SoK: Distributed Randomness Beacons

Kevin Choi*, Aathira Manoj*, and Joseph Bonneau*†
*New York University
†a16z crypto research

Abstract—**Motivated and inspired by the emergence of 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 trade-offs and characteristics. In this paper, we systematize the design of distributed randomness beacons (DRBs) as well as the cryptographic building blocks they rely on. We evaluate protocols on two key security properties, unbiasability and unpredictability, and discuss common attack vectors for predicting or biasing the beacon output and the countermeasures employed by protocols. We also compare 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 goal for millennia, dating at least to the earliest known use of dice around 3000 BCE. Today, public randomness is crucial to applications including gambling and lotteries [28], electronic voting [2], selecting parameters for cryptographic protocols [10], [74], leader election in proof-of-stake protocols [61], [72], and blockchain sharding [4], [73].

The concept of a *randomness beacon* was first formalized by Rabin [84] to describe an ideal service that regularly emits fresh random values that no party can manipulate or predict. 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 [56], [70] or random.org [66] 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 security of the beacon. In particular, it is straightforward to design a malicious beacon that outputs statistically random values which are predictable given a trapdoor. For example, given a semantically secure encryption scheme the underhanded beacon can simply use a secret key to encrypt a counter in each interval. Security of the underlying encryption scheme guarantees this is indistinguishable from random without access to the key, but completely predictable given the key.

Implicit Beacons. Another approach is to construct a beacon using publicly available implicit sources of entropy such as stock market data [41] or proof-of-work (PoW) blockchains like Bitcoin [16], [28], [78], [97]. 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.). These beacons are plausibly secure and low-cost in practice, but they still lack formal models of security. As a result, while we consider these important targets for future research, we will not discuss them in detail in this work.

Distributed Randomness Beacons. A natural approach to reduce trust in a centralized beacon is a multi-party *distributed randomness beacon* (DRB). DRB protocols are designed to remain secure and live despite some fraction of malicious participants. DRB protocols are typically epoch-based, producing fresh random output in each epoch.

The goal of this paper is to systematize current research on DRBs. We propose a general framework encompassing all DRB protocols in the landscape. To aid comparison and discussion of properties, we provide an overview of these protocols along with the 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 [21], and the definition of an ideal DRB in Section II. Section III introduces protocols using *delay functions* (verifiable delay functions [23] and timed commitments [27]), which offer the best fault tolerance (dishonest majority) and simplicity, assuming secure delay functions can be implemented in practice. In Section IV to VII, we introduce non-delay-based DRB protocols categorized by the number of nodes contributing *marginal entropy* (i.e. per-epoch randomness that is independently generated at a node level) in each epoch. 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 [83], [100], threshold secret sharing [34], [90], and threshold encryption [49]. Section VI covers committee-based protocols in which each epoch includes an extra committee selection step, after which only a committee (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 and Table I.

II. Preliminaries

A. System Model

We consider a system with a fixed set of n participants $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ (also called nodes). We may also write $\mathcal{P} = \{1, 2, ..., n\}$ for the purpose of algebraic formulations. Of the n, up to t nodes may be *faulty* (also called *malicious* or *Byzantine*) and engage in incorrect (arbitrary) behavior during a protocol run. 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 $\mathcal{A}$ which runs in probabilistic polynomial time (PPT). In particular, this means $\mathcal{A}$ cannot break standard cryptographic primitives such as hash functions, digital signatures, etc. For delay-based protocols, we also assume the adversary cannot compute delay functions in fewer than T time steps. 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 malicious nodes. Additionally, $\mathcal{A}$ has the power to perform a *grinding attack*, in which $\mathcal{A}$ privately precomputes and iterates through polynomially many combinations of inputs to an algorithm in order to derive a desirable output. By default, we assume a (t-limited) *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 (we assume a model where nodes remain corrupted once corrupted).

We denote our computational model's security parameter by λ. We call a function $\mathsf{negl}(\lambda)$ *negligible* if for all $c > 0$ there exists a λ_0 such that $\mathsf{negl}(\lambda) < \frac{1}{\lambda^c}$ 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 [13]. In the context of a distributed randomness beacon, we use τ to denote epoch number and Ω_τ to denote the *beacon output* (i.e. the distributed randomness output) in epoch τ. The *entropy-providing committee* denoted by $\mathcal{C}_\tau$ refers to a subset of nodes (hereafter called *entropy providers*) that proactively generate and provide marginal entropy in epoch τ.

B. Strawman Protocol: Rock-Paper-Scissors

Distributed randomness assuming perfect synchrony ($\Delta = 0$) is straightforward. Consider the following one-round protocol where each participant i broadcasts its *entropy contribution* (i.e. independently generated randomness) $e_i \xleftarrow{\$} \mathbb{Z}_p$ to every other participant at the same time. The protocol's random output Ω is calculated (via modular addition in $\mathbb{Z}_p$) as:

$$\Omega = \sum_{i=1}^{n} e_i \tag{1}$$

Repeating this protocol periodically would yield a DRB. This protocol is simple—in fact, it is essentially what humans approximate when playing rock-paper-scissors (with $e_i \xleftarrow{\$} \mathbb{Z}_3$). Under perfect synchrony, it 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 $\{P_1, P_2, P_3\}$ producing $\Omega = e_1 + e_2 + e_3$. 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 Ω to any value $\tilde{\Omega}$ by choosing $e_3 = \tilde{\Omega} - e_1 - e_2$. Effectively, the protocol cannot tolerate any malicious participants without perfect synchrony. Indeed, humans may attempt to cheat in rock-paper-scissors by quickly adjusting their play in reaction to what their opponent is playing.

C. Commit-Reveal

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

1) <u>Commit</u>. 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 [21], [44]. If participants sample e_i from a suitably large space, it is also secure to simply publish $c_i = H(e_i)$.
2) <u>Reveal</u>. Once all participants have shared their corresponding commitments, each participant P_i then opens its commitment by revealing the pair (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 Ω can be computed as in Equation 1. If any of the checks do not pass, however, the protocol aborts and outputs $\perp$.

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

D. Ideal Distributed Randomness Beacons

Clearly, a DRB should prevent any one participant from tampering with (e.g. predicting, biasing, or aborting) the output. We formalize the security properties of an ideal DRB as follows:

```
$\mathcal{G}^{\text{unbias}}_{\mathcal{A},b,\mathsf{DRB}}(\lambda)$
$pp_\tau \leftarrow \mathsf{DRB}^\tau.\mathsf{Setup}(\lambda)$
$\sigma \leftarrow \mathcal{A}_0(pp_\tau)$
$\Omega_{\tau,1} \leftarrow \mathsf{DRB}^\tau_{\mathcal{A}_1(\sigma)\cup(\mathcal{P}\setminus\mathcal{A}_1(\sigma))}$
$\Omega_{\tau,0} \leftarrow \mathsf{DRB}^\tau_{\mathcal{P}}$
$b' \leftarrow \mathcal{A}_2(\sigma, \Omega_{\tau,b})$
Return $b = b'$
```

Fig. 1: Security game for DRB unbiasability.

Definition II.1 (Ideal distributed randomness beacon)**.** An ideal distributed randomness beacon satisfies the following security properties:[1]

1) Unbiasability. A DRB is *unbiasable* if, for any PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, the adversary's advantage in the game depicted in Figure 1, given by

$$\mathsf{Adv}^{\text{unbias}}_{\mathcal{A},\mathsf{DRB}}(\lambda) = \left|\Pr\left[\mathcal{G}^{\text{unbias}}_{\mathcal{A},1,\mathsf{DRB}}(\lambda) = 1\right] - \Pr\left[\mathcal{G}^{\text{unbias}}_{\mathcal{A},0,\mathsf{DRB}}(\lambda) = 1\right]\right|$$

is negligible, i.e. $\mathsf{Adv}^{\text{unbias}}_{\mathcal{A},\mathsf{DRB}}(\lambda) \le \mathsf{negl}(\lambda)$.

2) Liveness. We define liveness [39] by requiring that the advantage of $\mathcal{A}$ denoted by $\Pr[\Omega_\tau = \bot]$ (i.e. the probability that the beacon output at the end of epoch τ is null) is negligible, given a DRB that runs among honest participants and $\mathcal{A}$.

3) Unpredictability. We define two types of unpredictability. Suppose a DRB's epoch τ starts at time $T_{\tau,0}$ and finalizes (Ω_τ becomes publicly available) at $T_{\tau,1}$ in the optimistic case (if every node is honest and online) and at $T_{\tau,2}$ in the worst case.

   - A DRB is *α-intra-unpredictable* ($\alpha > 0$) if $\mathcal{A}$ participating in DRB^τ (Figure 1) cannot predict any property of Ω_τ at time $T_{\tau,2} - \alpha$ with non-negligible advantage:

   $$\Pr[\Omega_\tau \in Y] \le \frac{|Y|}{2^{\ell_{\Omega_\tau}(\lambda)}} + \mathsf{negl}(\lambda)$$

   where Y denotes the set of possible values predicted by $\mathcal{A}$ and $\ell_{\Omega_\tau}(\lambda)$ denotes the bit-length of Ω_τ. It is *α-intra-predictable* otherwise.
   - A DRB is *β-inter-unpredictable* ($\beta \ge 1$) if $\mathcal{A}$ cannot predict any property of $\Omega_{\tau+\beta'}$ (as defined above) for any $\beta' \ge \beta$ before $T_{\tau,2}$ with non-negligible advantage.

Unbiasability. In the unbiasability game $\mathcal{G}^{\text{unbias}}_{\mathcal{A},b,\mathsf{DRB}}$ depicted in Figure 1, the adversary algorithm $\mathcal{A}_0$ first precomputes an advice string (σ) given the DRB's public parameters pp_τ in epoch τ (which may include the previous beacon output, a list of participants in epoch τ, the identity of the epoch leader, etc. depending on protocol specifications). Our definition is quite general in that the advice string may encode any biasing "strategy." This string is taken by $\mathcal{A}_1$, which then (statically and within epoch τ) corrupts up to t nodes in the honest node set $\mathcal{P} = \{1, \ldots, n\}$, interacts with $\mathcal{P} \setminus \mathcal{A}_1$, and outputs the beacon output $\Omega_{\tau,1}$. In contrast, $\Omega_{\tau,0}$ denotes the honest beacon output generated by $\mathcal{P}$ (and not involving $\mathcal{A}_1$). For both, the notation $\mathsf{DRB}^\tau_{\tilde{\mathcal{P}}}$ denotes a DRB in epoch τ with a participant set $\tilde{\mathcal{P}}$. Then $\mathcal{A}_2$ distinguishes the two cases given the same advice string. A DRB is unbiasable if the adversary can do so with negligible probability. Our definition includes the possibility of *private* biasing, e.g. the adversary biases the result in a way that requires a secret key to detect. This means that the biased beacon outputs can even appear pseudorandom (indistinguishable from uniform distribution) to an outsider, a notion not considered in previous works [19], [28], [46].

Liveness. While liveness implies a notion called *guaranteed output delivery* [46], [89] (all honest nodes receive Ω_τ at the end of epoch τ), it is in turn implied by unbiasability (due to the fact that $\Omega_{\tau,0}$ from Figure 1 is never null). For instance, commit-reveal, due to the last-revealer attack, does not satisfy liveness and thus is biasable.

Unpredictability. We note that β-inter-unpredictability (though in different forms) has been considered in previous works [19], [20] for $\beta \ge 1$, but we extend the notion to "$\beta = 0$" and explicitly consider α-intra-unpredictability in conjunction with β-inter-unpredictability, in order to exhibit variations across all possible DRBs. While neither implies the other, both are defined using the same probability formulation (involving $\Pr[\Omega \in Y]$), which has not been considered in previous works [19], [46], [89] and captures cases where (say) $\mathcal{A}$ predicts the first bit of Ω is 1 (in which case Y is a set of possible beacon output values whose first bit is 1), predicts the middle 10 bits make a prime number, etc. We also note that $\tilde{\beta}$-inter-unpredictability implies β-inter-unpredictability for all $\beta > \tilde{\beta}$, and that α-intra-unpredictability for all $\alpha > 0$ implies unbiasability. The reason is that biasability allows $\mathcal{A}$ in $\mathcal{G}^{\text{unbias}}_{\mathcal{A},b,\mathsf{DRB}}$ to make a prediction towards its biasing strategy encoded in σ, implying that there exists $\alpha > 0$ such that the protocol is α-intra-predictable.

III. Delay-Based Protocols

One way to prevent the last-revealer attack is to compute Ω_τ using a *delay function* after combining each node's entropy contribution. If the delay is suitably long, no participant can predict what effect a potential contribution will have on the output before its contribution must be published. Typically, a *verifiable delay function* (VDF) [23], [24] is used to accomplish this while maintaining efficient verifiability of the result.

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 significantly less time to verify. It is 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 delay parameter T.

[1]We present a game-based security definition here, as it is the most commonly used in the literature. Other formulations, such as ideal functionalities as used in UC-security [33], are possible.

- $\mathsf{Eval}(pp, x) \rightarrow (y, \pi)$ computes y in T sequential steps and (optionally) a proof π, given pp and an input x.
- $\mathsf{Verify}(pp, x, y, \pi) \rightarrow \{0, 1\}$ outputs 1 if y is the unique correct evaluation of the VDF on input x and 0 otherwise.

Two well-known VDF proposals, due to Pietrzak [82] and Wesolowski [98], make use of the (believed) inherently sequential nature of repeated squaring in a group of unknown order. VDFs can also be constructed from incrementally verifiable computation [23], [71] or isogenies [48]. 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 [88]).

A. Modifying Commit-Reveal

The Unicorn protocol [74] 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 modern VDF in place of Sloth to achieve faster (constant time) verification time for a given delay parameter. We refer to VDF-enhanced Unicorn as *Unicorn++*. It runs as follows:

1) **Collect.** Every participant P_i broadcasts its entropy contribution e_i between time t_0 and t_1 (assuming synchronized clocks). At t_1, they are combined into $x_\tau = H(e_1, \ldots, e_n)$.
2) **Evaluate.** Some party evaluates the VDF with x_τ and a chosen delay parameter T (part of pp) via
$$y_\tau, \pi_\tau = \mathsf{VDF.Eval}(pp, x_\tau)$$
such that $\Omega_\tau = H(y_\tau)$, which is posted and can be efficiently verified by any observer using π_τ via $\mathsf{VDF.Verify}$.

As long as T is longer than the duration of $t_1 - t_0$, 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 is possible to outsource this computation, even in a decentralized manner [94], as it does not matter for security who evaluates since VDFs are deterministic and verifiable.

We note a variation of above [28], [30], replacing or bolstering the participant entropy contributions with stock prices [41] or PoW blockchain headers [16], [78], [97] (which are otherwise susceptible to manipulation) to supply x_τ 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. Adding Recovery to Commit-Reveal

Another way to modify commit-reveal is to leverage a different class of delay functions called *timed commitments* [27] in place of regular commitments used in commit-reveal. The idea is simple: timed commitments are commitments with an additional slow recovery process (the committed value can be recovered in T sequential steps but not before) in case the committer withholds.

Definition III.2 (Timed commitment)**.** A *timed commitment* is a commitment with an additional algorithm whereby the committed value can be recovered (or *forced open*) in T sequential steps but not before:

- $\mathsf{ForceOpen}(c) \rightarrow (x, r_0)$ outputs the committed value (x, r_0) in T sequential steps given a commitment $c = \mathsf{Com}(x, r_0)$.

This recovery process avoids the last-revealer attack, as a withholding participant's contribution can be recovered. Thus, the resulting distributed randomness protocol can be seen as a "commit-reveal-recover" protocol (Section V). This approach was suggested by Boneh and Naor [27] though not specified in detail. Thyagarajan et al. [93] proposed leveraging *homomorphic* timed commitments to combine contributions and only require one delay function even if *all* participants refuse to open (rather than one computation per withholder). Bicorn [40] realizes the above logic in a simple, efficient DRB with comparable overhead to basic commit-reveal.

The advantage of Bicorn over Unicorn++ is that in the optimistic case (where every participant is honest) the protocol has no delay, analogous to a simple commit-reveal. Context may be important to consider if the optimistic case is unlikely to occur (e.g. due to poor network conditions or too many participants), in which case Unicorn++ is simpler and also requires only one delay function computation.

All of the protocols in this family share a fundamental predictability downside: if only one participant (or a colluding coalition) withholds while all others reveal, then the attacker(s) can simulate the optimistic case and learn Ω_τ early. As this implies T-intra-unpredictability (with delay parameter T), a protocol should consider Ω_τ to be potentially available to adversaries as of $T_{\tau,1}$ (optimistic case), even if it is not publicly known until $T_{\tau,2}$ (worst case).

C. Chain of VDFs

A disadvantage of above approaches is that each epoch may require consensus [37] on inputs to the delay function, incurring communication cost. Also, the rate at which beacon outputs are generated is limited by T (by default in Unicorn++ and in Bicorn's $\mathsf{ForceOpen}$ case). RandRunner [88] 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-epoch consensus while allowing each epoch's duration to be independent of T in the optimistic case.

Namely, RandRunner uses Pietrzak's VDF [82], where knowledge of a *trapdoor* allows an efficient evaluation of a VDF without T sequential steps unlike $\mathsf{VDF.Eval}$ (but with T steps otherwise). In its setup, each P_i broadcasts pp_i (each corresponding to a different VDF per participant). Then the idea is that, in each epoch, $H(\Omega_{\tau-1})$ is input to the VDF of the epoch leader—selected via either *round-robin* (i.e. taking turns in some permuted order) or *random selection* (i.e. using $\Omega_{\tau-1}$ as seed), discussed in Section VI-A1. In the optimistic case, the honest leader (the only one that knows its trapdoor) efficiently evaluates and publishes the VDF output as the beacon output while in the faulty case (if the leader withholds), others can evaluate the same value albeit more slowly.

Thus, RandRunner optimistically generates each beacon output rapidly with only $O(n)$ communication complexity. Adversarial leaders can increase the epoch duration to T and the communication complexity to $O(n^2)$.[2] The protocol exhibits two other beneficial properties. First, liveness is retained even with a dishonest majority and when network connectivity breaks down completely, as one can simply compute the beacon outputs over time via VDF.Eval. Second, it is impossible to bias the beacon once bootstrapped such that even the strongest adversary can only predict but not bias. The trade-off is that RandRunner can never achieve the ideal 1-inter-unpredictability property due to the existence of leaders that can withhold and adversaries with higher compute power. In other words, β-inter-unpredictability can be achieved only with $\beta > 1$, though β can be bounded [88] with assumptions.

IV. Commit-Reveal-Punish

Another approach to preventing last-revealer attacks is *commit-reveal-punish*, which assumes that all participants are rational entities and use financial penalties to discourage withholding. This requires some form of *escrow* (e.g. smart contracts on Ethereum [99]) to collect initial deposits from the participants 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 [6], [17], [83] or by tolerating some number of withholding participants via threshold commit-reveal [100]. These two approaches are summarized below.

A. Enforcing Every Reveal

Extending basic commit-reveal, RANDAO [83] implements commit-reveal-punish in a straightforward way. Each participant is required to deposit coins at the time of commitment, which are slashed if that participant withholds its value during the reveal phase. The drawback of this approach is twofold. First, honest failures are indistinguishable from withholding and must also be punished. Attackers might exploit this by trying to block victim nodes from publishing (e.g. by bidding up the price of gas in a smart contract platform). Second, a high deposit of $O(n^2)$ coins is required to ensure fairness [6], [17]. Thus, RANDAO is suitable only if participants are expected to be highly available and possess an ample supply of coins. Practical deployment also requires 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 is not known in advance.

[2]We assume consensus at a protocol level incurs $O(n^2)$ communication cost (bitwise) by default.

B. Rational Threshold Commit-Reveal

Economically Viable Randomness (EVR) [100] 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 Ω_τ) 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 coalition would inform.

EVR requires multiple cryptographic building blocks (introduced here, as they are used in other DRBs throughout the paper). EVR uses Escrow-DKG [101], an extension of DKG (distributed key generation) [59], [63], [65], [81], 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)**.** The dealer in a (t,n)-*secret sharing* shares a secret to n participants such that any subset of $t+1$ or more participants can reconstruct the secret, but smaller subsets cannot.

Definition IV.2 (Shamir's secret sharing)**.** A concrete realization of (t,n)-secret sharing, *Shamir's secret sharing* [91] 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* (Appendix H1), but smaller subsets cannot. In this paper, we use (t,n)-secret sharing and Shamir's secret sharing interchangeably.

Definition IV.3 (Verifiable secret sharing)**.** *Verifiable secret sharing* (VSS) [54], [80] protects a (t,n)-secret sharing scheme against a malicious dealer sending incorrect shares by enabling verification of each share. VSS can be described by the following algorithms (see Appendix A for details):

- $\mathsf{Setup}(\lambda) \rightarrow pp$ 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 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 $\mathsf{ShareVerify}$.

Definition IV.4 (Distributed key generation)**.** A *distributed key generation* (DKG) [59], [81] allows n participants to collectively generate a *group public key* (with an implicit *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).

- $\mathsf{DKG}(1^\lambda, t, n) \rightarrow (sk_i, pk_i, pk)$ 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 implicit group secret key sk given security parameter 1^λ, t, and n.

The intuition behind DKG is that it allows $t+1$ (but not less) out of n nodes to jointly *use* sk via Lagrange interpolation without necessarily *knowing* sk. See Appendix B for details. Unlike secret sharing schemes, one DKG setup can lead to an unlimited number of usages, as the group secret key is never computed explicitly during normal use.

The crux of EVR is adopting Escrow-DKG. It is first different from a classic DKG in that an escrow platform Escrow (e.g. smart contract) disincentivizes misbehavior. Second and more importantly, Escrow-DKG's implicit group secret key sk is in fact the beacon output that becomes computed and publicized (unlike traditional DKGs in which the group secret key is never revealed). EVR proceeds in four phases:

1) **Setup.** Every participant registers by depositing 1 coin per secret (i.e. entropy contribution), and Escrow accordingly sets the threshold parameter $t = 2n/3$ required for Escrow-DKG. It also sets the *illicit profit bound* (i.e. extra profit an adversary can gain as a result of using EVR's output as opposed to an ideal beacon) to $n - t = n/3$ and the *informing reward* to 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 Ω_τ is incentivized to inform Escrow to earn an informing reward obtained via collective punishment.
4) **Reveal.** $\Omega_\tau = sk$ is reconstructed once $t+1$ (or more) participants reveal their sk_i's. Initial deposits are returned after verification by Escrow. If Ω_τ is not reconstructed by the end, Escrow also initiates collective punishment.

While a malicious node in EVR might withhold to abort the protocol during Reveal or collude to learn Ω_τ before Reveal, security comes from the fact that both are disincentivized. First, setting the illicit profit bound to $n-t$ makes withholding unprofitable, as the $n-t$ or more participants needed to successfully abort EVR would earn an amount bounded by the illicit profit bound at the cost of losing their deposits. This prevents biasability. Second, setting the informing reward to n makes informing more profitable than any illicit profit. Thus, any coalition of nodes colluding to preemptively learn Ω_τ is economically unstable, as all nodes are incentivized to defect and act as 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 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 [100]).

V. COMMIT-REVEAL-RECOVER

Without using escrow to enforce desired behavior, *commit-reveal-recover* variants 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 this creates an inherent trade-off: while a smaller value of t helps tolerate more honest faults, it also means that a smaller subset can collude to predict the beacon output in advance.

A. From Threshold Secret Sharing

Commit-reveal-recover variants often use *publicly verifiable secret sharing* (PVSS) [34], [90] 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) is a VSS with the following additional algorithms to enable public verification: $\mathsf{PVSS.KeyGen}$ (which generates secret-public key pair per participant), $\mathsf{PVSS.Enc}$ (for public-key encryption), and $\mathsf{PVSS.Dec}$ (decryption). The idea is that $\mathsf{PVSS.ShareGen}$ uses above to encrypt and decrypt PVSS shares and also to generate public proofs, e.g. non-interactive zero-knowledge (NIZK) proofs. Then $\mathsf{PVSS.ShareVerify}$ can be run by anyone (not just the participants). See Appendix C 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 Ω_τ via Lagrange interpolation[3] ($\mathsf{PVSS.Recon}$) 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 a 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. The trade-off is additional communication cost, which amplifies if $O(n)$ Lagrange interpolations need to take place. Scrape [34] adopts this technique.

[3]In this paper, we assume one Lagrange interpolation at a protocol level incurs $O(n^2)$ and $O(n^3)$ communication cost (bitwise) in the optimistic and worst cases, respectively.

Scrape. With its own PVSS scheme [34] designed for efficiency, Scrape runs as follows after the initial generation ($\mathsf{PVSS.KeyGen}$) of (sk_i, pk_i) for each of the n nodes.

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

Note that Scrape, in the optimistic case (without Recover), is just 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 [35] provides an improved amortized communication complexity of $O(n)$ per beacon output by generating a batch of $O(n^2)$ beacon outputs per epoch (as opposed to one). This is achieved by two techniques: packed Shamir secret sharing and linear t-resilient functions [35]. As packed Shamir secret sharing allows sharing of $O(n)$ secrets (as opposed to one) per instance while linear t-resilient functions allow outputting of $O(n)$ values (as opposed to one) in the final randomness aggregation step, each of these techniques multiplicatively contributes $O(n)$ to the number of beacon outputs produced per epoch.

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, all nodes can distribute their PVSS shares (*share*), perform Lagrange interpolation per secret for a total of $O(n)$ times (*reconstruct*), and aggregate the interpolated secrets to output Ω_τ (*aggregate*). While the resulting *share-reconstruct-aggregate* saves a round of communication (Reveal) 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 epochs will require recovery due to faulty participants. RandShare [92] uses this technique.

3) Share-Aggregate-Reconstruct: Another alternative is to harness the homomorphic property of PVSS, due to which only one, as opposed to $O(n)$, Lagrange interpolation reconstructs Ω_τ if nodes perform *aggregate* before *reconstruct*, hence *share-aggregate-reconstruct*. SecRand [64] uses this technique to reduce communication overhead accordingly.

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 [49] (which does not rely on PVSS), 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 require a DKG, which may be run multiple times to refresh keys. In this section, we summarize how a protocol like HERB [39] uses threshold encryption to construct a DRB.

The main idea is simple: n participating nodes run a DKG, encrypt their respective entropy contributions under the group public key pk, homomorphically combine all ciphertexts into one group ciphertext, and jointly (requiring at least $t+1$ nodes) decrypt the group ciphertext via one Lagrange interpolation. Effectively, the DKG is what makes this possible, as it allows the usage of sk (to decrypt a ciphertext under pk) without knowing it (recall from Definition IV.4).

HERB achieves a communication complexity of $O(n^2)$ and $O(n^3)$ in the optimistic and worst cases, respectively. 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.

VI. Committee-Based Protocols

All aforementioned commit-reveal variants include every node in the entropy-providing committee $\mathcal{C}_\tau$ for every epoch. Incorporating marginal entropy from all nodes scales poorly with large numbers of participants, and hence a natural optimization is to select a smaller subset of nodes to contribute marginal entropy in each epoch (i.e. reduce $|\mathcal{C}_\tau|$).

In this section, we consider DRBs that are committee-based, with $\mathcal{C}_\tau$ such that $1 \leq |\mathcal{C}_\tau| < n$. Committee-based protocols proceed in two steps: *committee selection* and *beacon output generation*. As the names suggest, $\mathcal{C}_\tau$ is agreed upon during committee selection while the beacon output Ω_τ 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 epoch.

A. Step 1. Committee Selection

The first step of a committee-based DRB involves selecting $\mathcal{C}_\tau$ in a way agreed 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 $\mathcal{C}_\tau$.

Round-Robin (RR). A simple example is *round-robin* (RR), in which nodes simply take predetermined turns being selected. 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 $i \equiv \tau \pmod n$. Protocols like BRandPiper [19] (in which the epoch leader is the only active entropy provider) adopt RR as their leader selection mechanism due to its innate fairness property [8] (also known as chain quality [58] 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 $\Omega_{\tau-1}$) to derive $\mathcal{C}_\tau$. In HydRand [89] and GRandPiper [19], $\mathcal{C}_\tau$ consists of a node $i \equiv \Omega_{\tau-1} \pmod{\tilde{n}}$ where $\tilde{n}$ is the number of eligible nodes. Ouroboros [72] uses a similar process called follow-the-satoshi [18], [72] which selects nodes weighted by stake.

Randomized selection means that some nodes may, in theory, never be selected and therefore never be able to contribute entropy. A more serious concern is that an adversary can attempt to bias Ω_τ via grinding in order to bias $\mathcal{C}_{\tau+1}$ (which can bias $\Omega_{\tau+1}$ and so on). 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 trade-off of RR is that denial-of-service (DoS) becomes indefinitely possible (for all epochs τ for $\tau > \tilde{\tau}$ given $\Omega_{\tilde{\tau}}$) since each committee is publicly known in advance. All in all, RR gains unbiasability (due to determinism) at the cost of indefinite DoS, while RS reduces the risk of DoS, i.e. that only for epoch $\tau + 1$ (due to randomization given Ω_τ), at the cost of potential grinding attacks.

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 an epoch leader (either by RR or RS) while the second involves selection of $\mathcal{C}_\tau$ 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 $|\mathcal{C}_\tau|$ needs to be greater than t so that a malicious leader wouldn't be able to choose $\mathcal{C}_\tau$ maliciously. RandHound [92], SPURT [46], and OptRand [20] demonstrate such LS.

- RandHound. As instantiated in RandHerd [92], 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 $H(C \,\|\, pk_i)$ given a public configuration parameter C (assuming its randomness) such that node $\mathrm{argmin}_i\, H(C \,\|\, pk_i)$ 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 and OptRand. Unlike RandHound, SPURT and OptRand adopt RR as the first step of LS, with nodes simply taking turns as an epoch leader. Then the leader chooses $\mathcal{C}_\tau$ based on received encrypted messages.

Given an underlying DRB that utilizes a leader to orchestrate communication, 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 a *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 $\mathcal{C}_\tau$ (i.e. has won a lottery to serve on the committee). The general formulation of a private lottery is 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), which 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 an adversary can 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 $f_{priv}(\cdot)$.

A prime example of a private lottery is one based on VRFs (verifiable random functions [51], [77]), which output a pseudorandom value (as well as a proof for verification) given secret key sk and input x (see Appendix D). Most notably, Algorand [61] uses VRFs to realize a lottery every epoch. Quite naturally, one's private input to $\mathsf{VRF}_{sk}(\cdot)$ is its secret key. The lottery[4] is given by

$$\mathsf{VRF}_{sk}(\Omega_{\tau-1} \,\|\, role) < target$$

where $role$ is some parameter specific to Algorand. As both $\Omega_{\tau-1}$ 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 in advance. Similar private lotteries are used by Ouroboros Praos [47], Caucus [8] (where a hash chain replaces VRFs), and NV (from Nguyen-Van et al. [79]). See Table II for details.

Private lotteries provide two notable benefits: resilience to DoS attack (due to its property of delayed unpredictability [8] 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 permis-

[4]While there are multiple versions of Algorand, we consider its first version, as they do not differ fundamentally.

sionless 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 concrete committee $\mathcal{C}_\tau$, the next step is to output Ω_τ. While a typical commit-reveal-recover run among nodes in $\mathcal{C}_\tau$ may be sufficient to realize a DRB, other approaches provide different trade-offs. We classify variants which require *fresh* (independently generated on the spot) per-node entropy (contribution) and those which combine previous beacon output with *precommitted* (independently generated but precommitted, hence ungrindable) per-node entropy.

1) Fresh Per-Node Entropy: Beacon output generation approaches involving fresh (also referred to as true randomness [36], [46] as opposed to pseudorandomness) per-node entropy are typically commit-reveal-recover variants from Section V. Some protocols in this family include the following:
Share-Reconstruct-Aggregate. In Ouroboros, nodes in $\mathcal{C}_\tau$ (i.e. slot leaders of epoch τ) perform a RandShare-style share-reconstruct-aggregate using PVSS to output Ω_τ. RandHound uses a similar approach, facilitated by an epoch leader.
Share-Aggregate-Reconstruct. In SPURT, OptRand, and BRandPiper, nodes in $\mathcal{C}_\tau$ perform a SecRand-style share-aggregate-reconstruct to output Ω_τ. BRandPiper has a twist: it utilizes the idea of buffering PVSS shares in advance. While there is one entropy provider per epoch, n secrets (one from each node) are combined such that it provides the ideal 1-inter-unpredictability property as opposed to t-inter-unpredictability (as in HydRand or GRandPiper). The trick is that each epoch leader generates n fresh secrets (entropy contributions) that become combined with others' secrets in the next n epochs, respectively. In an epoch, one node distributes $O(n^2)$ PVSS shares (buffered by other nodes) whereas, in a typical share-aggregate-reconstruct like SPURT and OptRand, each of $O(n)$ nodes distributes $O(n)$ PVSS shares (with no buffering).
From Threshold Encryption. Similar to HERB, entropy providers in NV [79] 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 Ω_τ. Naturally, this Requester version of NV can be modified into what we call *NV++*, which differs from NV in two ways. First, nodes in $\mathcal{C}_\tau$ (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 $\mathcal{C}_\tau$ (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).

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

- HydRand and GRandPiper. Each epoch, an entropy provider (i.e. epoch leader) in HydRand commits its entropy that becomes opened (revealed) in the next epoch it is selected as the leader again. In other words, the epoch leader's precommitted entropy $e_{\tilde{\tau}}$ from its last epoch $\tilde{\tau}$ of leadership is the one that becomes combined with $\Omega_{\tau-1}$ in the form of $h^{e_{\tilde{\tau}}}$ to generate

$$\Omega_\tau = H(\Omega_{\tau-1} \parallel h^{e_{\tilde{\tau}}})$$

  while PVSS recovery is used in case the leader fails to open $e_{\tilde{\tau}}$ in epoch τ. Notable in HydRand is the fact (achieving ungrindability of $h^{e_{\tilde{\tau}}}$) that one honest node must be present in any $t+1$ consecutive epochs due to the requirement that a leader cannot gain another leadership in the next t epochs. 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 epoch 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 given by

$$\Omega_\tau = \mathsf{VRF}_{sk}(\Omega_{\tau-1} \parallel \tau)$$

  combining the previous output $\Omega_{\tau-1}$ 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_τ in epoch τ) from an entropy provider's private hash chain in Caucus corresponds to that node's precommitted entropy. The beacon output

$$\Omega_\tau = h_\tau \oplus \Omega_{\tau-1}$$

  naturally follows its committee selection mechanism $H(h_\tau \oplus \Omega_{\tau-1}) < target$. See Table II for details.

VII. Protocols With No Marginal Entropy

It is possible to devise a protocol where no node contributes any marginal entropy ($|\mathcal{C}_\tau| = 0$) as the beacon runs, producing the beacon output solely via cryptographic pseudorandomness. This can improve efficiency as no node needs to generate and communicate fresh entropy. However, the beacon becomes predictable forever (β-inter-unpredictability fails for all β) if compromised (perhaps undetectably).

Such a DRB can be based on a *distributed verifiable random function* [31], [32], [57] (DVRF, also known as threshold VRF or TVRF [36]). 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-epoch VRF output (as well as its proof), as if the computation involves one master node with sk.

Definition VII.1 (Distributed verifiable random function)**.** A *distributed verifiable random function* (DVRF) is a VRF where any $t+1$ out of n nodes can jointly compute a pseudorandom output while any t Byzantine nodes cannot. It can be described by the following 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 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

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

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

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

$$\Omega_\tau = \mathsf{VRF}_{sk}(f(\Omega_{\tau-1}))$$

There is no marginal entropy contributed by the participants, as f typically takes a form resembling $f(\Omega_{\tau-1}) = H(\tau \parallel \Omega_{\tau-1})$. The ideal 1-inter-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) [77] is equivalent to a VRF, a unique digital signature (which is a VUF [51]) can be made into a DVRF by computing its threshold variant [22] and hashing the output (assuming a hash function as a random oracle [13]). Dfinity [32] and drand [1] (while differing slightly in minor details) both use the BLS signature scheme [26] to realize a DRB as

$$\Omega_\tau = H(\mathsf{Sign}_{sk}(\tau \parallel \Omega_{\tau-1}))$$

where $\mathsf{Sign}_{sk}(\cdot)$ is a threshold BLS signature computed by at least $t+1$ nodes with sk as the implicit group secret key generated via DKG. The actual computation involves combining of partial signatures computed using sk_i (see Appendix H2).

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

- RandHerd [92]. Two modifications are made in RandHerd. First, a form of "sharding" into groups (each of size c) allows reduction of overall communication complexity. Second, the underlying signature scheme used is Schnorr instead of BLS. Each Ω_τ is a threshold Schnorr signature on message $m = t_\tau$ where t_τ denotes the timestamp at the epoch's beginning. As m can technically be chosen (and thus biased) by the leader, one simple improvement can be setting $m = \tau \parallel \Omega_{\tau-1}$ a la Dfinity or drand.
- DDH-DRB and GLOW-DRB [57]. These two DRBs modify Dfinity-DVRF (i.e. each epoch of Dfinity) and explore space-time trade-off by using DLEQ NIZKs (Appendix H3) in place of pairing equations (Appendix G). See Appendix E and F for details.
- Strobe [12]. In Strobe, threshold RSA decryption conceptually replaces the threshold BLS process. Note that RSA decryption and BLS signature are similar in that one needs a secret value (decryption key d and signer's sk, respectively) to perform the respective operations. The analogy is that threshold BLS distributes sk in a threshold manner (via DKG) while threshold RSA distributes d (not via DKG). The difference is that the latter requires a trusted setup (knowledge of factors of N, the RSA modulus), and this is Strobe's main downside. Its benefit of using threshold RSA decryption is equally clear: the simple relationship $\Omega_\tau^d = \Omega_{\tau-1} \pmod N$ allows efficient generation of all past beacon outputs (a novel property of a DRB called *history generation*).

VIII. Discussion

A. Relation to Collective Coin Flipping Protocols

Conceptually, distributed randomness is not a new line of research. Dating to Blum's classic work on coin flipping over the phone [21], distributed randomness has been much researched, albeit in a different context as elaborated below. Namely, Ben-Or and Linial in their seminal work [14], [15] introduced the *full information* model for the *collective coin flipping* problem, in which n participants with unbounded computational power communicate only via a single broadcast channel to generate a common random bit (such that an honest majority is required [29], [86] and thus assumed). Numerous works exist in this setting, largely classifiable into different types of adversaries dealt with: static [3], [5], [15], [29], [53], [68], [85], [86], adaptive [15], [50], [62], [67], [69], [75], and variants of adaptive [7], [43], [52], [62], [76]. See [67], [69] for this line of research.

Overall, these works concern upper and lower bounds on corruption threshold, bias (deviation from coin flipping probability 1/2), and round complexity, all of which provide interesting theoretical insights. Nonetheless, these bounds are often asymptotic (hence not practical) and are grounded in a more lax definition of security where it is sufficient that bias is bounded (but can still be nontrivial). This is in contrast to the modern literature on DRBs considered in this paper, which aims to design protocols which are as unbiasable as possible, output multiple bits per epoch, have explicit round complexity and fault tolerance, and assume computationally bounded adversaries in a cryptographic setting as well as point-to-point communication channels in the first place.

Outside the full information model (such that cryptography is allowed), the well-known lower bound by Cleve [42] states that for any r-round coin flipping protocol there exists an efficient adversary controlling half or more of the participants that can bias the output by $\Omega(1/r)$. In other words, it is impossible to have an unbiasable coin flipping protocol with a dishonest majority.

While this may seem to contradict the fault tolerance of delay-based DRBs from Section III, we note that delay functions help circumvent Cleve's impossibility result in the following two ways. First, timed commitments allow recovery of a value that is withheld (either due to honest or Byzantine fault) and lost from an honest node's perspective. In Cleve's proof, the notion of a "default bit" is used in such withholding situation whereas timed commitments effectively deprecate this default bit mechanism, sidestepping the proof logic.

Second, an implicit assumption in Cleve's model is that a Byzantine node is capable of grinding through possibilities to its liking and can arbitrarily choose which messages to output based on inputs from other nodes that are honest. However, VDFs limit this capability such that it is not possible for even a dedicated attacker to grind through possibilities in an attempt to fix an output of some computation if a VDF is applied. As a result of above, delay-based DRBs are able to enjoy both the highest fault tolerance and unbiasability without violating any classical lower bounds. In a similar vein, Bailey et al. [11] recently showed that VDFs can circumvent some classical impossibility results for general multiparty computation (MPC), of which DRBs are a special case.

B. Withholding Attacks

In a withholding attack, an adversary can influence the outcome by not publishing some information. Any leader-based protocol is vulnerable due to the inherent reliance on a leader's availability, affecting the protocol's liveness (as well as unbiasability and potentially unpredictability). Any protocol with a private lottery is also fundamentally vulnerable.

1) **Protocols with a leader.** RandHound, RandHerd, and SPURT suffer from the leader unavailability issue in case the leader withholds such that their liveness is affected and a beacon output can be aborted.[5] A fallback is needed if 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, as there is no accountability. There are two possible remedies. First, we can require all participants to post their lottery outputs every single epoch 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, SSLE (single secret leader election) [25] can be used to guarantee one winner per epoch, enabling detection of withholding. The guarantee of one winner as opposed to the expectation of one winner is what differentiates SSLE. While this makes withholding obvious, it does not prevent withholding by itself, nor does it detect *who* withholds in the case of withholding. Designing efficient SSLE protocols remains an open and active research area.

[5]In contrast to a leader in SPURT, that in RandHound or RandHerd can abort after seeing the beacon output in plaintext.

C. Adaptive Security

A DRB is *adaptively secure* if its security properties remain unaffected against an adaptive adversary instead of a static one. Here, we discuss a way to remedy adaptive attacks.

1) **Requiring lottery winners (in protocols with private lotteries) to broadcast marginal entropy and proof of selection into $\mathcal{C}_\tau$ in the same message.** While some private lottery schemes may involve less than or equal to t entropy providers per epoch, the fact that one message (per entropy provider) comprises both announcement of winning the lottery and provision of marginal entropy allows adaptive security (e.g. as in Algorand). By the time an adversary knows which nodes to corrupt adaptively in an epoch (after the nodes reveal their identity as entropy providers), there is no extra step left to be corrupted. A similar line of research has recently emerged in the MPC literature in a model called YOSO (You-Only-Speak-Once) [60], inspired by Algorand.

On the one hand, it is important for the sake of adaptive security that there is no central point of dependency in any step of a protocol. Otherwise, participating nodes depend on the leader (functioning as either an orchestrator or an entropy provider) such that an adversary can adaptively corrupt such leaders. On the other hand, we note that a proof of adaptive security does not follow immediately from a protocol's lack of leaders and can in fact be tricky to show. Only recently has it been shown that threshold BLS is adaptively secure [9].

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 observer (not participating in the protocol) 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 [34] run. *Communication Complexity*

TABLE I: DRB Comparison

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

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 [34] is used as the default PVSS scheme. Max Damage refers to the maximum damage possible when $n-1$ adversarial nodes cooperate (see Section VIII-D). ^{r} In a non-rushing adversary model, max damage would be predict rather than bias. * Each committee consists of a leader by default or by expectation. † PBB (public bulletin board) is assumed.
‡ T denotes VDF's delay parameter, and verification of Pietrzak's VDF is logarithmic in T. § $\beta = t$ for RandRunner's β-inter-unpredictability assuming a dishonest minority without any computational advantage. See [88] for more scenarios.

concerns bitwise point-to-point communication among nodes by default. Alternatively, we consider a *public bulletin board* (PBB) (e.g. blockchain) as a reliable information exchange medium in protocols where it is intrinsic. In a PBB model, we consider both the bitwise writing cost (amount of data posted to PBB) and reading cost (by all nodes where each node only reads data relevant to it). In the absence of PBB, Byzantine consensus [37] incurs $O(n^2)$ cost per decision by default.

Max Damage refers to the maximum damage possible when $n-1$ rushing [59] adversarial nodes cooperate. The reason for this column is to observe the consequence of when the honest majority assumption fails, which has happened in practice (e.g. a $625 million Axie Infinity's Ronin hack in 2022 [87]). A rushing adversary may delay sending messages until *after* reading messages sent by all honest nodes in a given round of communication. There can exist a separation between what rushing versus non-rushing adversaries can do especially in protocols from Section V: if one can generate its entropy contributions after seeing (or otherwise simultaneously with) all the honest nodes' entropy contributions, then biasing (or otherwise predicting) is possible. The same fundamental reasoning applies to (say) Ouroboros, where slot leaders (i.e. entropy providers) communicate in sequential slots (and hence the adversary can reconstruct an honest node's entropy contribution before generating its own). 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. PVSS.KeyGen or for private lottery schemes) and VDF.Setup incur $O(n^3)$ recovery cost without PBB (and $O(n^2)$ with PBB) while we conservatively assume each DKG incurs $O(n^4)$ recovery cost (although it can be optimized [65]). Finally, we note that it is possible to employ multiple DRBs as subprotocols to a multi-tiered DRB (an approach taken by Mt. Random [36]) in order to combine and take advantage of various DRB properties at the same time.

IX. Concluding Remarks

Our systematization highlights important insights both for practitioners and researchers. Based on our comparative framework, we would advise practitioners planning to deploy a DRB to consider the following high-level guidelines:

- Delay-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 be able to compute a VDF (preferably by running specialized hardware), which can add latency to 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.
- 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.
- Vulnerable to withholding, protocols based on private lotteries can generate biased outputs. Though the guarantee of a single lottery winner every epoch 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.
- Most existing DRBs assume synchronous communication, which may fail in practice. Extending protocols to handle asynchrony is an important challenge.
- Most papers today use game-based security definitions. Universal Composability (UC) security proofs [33] could be a useful tool for proving more robust and modular security results.
- Finally, there is a gap between the systems-based literature on DRBs and the traditional cryptographic literature on randomness extractors [95], [96], 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.

Acknowledgments

The authors would like to thank our anonymous shepherd, as well as Ling Ren, Zachary DeStefano, Philipp Schindler, Bernardo David, and Adithya Bhat for helpful conversations. Kevin Choi and Joseph Bonneau were supported by DARPA under Agreement No. HR00112020022. Joseph Bonneau was also supported by a16z crypto research and was previously on the technical advisory board of Algorand. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government, DARPA, a16z, Algorand or any other supporting organization.

References

[1] “Drand,” https://drand.love/.

[2] B. Adida, “Helios: Web-based open-audit voting,” in *USENIX Security*, 2008.

[3] M. Ajtai and N. Linial, “The influence of large coalitions,” *Combinatorica*, vol. 13, no. 2, 1993.

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

[5] N. Alon and M. Naor, “Coin-flipping games immune against linear-sized coalitions,” *SIAM Journal on Computing*, vol. 22, no. 2, 1993.

[6] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, “Secure Multiparty Computations on Bitcoin,” in *IEEE Security & Privacy*, 2014.

[7] J. Aspnes, “Lower bounds for distributed coin-flipping and randomized consensus,” *Journal of the ACM (JACM)*, vol. 45, no. 3, 1998.

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

[9] R. Bacho and J. Loss, “On the adaptive security of the threshold BLS signature scheme,” in *ACM CCS*, 2022.

[10] T. Baignères, C. Delerablée, M. Finiasz, L. Goubin, T. Lepoint, and M. Rivain, “Trap Me If You Can – Million Dollar Curve,” 2015.

[11] B. Bailey, A. Miller, and O. Sattath, “General partially fair multi-party computation with vdfs,” Cryptology ePrint Archive, Paper 2022/1318, 2022.

[12] D. Beaver, K. Chalkias, M. Kelkar, L. K. Kogias, K. Lewi, L. de Naurois, V. Nicolaenko, A. Roy, and A. Sonnino, “STROBE: Stake-based Threshold Random Beacons,” 2021.

[13] M. Bellare and P. Rogaway, “Random oracles are practical: A paradigm for designing efficient protocols,” in *ACM CCS*, 1993.

[14] M. Ben-Or and N. Linial, “Collective coin flipping, robust voting schemes and minima of banzhaf values,” in *FOCS*, 1985.

[15] ——, “Collective coin flipping,” *Advances in Computing Research*, 1989.

[16] I. Bentov, A. Gabizon, and D. Zuckerman, “Bitcoin beacon,” *arXiv preprint arXiv:1605.04559*, 2016.

[17] I. Bentov and R. Kumaresan, “How to use Bitcoin to design fair protocols,” in *CRYPTO*, 2014.

[18] 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.

[19] A. Bhat, N. Shrestha, A. Kate, and K. Nayak, “RandPiper-Reconfiguration-Friendly Random Beacons with Quadratic Communication,” *ACM CCS*, 2021.

[20] ——, “OptRand: Optimistically responsive distributed random beacons,” 2023.

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

[22] A. Boldyreva, “Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme,” in *PKC*, 2003.

[23] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, “Verifiable delay functions,” in *CRYPTO*, 2018.

[24] D. Boneh, B. Bünz, and B. Fisch, “A Survey of Two Verifiable Delay Functions,” Cryptology ePrint Archive, Paper 2018/712, 2018.

[25] D. Boneh, S. Eskandarian, L. Hanzlik, and N. Greco, “Single secret leader election,” in *AFT*, 2020.

[26] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the Weil pairing,” in *Asiacrypt*, 2001.

[27] D. Boneh and M. Naor, “Timed commitments,” in *CRYPTO*, 2000.

[28] J. Bonneau, J. Clark, and S. Goldfeder, “On Bitcoin as a public randomness source,” Cryptology ePrint Archive, Paper 2015/1015, 2015.

[29] R. B. Boppana and B. O. Narayanan, “Perfect-information leader election with optimal resilience,” *SIAM Journal on Computing*, vol. 29, no. 4, 2000.

[30] B. Bünz, S. Goldfeder, and J. Bonneau, “Proofs-of-delay and randomness beacons in Ethereum,” *IEEE Security and Privacy on the blockchain*, 2017.

[31] C. Cachin, K. Kursawe, and V. Shoup, “Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography,” *Journal of Cryptology*, 2005.

[32] J. Camenisch, M. Drijvers, T. Hanke, Y.-A. Pignolet, V. Shoup, and D. Williams, “Internet computer consensus,” in *ACM PODC*, 2022.

[33] R. Canetti, “Universally composable security: A new paradigm for cryptographic protocols,” in *FOCS*, 2001.

[34] I. Cascudo and B. David, “SCRAPE: Scalable randomness attested by public entities,” in *Applied Cryptography and Network Security*, 2017.

[35] ——, “Albatross: publicly attestable batched randomness based on secret sharing,” in *Asiacrypt*, 2020.

[36] I. Cascudo, B. David, O. Shlomovits, and D. Varlakov, “Mt. random: Multi-tiered randomness beacons,” *Cryptology ePrint Archive*, 2021.

[37] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in *OSDI*, 1999.

[38] D. Chaum and T. P. Pedersen, “Wallet databases with observers,” in *CRYPTO*, 1992.

[39] A. Cherniaeva, I. Shirobokov, and O. Shlomovits, “Homomorphic Encryption Random Beacon,” Cryptology ePrint Archive, Paper 2019/1320, 2019.

[40] K. Choi, A. Arun, N. Tyagi, and J. Bonneau, “Bicorn: An optimistically efficient distributed randomness beacon,” in *Financial Crypto*, 2023.

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

[42] R. Cleve, “Limits on the security of coin flips when half the processors are faulty,” in *TOC*, 1986.

[43] R. Cleve and R. Impagliazzo, “Martingales, collective coin flipping and discrete control processes,” *Manuscript*, 1993.

[44] I. Damgård, “Commitment schemes and zero-knowledge protocols,” in *School organized by the European Educational Forum*, 1998.

[45] ——, “On σ-protocols,” *Lecture Notes, University of Aarhus*, 2002.

[46] S. Das, V. Krishnan, I. M. Isaac, and L. Ren, “SPURT: Scalable Distributed Randomness Beacon with Transparent Setup,” Cryptology ePrint Archive, Paper 2021/100, 2021.

[47] B. David, P. Gaži, A. Kiayias, and A. Russell, “Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain,” in *Eurocrypt*, 2018.

[48] L. De Feo, S. Masson, C. Petit, and A. Sanso, “Verifiable delay functions from supersingular isogenies and pairings,” in *Eurocrypt*, 2019.

[49] Y. Desmedt and Y. Frankel, “Threshold cryptosystems,” in *CRYPTO*, 1990.

[50] Y. Dodis, “Impossibility of black-box reduction from non-adaptively to adaptively secure coin-flipping,” in *ECCC*, 2000.

[51] Y. Dodis and A. Yampolskiy, “A verifiable random function with short proofs and keys,” in *PKC*, 2005.

[52] O. Etesami, S. Mahloujifar, and M. Mahmoody, “Computational concentration of measure: Optimal bounds, reductions, and more,” in *SODA*, 2020.

[53] U. Feige, “Noncryptographic selection protocols,” in *FOCS*, 1999.

[54] P. Feldman, “A practical scheme for non-interactive verifiable secret sharing,” in *FOCS*, 1987.

[55] A. Fiat and A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” in *Eurocrypt*, 1986.

[56] M. J. Fischer, M. Iorga, and R. Peralta, “A public randomness service,” in *SECRYPT*, 2011.

[57] D. Galindo, J. Liu, M. Ordean, and J.-M. Wong, “Fully Distributed Verifiable Random Functions and their Application to Decentralised Random Beacons,” Cryptology ePrint Archive, Paper 2020/096, 2020.

[58] J. Garay, A. Kiayias, and N. Leonardos, “The Bitcoin backbone protocol: Analysis and applications,” in *Eurocrypt*, 2015.

[59] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, “Secure distributed key generation for discrete-log based cryptosystems,” in *Eurocrypt*, 1999.

[60] C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J. B. Nielsen, T. Rabin, and S. Yakoubov, “Yoso: you only speak once,” in *CRYPTO*, 2021.

[61] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, “Algorand: Scaling Byzantine Agreements for Cryptocurrencies,” in *SOSP*, 2017.

[62] S. Goldwasser, Y. T. Kalai, and S. Park, “Adaptively secure coin-flipping, revisited,” in *ICALP*, 2015.

[63] J. Groth, “Non-interactive distributed key generation and key resharing,” Cryptology ePrint Archive, Paper 2021/339, 2021.

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

[65] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu, "Aggregatable distributed key generation," in *Eurocrypt*, 2021.
[66] M. Haahr, "Random.org: True random number service," www.random.org, 2010.
[67] I. Haitner and Y. Karidi-Heller, "A tight lower bound on adaptively secure full-information coin flip," in *FOCS*, 2020.
[68] J. Kahn, G. Kalai, and N. Linial, "The influence of variables on boolean functions," *FOCS*, 1989.
[69] Y. T. Kalai, I. Komargodski, and R. Raz, "A lower bound for adaptively-secure collective coin flipping protocols," *Combinatorica*, vol. 41, no. 1, 2021.
[70] J. Kelsey, L. T. Brandão, R. Peralta, and H. Booth, "A reference for randomness beacons: Format and protocol version 2," National Institute of Standards and Technology, Tech. Rep., 2019.
[71] D. Khovratovich, M. Maller, and P. R. Tiwari, "MinRoot: Candidate Sequential Function for Ethereum VDF," Cryptology ePrint Archive, Paper 2022/1626, 2022.
[72] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO*, 2017.
[73] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *IEEE Security & Privacy*, 2018.
[74] A. K. Lenstra and B. Wesolowski, "A random zoo: sloth, unicorn, and trx," Cryptology ePrint Archive, Paper 2015/366, 2015.
[75] D. Lichtenstein, N. Linial, and M. Saks, "Some extremal problems arising from discrete control processes," *Combinatorica*, vol. 9, no. 3, 1989.
[76] S. Mahloujifar and M. Mahmoody, "Can adversarially robust learning leveragecomputational hardness?" in *Algorithmic Learning Theory*, 2019.
[77] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *FOCS*, 1999.
[78] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[79] 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 *IEEE International Conference on Blockchain*, 2019.
[80] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO*, 1991.
[81] ——, "A threshold cryptosystem without a trusted party," in *Eurocrypt*, 1991.
[82] K. Pietrzak, "Simple verifiable delay functions," in *ITCS*, 2018.
[83] Y. Qian, "RANDAO: Verifiable Random Number Generation," 2017.
[84] M. O. Rabin, "Transaction protection by beacons," *Journal of Computer and System Sciences*, 1983.
[85] A. Russell, M. Saks, and D. Zuckerman, "Lower bounds for leader election and collective coin-flipping in the perfect information model," in *TOC*, 1999.
[86] M. Saks, "A robust noncryptographic protocol for collective coin flipping," *SIAM Journal on Discrete Mathematics*, vol. 2, no. 2, 1989.
[87] J. Scharfman, "Decentralized finance (defi) fraud and hacks: Part 2," in *The Cryptocurrency and Digital Asset Fraud Casebook*, 2023.
[88] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. Weippl, "RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness," Cryptology ePrint Archive, Paper 2020/942, 2020.
[89] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "HydRand: Practical Continuous Distributed Randomness," in *IEEE Security & Privacy*, 2020.
[90] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic voting," in *CRYPTO*, 1999.
[91] A. Shamir, "How to share a secret," *Communications of the ACM*, 1979.
[92] 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 *IEEE Security & Privacy*, 2017.
[93] S. A. K. Thyagarajan, G. Castagnos, F. Laguillaumie, and G. Malavolta, "Efficient CCA Timed Commitments in Class Groups," in *ACM CCS*, 2021.
[94] S. A. K. Thyagarajan, T. Gong, A. Bhat, A. Kate, and D. Schröder, "OpenSquare: Decentralized Repeated Modular Squaring Service," in *ACM CCS*, 2021.
[95] L. Trevisan, "Extractors and pseudorandom generators," *Journal of the ACM*, 2001.
[96] L. Trevisan and S. Vadhan, "Extracting randomness from samplable distributions," in *FOCS*, 2000.
[97] G. Wang and M. Nixon, "RandChain: Practical Scalable Decentralized Randomness Attested by Blockchain," *IEEE International Conference on Blockchain*, 2020.
[98] B. Wesolowski, "Efficient verifiable delay functions," in *Eurocrypt*, 2019.
[99] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014.
[100] D. Yakira, A. Asayag, I. Grayevsky, and I. Keidar, "Economically viable randomness," *CoRR*, 2020.
[101] D. Yakira, I. Grayevsky, and A. Asayag, "Rational threshold cryptosystems," *arXiv preprint arXiv:1901.01148*, 2019.

Appendix

A. 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 [54] and Pedersen-VSS [80] 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 secret.

- $\mathsf{ShareGen}(s) \to (\{s_i\}, C)$ 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.
- $\mathsf{ShareVerify}(s_i, C) \to \{0, 1\}$ 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 not passing $\mathsf{ShareVerify}$.
- $\mathsf{Recon}(A, \{s_i\}_{i \in A}) \to s$ outputs the secret s by performing Lagrange interpolation (see Appendix H1) 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.

B. Distributed Key Generation (DKG)

One of the best known DKG schemes is Joint-Feldman, proposed by Pedersen [81].

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

- $\mathsf{DKG}(1^\lambda, t, n) \to (sk_i, pk_i, pk)$ 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 $s_{ij} = p_i(j)$ in 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 subshares it receives from other participants by performing verification steps of Feldman-VSS on each subshare. If verification for 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 individual secret key is computed as $sk_j = \sum_{i \in \mathcal{C}} s_{ij}$. Though not computed explicitly, the group secret key sk is equal to both $\sum_{i \in \mathcal{C}} a_{i0}$ and the Lagrange interpolation involving the individual secret keys $\{sk_j\}_{j \in \mathcal{C}}$.

C. 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 simplest PVSS schemes used in practice is one by Schoenmakers [90]. 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 $0 \leq j \leq t$ 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 H3). 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.

D. Verifiable Random Function (VRF)

A VRF [51], [77] is a function that, given an input x and a secret key sk, generates a unique, pseudorandom output y as well as a proof π verifying that the computation has been done correctly. Due to π, it is possible to repeatedly generate new pseudorandom outputs with one sk and varying inputs in a verifiable manner whereas otherwise (e.g. using a classical pseudorandom function) one needs to divulge the secret key and sacrifice its reusability for public verification purposes. It can be represented by the following tuple of algorithms:

- $\mathsf{Prove}(sk, x) \to (F_{sk}(x), \pi_{sk}(x))$ generates the pseudorandom output $F_{sk}(x)$ and its proof of correctness $\pi_{sk}(x)$ given input x and secret key sk.
- $\mathsf{Verify}(pk, x, y, \pi) \to \{0, 1\}$ outputs 1 if it is verified that $y = F_{sk}(x)$ using the proof π and 0 otherwise.

E. 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 H3).
- $\mathsf{PartialVerify}(pk_i, x, y_i, \pi_i)$ is equivalent to $\mathsf{DLEQ\text{-}Verify}(g, pk_i, H(x), y_i, \pi_i)$ (Appendix H3) 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 $y = \prod_{i\in A} y_i^{\lambda_{0,i,A}}$ and $\pi = \{(y_i, \pi_i)\}_{i\in A}$. Details related to Lagrange coefficients $\lambda_{0,i,A}$ are included in Appendix H1.
- $\mathsf{Verify}(pk, \{pk_i\}, x, y, \pi)$ verifies all partial proofs via $\mathsf{PartialVerify}$ for all $i \in A$ from π and checks $y = \prod_{i\in A} y_i^{\lambda_{0,i,A}}$.

F. 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 H2) 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 pairing equation $e(\pi, g_2) = e(H_1(x), pk)$.

G. 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 $\mathsf{PartialVerify}$ 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 $\mathsf{PartialVerify}$ (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.

H. 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 [26] 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 [22] 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 implicit 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 [38], the Σ protocol [45] 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 [55]. 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 $\mathbb{G}_q$.

```
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 $w \xleftarrow{\$} \mathbb{Z}_q$
```

TABLE II: Committee-Based DRB

			Step 2: Beacon Output Generation	
			Fresh per-node entropy	$\Omega_{\tau-1}$ & precommitted per-node entropy
Step 1: Committee Selection	Public	RR	**BRandPiper** *Step 1*: Node $i \equiv \tau \pmod{n}$ *Step 2*: Share-aggregate-reconstruct	
		RS	**Ouroboros** *Step 1*: Follow-the-satoshi [18], [72] *Step 2*: Share-reconstruct-aggregate	**HydRand** *Step 1*: Node $i \equiv \Omega_{\tau-1} \pmod{\tilde{n}}$ *Step 2*: $\Omega_\tau = H(\Omega_{\tau-1} \parallel h^{e_{\tilde{\tau}}})$ **GRandPiper** *Step 1*: Node $i \equiv \Omega_{\tau-1} \pmod{\tilde{n}}$ *Step 2*: $\Omega_\tau = H(h^{e_{\tilde{\tau}}}, \Omega_{\tau-1}, ..., \Omega_{\tau-t})$
		LS	**RandHound** *Step 1*: Node $\text{argmin}_i\, H(C \parallel pk_i)$ *Step 2*: Share-reconstruct-aggregate **SPURT, OptRand** *Step 1*: Node $i \equiv \tau \pmod{n}$ *Step 2*: Share-aggregate-reconstruct	
	Private	VRF	**NV++** *Step 1*: $VRF_{sk}(\Omega_{\tau-1} \parallel nonce) < target$ *Step 2*: Threshold ElGamal	**Algorand** *Step 1*: $VRF_{sk}(\Omega_{\tau-1} \parallel role) < target$ *Step 2*: $\Omega_\tau = VRF_{sk}(\Omega_{\tau-1} \parallel \tau)$ **Ouroboros Praos**[1] *Step 1*: $VRF_{sk}(\Omega_{\tau-1} \parallel slot \parallel \mathsf{TEST}) < target$ *Step 2*: $\Omega_\tau = H(\Omega_{\tau-1} \parallel epoch \parallel \rho_1 \parallel ... \parallel \rho_K)$
		Hash chain		**Caucus**[2] *Step 1*: $H(h_\tau \oplus \Omega_{\tau-1}) < target$ *Step 2*: $\Omega_\tau = h_\tau \oplus \Omega_{\tau-1}$

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 beacon output generation, see Sections VI-B1 and VI-B2.

[1] The protocol is a variant of Algorand. While $|\mathcal{C}_\tau|$ is expected to be one in Algorand (with 1 final winner per lottery and 1 lottery per epoch), that in Ouroboros Praos is expected to be K where each epoch consists of K slots and thus K per-slot lotteries. Parameters $slot$ and $epoch$ denote the slot and epoch numbers, respectively, and $\rho_i = VRF_{sk_i}(\Omega_{\tau-1} \parallel slot_i \parallel \mathsf{NONCE})$ 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 $h_\tau = H(h_{\tau+1})$ for all $\tau = 1, ..., m-1$ 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_τ at a time (i.e. h_τ in epoch τ). Each node independently generates a private hash chain. One downside is that the hash chain needs to be periodically regenerated, as m is finite.

2) $e = H(h_1, h_2, A_1, A_2)$
3) $s = w - \alpha \cdot e \pmod{q}$
4) $\pi = (e, s)$

$\mathsf{DLEQ\text{-}Verify}(g_1, h_1, g_2, h_2, \pi)$

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) $b = \begin{cases} 1 & \text{if } e' = e \\ 0 & \text{otherwise} \end{cases}$