HydRand: Efficient Continuous Distributed Randomness

Philipp Schindler*, Aljosha Judmayer*, Nicholas Stifter*†, Edgar Weippl*†
*SBA Research
†Christian Doppler Laboratory for Security and Quality Improvement in the Production System Lifecycle (CDL-SQI), TU Wien
Email: (firstletterfirstname)(lastname)@sba-research.org

***Abstract*—A reliable source of randomness is not only an essential building block in various cryptographic, security, and distributed systems protocols, but also plays an integral part in the design of many new blockchain proposals. Consequently, the topic of publicly-verifiable, bias-resistant and unpredictable randomness has recently enjoyed increased attention. In particular *random beacon protocols*, aimed at *continuous* operation, can be a vital component for current Proof-of-Stake based distributed ledger proposals. We improve upon previous random beacon approaches with *HydRand*, a novel distributed protocol based on publicly-verifiable secret sharing (PVSS) to ensure unpredictability, bias-resistance, and public-verifiability of a *continuous* sequence of random beacon values. Furthermore, HydRand provides guaranteed output delivery of randomness at regular and predictable intervals in the presence of adversarial behavior and does not rely on a trusted dealer for the initial setup. Compared to existing PVSS based approaches that strive to achieve similar properties, our solution improves scalability by lowering the communication complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$. Furthermore, we are the first to present a detailed comparison of recently described schemes and protocols that can be used for implementing random beacons.**

I. Introduction

The question of how to generate trustworthy random values among a set of mutually distrusting participants over a message passing network was first addressed by Blum in 1983, thereby introducing the notion of *coin tossing protocols* [10]. Distributed randomness also forms a key component of *asynchronous* consensus protocols in the form of local [6] and *common coin* designs [38], [20].

Lately, coin tossing protocols have received increased attention, in part because generating shared randomness is proving to be a vital component of most distributed ledger approaches (e.g. [8], [22], [33]) that aim to replace the computationally intensive *Proof-of-Work* (PoW) mechanism as found in Bitcoin [36] and similar cryptocurrencies. Specifically, *Proof-of-Stake* (PoS) blockchain proposals, which rely on virtual resources in the form of digital assets, call for manipulation resistant and unpredictable leader election as part of a secure protocol design [33]. The distributed generation of trustworthy random values can hence be considered a complementary problem to the development of such protocols.

Random beacon protocols aim to generate publicly-verifiable, bias-resistant and unpredictable randomness[1] in distributed environments. The concept of a random *beacon* was first formalized by Rabin [39], where a service that emits a fresh random number at regular intervals is proposed. Potential application areas for random beacons are broad and, as described in [44], [21], [15], include:

- the secure generation of protocol parameters for cryptographic schemes [4], [34]
- privacy preserving messaging services [49], [47], [30]
- protocols for anonymous browsing, including Tor hidden services [46], [31], [28]
- electronic voting protocols [1]
- publicly-auditable selections [15]
- gambling and lottery services [15]

With the emergence of blockchain protocols additional areas that demand secure sources of public randomness, such as sharding approaches [23], were formed. In particular smart contracts often draw upon insecure sources of randomness or trusted third parties [2], [17] such as the NIST random beacon, Random.org or Oraclize.it.

The revealed backdoor in the Dual Elliptic Curve PRNG [9], the unreliability of proprietary beacons [15], and the possibility of a centralized provider buffering, manipulating, and benefiting from prior knowledge of the provided randomness [15] are only a few of many reasons in favor of distributed randomness beacons where trust is spread among participants.

Considering distributed approaches, the following properties, as outlined in [3], [15], [44], are desiderata of a random beacon protocol:

1) **Availability/Liveness:**
Any single participant or colluding adversary should not be able to prevent progress.
2) **Unpredictability:** Correct, as well as adversarial nodes, should not be able to predict (precompute) future random beacon values.
3) **Bias-Resistance:** Any single participant or colluding adversary should not be able to influence future random beacon values to their advantage.
4) **Public-Verifiability:** Third parties not directly participating in the protocol should also be able to verify generated values. As soon as a new random beacon value becomes available, all parties can verify the correctness of the new value using public information only.

Additionally, we follow the notion of [33], [21] where **guaranteed output delivery** (G.O.D.) [40] i.e., the inability for an adversary to prevent correct participants of the protocol from obtaining an output, is also considered as an important property of random beacon protocols. In particular, if an adversary is not sufficiently restricted by how much it can affect the timing of the random beacon's output in system

[1] In the following we will simply refer to this by the term *randomness*.

models with bounded delays, both unpredictability and bias-resistance are weakened because the adversary can influence if an application is able to receive the output before a certain time or not.

Another particular desirable property for random beacons in the context of (permissionless) distributed ledgers is the **avoidance of an initial trusted setup**, e.g. a trusted dealer, [44].

Current random beacon protocols aim to provide solutions by employing different techniques, reaching from Proof-of-Delay [17], [19] and incentive based solutions [18], [41] over publicly-verifiable secret sharing (PVSS) [3], [21], [33], [44] and unique signatures [22], [25] to utilizing Bitcoin itself as a source of randomness [7], [15]. The diversity of these approaches, as well as the differences in their underlying assumptions and characteristics, make them difficult to compare and not equally suited for all use-cases. Furthermore, various recently described (PoS) blockchain schemes utilize or provide a random beacon as part of their protocol design and are therefore not easily comparable or deployable as a stand-alone protocol.

A. Contribution

We present *HydRand*, a PVSS based distributed random beacon protocol geared towards the *continuous* provision of randomness at regular intervals in a Byzantine failure setting. HydRand provides *guaranteed output delivery*, i.e., it guarantees the generation of new, *bias-resistant* randomness in every round of the protocol. As a hybrid approach, HydRand provides both a probabilistic guarantee for *unpredictability*, which ensures that a successful prediction of future random beacon values becomes exponentially unlikely, as well as unpredictability with absolute certainty for applications which wait for at least $f + 1$ rounds before using a future protocol output. The protocol assumes a synchronous system model and $n = 3f + 1$ participants. In respect to previous approaches based on PVSS, the communication complexity is hereby lowered from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ as HydRand only requires at most one PVSS distribution/recovery operation per round. Our protocol is described in a self contained manner and neither relies on a trusted dealer nor on a distributed key generation (DKG) protocol.

Moreover, to the best of our knowledge, we are the first to provide an extensive comparison of state of the art random beacon protocols in this field that considers and analyses a variety of key characteristics and assumptions.

B. Paper Structure

The assumed system model is presented in section II. Section III provides a high level overview of the HydRand protocol and highlights the employed variant of PVSS, which constitutes one of the main cryptographic primitives that is utilized. The protocol construction is outlined in detail in section IV. An extensive analysis including proofs showing that HydRand achieves the desired properties is presented in section V. Section VI compares HydRand to other related schemes, while sections VII, VIII and IX discuss the paper, present our evaluation results and conclude the paper. An execution example highlighting the protocol functionality and its phases, as well as a quick reference for utilized symbols and notations can be found in the appendix.

II. System and Threat Model

We assume a fixed set of known participants, hereby referred to as *nodes*, of size $n = 3f + 1$, of which at most f nodes may exhibit Byzantine failures and can deviate arbitrarily from the specified protocol. A node is considered to be *correct* if it does not engage in any incorrect behavior during the entirety of the protocol execution, else it is considered to be *faulty*. The terms *Byzantine* or *malicious* are used synonymously to refer to faulty nodes. The set of all nodes is denoted by $\mathcal{P} = \{1, 2, ..., n\}$ and each node $i \in \mathcal{P}$ is assumed to have a private/public key pair $\langle sk_i, pk_i \rangle$. The public keys of these keypairs are known to all participants. A synchronous system model with a fully connected network of authenticated and reliable bidirectional point-to-point messaging channels is assumed. We argue that the chosen timing model is reasonable for a small to moderate set of participants, and defer an analysis of our protocol in other system models to future work. Further, for many application areas of random beacons, e.g, in the context of cryptocurrencies, partially synchronous and synchronous system models are prevalent. Here, a synchronous random beacon protocol that also provides guaranteed output delivery may be necessary if strong notions of bias-resistance are a requirement.

III. Protocol Overview

The aim of HydRand is to provide a bias-resistant, publicly-verifiable and unpredictable stand-alone random beacon which emits random values at a regular interval. We target HydRand at a permissioned setting with a fixed set of participants and assume a known upper bound[2] on both computation and message transmission times.

For the protocol setup it is assumed that all participants exchanged their public keys and prepared an initial commitment using publicly-verifiable secret sharing (PVSS). The protocol operation itself is separated into *rounds*, where each round consists of three distinct *phases* – propose, acknowledge and voting. We describe these phases in detail in section IV. In each round, the previously generated random value is used for uniquely determining the current round *leader*. This leader has two choices: (i) The leader *reveals* the correct secret value he has committed himself to the last time[3] he was leader and attaches his next commitment. (ii) The leader does not reveal his secret value and therefore cannot attach another commitment. In the latter case, the previously committed secret value is *reconstructed* by $f+1$ other nodes, including at least one correct participant. The properties of the underlying PVSS scheme ensure that the random beacon value obtained by reconstruction is always equal to the value that is obtained

[2] We assume that a message sent at the beginning of one phase is received within that same phase.

[3] The initial commitment from the protocol setup is used the first time.

when a leader reveals his secret – this establishes *bias-resistance*. Additionally, *guaranteed output delivery* follows because the protocol outputs a random beacon value at each round, independent of the actions of the (potentially adversarial) leader and other faulty nodes.

In case the leader's previous commitment is reconstructed, the leader is excluded from being eligible as leader in future rounds since no new valid commitment was provided. However, the presented protocol could also be adapted to facilitate that temporarily failed nodes may rejoin $f+1$ rounds after a fresh commitment is provided and agreed upon. A correct leader constructs a new *dataset*, which includes: (i) the secret value they previously committed themself to, (ii) a new commitment to a uniformly random sampled value and (iii) a reference to the dataset of the previous round. The leader signs this dataset using their private key and broadcasts this message and signature to all other nodes in the network. After receiving and verifying the dataset, each node can compute the new random value of the beacon.

In case a leader is faulty and does not broadcast any data, other participants can collaborate to reconstruct the missing secret value, i.e. the value the leader has previously committed himself to in (ii). The reconstructed value can be used by any node to obtain the new random beacon value and thereby advance the protocol to the next round and leader. This process is repeated until eventually a correct leader is selected that creates a new dataset that accounts for all reconstructed datasets in between.

To ensure that a correct node is selected as leader after (at most) $f+1$ rounds, all previously selected leaders of the last f rounds are exempt from becoming leader in the current round. Since malicious nodes are unable to determine how an unrevealed commitment of an honest leader will influence future random beacon values, they cannot precompute any future output once a correct node is selected as leader. Moreover, correct participants converge on a single history after a correct node is selected as leader, because correct leaders are required to build on top of a single dataset and never sign different datasets in the same round. The correct node acts as a barrier for *unpredictability* and anchor for agreement on the protocol state. Unpredictability is thereby ensured with certainty for any round after $f+1$ rounds in the future. *Public-verifiability* is established by leveraging the properties of the underlying PVSS scheme.

A. Publicly-Verifiable Secret Sharing

We rely on PVSS as a primary building block in the HydRand protocol. More specifically, we make use of Scrape's PVSS protocol [21], which is an optimization of Schoenmakers' PVSS scheme [43], and allows a node (dealer) to efficiently share a secret value $s \in \mathbb{Z}_q$ among a set of n recipients, such that any subset of at least t recipients is able to recover/reconstruct the value $h^s \in \mathbb{G}_q$, where h is one of two independent generators of the group $\mathbb{G}_q$ and the prime q denotes the order of this group. The value of the reconstruction threshold t is set in a way that does not enable a colluding adversary to successfully recover a shared secret without requiring the collaboration of at least one correct node, i.e. $t = f + 1$. A key property of a *publicly-verifiable* secret sharing protocol is that, upon receiving the secret shares, not only the recipients but any third party with access to the public keys of the participants can verify the correctness of the shares prior to reconstruction of the secret. We use the term *PVSS commitment*, denoted by $Com(s)$, to refer to the result of the share distribution process of Scrape's PVSS. To form a PVSS commitment, a dealer provides:

- The encrypted shares for a secret s, i.e. one encrypted share $\hat{s_i}$ for each node i, encrypted with the receiver's public key.
- The commitments $v_1, v_2, ..., v_n$ to the shares for each node.
- A non-interactive zero-knowledge (NIZK) proof ensuring the correctness of the encrypted shares

For additional details regarding Scrape we refer the reader to [21].

B. Design Rationale

A malicious leader can try to construct and send different commitments, and hence different datasets, to other participants of the protocol or selectively withhold information to bias the resulting sequence of random beacon values. Hence, some form of (Byzantine) *consensus protocol* is necessary for participants to agree either on a single, valid commitment or the fact that the leader was faulty. In this respect, HydRand leverages on its intended application as a *continuous* random beacon to amortize the communication overhead of Byzantine agreement (BA) that is incurred at each round. Specifically, HydRand introduces a variant of repeated Byzantine agreement that defers consensus decisions for up to $f+1$ rounds, and combines data from multiple consensus instances that are executed with every consecutive new round of the HydRand protocol. By exempting a current leader to be re-elected within the next f rounds, enough time is given to reach agreement if the leader was faulty or not. Thereby, the overall communication (bit) complexity in regard to PVSS based random beacon schemes with comparable guarantees is reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.

IV. Protocol Details

HydRand proceeds in rounds, where each round $r \geq 1$ consists of three phases: *propose*, *acknowledge* and *vote*. Further, each round has a uniquely associated leader $\ell_r \in \mathcal{P}$ that is selected through the randomness generated by the protocol. When referring to the current round's leader, we may omit the subscript and simply denote the leader by ℓ.

Each round, ℓ_r is selected uniformly at random from the set of all nodes that *were not leader* during the last $f+1$ rounds[4]. At the end of a round all nodes learn a new random beacon value R_r. For simplicity, we hereby assume that correct nodes agree on the initial random beacon value R_0 used to select the

[4] The detailed leader selection mechanism is described in section IV-D.

leader of round 1, as well as the set of initial commitments of all nodes. R_0 becomes public knowledge only after the set of initial commitments was defined during setup.[5]

To simplify our notation, we assume that the sender of a broadcast is also a recipient of that message. Similarly, the dealer in the PVSS protocol also provides a share for himself. We use $\langle m \rangle_i$ to denote the message m a node i cryptographically signed with its private key sk_i. We further assume, that each correct node discards invalidly signed messages and processes only messages for the current round and phase.

A. Propose Phase

During this phase the round leader ℓ reveals his previously committed value s_ℓ and provides a new commitment $Com(s_\ell^\star)$. For this purpose, it is the leader's task to propose a new dataset D_r for the current round r. As a performance optimization, we split a dataset into two parts: a header and a body. For certain operations, we only require sending the header of the dataset. The header $header(D_r)$ of dataset D_r contains:

- the hash of the dataset's body $H(body(D_r))$
- the current round index r
- the round's random beacon value R_r
- the revealed secret value s_ℓ
- the round index $\overleftarrow{r}$ of the previous dataset $D_{\overleftarrow{r}}$
- the hash $H(D_{\overleftarrow{r}})$ of the previous dataset $D_{\overleftarrow{r}}$ if $\overleftarrow{r} > 0$
- a list of random beacon values $\{R_k, R_{k+1}, ...\}$ for all recovered rounds between $\overleftarrow{r}$ and r (if any)
- the Merkle tree root hash M_r over all encrypted shares in the new commitment $Com(s_\ell^\star)$

We use $H(D_r) = H(header(D_r))$ to denote the cryptographic hash of the dataset D_r. The body $body(D_r)$ of dataset D_r contains:

- a confirmation certificate $CC(D_{\overleftarrow{r}})$, which confirms that $D_{\overleftarrow{r}}$ was previously accepted as a valid dataset
- a recovery certificate $RC(k)$ for all rounds $k \in \{\overleftarrow{r} + 1, \overleftarrow{r} + 2, ..., r - 1\}$, which confirms that there exists a recovery for all rounds between $\overleftarrow{r}$ and r If $\overleftarrow{r} = r - 1$ then no such intermediate round exists and this value is omitted.
- the commitment $Com(s_\ell^\star)$ to a new randomly chosen secret $s_\ell^\star$

The leader selects $\overleftarrow{r} < r$ as the most recent regular round, for which the leader is not aware of any successful recovery. As we prove in section V-A, such a round always exists and the leader is in possession of the confirmation certificate $CC(D_{\overleftarrow{r}})$ required for the dataset's body.

After the construction of the above dataset, a correct leader ℓ broadcasts a signed *propose* message $\big\langle propose, \langle header(D_r) \rangle_\ell, body(D_r) \big\rangle_\ell$ to all nodes. Each node i, which receives such a message from the leader before the end of the propose phase, checks the validity of the dataset D_r. For this purpose i verifies that D_r is constructed as previously defined and properly signed. This includes a check that the revealed secret s_ℓ corresponds to the previous commitment $Com(s_\ell)$ of the current leader. Additionally, the validity of the confirmation and recovery certificates is checked. A *confirmation certificate* $CC(D_{\overleftarrow{r}})$ for dataset $D_{\overleftarrow{r}}$ is valid iff it consists of $f + 1$ signed messages of the form $\langle confirm, \overleftarrow{r}, H(D_{\overleftarrow{r}}) \rangle_i$ from $f+1$ different senders i. Similarly, a *recovery certificate* $RC(k)$ for some round k is a collection of $f+1$ signed messages of the form $\langle recover, k \rangle_i$ from $f + 1$ different senders.

B. Acknowledge Phase

If a node i receives a valid dataset D_r from the round's leader ℓ during the propose phase, it constructs and broadcasts a signed acknowledge message $\big\langle \langle acknowledge, r, H(D_r) \rangle_i, \langle header(D_r) \rangle_\ell \big\rangle_i$ thereby also forwarding the revealed secret value s_ℓ as part of the header. Further, each node i collects and validates acknowledge messages from other nodes.

C. Vote Phase

Each node i checks the following conditions:

- During the current propose phase a valid dataset D_r was received.
- During the current acknowledge phase at least $2f + 1$ valid acknowledge messages from different senders have been received.
- All acknowledge messages received refer to the dataset's hash $H(D_r)$. Valid acknowledge messages for more than one value of $H(D_r)$ form a cryptographic proof of leader equivocation.[6]

If all conditions are met, node i broadcasts a signed confirmation message: $\langle confirm, r, H(D_r) \rangle_i$ Otherwise node i, broadcasts a recover message: $\big\langle \langle recover, r \rangle_i, s_\ell, Com(s_\ell)[s_i], \hat{s_i}, M_k[\hat{s_i}], R_{r-1} \big\rangle_i$ Here, $Com(s_\ell)[s_i]$ denotes i's decrypted share s_i and its share decryption proof according to Scrape's PVSS, which cryptographically proves that s_i is a valid decryption of $\hat{s_i}$ under i's secret key. Round k denotes the round in which ℓ has provided the commitment $Com(s_\ell)$ and a Merkle tree root hash M_k. The Merkle branch $M_k[\hat{s_i}]$ proves that the encrypted share $\hat{s_i}$ was previously distributed as part of $Com(s_\ell)$ and therefore also of D_k. The values $\hat{s_i}$ and $M_k[\hat{s_i}]$ are required to enable nodes which are not in possession of $Com(s_\ell)$ to verify the share decryption proof for s_i. R_{r-1} is included for efficient external verification.

Correct nodes always include values for s_ℓ, $Com(s_\ell)[s_i]$, $\hat{s_i}$ and $M_k[\hat{s_i}]$ if they are in possession of the required data. Otherwise the unknown value(s) are omitted. This can happen if an adversary selectively sent the previous dataset D_k to a subset of nodes. Therefore, upon receiving recovery messages from other nodes, correct nodes accept messages with omitted values. The protocol guarantees that at least $f+1$ correct nodes

[5] In practice this initial random value can, for example, be obtained via *Proof-of-Delay* [17] or a *Proof-of-Work* [7].

[6] In a (PoS) cryptocurrency setting, the protocol could be extended such that this equivocation proof is used to seize some form of security deposit from the leader.

have received the dataset with a valid confirmation certificate, and hence are able to provide the necessary shares required for reconstructing the respective secret. An example is presented in appendix A.

At the end of this phase each node i can obtain the round's random beacon value R_r. We distinguish between the following two cases: (i) node i already knows the secret value s_ℓ, because it received the dataset D_r or an acknowledge message for D_r, and (ii) node i has received at least $f+1$ valid recover messages which include at least $f+1$ decrypted secret shares for s_ℓ. In the latter case the reconstruction procedure of Scrape's PVSS can be executed to produce the value h^{s_ℓ}. In both cases R_r is then obtained by computing:

$$R_r \leftarrow H(R_{r-1} \mid\mid h^{s_\ell}) \qquad \text{(Definition 1)}$$

D. Leader selection

At the beginning of each round $r \geq 1$, a node i determines the round's leader ℓ_r based on the available local information it gathered so far. For this purpose node i uses the randomness R_{r-1} of the previous round to deterministically select ℓ_r from the set $\mathcal{L}_r$ of potential leaders. We denote the canonical representation of $\mathcal{L}_r$ as $\langle l_0, l_1, ..., l_{|\mathcal{L}_r|-1} \rangle$ and obtain ℓ_r as follows:

$$\ell_r \leftarrow l_{(R_{r-1} \bmod |\mathcal{L}_r|)} \qquad \text{(Definition 2)}$$

Let $D_{\overleftarrow{r}}$ denote the most recent valid dataset, for which node i is *not* in possession of a corresponding recovery certificate $RC(\overleftarrow{r})$. If no such dataset exists we set $\overleftarrow{r} = 0$. Now we introduce a method to determine the set of *recovered nodes* $rn(\cdot)$ as a component needed for the definition of $\mathcal{L}_r$. Intuitively, the set $rn(\cdot)$ contains all nodes, which have not provided valid datasets for some round where the node was selected as leader. We define the set of all leaders that were recovered in some round up to a referenced dataset as follows:

$$rn(D_x) = \begin{cases} \emptyset & \text{if } x = 0 \\ \{\ell_k \mid RC(k) \in D_x\} \cup rn(D_{\overleftarrow{x}}) & \text{otherwise} \end{cases} \qquad \text{(Definition 3)}$$

Here $D_{\overleftarrow{x}}$ denotes the previous dataset referenced by D_x. This function is used to construct the set of available nodes $\mathcal{P}_r$ for round r recursively by excluding all nodes which have been selected as leader in a round for which a valid reconstruction certificate exists:

$$\mathcal{P}_r = \mathcal{P} \setminus rn(D_{\overleftarrow{r}}) \qquad \text{(Definition 4)}$$

Based on this notion, the definition of the set of potential leaders $\mathcal{L}_r$ for round r follows:

$$\mathcal{L}_r = \mathcal{P}_r \setminus \{\ell_{r-f}, \ell_{r-f+1}, ..., \ell_{r-1}\} \qquad \text{(Definition 5)}$$

Intuitively, the set $\mathcal{L}_r$ only includes nodes that were not selected as leader for at least f rounds in the past and have not been reconstructed in any previous round, i.e., nodes that distributed valid datasets for all rounds in which they were selected as leader.

V. Protocol Properties

In the following, we show that HydRand achieves the desirable properties of a random beacon protocol as outlined in section I: *liveness*, *guaranteed output delivery*, *unpredictability*, *bias-resistance*, and *public-verifiability*. We furthermore show that our protocol also achieves *uniform agreement*. In our proofs we may refer to the definitions introduced in section IV.

A. Liveness and Guaranteed Output Delivery

To show that HydRand satisfies liveness and guaranteed output delivery, we first introduce and prove several primary lemmas. We show that (i) correct nodes are always able to provide a valid dataset if they are selected as leader, (ii) correct nodes can never be recovered and (iii) the set of potential leaders always contains at least $f+1$ correct nodes. Using these results, we infer that correct nodes can always output the round's random beacon value by the end of the round, given that they know the value for the previous round. Finally, we use an inductive argument to prove liveness and guaranteed output delivery of our protocol.

Lemma 1. *(Possibility of construction of valid datasets) For each round r a correct leader ℓ_r can construct a valid dataset D_r.*

Proof. Implicit agreement by all correct nodes on the current round number r follows from the synchronous system model and fixed duration of phases. A correct leader is in possession of its own secret s_ℓ and thus knows R_r. Furthermore, the leader can always construct a new PVSS commitment for a new secret $Com(s_\ell^\star)$ and is able to provide a valid value for M_r. Therefore, it only remains to be shown that each correct node is able to provide the required confirmation certificate $CC(\cdot)$ and recovery certificates $RC(\cdot)$. During the vote phase of every previous round, correct nodes have either broadcast a recover or confirm message. As there are at least $2f+1$ correct nodes, each node is guaranteed to receive at least $f+1$ recover messages or at least $f+1$ confirm messages (or both) for each of these rounds. As $f+1$ recover messages form a recovery certificate and $f+1$ confirm messages form a confirmation certificate, each node is in possession of a recovery certificate or a confirmation certificate (or both) for every previous round, and is hence able to provide the required certificates for D_r. □

Lemma 2. *(No recovery of correct leaders) If leader ℓ_r is correct, there does not exist a node i, which is in possession of a valid recovery certificate $RC(r)$.*

Proof. A correct leader ℓ_r sends valid proposal D_r to all nodes during the propose phase. By lemma 1, ℓ_r can always construct such a dataset. As all correct nodes consider D_r as valid, at least $2f+1$ nodes broadcast acknowledge messages for D_r during the acknowledge phase. All $2f+1$ correct nodes therefore receive at least $2f+1$ valid acknowledge messages for D_r. Since there cannot exist a valid acknowledge for a different dataset D'_r (a correct leader only provides his signature for D_r) all correct nodes broadcast *confirm* messages during the vote phase. As correct nodes only broadcast either confirm or recover messages, there are at most f recover messages (from Byzantine nodes). A valid recovery certificate $RC(r)$ however requires at least $f+1$ recover messages from different nodes, and therefore cannot exist. □

Lemma 3. *(Availability of leaders) For each round $r \geq 1$, the set of potential leaders $\mathcal{L}_r$ contains at least $f+1$ correct nodes.*

Proof. We first show that for each round r, the set of available nodes $\mathcal{P}_r$ contains at least $2f+1$ correct nodes. By Definition 3 and Definition 4 (see section IV-D), we ensure that only leaders ℓ_k for some round k, in which a recovery certificate $RC(k)$ exists, are excluded from the set $\mathcal{P}$ to form $\mathcal{P}_r$. As we have shown in lemma 2 there are no recovery certificates for rounds with correct leaders. Therefore correct nodes cannot be excluded from $\mathcal{P}$ to form $\mathcal{P}_r$, and thus $\mathcal{P}_r$ contains at least $2f+1$ correct nodes.

Using the above result and Definition 5, which excludes at most $f+1$ nodes from $\mathcal{P}_r$ to form $\mathcal{L}_r$, $\mathcal{L}_r$ contains at least $f+1$ correct nodes. □

Lemma 4. *(Liveness) If a correct node knows the random beacon value R_{r-1}, it can output the random beacon value R_r by the end of round r (independent of the actions of the round's leader ℓ_r).*

Proof. Following lemma 3 we guarantee the existence of a leader ℓ_r. Since $\ell_r \in \mathcal{L}_r$ and $\mathcal{L}_r \subset \mathcal{P}_r$, we know that $\ell_r \in \mathcal{P}_r$. By applying Definition 4 we get $\ell_r \notin rn(D_{\overleftarrow{r}})$. Hence, there exists some history of datasets with head $D_{\overleftarrow{r}}$, in which there does not exist a recovery certificate $RC(k)$ for any round $k < \overleftarrow{r}$ in which ℓ_r was also leader. Such a history for any valid dataset D_k can only exist if at least one correct node confirmed that D_k was correctly distributed and acknowledged by $2f+1$ nodes by providing a confirm message. Hence, at least $f+1$ correct nodes know a common dataset D_k for all rounds k where ℓ_r was previously selected as leader. In addition, all nodes know the shares for ℓ_r's first commitment as defined in the protocol setup. Thus, at least $f+1$ correct nodes can broadcast the decrypted share in case a recovery of the leader ℓ_r in round r is necessary. Hence all nodes learn the value h^{s_ℓ} corresponding to ℓ_r's last commitment $Com(s_\ell)$, and thus obtain R_r using h^{s_ℓ} and R_{r-1} via Definition 1. □

Theorem 1. *(Guaranteed Output Delivery) For each round r all correct nodes output a new random beacon value R_r.*

Proof. We use lemmas 3 and 4 and prove the theorem by induction on the round index r. For the base case we have an agreed random beacon value R_0 as given by the protocol setup. For the induction step, we assume that R_{r-1} is known by all correct nodes. Lemma 3 ensures that the set of potential leaders $\mathcal{L}_r$ contains at least $f+1$ correct nodes. Therefore, Definition 2 can always be applied to a selected leader ℓ_r using $\mathcal{L}_r$ and R_{r-1}. Hence, we can use lemma 4, to show that by the end of round r each correct node outputs a value R_r. □

B. Agreement

In the following, we show that all correct nodes agree on a common sequence of random beacon values. We start by showing that (i) within $f+1$ rounds a correct node is selected as leader and (ii) all correct nodes agree on a common set of potential leaders and use this two results to prove that uniform agreement is satisfied for the random beacon values in HydRand.

Lemma 5. *(Selection of correct leaders) In each interval $\{k, k+1, k+2, ..., k+f\}$ of $f+1$ consecutive rounds there is at least one round $\overleftarrow{k} \in \{k, k+1, k+2, ..., k+f\}$, such that the leader $\ell_{\overleftarrow{k}}$ of that round is correct.*

Proof. We assume that there is no correct leader in $\{\ell_k, \ell_{k+1}, \ell_{k+2}, ..., \ell_{k+f}\}$ and derive a contradiction. We apply the definition of the set of potential leaders for round $k+f$:

$$\mathcal{L}_{k+f} = \mathcal{P}_{k+f} \setminus \{\ell_k, \ell_{k+1}, ..., \ell_{k+f-1}\}$$

Notice that $\{\ell_k, \ell_{k+1}, ..., \ell_{k+f-1}\}$ denotes a set of f Byzantine nodes. As there are only f Byzantine nodes in total, $\mathcal{L}_{k+f}$ cannot contain any Byzantine nodes. However, the Byzantine node ℓ_{k+f} is assumed to be leader of round $k+f$ and therefore $\ell_{k+f} \in \mathcal{L}_{k+f}$, which completes the contradiction. □

Lemma 6. *(Agreement on potential leaders) If a node constructs a valid set of potential leaders $\mathcal{L}_r$ in round r, then every correct node constructs the same value for $\mathcal{L}_r$.*

Proof. Using lemma 5, for the interval $\{r-f-1, r-f, ..., r-1\}$, we know that there is some round $\overleftarrow{r}$ with a correct leader $\ell_{\overleftarrow{r}}$ in this interval. Using lemma 1, we know that $\ell_{\overleftarrow{r}}$ is able to construct a valid dataset $D_{\overleftarrow{r}}$ in round $\overleftarrow{r}$. As $\ell_{\overleftarrow{r}}$ is correct, it has distributed this dataset to all nodes during the propose phase of round $\overleftarrow{r}$. All correct nodes therefore acknowledge $D_{\overleftarrow{r}}$ in the acknowledge phase of round $\overleftarrow{r}$. Since there are at least $2f+1$ correct nodes, all correct nodes receive at least $2f+1$ valid acknowledge messages for $D_{\overleftarrow{r}}$ by the end of the acknowledge phase. No node can receive a valid acknowledge for some different dataset $D'_{\overleftarrow{r}}$, because the correct leader $\ell_{\overleftarrow{r}}$ does not provide a signature for a different value. Therefore, all correct nodes broadcast confirm messages for $D_{\overleftarrow{r}}$. As all correct nodes broadcast either one confirm or one recovery message, there are at most f recover messages (by Byzantine nodes). Therefore, no valid recovery certificate $RC(\overleftarrow{r})$ exists for round $\overleftarrow{r}$. Thus, any valid future dataset needs to (indirectly) reference the common and unique dataset $D_{\overleftarrow{r}}$. Consequently, we established agreement on $D_{\overleftarrow{r}}$ and its common history provided by the references to the predecessor datasets.

As the set of available nodes $\mathcal{P}_{\overleftarrow{r}}$ for round $\overleftarrow{r}$ is defined using only the agreed set of all nodes $\mathcal{P}$ and $D_{\overleftarrow{r}}$, $\mathcal{P}_{\overleftarrow{r}}$ is also agreed upon. Since the definition of $\mathcal{L}_r$ does not depended on whether or not leaders are recovered during the rounds $\{r-f, r-f+1, ..., r-1\}$ and $\overleftarrow{r} \geq r-f-1$, agreement on the set $\mathcal{L}_r$ follows. □

Theorem 2. *(Uniform Agreement) If a node outputs a valid random beacon value R_r in round r, then every node that outputs a valid beacon value in round r outputs the same R_r.*

Proof. We prove the theorem by induction on the round index r. For the base case we have an agreed common random beacon value R_0 as defined by the protocol setup.

For the induction step, we assume that every node that outputs a valid beacon value in round $r-1$ outputs the same R_{r-1}. We have agreement on R_{r-1} by the induction hypothesis and show agreement on the set of potential leaders $\mathcal{L}_r$ in lemma 6. As the leader selection mechanism given in Definition 2 only depends on those two arguments, all correct nodes agree on a common unique leader ℓ_r. By applying lemma 4 we obtain that each correct node learns the leader's previously committed secret h^{s_ℓ}. By either checking the revealed value of s_ℓ against the leaders commitment or verifying the validity of the share decryption proof according to Scrape's PVSS description [21], uniqueness of a valid h^{s_ℓ} and consequently of R_r is ensured. □

C. Unpredictability

Intuitively, the prediction of a future random beacon value by the adversary is only possible if the adversary is selected as leader for that particular round, as well as all rounds before that point, because each round's random beacon value depends on a secret value only known to the round leader. As we prove below, this is impossible for $f+1$ consecutive rounds.

However, even before this bound is reached, the possibility of successful prediction decreases exponentially in the number of rounds to predict. The probability of successful prediction of ω future random beacon values, where $\omega < f+1$, can be characterized by a hypergeometric distribution with population size n, ω draws (the number of values to predict) and f success states (adversarial nodes) in the population. The prediction is possible, iff all of the ω draws pick one of the success states. Figure 1 shows the probabilities for different values of n, under the $n = 3f+1$ security assumption. For large values of n, the probability converges to a geometric distribution.

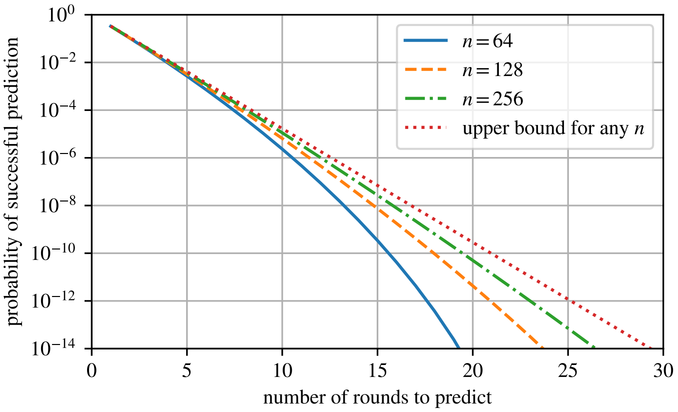

Fig. 1: Unpredictability guarantees for different numbers of nodes n, assuming a 33% adversary, i.e. $f = \lceil \frac{n}{3} \rceil$ - 1

Theorem 3. *(Unpredictability) At the beginning of round r, no node can predict the outcome R_{r+f} of the random beacon protocol in round $r+f$.*

Proof. By applying lemma 5 we show that there is at least one correct leader during the interval of $f+1$ consecutive rounds $\{r, r+1, r+2, ..., r+f\}$. Let k denote any round during this interval in which the leader ℓ_k is correct. As ℓ_k follows the protocol, it has not distributed its secret value s_{ℓ_k} to any node at the beginning of round r. Additionally, no correct node will provide a decrypted secret share, which could be used in the recovery process of the secret value. Therefore only f secret shares are available for an adversary to try and recover the secret in order to compute R_k (and potentially consecutive random beacon values). However, the protocol defines the reconstruction threshold t used by the PVSS scheme to be $f+1$. Therefore, an adversary cannot obtain the underlying secret before it is revealed or recovered during round k. Consequently, R_k and all consecutive random beacon values (including R_{r+f}) are unpredictable at the start of round r. □

D. Bias-Resistance

Theorem 4. *(Bias-Resistance) No node i can, for any round r, influence the value R_r of the random beacon protocol in a meaningful (i.e. predictable) way.*

Proof. This property follows from unpredictability and the fact that the protocol is constructed in a way that ensures that any action a (Byzantine) nodes takes in some round r, can only influence the value of the random beacon at round $r+f+1$ or later. In theorem 3 we have shown that the random beacon value at round $r+f$ is unpredictable at the beginning of round r. Therefore, a node cannot influence the random beacon values for rounds r to $r+f$, and may only influence values at round $r+f+1$ or later in an unpredictable manner. □

E. Public-Verifiability

Theorem 5. *(Public-Verifiability) For each round r, an external verifier can check the correctness of the random beacon value R_r, at the end of round r.*

Proof. The external verifier receives from any correct node (i.e. after querying at most $f+1$ nodes) its history up to and including round r. The verifier can, by following the protocol rules, only obtain a random beacon value R_r iff the provided data is correct. Additionally, an external verifier can obtain and verify recovered random beacon values between the last valid dataset $D_{\overleftarrow{r}}$ and the current dataset D_r for all rounds $k \in \{\overleftarrow{r}+1, \overleftarrow{r}+2, ..., r-1\}$. □

Lemma 7. *(Efficient-Verification) For each round r, an external verifier can check the correctness of the random beacon value R_r in $\mathcal{O}(n)$, at the end of round r (without validation of all previous rounds).*

Proof. We distinguish two cases: (i) the leader of round r provided a valid dataset D_r in time. The confirmation certificate $CC(D_r)$ is hereby available at the end of round

r and can be used to verify the correctness of D_r (and hence the included R_r) by verifying $f+1$ signatures; (ii) the leader of round r was recovered. In this case, an external verifier requests the necessary information to simulate a node's execution of the recovery step of round r, i.e. $f+1$ recover messages from round r as well as the header $header(D_{r'})$ and confirmation certificate $CC(D_{r'})$ of the failed leader's previously distributed dataset $D_{r'}$. The simulation of a node's recovery requires $f+1$ validations of decrypted PVSS shares, combining the shares via Lagrange interpolation, and checking $f+1$ signatures to verify $CC(D_{r'})$, and can thus be performed in $\mathcal{O}(n)$. □

VI. Comparison of Random Beacon Protocols

Recent years have seen a substantial amount of new research related to the generation of publicly-verifiable (distributed) randomness being published in academia as well as the industry. Thereof, we distinguish between the following types of protocols:

1) Stand-alone protocols, that are specifically designed to provide randomness. This includes the approach described within the first Ouroboros Proof-of-Stake protocol [33], the Scrape protocol [21], the Rand* protocol family [44], as well as our HydRand protocol.
2) Protocols designed for the purpose of generating randomness, leveraging resources of existing systems, namely Caucus [3] and Proof-of-Delay [17], [19].
3) Protocols that produce randomness as a byproduct of their operation, including Algorand [22], the BA protocol by Cachin et al., Dfinity [32] and Ouroboros Praos [24].

Additionally, we include Proof-of-Work blockchains, as first described by S. Nakamoto [36], as a source of public-verifiable randomness [16] in our comparison.

Proof-of-Work and Proof-of-Delay inherently require a substantial amount of computational resources to ensure security. When directly relying on the block hashes of a PoW blockchain as a source of randomness, bias-resistance can generally not be ensured. A miner can, with non-negligible probability, pick/reject random beacon values which suit him by choosing to withhold a valid PoW solution in favor of some other block(s). Hence, random beacon values derived by this mechanism may not be guaranteed to be uniformly distributed.

Proof-of-Delay, as described by Bünz et al. [17], addresses this problem by employing a delay function on top of the PoW block hash. Here, a cryptographic hash function or symmetric encryption algorithm is applied iteratively to the block hash to produce the randomness. The number of iterations Δ is a security parameter of the protocol and must be selected in a way that ensures that no adversary can finish this inherently sequential computation within the typical confirmation time of a block. While the adversary can still withhold its value and influence the protocol's output, they can only do so blindly without knowing the effects at the time of the decision, which ensures bias-resistance. However, full verification of a random beacon value is slow, as it requires the same sequential recomputing.

Algorand [22], Ouroboros Praos [24] and Caucus [3] are comparable in their approach of combining the previous public randomness with a (verifiable) source of private randomness, i.e., in the form of a VRF or hash chain, from an eligible leader to form the next random value. However, leader uniqueness by itself is not guaranteed and additional consensus rules are necessary to reach agreement. In this respect Algorand implements a Byzantine agreement protocol with finality, whereas Ouroboros Praos is a Proof-of-Stake blockchain protocol with eventual agreement, and Caucus is implemented within a smart contract that leverages the consensus protocol provided by the underlying Ethereum blockchain.

Cachin et al. [20] and Dfinity [32] both employ *unique* threshold signatures as a core primitive in their construction. The BLS signature scheme of Boneh et al. [14], [13], is a suitable candidate as its signatures are unique and both the signing process as well as the aggregation process are non-interactive. The main idea is that all nodes (i) provide a signature share on some common value (e.g. a round number), (ii) verify the received signatures shares and (iii) combine the valid shares to obtain the next random beacon value. As long as a threshold of nodes contributes valid signature shares the aggregation succeeds. Both approaches require the secure distribution of a shared private key as a precondition. While a trusted dealer is assumed in [20], Dfinity uses a distributed key generation protocol to establish this key.

The approaches described in the initial Ouroboros protocol [33], Scrape [21] and RandShare [44] all rely on PVSS as an underlying primitive. The general idea is that each node first privately generates a random secret value, and then sends out a publicly-verifiable commitment and shares of this secret using PVSS to all nodes. After verification and filtering out invalid commitments, the nodes begin to reveal their respective secrets. If a node fails to reveal or maliciously withholds its value, the other nodes step in and collectively recover the secret from the shares they received previously. Finally, all revealed/reconstructed secrets are combined to form the randomness.

RandHound [44] and RandHerd [44] are also protocols based on PVSS, but operate in a different manner. RandHound is a one-shot protocol, where a client divides nodes into multiple smaller groups and combines the randomness generated by these groups to form a random beacon value, whereas RandHerd is tailored towards continuous operation. The latter uses RandHound to establish a fair division of nodes, executes a distributed key generation protocol within these groups, and leverages on *collective signing* [45] to produce a sequence of random beacon values.

A. Comparison Overview

In this section, the results of our comparison of the herein presented and discussed approaches for generating publicly-verifiable distributed randomness are outlined. We highlight

TABLE I: Comparison of approaches for generating publicly-verifiable randomness

		Communication model	Liveness / failure probability◇	Comm. complexity (overall protocol)	Unpredictability	Bias-Resistance	Comp. complexity (per node)	Verif. complexity (per passive verifier)	Characteristic cryptographic primitive(s)	Trusted dealer or DKG required
[22]	Algorand	semi-syn.	10^{-12}	$\mathcal{O}(cn)^*$	$\overset{t}{\nearrow}$	✗	$\mathcal{O}(c)^*$	$\mathcal{O}(1)^*$	VRF	no
[20]	Cachin et al.	asyn.	✓	$\mathcal{O}(n^2)$	✓	✓	$\mathcal{O}(n)$	$\mathcal{O}(1)$	uniq. thr. sig.	yes
[3]	Caucus	syn.	✓	$\mathcal{O}(n)$	$\overset{t}{\nearrow}$	✗	$\mathcal{O}(1)$	$\mathcal{O}(1)$	hash func.	no
[32]	Dfinity	syn.	10^{-12}	$\mathcal{O}(cn)$	✓	✓	$\mathcal{O}(c)$	$\mathcal{O}(1)$	BLS sig.	yes#
[33]	Ouroboros	syn.	✓	$\mathcal{O}(n^3)$	✓	✓	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	PVSS	no
[24]	Ourob. Praos	semi-syn.	✓	$\mathcal{O}(n)^*$	$\overset{t}{\nearrow}$	✗	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	VRF	no
[36]	Proof-of-Work	syn.	✓	$\mathcal{O}(n)$	$\overset{t}{\nearrow}$	✗	very high§	$\mathcal{O}(1)$	hash func.	no
[17]	Proof-of-Delay	syn.	✓	$\mathcal{O}(n)$	✓	✓	very high§	$\mathcal{O}(\log \Delta)^\circ$	hash func.	no
[44]	RandShare	asyn.	✗†	$\mathcal{O}(n^3)$	✓	✓	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	PVSS	no
[44]	RandHound	syn.	0.08%	$\mathcal{O}(c^2 n)^\ddagger$	✓	✗	$\mathcal{O}(c^2 n)$	$\mathcal{O}(c^2 n)$	PVSS	no
[44]	RandHerd	syn.	0.08%	$\mathcal{O}(c^2 \log n)^\ddagger$	✓	✓	$\mathcal{O}(c^2 \log n)$	$\mathcal{O}(1)$	PVSS/CoSi	yes#
[21]	Scrape	syn.	✓	$\mathcal{O}(n^3)$	✓	✓	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	PVSS	no
	HydRand	syn.	✓	$\mathcal{O}(n^2)$	$\overset{t}{\nearrow}$✓	✓	$\mathcal{O}(n)$	$\mathcal{O}(n)$	PVSS	no

◇ For the failure probability we give the upper bound for the parameterization of the protocol as suggested by the respective authors.

* The approach for generating randomness is not described in a standalone matter and requires additional communication and verification steps for the underlying consensus protocol or the implementation of e.g. a bulletin board. The herein presented values do not account for this additional complexity.

‡ In contrast to Algorand and Dfinity, the parameter c in RandHound/RandHerd actually depends on n and is thus not constant. This is a direct consequence of sharding n nodes into groups of size c, as the protocols fail to provide availability if any single group fails. Keeping c constant while increasing n leads to a higher number of groups m, and thus increases the probability of a lifeness failure. To counter this effect requires a security/performance tradeoff where c also has to be increased as n grows. In section VI-C we further outline that c is a relevant factor in practice, in particular if one wants to achieve a similar liveness guarantee as e.g. Algorand or Dfinity.

§ The complexity is not dependent on the number of nodes n, but the involved Proof-of-Work is inherently computationally demanding.

# In Dfinity's and RandHerd's approach nodes are split into smaller groups. Within each of these groups a distributed key generation protocol is run.

$\overset{t}{\nearrow}$ The protocols provide probabilistic guarantees for unpredictability, which quickly (exponentially in the waiting time) get stronger the longer a client waits after it commits to use a future protocol output. For HydRand, we indicate that unpredictability with absolute certainty is reached after f rounds using the additional ✓symbol.

° We refer to the verification executed within the Smart Contract via an *interactive* challenge/response protocol. It has logarithmic complexity $\mathcal{O}(\Delta)$ in the security parameter Δ, which describes how many iterations of the hash function are applied to the seed.

† The protocol only provides liveness with additional synchrony assumptions. See section VI-C for a detailed discussion.

that a broad comparison was performed by not only considering protocols specifically targeted at implementing random beacons, but also by including approaches that can readily provide a random beacon functionality as a byproduct of their intended application, such as the provision of a distributed public ledger. Consequently, the underlying models, assumptions, notations, as well as the context differ from protocol to protocol and render an evaluation of the herein presented approaches a non-trivial task. We conducted the comparison to the best of our knowledge, contacted the respective protocol authors to try and clarify ambiguities and explicitly state whenever we were unable to adequately determine certain properties or had to estimate them.

The main results are presented in table I and the various protocol properties are discussed in greater detail in the following subsections. For the presented complexity evaluations, n refers to the number of protocol participants, and c denotes the size of some subset of nodes, if one is assumed in the specific protocol. Notice that the subset size c is protocol dependent and, although typically constant, a non-negligible factor for the resulting communication complexity in practice (see section VI-D for a more detailed discussion).

B. Communication Model

We classify the communication model of the analyzed protocols into three categories, namely synchronous, semi-synchronous and asynchronous protocols. We call a protocol synchronous, if a fixed known upper bound on message propagation delay is assumed. If no assumption on this delay is imposed by the protocol and messages are only eventually delivered, we categorize the protocol as asynchronous. If some weaker assumptions in regard to synchrony are made, we informally use the term semi-synchronous. This applies for instance to Algorand and Ouroboros Praos, where the underlying assumptions are outlined in detail, but are not readily comparable to other definitions of partial-synchrony, such as those first established in the context of distributed consensus [26], [27].

Dfinity [32] is aimed at a semi-synchronous setting, however security proofs are currently only published for the synchronous case.

We inferred the synchrony assumption from the protocol description or the underlying protocol whenever they have not been stated explicitly. Currently deployed Proof-of-Work blockchains such as Bitcoin and Ethereum assume a synchronous communication model. As Proof-of-Delay [17] and Caucus [3] are built on top of such Proof-of-Work blockchains, these protocols are also classified as synchronous. In [44], RandShare is described within an asynchronous setting[7]. For RandHound and RandHerd synchronicity is indicated in various paragraphs, e.g. III. A. for RandHound and IV. B. 1) for RandHerd [44].

C. Liveness/Availability

In regard to liveness, we distinguish between three different protocol types:

1) protocols which achieve liveness unconditionally (in the respective system model)
2) protocols which have a (configurable) but non-zero probability of a liveness failure
3) protocols which do not provide liveness (in the respective system model)

We mark protocols of the first type with a ✓ symbol in our comparison table. This category also includes HydRand, which achieves liveness and guaranteed output delivery in the respective system model unconditionally. For protocols of the second type, namely Algorand, Dfinity, RandHound and RandHerd, we give a typical failure probability as described by the respective authors. The authors of Algorand and Dfinity consider failure probabilities of at most 10^{-12} [22] and $2^{-40} \approx 10^{-12}$ [32] as suitable for the respective setting, whereas a typical failure probability of 0.08% [44] is stated for the exemplary configuration in the RandHound and RandHerd protocols.

For all of the above protocols, the failure probability can be adjusted through a security parameter. For example, to lower the failure probability of RandHound and RandHerd to a level of 10^{-12}, the group size c can be increased. By applying the formula given in Syta et al. [44], we observe an increase in group size from $c = 32$ to $c = 125$ to achieve this failure rate against an adversary controlling less than $1/3$ of the nodes. Consequently, performance is decreased, as the communication complexities of both protocols contain c as a quadratic factor.

The RandShare protocol is described in an asynchronous communication model (under a $n = 3f + 1$ adversary assumption). However, a closer analysis of the protocol shows that further synchronicity assumptions are required and therefore RandShare does not guarantee liveness under full asynchrony. The problem arises in paragraph II. D. 2. 1) [44], where $s_j(i)$ is used to denote the secret share of the secret $s_j(0)$, which node j sends privately to node i.

> Initialize a bit-vector $V_i = (v_{i1}, ..., v_{in})$ to zero, to keep track of valid secrets $s_j(0)$ received. Then wait until a message with share $s_j(i)$ from each $j \neq i$ has arrived.

In an asynchronous setting a node i cannot wait to receive a message from *each* other node j, as Byzantine nodes might never send such a message. Similarly, a node should not broadcast a negative vote in case no value $\hat{A}_j$ is received, as described in paragraph II. D. 2. 3), because this would imply a time bound for being able to send valid votes.[8]

D. Communication Complexity

In Table I, we outline the communication complexity of different approaches that provide randomness either as a standalone service or by deriving it from the characteristics of the underlying protocol. Thereby we consider the overall bits transmitted for all nodes per round, i.e. per produced random beacon value.

The different approaches exhibit a wide range of communication complexities. In the simplest scenario, where a Proof-of-Work blockchain forms the basis for the random beacon, a successful miner only has to perform one broadcast, leading to a complexity of $\mathcal{O}(n)$. This also applies for the Proof-of-Delay approach. Caucus also provides a low communication complexity of $\mathcal{O}(n)$ by leveraging the properties of the underlying Smart Contract platform.

For the Algorand and Ouroboros Praos protocols, an analysis of the communication complexity is not provided in the respective publications [22], [24]. We infer that Ouroboros Praos has a communication complexity in $\mathcal{O}(n)$, because the protocol only provides guarantees for eventual consensus and is based upon many of the design principles of Proof-of-Work blockchains, whereas protocols like Algorand, which provide consensus finality, generally operate at a higher per round communication cost. Both protocols use a similar approach based on private randomness, where a verifiable random function (VRF) is used to compute and verify a local source of randomness. The leader's local randomness is then combined with the previous global randomness to obtain the next global randomness. Used in this way, the communication complexity is only dependent on the underlying agreement protocol and does not incur any additional overhead.

To optimize the amount of data transmitted, the Algorand and Dfinity protocols perform certain communication-heavy operations only within a single subset of nodes. RandHound and RandHerd employ sharding to split nodes into multiple

[7] see section VI-C for detailed discussion on liveness problems in this setting

[8] Even if this issue is corrected, i.e., by modifying the protocol to only wait for $2f+1$ shares, and broadcasting negative votes only after receiving $2f+1$ valid messages, the protocol can not guarantee liveness, as the threshold of $2f + 1$ positive votes as described in paragraph II. D. 2. 4) may never be reached, and consequently the protocol aborts as stated in paragraph II. D. 3. 2).

smaller groups, where some operations are performed independently within all groups, and the results from individual groups are then combined in a final step. The required sizes for these subgroups typically depend on the assumptions in regard to the adversarial power and the described failure probability. Algorand is designed for a very large number of nodes, and the group size is $c \approx 1000$ [29]. Dfinity outlines a group size of $c = 405$ under a $n = 3f + 1$ security assumption and a failure probability of $2^{-40} \approx 10^{-12}$. As the authors outline, for small values of n, Dfinity's random beacon protocol may also be executed by all nodes, i.e. without selecting a committee as a subset of all nodes. In this case n nodes broadcast a signature share to all other nodes, leading to a complexity of $\mathcal{O}(n^2)$.

For RandHound and RandHerd, group sizes of 16, 24, 32 and 40 are considered by the authors. As we outline in Section VI-C, a group size of $c \geq 125$ is required to establish a failure probability similar to Algorand or Dfinity.

The approaches employed by Ouroboros, RandShare and Scrape are similar, where each node in the protocol employs a PVSS scheme to commit to a secret value. This involves the distribution of the PVSS shares, i.e. each node has to broadcast a message of size $\mathcal{O}(n)$ to all other nodes. The resulting communication complexity of $\mathcal{O}(n^3)$ is a major drawback of these approaches, however in this context (P)VSS can also help to achieve guaranteed output delivery [39].

HydRand is similar in this respect, as it also uses PVSS as an underlying primitive, but improves efficiency by a factor of $\mathcal{O}(n)$ because only a single node has to perform the distribution of PVSS shares per round. HydRand's communication complexity of $\mathcal{O}(n^2)$ includes all messages required to establish Byzantine agreement. The communication complexity is reduced by shifting the transmission of messages of size n to the leader and employing cryptographically signed *conformation/recovery certificates* to converge on a history of datasets. Messages that need to be broadcast by all nodes are always of constant size. In the evaluation section VIII, we provide information on the produced network traffic for different numbers of nodes (n) in practice.

E. Unpredictability

Unpredictability is a key property related to randomness that is provided by all compared protocols. We distinguish between the following two types of unpredictability that the protocols achieve:

1) all future random beacon values are fully unpredictable for all participants
2) the probability of predicting future random beacon values decreases exponentially with the number of rounds to predict

Protocols, where each round's random beacon is dependent on the input of a (Byzantine) quorum of participants, namely the protocols by Cachin et al., Ouroboros, Dfinity, RandShare, RandHound, RandHerd and Scrape, fall into the first category. For Proof-of-Work, Algorand, Caucus, Ouroboros Praos this is not the case and the next random beacon depends on a single node's (i.e. the miner's or the leader's) secret value. Clearly, since this node knows the next random number in advance it is able to predict the next random beacon value. In case adversarial nodes mine a sequence of blocks or are selected repeatedly as leader, prediction of more than one value is possible if they collude. This issue is typically addressed by a random selection of the respective leader, rendering prediction unlikely quickly. As long as the leader selection process ensures that honest nodes are selected with non-negligible probability, the probability of successful prediction decreases exponentially with the number of rounds to predict.

Proof-of-Delay can, in principle, achieve full unpredictability or unpredictability with high probability even though the next random beacon value depends on the output of a single node, because the leader (e.g. the miner who finds a valid PoW) does not immediately know the resulting random number that is derived from their output. If the leader tries to predict a future value, it has to withhold their output until it is able to finish the sequential computation required for the delay function. Depending on the underlying synchrony assumptions and consensus protocol, withholding the solution (e.g. block) for too long will either exclude the leader's output with certainty or high probability, as the delay parameter can be set much greater than the time bounds used for consensus.

In the context of unpredictability, HydRand offers both unpredictability with exponentially increasing probability for at most f rounds, as well as *full unpredictability* after $f + 1$ rounds. We provide a detailed analysis in section V-C, outlining the necessary waiting times to achieve an error margin of 10^{-12} for different participant numbers when waiting less than $f + 1$ rounds to achieve guaranteed unpredictability.

F. Bias-Resistance

Bias-resistance is the property that ensures a protocol's output cannot be manipulated by a (colluding) adversary, i.e. each random beacon value should be uniformly drawn from the set of possible values. Following the work in Cascudo at el. [21], we observe that bias-resistance is closely related to the property of guaranteed output delivery. In case an adversary can learn a candidate output and subsequently prevent the random beacon protocol from producing that output, the resulting beacon values are no longer guaranteed to be uniformly distributed. Even if an adversary is only able to prevent the output of a random beacon value to be available at some specific time, without having previously gained knowledge of the candidate value itself, bias resistance may not be guaranteed. Here, the synchrony requirements of the application(s) toward the delivery of new random beacon values determine biasability. In either cases, further security assumptions and additional primitives (e.g. PVSS or threshold signatures and $n > 2$ participants) are necessary if bias-resistance is to be guaranteed.

For all (of the compared) protocols, where the last interacting party can influence the random beacon value, this strong form of bias-resistance can not be ensured. This does not necessarily imply that an adversary can arbitrarily manipulate the probability distribution or, even worse, select a specific output. For example, the respective publications for Algorand and Ouroboros Praos show techniques to efficiently use this somewhat biasable form of randomness for the purpose of leader selection.[9] However, if the specific application requires a true uniform distribution of random beacon values, only protocols that provide the previously outlined strong notion of bias-resistance should be considered, namely the protocols by Cachin et al., Dfinity, Ouroboros, Proof-of-Delay, RandShare, RandHerd, Scrape and HydRand.

G. Computation and Verification Complexity

For our analysis we distinguish between (i) computation complexity, which describes the amount of operations an active protocol participant has to perform during one round of the protocol, and (ii) verification complexity, referring to the amount of computation an external (passive) observer of the protocol has to perform in order to verify the correctness of one random beacon value.

A main drawback of using Proof-of-Work and Proof-of-Delay as a source of randomness is the high computational complexity, as both approaches inherently rely on solving cryptographic puzzles as part of their security model. The other protocols herein considered have a computational complexity of at most $\mathcal{O}(n^3)$. The protocols RandShare and Ouroboros, which require $\mathcal{O}(n^3)$ due to the involved PVSS instances, may be optimized by updating the employed PVSS scheme to the variant introduced by Scrape [21]. The VRF based approaches from Algorand, Ouroboros Praos as well as Caucus (after the initial setup) are very efficient, because they only require the verification of a VRF or hash preimage.

In regard to verification complexity, when applying the optimization of the PVSS protocol introduced by Scrape, all protocol outputs can be verified reasonably efficiently, i.e. within $\mathcal{O}(n^2)$. For Proof-of-Delay, the high verification complexity may also be addressed by employing the very recently described verifiable delay functions, i.e. [11], [48], [37], [12].

The most efficient protocols in regard to verification are based on threshold signatures, VRFs and Proof-of-Work. We consider these approaches most suitable for verification within smart contracts or embedded devices, if fast implementations of the required cryptographic primitives are available within the specific platform.

VII. Discussion

Through the comparison in section VI we outline that there exist a variety of different approaches that can be used to implement a random beacon protocol. In the following, we briefly summarize their most defining characteristics.

Both Proof-of-Work [36] and Proof-of-Delay [17] are approaches that are well suited for a larger number of participants and can easily leverage on existing Proof-of-Work blockchains. While Proof-of-Work alone is not sufficient to establish bias-resistance, Proof-of-Delay can serve as an augmentation to achieve this guarantee with high probability. However, both approaches require a very high amount of computational resources. Proof-of-Delay may also serve as a suitable bootstrapping mechanism for generating an initial random value to be used in other protocols.

Ouroboros [33], RandShare [44], and Scrape [21] are PVSS based protocols. While the produced randomness of these approaches satisfies strong notions of unpredictability and bias-resistance, their high communication overhead significantly impacts scalability. Consequently, these protocols seem most suitable for a small scale setting (e.g. a private/consortium blockchain) or as an alternative for a Proof-of-Delay bootstrapping mechanism without the computational requirements.

Caucus [3] is an approach that can be deployed and efficiently verified within Smart Contracts but unfortunately cannot ensure bias-resistance.

Algorand [22] targets a (very) large set of nodes while still being able to provide consensus finality without requiring strong synchrony assumptions. As a trade-off, the protocol can not ensure a strong notion of bias-resistance. In this regard Ouroboros Praos [24] makes a similar trade-off to achieve better scalability at the cost of consensus finality and also weakening bias-resistance.

The randomness produced by the threshold signature based protocols of Cachin et al. [20] and Dfinity [32] provide strong bias-resistance. Additionally, Cachin et al. is the only protocol in our comparison that is proven secure in an asynchronous communication model. Dfinity's approach scales to a larger number of nodes, but security is only proven in a synchronous system model. The drawback of both protocols is their reliance on cryptographic primitives that are based on elliptic curve pairings, which are not yet well-established. E.g. Menezes et al. [35] and subsequently Barbulescu et al. [5] showed the security level of a commonly used pairing-friendly curve is in fact 2^{110} or 2^{100} instead of the targeted 2^{128}. Also these protocols require a trusted dealer or distributed key generation protocol.

RandHound [44] and RandHerd [44] employ a sharding approach to achieve good scalability for a large number of participants. RandHound does not provide a strong notion of bias-resistance while RandHerd requires additional view-change and agreement protocols when a leader is Byzantine or non-available.

HydRand, is a dedicated random beacon protocol tailored towards continuous operation and assumes a small to medium set of nodes. The protocol can provide the same strong properties as the related PVSS based approaches while reducing the communication overhead by $\mathcal{O}(n)$. This is achieved without requiring a trusted dealer or DKG in the initial setup, thereby avoiding the introduction of additional security assumptions and implementation complexity. Moreover, a detailed analysis and security proofs of the protocol's properties and guarantees are provided. HydRand furthermore ensures *guaranteed output delivery*, i.e. a new random beacon value is guaranteed to be produced at each round, regardless of the adversary's actions.

[9] Both publications are aware of, and analyze the fact that the distribution of random numbers produced by their approaches is not uniform and consider the potential implications [22], [24].

This is of particular importance for application scenarios in which strong synchrony requirements or gapless delivery of new random beacon values is required.

VIII. Evaluation

To outline the feasibility and practicability of HydRand, we develop a fully functional protocol prototype in Python, and make our code publicly available on Github [42]. The evaluation was performed by executing the HydRand protocol on Amazon EC2 *t2.micro* instances (1 GiB of RAM, one virtual CPU core, 60-80 Mbit/s network bandwidth). To simulate an execution over the internet, instances were spread equally over multiple data centers in eight AWS regions (Canada, London, Ireland, N. California, N. Virginia, Paris, Singapore and Tokio).

Executions were performed both, with correct nodes only, as well as considering up to f simulated node failures. The synchronous round duration was derived experimentally and leaves room for improvement, as both the resource and network capacity of the instances can be adjusted, and the protocol code may be further optimized. Figure 2 presents the throughput our protocol achieves within the aforementioned setup conditions for different numbers of nodes (n) where failures can occur.

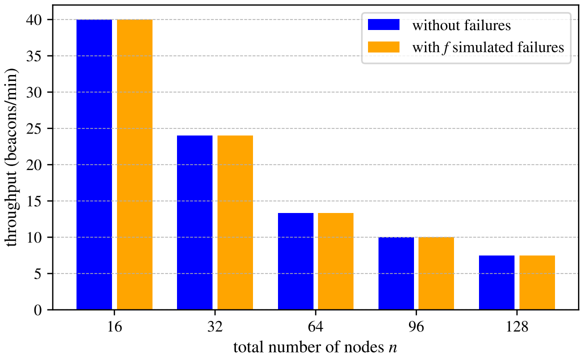

Fig. 2: Random beacon throughput per minute (bpm) for different numbers of nodes (n) with f simulated failures.

Figure 3 outlines the average bandwidth used per node over the duration of 30 minutes of protocol execution, with and without simulated failures. It should be noted that the protocol parametrization, specifically the round duration, is the same only for executions with the same n, i.e. for faulty and correct executions for a particular n, and corresponds to that of Figure 2. In the presented data it appears executions with simulated failures grow linearly with the size of n, whereas the average amount of data transferred per node in failure-free runs appears almost constant for different sizes of n. This is expected, given that executions with simulated failures induce larger message exchanges between participants if a leader has to be recovered and therefore consume more bandwidth than runs without failures.

In regards to verification performance, we measure that an (external) client can publicly verify the correctness of a round's random beacon in ≈ 57ms, considering the worst case in a setting with with 128 nodes. The corresponding proof, which enables non-interactive verification, is ≈ 26kB in size.

The results of our research prototype evaluation highlight that the presented HydRand protocol is practicable for realistic deployment scenarios. Data from our performed executions suggests that the beacon throughput for large n was restricted by the computational capacity of the AWS instances. An evaluation of the effects of different parametrizations, including the utilization of more powerful instances, as well as an analysis of resource consumption under more complex adversarial behavior, is deferred to future work.

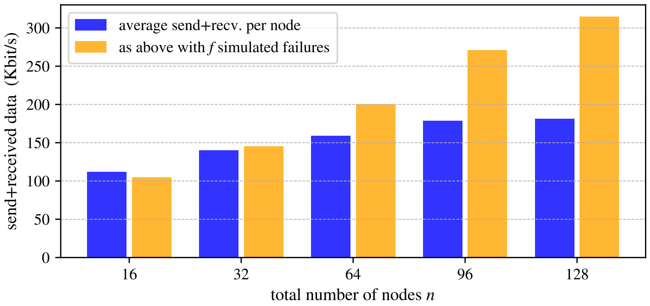

Fig. 3: Average used bandwidth per node in Kbit/s for different total (n), with and without simulated failures.

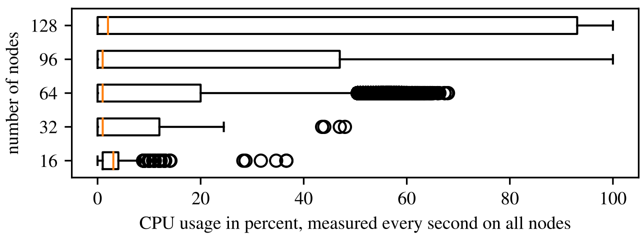

Fig. 4: Total CPU utilization (%), measured every second on all nodes (n) for different runs (y axis).

IX. Conclusion

We present HydRand, a synchronous random beacon protocol that tolerates up to one third Byzantine failures and show that it provides *liveness*, *public-verifiability*, *bias-resistance*, and probabilistic as well as hard bounds for *unpredictability*. HydRand ensures *guaranteed output delivery*, namely that randomness is produced at regular intervals, even under adversarial conditions. The protocol is designed for stand-alone use, but could also find utility in the context of current and future Proof-of-Stake and permissioned blockchain or consensus protocols. Further, we provide the first in-depth comparison and discussion of novel approaches for generating publicly-verifiable randomness, which enables researchers to compare current as well as future designs objectively with each other. Thereby, we highlight that HydRand achieves various desirable properties in a unique way without incurring major drawbacks: (i) it is a stand-alone protocol that can be readily adapted for different use-cases, (ii) it neither requires a trusted dealer nor a distributed key generation protocol, and (iii) it offers strong guarantees for the produced randomness while improving upon the performance and scalability of previous solutions with comparable guarantees.

Furthermore, we develop and evaluate a fully functional protocol prototype in Python to demonstrate the feasibility and practicability of HydRand and make our code publicly available on Github [42].

Acknowledgments

We thank the anonymous reviewers for the valuable insights and feedback. We further thank all of the contacted authors, which provided additional information and helped to clarify any ambiguties for our comparison.

References

[1] B. Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.

[2] N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on Ethereum smart contracts. In *International Conference on Principles of Security and Trust*, pages 164–186. Springer, 2017.

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

[4] T. Baigneres, C. Delerablée, M. Finiasz, L. Goubin, T. Lepoint, and M. Rivain. Trap me if you can-million dollar curve. *IACR Cryptology ePrint Archive*, 2015:1249, 2015.

[5] R. Barbulescu and S. Duquesne. Updating key size estimations for pairings. *Journal of Cryptology*, pages 1–39, 2017.

[6] M. Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30. ACM, 1983.

[7] I. Bentov, A. Gabizon, and D. Zuckerman. Bitcoin beacon, 2016.

[8] I. Bentov, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake, 2016.

[9] D. J. Bernstein, T. Lange, and R. Niederhagen. Dual ec: a standardized back door. In *The New Codebreakers*, pages 256–281. Springer, 2016.

[10] M. Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 15(1):23–27, 1983.

[11] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. *IACR Cryptology ePrint Archive*, 2018:601, 2018.

[12] D. Boneh, B. Bünz, and B. Fisch. A survey of two verifiable delay functions. 2018.

[13] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Eurocrypt*, volume 2656, pages 416–432. Springer, 2003.

[14] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. *Advances in Cryptology ASIACRYPT 2001*, pages 514–532, 2001.

[15] J. Bonneau, J. Clark, and S. Goldfeder. On Bitcoin as a public randomness source. *IACR Cryptology ePrint Archive*, 2015:1015, 2015.

[16] J. Bonneau, J. Clark, and S. Goldfeder. On bitcoin as a public randomness source, 2015. Accessed: 2015-10-25.

[17] B. Bünz, S. Goldfeder, and J. Bonneau. Proofs-of-delay and randomness beacons in Ethereum. In *S&B '17: Proceedings of the 1st IEEE Security & Privacy on the Blockchain Workshop*, April 2017.

[18] V. Buterin. Validator Ordering and Randomness in PoS, 2016. Accessed: 2018-04-24.

[19] V. Buterin. Randao++, 2017. Accessed: 2018-04-24.

[20] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 123–132. ACM, 2000.

[21] I. Cascudo and B. David. Scrape: Scalable randomness attested by public entities, 2017.

[22] J. Chen and S. Micali. Algorand. *arXiv preprint arXiv:1607.01341*, 2017.

[23] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[24] B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. Technical report, Cryptology ePrint Archive, Report 2017/573, 2017., 2017.

[25] Dfinity Stiftung. Threshold Relay: How to Achieve Near-Instant Finality in Public Blockchains using a VRF, 2017. Accessed: 2017-08-20.

[26] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. volume 34, pages 77–97. ACM, 1987.

[27] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. volume 35, pages 288–323. ACM, 1988.

[28] M. Ghosh, M. Richardson, B. Ford, and R. Jansen. A torpath to torcoin: proof-of-bandwidth altcoins for compensating relays. Technical report, NAVAL RESEARCH LAB WASHINGTON DC, 2014.

[29] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[30] S. Goel, M. Robson, M. Polte, and E. Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

[31] D. Goulet and G. Kadianakis. Random number generation during tor voting. *Tors protocol specifications–Proposal*, 250, 2015.

[32] T. Hanke, M. Movahedi, and D. Williams. Dfinity technology overview series consensus system, 2018. Rev. 1.

[33] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol, 2016.

[34] A. K. Lenstra and B. Wesolowski. A random zoo: sloth, unicorn, and trx. *IACR Cryptology ePrint Archive*, 2015:366, 2015.

[35] A. Menezes, P. Sarkar, and S. Singh. Challenges with assessing the impact of nfs advances on the security of pairing-based cryptography. In *International Conference on Cryptology in Malaysia*, pages 83–108. Springer, 2016.

[36] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Dec 2008.

[37] K. Pietrzak. Simple verifiable delay functions. *IACR Cryptology ePrint Archive*, 2018:627, 2018.

[38] M. O. Rabin. Randomized byzantine generals. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 403–409. IEEE, 1983.

[39] M. O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256–267, 1983.

[40] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85. ACM, 1989.

[41] randao.org. Randao: Verifiable Random Number Generation, 2017. Accessed: 2018-04-24.

[42] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl. Python implementation of the HydRand protocol, 2019. https://github.com/PhilippSchindler/hydrand. Accessed: 2019-04-01.

[43] B. Schoenmakers. A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting. In *Annual International Cryptology Conference*, pages 148–164. Springer, 1999.

[44] 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 *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 444–460. IEEE, 2017.

[45] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping authorities" honest or bust" with decentralized witness cosigning. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 526–545. Ieee, 2016.

[46] P. Syverson, R. Dingledine, and N. Mathewson. Tor: The secondgeneration onion router. In *Usenix Security*, 2004.

[47] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152. ACM, 2015.

[48] B. Wesolowski. Efficient verifiable delay functions. *IACR Cryptology ePrint Archive*, 2018:623, 2018.

[49] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. In *OSDI*, pages 179–182, 2012.

Appendix A
Example Protocol Execution

Figure 5 shows four rounds of an example execution of the HydRand protocol in a setting of $f = 2$ Byzantine nodes. The sequence of randomly selected leaders in this example execution includes a worst case scenario, where f distinct leaders were drawn from the set of Byzantine nodes (nodes n_4 and n_5), followed by a correct node and then again the first Byzantine node (n_4).

Round r_1: In this execution the first node that gets selected as the leader (i.e., node n_4) belongs to the set of Byzantine nodes. This leader selectively sends a *propose* message only to a subset of correct nodes. In our case the nodes n_1, n_2 and n_3. Moreover, the Byzantine node n_5 only sends *acknowledge* messages to the very same nodes (n_1, n_2, n_3). After that phase,

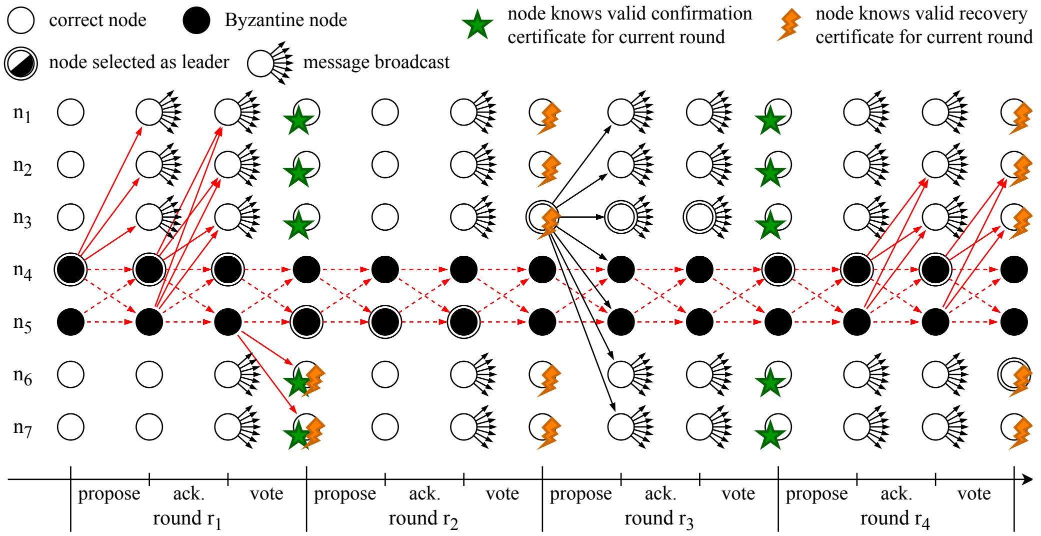

Fig. 5: Example execution of four rounds of the HydRand protocol with $n = 3f + 1 = 7$.

the Byzantine node n_5 sends a *recover* message to the nodes n_6 and n_7.

This leads to a situation where the correct nodes n_1, n_2 and n_3 receive $2f + 1$ acknowledge messages. Therefore, those nodes (n_1, n_2 and n_3) broadcast *confirm* messages which together form a valid confirmation certificate known to every node. Further, the nodes n_6 and n_7 as well as the adversary are in possession of a valid recovery certificate $RC(r_1)$, as nodes n_5, n_6 and n_7 sent out *recover* messages.

Round r_2: The next node (n_5) that is selected as leader is also in the set of Byzantine nodes and does not broadcast any message. Therefore, the secret value of the rounds leader gets reconstructed at the end of the *vote* phase and all nodes are only in possession of a *reconstruction certificate* $RC(r_2)$ for this round.

Round r_3: The leader (n_3) of this round belongs to the set of correct nodes and has received $f + 1$ *confirm* messages in round r_1. Moreover, node n_3 is not in possession of a valid recovery certificate for r_1 since he has only received f *recover* messages, i.e. from node n_6 and n_7 but not from node n_5. Therefore, the leader broadcasts a new dataset D_3 containing a valid *confirmation certificate* $CC(D_1)$ for round r_1, as well as a *recovery certificate* $RC(r_2)$ for round r_2.

After receiving the *propose* message, all correct nodes, including n_6 and n_7, are safe to assume that at least $f+1$ correct nodes are in possession of dataset D_1. The justification for this assumption comes from the fact that the *propose* message contains a *confirmation certificate* composed of $f + 1$ signed messages including the hash $H(D_1)$ of D_1. This necessarily includes at least one honest node which, per definition, only sends a confirm message if it has received $2f + 1$ valid *acknowledge* messages in advance. Therefore, at least $f + 1$ correct nodes have to be in possession of dataset D_1. As a result, all correct nodes accept this rounds new dataset D_3 containing $CC(D_1)$. This holds true, even for nodes n_6 and n_7 although they have not received dataset D_1.

If node n_6 or n_7 would have been selected as leader in round r_3, then this node would have constructed a dataset D_3 that contains a valid *recovery certificate* for round r_1 and r_2 as well. In that case the nodes n_1, n_2 and n_3 would have discarded their dataset D_1.

Round r_4: In this round node n_4 is again selected as leader. This is valid since f rounds have passed since this node has been selected as leader. Therefore, at least one correct node was selected as leader in between – in this case node n_3. Since there is no *recovery certificate* $RC(r_3)$ for round r_3 available, all further leaders have to include the *confirmation certificate* $CC(D_3)$ for round r_3 to extend upon the chain of valid datasets. Otherwise their future datasets would not be valid and rejected by all correct nodes. Therefore, all nodes including node n_4, have to accept the view of node n_3 in this case.

In our example, node n_4 attempts to stall the protocol by selectively releasing a new dataset D_4 only to the nodes n_2, n_3. But since those nodes are not able to reach the required number of $2f+1$ *acknowledge* messages (together with the Byzantine nodes n_4 and n_5), no correct node will send a *confirmation* message in the last phase of this round. As a result all correct nodes will send *reconstruct* messages leading to a total of $2f+1$ reconstruct messages, which is more than $f + 1$ and hence enough to form a *reconstruction certificate* and to reconstruct the leader's secret for round r_4 given the decrypted shares of n_1, n_2 and n_3.

Note that, although possible, the PVSS reconstruction of the secret from r_1 would not be necessary here, since in this example the leader of r_4 selectively sent out a new dataset and therefore revealed the secret to at least one correct node, namely n_2 and n_3. Per definition, correct nodes broadcast the revealed secret in their *acknowledge* messages. Therefore, all other correct nodes receive the revealed secret in round r_4 even if they have not received the dataset D_3 directly.

APPENDIX B
NOTATION REFERENCE

TABLE II: Symbols

Symbol	Description
f	number of Byzantine nodes
n	number of all nodes, defined as $n = 3f + 1$
t	reconstruction threshold for PVSS, defined as $t = f + 1$
i	a node as defined by context
r, k, x	some round as defined by context
ℓ	leader of the current round r
ℓ_x	leader of round x
$H(\cdot)$	cryptographic hash function
$\langle sk_i, pk_i \rangle$	private/public keypair of node i
$\langle m \rangle_i$	some message m signed using the secret key sk_i of node i
$\|\|$	string/list concatenation
R_x	randomness of round x
D_x	dataset of some round x, consists of a $header(D_x)$ and $body(D_x)$
$H(D_x)$	cryptographic hash of the $header(D_x)$
$\overleftarrow{x}$	previous round of round x, such that there exists a valid dataset for round $\overleftarrow{x}$
$D_{\overleftarrow{x}}$	previous dataset referenced in dataset D_x
$\mathcal{P}$	set of all nodes (processes), $\mathcal{P}$ is of size n
$\mathcal{P}_x$	set of available nodes for some round x, i.e., set of all nodes excluding recovered nodes till round x
$\mathcal{L}_x$	set of potential leaders for some round x, i.e., set of all nodes excluding recovered nodes till round x and excluding nodes that have been selected as leader within the last f rounds
$rn(D_x)$	set of recovered nodes upto block D_x

Symbol	Description
q	prime number q
$\mathbb{Z}_q$	ring of integers modulo q
$\mathbb{G}_q$	multiplicative group of order q, in which the discrete log problem hard
h	generator for the group $\mathbb{G}_q$
s	underlying secret value, a dealer wants to share with PVSS, $s \in \mathbb{Z}_q$
$Com(s)$	PVSS commitment to the value s, includes commitments to the coefficients of the underlying polynomial, encrypted shares and a NIZK correctness proof.
h^s	result of the reconstruction process for a commitment $Com(s)$
$\hat{s_i}$	encrypted share for node i, part of the commitment $Com(s)$
$Com(s)[s_i]$	node i's decrypted share for the commitment $Com(s)$, result of decrypting $\hat{s_i}$ using i's private key
s_ℓ	current leader's previously committed secret value.
$s_\ell^\star$	current leader's new randomly selected secret value.
$Com(s_\ell)$	current leader's previous commitment
$Com(s_\ell^\star)$	current leader's new commitment
$CC(D_x)$	*commit certificate* of dataset D_x that contains at least $f + 1$ valid *confirmation* messages.
$RC(x)$	*recovery certificate* of round x that contains at least $f + 1$ valid *recover* messages.
M_x	root of a Merkle tree for the shares $\hat{s_1}, \hat{s_2}, ..., \hat{s_n}$ for ℓ_x's commitment $Com(s_{\ell_x})$ in round x
$M_x[\hat{s_i}]$	merkle branch for $\hat{s_i}$, showing that $\hat{s_i}$ is under the Merkle root M_x (and thus part of D_x)

TABLE III: Messages

Message	Description
$\left\langle propose, \langle header(D_r) \rangle_\ell, body(D_r) \right\rangle_\ell$	The message that is broadcasted by correct leaders in the *propose* phase of each round.
$\left\langle \langle acknowledge, r, H(D_r) \rangle_i, \langle header(D_r) \rangle_\ell \right\rangle_i$	The message that is broadcasted by correct nodes that received a valid *propose* messages from the leader of the current round. Broadcasting this messages ensures that the leader cannot equivocate.
$\left\langle confirm, r, H(D_r) \right\rangle_i$	The message that is broadcasted by correct nodes that received $2f + 1$ valid *acknowledge* messages from other nodes during this round. Any node which received $f + 1$ of these messages can construct a valid *confirmation certificate* for round r.
$\left\langle \langle recover, r \rangle_i, s_\ell, Com(s_\ell)[s_i], \hat{s_i}, M_k[\hat{s_i}], R_{r-1} \right\rangle_i$	The message that is broadcasted by correct nodes that did not receive a valid *propose* message from the leader at the beginning of this round. Any node which received $f + 1$ of these messages can reconstruct a valid *recovery certificate* for round r.
$(f + 1) \times \langle confirm, r, H(D_r) \rangle_i$	commitment certificate $CC(D_r)$ for dataset D_r with hash $H(D_r)$ (valid if it contains correctly signed messages from $f + 1$ different nodes i)
$(f + 1) \times \langle recover, r \rangle_i$	recovery certificate $RC(r)$ for round r (valid if it contains correctly signed messages from $f + 1$ different nodes i)