SPURT: Scalable Distributed Randomness Beacon with Transparent Setup

Sourav Das, Vinith Krishnan, Irene Miriam Isaac and Ling Ren
University of Illinois at Urbana-Champaign
{souravd2, vinithk, irenemi2, renling}@illinois.edu

Abstract—**Having shared access to high-quality random numbers is essential in many important applications. Yet, existing constructions of distributed random beacons still have limitations such as imperfect security guarantees, strong setup or network assumptions, or high costs. In this paper, we present SPURT, an efficient distributed randomness beacon protocol that does not require any trusted or expensive setup and is secure against a malicious adversary that controls up to one-third of the nodes in a partially synchronous network. We formally prove that each output of SPURT is unpredictable, bias-resistant, and publicly verifiable. SPURT has an amortized total communication cost of $O(\lambda n^2)$ per beacon output where λ is the security parameter. While designing SPURT, we augment existing State Machine Replication (SMR) protocols to ensure that all nodes decide nearly simultaneously. We also design a publicly verifiable secret sharing (PVSS) scheme whose security is based on the standard Decisional Bilinear Diffie-Hellman assumption and does not require a Random Oracle. We implement SPURT and evaluate it using a network of up to 128 nodes running in geographically distributed AWS instances. Our evaluation shows that SPURT can produce about 84 beacon outputs per minute in a network of 32 nodes and is comparable to systems with stronger assumptions or weaker security.**

I. INTRODUCTION

A reliable source of a continuous stream of shared randomness, also known as a *random beacon*, is crucial for many distributed protocols. Applications of random beacon include leader election in proof-of-stake based blockchains [4], [41], blockchain sharding [7], [54], [57], [76], scaling smart contracts [31], annonymous communications [5], [42], [73], [74], solving consensus under asynchrony [37], anonymous browsing [33], [40], [44], publicly auditable auctions and lottery [18], electronic voting [6], cryptographic parameter generations [9], [55].

The simplest approach to achieve a random beacon is to rely on a single node or organization such as NIST random beacon [52] or Random.org [47]. This is undesirable due to security incidents such as a backdoor in the Dual elliptic curve pseudorandom number generator [12] and 1969 US conscription lottery [71]. Such an approach is also unreasonable in systems such as blockchains, where using a trusted party for randomness generation defeats the blockchain's main objective of avoiding central authorities.

A natural approach to remove the trusted third party is to decentralize the process of generating randomness among many nodes using a distributed protocol. As long as a large fraction (majority or supermajority) of nodes faithfully follow the protocol, the protocol will produce the shared randomness with desired properties. Briefly, any randomness beacon protocol should be *available* and each beacon output should be *unpredictable, bias-resistant* and *publicly verifiable*. Informally, *unpredictability* requires that no one can compute any non-trivial information about future beacon outputs, *bias-resistance* requires that beacon outputs are independently sampled from a uniform distribution, and *public verifiability* enables external clients to validate the correctness of beacon outputs.

Existing works. Starting from Blum's two-node coin tossing protocol [15], a long line of works have looked into the problem of generating shared randomness under different system models [2], [8], [15], [21], [24], [25], [32], [41], [53], [60], [68], [72]. Due to its use in practical blockchain systems, which typically involves a large number of nodes [31], [54], [57], [76], recent randomness beacon protocols put an emphasis on scalability. Specifically, it is desirable to construct a beacon protocol that has low latency, low communication complexity, and low computation cost per node per beacon output. Also, since many of these protocols are decentralized and seek to eliminate trusted entities, it is preferable that the beacon protocol does not rely on a trusted setup.

Despite decades of research and many breakthroughs, existing distributed randomness beacon protocols have scalability issues, require strong cryptographic or network assumptions, or do not provide the full suite of desired properties.

Notably, many protocols [2], [21], [23] assume an initial trusted setup, where a trusted party generates private keys for all nodes. Security of such protocols relies crucially on the trusted party's ability and willingness to keep these keys safe. Some protocols replace the trusted setup with a Distributed Key Generation (DKG) procedure [38], [72]. However, DKG comes with a high initial setup cost. Another limitation of using DKG, as observed in [13], is the inability/inefficiency to replace nodes. Whenever a participating node is to be replaced, we need to rerun the expensive DKG procedure. Thus, DKG-based solutions such as [2], [22], [23], [38] are efficient when nodes are fixed, but are not suitable in applications where nodes change frequently (e.g., proof-of-stake [41], [53]).

Only recently, the community has started to explore distributed random beacons without a trusted setup. These protocols currently either have poor efficiency and/or do not provide full security. For example, protocols such as [25], [29], [53] have at least $O(\lambda n^4)$ communication cost per beacon output, where λ is the security parameter and n is the number of nodes

Table I: Comparison of existing randomness beacon protocol.

	Network model	Fault Tolerance	Adaptive Adversary	Liveness / Availability	Unpredictability	Bias-resistance	Communication Cost (total)	Computation Complexity	Public Verification Complexity	Cryptographic Primitives	Setup Assumption
Cachin et al. [60]	async.	1/3	✗	✓	✓	✓	$O(\lambda n^2)$	$O(n)$	$O(1)$	Uniq. th-sig.	DKG
RandHerd [72]*	async.	1/3♣	✗	✓	✓	✓	$O(\lambda c^2 \log n)$♣	$O(c^2 \log n)$	$O(1)$	PVSS+CoSi	DKG
Dfinity [23]	partial sync.	1/3	✗	✓	✓	✓	$O(\lambda n^2)$	$O(n)$	$O(1)$	Uniq. th-sig.	DKG
Drand [2]	sync.	1/2	✗	✓	✓	✓	$O(\lambda n^2)$	$O(n)$	$O(1)$	Uniq. th-sig.	DKG
HERB [29]	sync.	1/3	✗	✓	✓	✓	$O(\lambda n^4)^{\ddagger}$	$O(n)$	$O(n)$	Partial HE	DKG
Algorand [41]	partial sync.	1/3♣	✗	✓	$\Omega(t)$	✗	$O(\lambda cn)$♣	$O(c)$	$O(1)$	VRF	CRS
Proof-of-Work [60]	sync.	1/2	✗	✓	$\Omega(t)$	✗	$O(\lambda n)$	very high	$O(1)$	Hash func.	CRS
Ouroboros [53]	sync.	1/2	✗	✓	✓	✓	$O(\lambda n^4)^{\ddagger}$	$O(n^3)$	$O(n^3)$	PVSS	CRS
Scrape [25]	sync.	1/2	✗	✓	✓	✓	$O(\lambda n^4)^{\ddagger}$	$O(n^2)$	$O(n^2)$	PVSS+Broadcast	CRS
Hydrand [68]	sync.	1/3	✗	✓	$t+1$	✓	$O(\lambda n^2 \log n)$	$O(n)$	$O(n)$	PVSS	CRS
RandRunner [67]	sync.	1/2	✓	✓	$t+1$	✓	$O(\lambda n^2)$	VDF	$O(1)$	VDF	CRS
GRandPiper [13]	sync.	1/2	✗	✓	$t+1$	✓	$O(\lambda n^2)$	$O(n^2)$	$O(n^2)$	PVSS	$q-$SDH
BRandPiper [13]	sync.	1/2	✓	✓	✓	✓	$O(\lambda n^3)$	$O(n^2)$	$O(n^2)$	VSS	$q-$SDH
SPURT	partial sync.	1/3	✗	✓	✓	✓	$O(\lambda n^2)$	$O(n)$	$O(n)$	PVSS+Pairing	CRS

* RandHerd uses RandHound as a one-time setup phase. RandHound is driven by a leader node and hence its liveness requires the leader to be honest. As presented, RandHerd is biasable and need additional techniques to be unbiasable.

♣ Algorand and Randherd use a randomly sampled committee of size c to run the protocol. This is an orthogonal technique and can be applied to most other protocols in the table to improve scalability at the cost of slightly reducing fault tolerance.

‡ Scrape, Ouroboros and HERB assume a broadcast channel (or blockchain) and every node uses the broadcast channel to share $O(n)$ groups elements. Even with best known broadcast protocols, its total communication complexity would be $O(\lambda n^4)$. HERB has three variants depending upon the underlying broadcast channel. We report their first variant since it uses standard metric for measuring broadcast channel communication cost.

running the protocol. A recent work Hydrand [68] reduces the communication cost to $O(\lambda n^2)$ but offers poor unpredictability, even against a semi-honest adversary. Very recently, a concurrent work Brandpiper [13] improves upon Hydrand to provide perfect unpredictability and increased fault tolerance. As a trade-off, it incurs $O(\lambda n^3)$ worst-case communication cost and makes the q-SDH assumption, which either requires a trusted setup or a secure multi-party computation protocol to generate public parameters.

Furthermore, all existing protocols without trusted setup rely on synchronous networks that assume a known upper bound on the network delay. If the delay assumption is violated, these protocols will fail to provide the desired security properties. They are hence less robust in wide area networks. Another advantage of a partially synchronous protocol over a synchronous one is that such protocol can be made responsive [62], i.e., they can generate beacons at the rate of true network speed, as opposed to some pre-determined conservative parameters.

An orthogonal and effective approach is the sampling technique [41], [72], which samples a subset of nodes to form a committee, and then runs a random beacon protocol in the committee. Note that after sampling, we still need a random beacon protocol to run on the committee. Again, if the committee rarely changes, a DKG-based random beacon protocol is suitable, but no efficient solution exists if the committee changes frequently.

We summarize existing works in Table I and will provide more details about each protocol in §VIII.

Our results. In this paper, we design SPURT, a distributed random beacon protocol that does not require any trusted or expensive setup. SPURT guarantees unpredictability, bias-resistance, availability, and public verifiability in a partially synchronous network [35] against a malicious adversary that controls up to one-third of the nodes. In a network of n nodes, SPURT's communication cost per beacon output (across all nodes) is $O(\lambda n^2)$ = where λ is a security parameter representing the size of group elements. Each node in SPURT performs $O(n)$ group exponentiation per beacon output. Furthermore, SPURT is *responsive* [62], i.e., it can produce beacon outputs at the actual speed of the network, as opposed to any pre-determined conservative parameters With these properties, we believe SPURT has good scalability and is suitable for applications with a large number of nodes deployed globally across the Internet (possibly by combining with the sampling technique).

While designing SPURT, we augment existing state machine replication (SMR) protocols to guarantee that all honest nodes decide nearly simultaneously, i.e., within two message delays, while maintaining quadratic communication. We also design a new publicly verifiable secret sharing (PVSS) scheme whose security relies on the standard Decisional Bilinear Diffie-Hellman (DBDH) assumption [17] and does not require a Random Oracle. These results can be of independent interests.

We implement SPURT in Golang and evaluate it with up to 128 geographically distributed nodes. We compare with recent works Hydrand [68] and Drand [2] in terms of throughput and network bandwidth usage. Note that Hydrand provides poor unpredictability, and Drand requires a DKG setup. Our evaluation illustrates that SPURT can generate beacon outputs at a rate comparable to or better than Drand and Hydrand. For example, with 32 nodes, SPURT can generate 84 beacon outputs every minute, which is approximately $1.1\times$ and $3.5\times$ higher than Drand and Hydrand, respectively. SPURT has a network bandwidth cost of 35 Kilobytes with 32 nodes, which is approximately $5\times$ higher than Drand and is 55% of Hydrand.

Paper organization. The rest of the paper is organized as follows. We describe the system model and an overview of SPURT in §II. In §III, we give preliminaries and notations. We then provide details of our new PVSS scheme in §IV. We describe SPURT in detail in §V and analyze its security and complexity in §VI. We present our prototype implementation and evaluation results in §VII. We describe related work in detail in §VIII and conclude with a discussion in §IX.

II. System Model and Overview of SPURT

In this section, we describe our system model, provide an overview of SPURT, and discuss the main challenges we face and how we address them.

A. System Model

We consider a network of n nodes connected via pair-wise authenticated channels. We assume that at most $t < n/3$ nodes can be malicious and they are controlled by a single adversary $\mathcal{A}$. The remaining nodes are honest and strictly follow the specified protocol. We assume a standard public-key infrastructure, i.e., every node in the system is aware of every other node's public key in the system. Throughout this paper, we assume that all messages exchanged between honest nodes are *digitally signed* by the sender, and the recipients validate the messages before processing them further.

We assume that at the start of the protocol, all honest nodes agree on public parameters $g_0, h_0 \in \mathbb{G}_0$ and $g_1, h_1 \in \mathbb{G}_1$, which are randomly and independently chosen generators of $\mathbb{G}_0$ and $\mathbb{G}_1$. This is a common reference string (CRS) setup. We assume $\mathcal{A}$ cannot break standard cryptographic constructions such as hash functions, signatures, and the ones specified in §III.

We assume the network is partially synchronous [35], i.e., it oscillates between periods of synchrony and periods of asynchrony. During periods of synchrony, all messages sent by honest replicas adhere to a known delay bound Δ. During periods of asynchrony, messages can be delayed arbitrarily.

B. Desired Properties of Randomness Beacon

Intuitively, a randomness beacon should be *bias-resistant* and *unpredictable*, i.e., beacon outputs are (computationally) indistinguishable from random and an adversary does not learn them much sooner than honest nodes.

Definition 1 (Bias-resistance and unpredictability)**.** Let $o_1, o_2, \ldots, o_i$ be the beacon outputs generated so far. The beacon protocol is bias-resistant and unpredictable, if for every future beacon output o_j where $j > i$, for all PPT adversaries $\mathcal{A}$, that knows the beacon output generated so far and associated protocol transcripts, there exists a negligible function $\epsilon(\lambda)$ such that:

$$|\Pr[\mathcal{A}(o_j) = 1] - \Pr[\mathcal{A}(u) = 1]| < \epsilon(\lambda) \qquad (1)$$

where $u \leftarrow U$ is a uniformly random element from a set U, and that all honest nodes learn o_j within a small constant number of rounds from the time the adversary learns it.

In addition to unpredictability and bias-resistance, any beacon protocol should also guarantee *availability*, i.e., the protocol keeps producing new beacon outputs, and *public-verifiability*, i.e., beacon outputs can be efficiently verified even by users that do not directly participate in the beacon generation protocol. A randomness beacon protocol in the partially synchronous model should ensure that every beacon output is unpredictable, bias-resistant, and publicly verifiable even during periods of asynchrony, and guarantees availability during periods of synchrony.

C. *Overview of* SPURT

Existing protocols that do not rely on trusted setup use publicly verifiable secret sharing (PVSS) schemes as a crucial building block. We will also start with this design paradigm. Briefly, the idea is that, for every beacon output, each node runs a concurrent instance of PVSS to share a randomly chosen secret with every other node. Once the sharing phase finishes for $n - t$ nodes, the shares are reconstructed and aggregated to compute the beacon output. This way, each beacon output has contributions from some honest nodes and thus remains hidden from the adversary before reconstruction.

The downside of naïvely using PVSS is that PVSS schemes assume broadcast channels. In fact, this is the major source of high communication complexity. A broadcast channel, when actually implemented using a distributed protocol, has a communication lower bound of $\Omega(n^2)$ [34]. Some works [25], [29], [53] need to send $O(n^2)$ amount of data over the broadcast channel and that leads to their $O(n^4)$ costs.

Another major downside of a broadcast channel is that it is impossible to achieve in partial synchrony, the network model we target. This is also why most previous works have to stick with the stronger synchrony model. This motivates us to revisit the use of broadcast channels. Previous works such as Scrape [25] and Hydrand [68] explicitly mentioned that for a beacon to be bias-resistant and available, the sub-protocol invoked for each beacon output must provide *guaranteed output delivery* [30]. The use of broadcast channels then becomes natural as it is the standard technique in the multi-party computation literature [43], [64] to achieve guaranteed output delivery. However, we observe that guaranteed output delivery is not necessary. Instead, we just need to ensure that the adversary cannot abort a beacon output *after* learning the output. This allows us to use a state machine replication protocol (SMR) (cf. §III-C) instead of broadcast channels, which enables us to handle a partially synchronous network.

One main design philosophy of SPURT is to lower the amount of data sent via the broadcast channel or state machine replication (SMR) protocol. We utilize the additive homomorphism of commitments and encrypted shares in PVSS and have a leader collect and aggregate the PVSS messages from all other nodes. Note that the leader can verify the PVSS messages from each node but cannot learn any information from them because PVSS shares are encrypted under the public keys of corresponding nodes.

The first main challenge we face is that a malicious leader may not correctly aggregate the PVSS messages. In fact, a malicious leader may not aggregate anything at all and may instead claim an adversarially chosen polynomial as the aggregated polynomial (PVSS commitments are polynomials, cf. §IV). Such an attack will immediately violate unpredictability and bias-resistance. Note that we cannot ask the leader to forward all pre-aggregation messages to all other nodes since that would consume $O(\lambda n^3)$ network bandwidth.

SPURT addresses this issue with the following novel approach. Each non-leader node checks a disjoint part of the aggregation result such that any subset of $t+1$ honest nodes collectively checks the entire aggregation. Validating part of the aggregation requires a message of size $O(\lambda n)$. So the leader can send the necessary information to each node using a total of $O(\lambda n^2)$ communication.

After aggregation, the leader is supposed to broadcast the aggregated PVSS message to all the nodes. However, note that the aggregated message is still $O(\lambda n)$ in size, and it would still consume too much bandwidth if we send it over the SMR channel. Thus, SPURT only sends the cryptographic digest of the aggregated message via SMR. Other pieces of data will be sent over pair-wise private channels. However, there are two challenges in this approach. First, in existing SMR protocols, some nodes may learn a decision much earlier than others. This will violate the unpredictability requirement of a randomness beacon. We thus augment existing SMR protocols with a Bracha-style amplification step [19] in §V-C to achieve nearly simultaneous decisions, i.e., nodes learn a decision within a small constant number of rounds. Second, a malicious leader may not send private messages to all nodes. In that case, some nodes may decide on the hash digest in the SMR protocol but do not have the full aggregated data to reconstruct the beacon output. We address this with a novel *two round* reconstruction phase. During the first round, every node that receives valid private data from the leader reconstructs the beacon output. These nodes then help all remaining nodes obtain the beacon output using one more round of communication.

III. PRELIMINARIES

This section describes the notations and tools we will use in SPURT. Let λ be the security parameter. Let $\mathbb{G}_0, \mathbb{G}_1$ and $\mathbb{G}_T$ be cyclic groups of prime order q and $\mathbb{Z}_q$ the group of integer modulo q. We denote an element x sampled uniformly from a finite set $\mathcal{M}$ by $x \leftarrow \mathcal{M}$. We denote vectors using boldface lowercase letters such as $\mathbf{x}$. We summarize the notations used in the paper in Table II and describe them in detail in the rest of this section.

Table II: Notations used in the paper

Notation	Description
$\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T$	Bilinear pairing gropus
g_0, h_0	Random generators in $\mathbb{G}_0$
g_1, h_1	Random generators in $\mathbb{G}_1$
λ	Security parameter
n	Total number of nodes
t	Maximum number of malicious nodes
pk_i, sk_i	Public and secret keys of i^{th} node.
r, L_r	epoch number, and the leader of epoch r
ht	height number
s_i	Secret chosen by i^{th} node
$p_i(\cdot)$	Polynomial chosen by i^{th} node to share s_i
$s_{i,j}$	$p_i(j)$, i.e., $p_i(\cdot)$ evaluated at j
$v_{i,j}$	Commitment of $s_{i,j}$ computed as $g_1^{s_{i,j}}$
$c_{i,j}$	Encryption of $s_{i,j}$ under pk_j computed as $pk_i^{s_{i,j}}$
$\mathsf{dleq}(\cdot)$	DDH based NIZK proof for equality of discrete logarithm
$C^\perp$	Dual of error correcting code C

A. Bilinear Pairings

SPURT and our new PVSS scheme Π_{DBDH} makes use of pairing. In particular, security of Π_{DBDH} relies on the the decisional version of the *Bilinear Diffie-Hellman* assumption (ref. Definition 7 in Appendix D).

Definition 2 (Bilinear Pairing)**.** Let $\mathbb{G}_0, \mathbb{G}_1$ and $\mathbb{G}_T$ be three cyclic groups of prime order q where $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$ are generators. A pairing is an efficiently computable function $e : \mathbb{G}_0 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ satisfying the following properties.

1) bilinear: For all $u, u' \in \mathbb{G}_0$ and $v, v' \in \mathbb{G}_1$ we have

$$e(u \cdot u', v) = e(u, v) \cdot e(u', v), \text{ and} \tag{2}$$

$$e(u, v \cdot v') = e(u, v) \cdot e(u, v') \tag{3}$$

2) non-degenerate: $g_T := e(g_0, g_1)$ is a generator of $\mathbb{G}_T$.

We refer to $\mathbb{G}_0$ and $\mathbb{G}_1$ as the pairing groups or source groups, and refer to $\mathbb{G}_T$ as the target group.

B. Zero knowledge Proof of Equality of Discrete Logarithm

SPURT and our new PVSS scheme Π_{DBDH} have steps that require nodes to produce zero-knowledge proofs about equality of discrete logarithms for a tuple of publicly known values. In particular, given groups $\mathbb{G}_0$ and $\mathbb{G}_1$ of prime order q, random generators $g_0 \in \mathbb{G}_0$ and $g_1 \leftarrow \mathbb{G}_1$ and a tuple (g_0, x, g_1, y), where $x \in \mathbb{G}_0$ and $y \in \mathbb{G}_1$, a prover $\mathcal{P}$ wants to prove to a verifier $\mathcal{V}$ in zero-knowledge, that there exists a witness α such that $x = g_0^\alpha$ and $y = g_1^\alpha$. Moreover, SPURT also requires *knowledge soundness*, i.e., the prover knows α.

We use two different protocols (for reasons to be described later) for the equality of discrete logarithm. The first protocol is the classic Chaum-Pedersen Σ-protcotol [28] in the random oracle model. For a given tuple (g_0, x, g_1, y), the Chaum-Pedersen protocol proceeds as follows.

1) $\mathcal{P}$ samples a random element $\beta \leftarrow \mathbb{Z}_q$ and sends (a_0, a_1) to $\mathcal{V}$ where $a_0 = g_0^\beta$ and $a_1 = g_1^\beta$.
2) $\mathcal{V}$ sends a challenge $e \leftarrow \mathbb{Z}_q$.

3) $\mathcal{P}$ sends a response $z = \beta - \alpha e$ to $\mathcal{V}$.
4) $\mathcal{V}$ checks whether $a_0 = g_0^z x^e$ and $a_1 = g_1^z y^e$ and accepts if and only both equations hold.

The Chaum-Pedersen Σ-protocol can be made non-interactive in the random oracle model using the Fiat-Shamir heuristic [36], [63]. This protocol guarantees completeness, knowledge soundness, and zero-knowledge. The knowledge soundness implies that if $\mathcal{P}$ convinces the $\mathcal{V}$ with non-negligible probability, there exists an efficient (polynomial time) extractor that can extract α from the prover with non-negligible probability. Throughout this paper, we will use the non-interactive variant of the Σ-protocol and denote it using $\mathsf{dleq}(\cdot)$. In particular, for any given tuple (g_0, x, g_1, y) where $x = g_0^\alpha$ and $y = g_1^\alpha$, the procedure $\mathsf{dleq.Prove}(\alpha, g_0, x, g_1, y)$ generates the proof π. Given the proof π and (g_0, x, g_1, y), $\mathsf{dleq.Verify}(\pi, g_0, x, g_1, y)$ verifies the proof.

The second equality of discrete logarithm protocol uses bilinear pairings in a straightforward way, and it does not require any interaction or additional proof. Given a tuple (g_0, x, g_1, y), the verifier can check whether $x = g^\alpha$ and $y = g_1^\alpha$ for some witness α, using the following equality check:

$$e(g_0, y) = e(x, g_1) \tag{4}$$

In the case of an honest prover equation (4) will hold because

$$e(g_0, y) = e(g_0, g_1^\alpha) = e(g_0, g_1)^\alpha = e(g_0^\alpha, g_1) = e(x, g_1)$$

C. State Machine Replication

A State Machine Replication is a distributed protocol run by a network of n nodes to decide on a sequence of values, one for each height. It provides the following properties.

- **Agreement/Safety.** If an honest node decides some value M in height ht, then for height ht, no honest node decides on a value M' such that $M' \neq M$ for height ht.
- **Validity/Liveness.** If an honest node proposes a value M, every honest node eventually decides M in some height.
- **Verifiability.** Whenever a node decides on a value, it can prove to other nodes and external parties the correctness of the decided value.

Note that, unlike regular SMR protocols that service clients [27] in our case, only participating nodes propose values, and all decided values must meet a certain external valid predicate $P(\cdot)$.

SPURT uses a variant of the HotStuff [75] protocol. HotStuff is an *epoch* based protocol, where each epoch has a leader, who proposes a value M to be output in the corresponding epoch. In every epoch, the HotStuff protocol finalizes the value in four steps: *Propose, Prepare, Pre-Commit*, and *Commit*. We present a simplified description of HotStuff in Figure 1. Specifically, we modify the protocol so that nodes multicast messages to others instead of sending them to the leader. We refer the reader to [75] for more details.

Let r be the current epoch and L be its leader. Also, let $\mathsf{ht} - 1$ be the latest finalized height.

Propose. L proposes a value M to be finalized at height ht by sending $\langle \texttt{PROPOSE}, M, r, \mathsf{ht}, X \rangle$ message to all the nodes. Here X is the view change certificate (if any) that validates that the proposal is safe.

Prepare. Each node j, upon receiving the proposal checks whether the proposal is consistent with HotStuff specifications using X, and $P(M)$ is true for an external predicate $P(\cdot)$. If both checks pass, node j sends $\langle \texttt{PREPARE}, M, r, \mathsf{ht} \rangle$ to all nodes.

Pre-Commit. Upon receiving $2t + 1$ PREPARE messages for the proposal M at height ht and epoch r, node j sends $\langle \texttt{PRECOMMIT}, M, r, \mathsf{ht} \rangle$ message to every node.

Commit. Upon receiving $2t+1$ PRECOMMIT messages for the proposal M at height ht and epoch r, node j sends $\langle \texttt{COMMIT}, M, r, \mathsf{ht} \rangle$ message to every node.

Each node outputs M upon receiving $2t + 1$ COMMIT messages corresponding to M.

Figure 1: Steady state of a modified HotStuff [75] protocol that does not use threshold signatures and has a communication cost of $O(|M|n^2)$ per decision.

IV. PVSS Scheme for Uniform Secrets

In this section, we will describe our PVSS scheme Π_{DBDH}. Π_{DBDH} builds upon the PVSS scheme from Scrape [25], which relies on a less standard Decisional Bilinear Squaring assumption [48]. Our new Π_{DBDH} scheme only relies on the much more standard Decisional Bilinear Diffie-Hellman (DBDH) assumption and does not require a Random Oracle. Due to space restrictions, we will directly describe the protocol and refer the readers to Appendix D for formal definition and desired security properties of PVSS.

Our PVSS scheme allows a node (dealer) to share a uniformly random (*uniform* for short) secret $s \in \mathbb{Z}_q$ among n nodes, such that any subset of at least $t + 1$ nodes can reconstruct $e(h_0^s, h_1)$ where $h_0 \in \mathbb{G}_0$ and $h_1 \in \mathbb{G}_1$ are uniformly random independent elements from the respective groups. The reconstruction threshold $t + 1$ ensures that an adversary controlling t nodes cannot recover $e(h_0^s, h_1)$ without contribution of at least one honest node. A key property of PVSS is that not only the participating nodes but any third party (with access to participating node's public keys) can verify, even before the reconstruction phase begins that the dealer has generated the shares correctly without having plaintext access to the shares. This property will be crucial to SPURT.

Π_{DBDH} has four procedures: $\mathsf{PVSS.Setup}$, $\mathsf{PVSS.Share}$, $\mathsf{PVSS.Verify}$, and $\mathsf{PVSS.Reconstruct}$. The $\mathsf{PVSS.Setup}$ procedure takes the security parameter λ as the input and generates four independent generators g_0, h_0, g_1, h_1 where $g_0, h_0 \in \mathbb{G}_0$ and $g_1, h_1 \in \mathbb{G}_1$. Here $\mathbb{G}_0$ and $\mathbb{G}_1$ are two pairing groups of order q. Note that the tuple (g_0, h_0, g_1, h_1) needs to be generated only once and can be reused across different execution of the protocol. During the setup step, each node i also samples their secret key $sk_i \in \mathbb{Z}_q$ and publishes their public key $pk_i = h_0^{sk_i}$.

$\mathsf{PVSS.Setup}(1^\lambda) \rightarrow (g_0, h_0, g_1, h_1, \{(sk_i, pk_i)\})$:
The setup algorithm chooses uniform random and independent generators $g_0, h_0 \in \mathbb{G}_0$ and $g_1, h_1 \in \mathbb{G}_1$ and publishes it in a public ledger. Each node i, then generates a secret key $sk_i \in \mathbb{Z}_q$, a public key $pk_i = h_0^{sk_i}$, and registers the public key pk_i by posting it to the public ledger, for $1 \leq i \leq n$.

During the sharing step, the dealer L, samples $s \in \mathbb{Z}_q$ and let $S = e(h_0^s, h_1)$ be the secret the dealer with public-private key pair (sk, pk) wants to share with set of nodes with public keys $\{pk_j\}_j$ for $j = 1, 2, \ldots, n$.

$\mathsf{PVSS.Share}(s, g_1, sk, \{pk\}_{j,j=1,2,\ldots,n}) \rightarrow (\mathbf{v}, \mathbf{c})$:

1) Sample uniform random $a_k \in \mathbb{Z}$ for $k = 1, 2, \ldots, t-1$ and let
$$p(x) = s + a_1 x + \ldots + a_t x^t;$$
2) Compute $s_j \leftarrow p(j)$; $v_j \leftarrow g_1^{s_j}$; $c_j \leftarrow pk_j^{s_j}$, $\forall j \in [n]$.
3) Multi-cast to all nodes $\mathbf{v} = \{v_1, v_2, \ldots, v_n\}$ and $\mathbf{c} = \{c_1, c_2, \ldots, c_n\}$ using a broadcast channel.

Upon receiving $(\mathbf{v}, \mathbf{c})$ from the dealer, each node validates them as follows.

$\mathsf{PVSS.Verify}(g_1, \mathbf{v}, \mathbf{c}, \{pk\}_{j,j=1,2,\ldots,n}) \rightarrow 0/1$:

1) Sample a random code word $\mathbf{y}^\perp \in C^\perp$ where $\mathbf{y}^\perp = [y_1^\perp, y_1^\perp, \ldots, y_n^\perp]$ and check whether
$$\prod_{k=1}^{n} v_k^{y_k^\perp} = 1_{\mathbb{G}_1} \tag{5}$$
where $1_{\mathbb{G}_1}$ is the identity element of $\mathbb{G}_1$.
2) Check whether $e(pk_j, v_j) = e(c_j, g_1)$ for all j.
3) Output 1 if both checks pass, otherwise output 0.

During the reconstruction step, each node j decrypts its share c_j to compute $\tilde{s}_j \leftarrow c_j^{1/sk_j}$, and multi-casts $\tilde{s}_j$ to all nodes. A node i upon receiving $\tilde{s}_j$ from node j checks if $e(h_0, v_j) = e(\tilde{s}_j, g_1)$. Let H be the set of indices of $t+1$ valid decrypted shares $\tilde{s}_j$.

$\mathsf{PVSS.Reconstruct}(h_0, h_1, \{\tilde{s}_k\}_{k \in H}) \rightarrow e(h_0^s, h_1)$:

1) Use Lagrange interpolation to compute
$$\prod_{k \in H} (\tilde{s}_k)^{\mu_k} = \prod_{k \in H} h_0^{\mu_k \cdot p(k)} = h_0^{p(0)} \tag{6}$$
where $\mu_k = \prod_{j \neq k} \frac{j}{j-k}$ are Lagrange coefficients.
2) Output $e(h_0^s, h_1)$.

Figure 2: Description of Π_{DBDH}.

After the setup step, the dealer uses PVSS.Share to share a secret s, other nodes or external users use PVSS.Verify to validate the shares, and PVSS.Reconstruct is used to recover $e(h_0^s, h_1)$. We describe them in detail in Figure 2.

The verification procedure of Π_{DBDH} uses properties of error-correcting code, specifically the Reed-Solomon code [65]. In particular, we use the observation by McEliece and Sarwate [58] that sharing of a secret x using a degree t polynomial among n nodes is equivalent to encoding the message $(x, a_1, a_2, \cdots, a_t)$ using a $[n, t+1, n-t]$ Reed-Solomon code. Let C be a $[n, k, d]$ linear error correcting code over $\mathbb{Z}_q$ of length n and minimum distance d. Also, let $C^\perp$ be the dual code of C i.e., $C^\perp$ consists of vectors $\mathbf{y}^\perp \in \mathbb{Z}_q^n$ such that for all $\mathbf{x} \in C$, $\langle \mathbf{x}, \mathbf{y}^\perp \rangle = 0$. Here, $\langle \cdot, \cdot \rangle$ is the inner product operation. The PVSS.Verify step uses the following basic fact (Lemma 1) of linear error correcting code. We refer readers to [25, Lemma 1] for its proof, and Appendix B for a brief description of the Reed Solomon and its dual code.

Lemma 1. *If $\mathbf{x} \in \mathbb{Z}_q^n \setminus C$, and $\mathbf{y}^\perp$ is chosen uniformly at random from $C^\perp$, then the probability that $\langle \mathbf{x}, \mathbf{y}^\perp \rangle = 1$ is exactly $1/q$.*

We provide more details about the verification step of Π_{DBDH} in Appendix B. Also in appendix D, we will define the required properties for PVSS such as correctness, verifiability, and IND1-Secrecy. We then prove that assuming DBDH hardness, Π_{DBDH} guarantees the desired correctness, verifiability, and IND1-Secrecy properties.

V. Spurt Design and Optimizations

In this section, we present the detailed design of Spurt. Spurt proceeds in epochs and each epoch has four phases. Each epoch has a designated leader chosen in any deterministic manner. For concreteness, we assume leaders are chosen in a round-robin order, i.e., the leader of epoch r is node $i = r \bmod n$. We will use L_r to denote the leader of epoch r. We next describe each phase in detail. Refer to Table II for notations.

A. Commitment Phase

For any given epoch r, let L_r be its leader. Each node i samples a uniformly random secret $s_i \leftarrow \mathbb{Z}_q$ and computes PVSS tuples using the PVSS.Share primitive described in §IV:
$$\mathbf{v}_i, \mathbf{c}_i \leftarrow \mathsf{PVSS.Share}(s_i, g_1, sk_i, \{pk\}_{j,j=1,2,\ldots,n}) \tag{7}$$
where $\mathbf{v}_i = \{v_{i,1}, \ldots, v_{i,n}\}$ and $\mathbf{c}_i = \{c_{i,1}, \ldots, c_{i,n}\}$. Node i also computes $\boldsymbol{\pi}_i = \{\pi_{i,1}, \ldots, \pi_{i,n}\}$ where
$$\pi_{i,j} = \mathsf{dleq.Prove}(g_1, v_j, pk_j, c_j, s_{i,j})$$
where $s_{i,j}$ is the share of secret s_i for node j. Node i then sends $(\mathbf{v}_i, \mathbf{c}_i, \boldsymbol{\pi}_i)$ to L_r.

Note that, here we use the DDH-based dleq proof due to its *knowledge soundness* property. This ensures that the secrets the adversary samples are independent of the secrets chosen by the honest nodes.

B. Aggregation Phase

L_r, upon receiving a tuple $(\mathbf{v}_i, \mathbf{c}_i, \boldsymbol{\pi}_i)$, first validates $\mathbf{v}_i$ and $\mathbf{c}_i$ using $\mathsf{PVSS.Verify}(g_1, \mathbf{v}_i, \mathbf{c}_i, \{pk_j\}_{j=1,2,\ldots,n})$. Then, for each j, L_r checks $\pi_{i,j}$ using dleq.Verify. We remark that since the leader anyway checks the equality of discrete

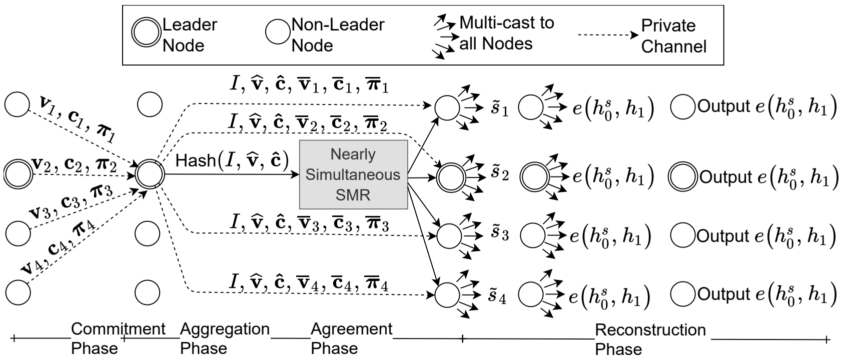

Figure 3: Messages sent during each phase of the SPURT. We describe contents of the messages i.e., the notations over the arrows in §V.

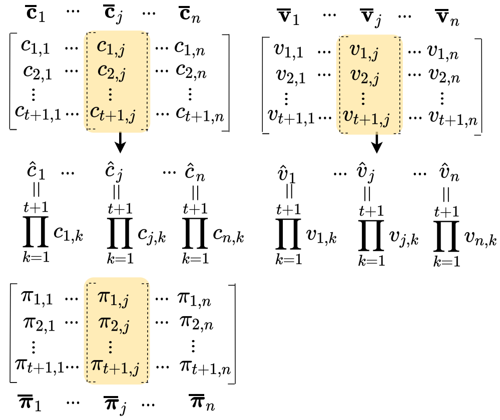

Figure 4: Aggregation phase at the leader.

logarithm using dleq.Verify, the leader need not perform step 2) of PVSS.Verify as this check is redundant with dleq.Verify.

Upon receiving $t+1$ such valid tuples, L_r aggregates them as follows. Let $I \subseteq [n]$ be the set of nodes that send valid messages during the commitment phase. L_r aggregates the commitments into $\hat{\mathbf{v}} = (\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_n)$, a commitment to the aggregated polynomial $\hat{p}(\cdot) = \sum_{i \in I} p_i(\cdot)$. For any ℓ, L_r computes $\hat{v}_\ell$ where

$$\hat{v}_\ell = \prod_{i \in I} v_{i,\ell} \tag{8}$$

L_r also aggregates the encrypted shares into $\hat{\mathbf{c}} = (\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_n)$, which would be encrypted shares for the aggregated secret $\hat{p}(0)$.

$$\hat{c}_\ell = \prod_{i \in I} c_{i,\ell} \tag{9}$$

Figure 4 illustrates this step using $I = \{1, 2, \ldots, t+1\}$ as an example. Observe that the $t+1$ messages received and validated by during L_r can be represented as three matrices shown in Figure 4. Here on, we refer to these matrices as the commitment matrix $\{v_{i,j}\}$, the ciphertext matrix $\{c_{i,j}\}$, and the proof matrix $\{\pi_{i,j}\}$. Let $\bar{\mathbf{c}}_j, \bar{\mathbf{v}}_j$ and $\bar{\boldsymbol{\pi}}_j$ be the j^{th} column of the ciphertext, commitment, and proof matrix respectively. Stated differently, $\bar{\mathbf{c}}_j$ is the set of encryptions sent by nodes in I that are encrypted under the public key of node j. $\bar{\mathbf{v}}_j$ and $\bar{\boldsymbol{\pi}}_j$ are j^{th} coordinate of commitments and dleq proofs sent by nodes in I, respectively. Without loss of generality, let $I = \{1, 2, \ldots, t+1\}$, then $\bar{\mathbf{c}}_j = \{c_{1,j}, c_{2,j}, \ldots, c_{t+1,j}\}$, $\bar{\mathbf{v}}_j = \{v_{1,j}, v_{2,j}, \ldots, v_{t+1,j}\}$, and $\bar{\boldsymbol{\pi}}_j = \{\pi_{1,j}, \pi_{2,j}, \ldots, \pi_{t+1,j}\}$. As shown in Figure 4, the aggregation result $\hat{c}_j$ is the product of all elements in $\bar{\mathbf{c}}_j$ and $\hat{v}_j$ is the product of all elements in $\bar{\mathbf{v}}_j$.

Next, L_r computes a cryptographic digest of $I \| \hat{\mathbf{v}} \| \hat{\mathbf{c}}$ where $\|$ denotes concatenation. We will use `digest` to denote this cryptographic digest. As illustrated in Figure 3, in the agreement phase, only the digest is sent via SMR. $I, \hat{\mathbf{v}}, \hat{\mathbf{c}}$ themselves and the original PVSS tuples are sent privately to corresponding nodes.

C. Agreement Phase

Let ht be the height in SMR chosen by L_r. Then, to each node j, L_r sends $(\texttt{digest}, \hat{\mathbf{v}}, \hat{\mathbf{c}}, I, \bar{\mathbf{v}}_j, \bar{\mathbf{c}}_j, \bar{\boldsymbol{\pi}}_j, \mathsf{ht})$ and proposes `digest` using the SMR protocol for height ht. Observe that in the above message, only $\bar{\mathbf{v}}_j$, $\bar{\mathbf{c}}_j$, and $\bar{\boldsymbol{\pi}}_j$ are recipient specific and everything else is common to all nodes. Essentially, the tuple L_r sends to each node is the full message corresponding to the digest `digest` for epoch r and height ht.

Upon receiving $(\texttt{digest}, \hat{\mathbf{v}}, \hat{\mathbf{c}}, I, \bar{\mathbf{v}}_j, \bar{\mathbf{c}}_j, \bar{\boldsymbol{\pi}}_j, \mathsf{ht})$ from L_r, node j validates them by checking:

1) The proposal is safe according to SMR,
2) `digest` is the cryptographic digest of $I \| \hat{\mathbf{v}} \| \hat{\mathbf{c}}$,
3) Let $\mathbf{y}^\perp = \{y_1^\perp, y_2^\perp, \ldots, y_n^\perp\}$ be a randomly chosen code word from the dual code $C^\perp$, then check

$$\prod_{k=1}^{n} \hat{v}_k^{y_k^\perp} = 1_{\mathbb{G}_2}. \tag{10}$$

   This check ensures that $\hat{\mathbf{v}}$ is a commitment to a polynomial of degree at most t.
4) Every tuple $(v_{i,j}, c_{i,j}, \pi_{i,j}) \in (\bar{\mathbf{v}}_j, \bar{\mathbf{c}}_j, \bar{\boldsymbol{\pi}}_j)$ is a valid dleq proof according to §III-B.
5) $\hat{c}_j = \prod_{i \in I} c_{i,j}$ and $\hat{v}_j = \prod_{i \in I} v_{i,j}$.

If all of the above checks pass, node j multi-casts $\langle \texttt{PREPARE}, \texttt{digest}, r, \mathsf{ht} \rangle$ to all other nodes. If any of the above checks fails or if j does not receive the required private information from L_r, j does not send the PREPARE message in the SMR protocol (cf. §III-C). An honest node, upon receiving $2t+1$ $\langle \texttt{PREPARE}, \texttt{digest}, r, \mathsf{ht} \rangle$ messages multi-casts $\langle \texttt{PRECOMMIT}, \texttt{digest}, r, \mathsf{ht} \rangle$, upon receiving $2t+1$ $\langle \texttt{PRECOMMIT}, \texttt{digest}, r, \mathsf{ht} \rangle$ messages multi-casts $\langle \texttt{COMMIT}, \texttt{digest}, r, \mathsf{ht} \rangle$. Unlike existing SMR, however, upon receiving $2t+1$ $\langle \texttt{COMMIT}, \texttt{digest}, r, \mathsf{ht} \rangle$, a node does not yet decide. Instead, it multi-casts $\langle \texttt{TERMINATE}, \texttt{digest}, r, \mathsf{ht} \rangle$ message to achieve nearly simultaneous decision, as we discuss below.

Nearly simultaneous decision. As stated above, each node, upon receiving $2t+1$ $\langle \texttt{COMMIT}, \texttt{digest}, r, \mathsf{ht} \rangle$ messages multi-casts $\langle \texttt{TERMINATE}, \texttt{digest}, r, \mathsf{ht} \rangle$. An honest node also multi-casts a $\langle \texttt{TERMINATE}, \texttt{digest}, r, \mathsf{ht} \rangle$ message upon receiving

$t+1$ matching TERMINATE messages. Upon receiving $2t+1$ matching TERMINATE messages, an honest node decides digest.

Multi-casting TERMINATE upon receiving $t+1$ matching TERMINATE messages is the Bracha amplification step [19] and is critical to nearly simultaneous decision. The intuition behind this step is that when any node receives $2t+1$ TERMINATE message, every honest node will receive at least $t+1$ TERMINATE message, and multi-cast TERMINATE message (if they haven't done it already). As a result, every honest node will decide on the proposal by the end of the next round.

D. Reconstruction Phase

Whenever an honest node *decides* on digest, it starts the reconstruction phase for that epoch and multi-casts its decrypted share to all other nodes. In particular, node j sends $\langle \texttt{RECONSTRUCT}, \tilde{s}_j, r, \mathsf{ht}\rangle$ to all other nodes where its decrypted share is computed as

$$\tilde{s}_j = \hat{c}_j^{\frac{1}{sk_j}} = h_0^{\sum_{i\in I} s_{i,j}} \tag{11}$$

Note that due to the *nearly simultaneous decision* property of our underlying SMR all honest nodes will decide within two message delays. Hence, they will start the reconstruction phase within two message delays as well.

Each node i who receives the correct $\mathbf{v}, \mathbf{c}, I$ during the agreement phase, upon receiving a tuple $\tilde{s}_j$, validates it using the pairing-based discrete logarithm equality check (cf.§ III-B) as below.

$$e(\tilde{s}_j, g_1) = e(h_0, \hat{v}_j) \tag{12}$$

We use the pairing-based discrete logarithm equality proof here because it does not require the prover to know the witness. This is crucial because, during the reconstruction phase, each node j can only recover $h_0^{\sum_{i\in I} s_{i,j}}$ and not $\sum_{i\in I} s_{i,j}$.

Let T be the set of nodes from which node i receives valid $\tilde{s}_j$ tuples. Upon receiving $t+1$ such valid tuples, i.e., when $|T| \geq t+1$, node i outputs the beacon for height ht as $e(h_0^s, h_1)$. Recall from §V-A, $s = \hat{p}(0) = \sum_{i\in I} s_i$. Honest nodes construct h_0^s using the Lagrange interpolation:

$$\prod_{k\in T} (\tilde{s}_k)^{\mu_k} = \prod_{k\in T} h_0^{\mu_k \cdot \hat{p}(k)} = h_0^{\hat{p}(0)} \tag{13}$$

where $\mu_k = \prod_{j\neq k} \frac{j}{j-k}$ are the Lagrange coefficients. Then, the beacon output for height ht is given as:

$$o_{\mathsf{ht}} = e(h_0^{\hat{p}(0)}, h_1) = e(h_0, h_1)^{\hat{p}(0)} \tag{14}$$

Each honest node upon reconstructing the beacon output o_{ht} multi-casts $\langle \texttt{BEACON}, o_{\mathsf{ht}}, r, \mathsf{ht}\rangle$ to all other nodes. If an honest node i has not output at height ht yet, it waits for $t+1$ identical $\langle \texttt{BEACON}, o_{\mathsf{ht}}, r, \mathsf{ht}\rangle$ messages and outputs o_{ht} as the beacon output for height ht.

Note that an additional round of communication during the reconstruction phase ensures that honest nodes who miss the original proposal from a malicious leader can recover the beacon output with the help of others. In particular, upon receiving $t+1$ matching $\langle \texttt{BEACON}, o_{\mathsf{ht}}, r, \mathsf{ht}\rangle$ messages, a node can safely output o_{ht} as the beacon output at height ht. This is because at least one of these BEACON messages is sent by an honest node and honest node never sends a BEACON message for an incorrect output.

E. Optimizations

Pre-aggregating data. Recall from §V-B, during the aggregation phase, the leader validates a total of $O(n^2)$ NIZK proofs. Moreover, the leader aggregates polynomial commitments from $t+1$ nodes. As a result, the leader performs $O(n^2)$ computation while other nodes each perform $O(n)$ computation. For a large n, the leader will become a bottleneck.

SPURT addresses this by having leaders pre-compute the messages of aggregation phase. In particular, at any epoch r, every node sends their PVSS shares for epoch $r+\tau$ to $L_{r+\tau}$. Here, τ is a system parameter. Since the leader selection rule in SPURT is deterministic, $L_{r+\tau}$ is fixed and known to all nodes in advance. $L_{r+\tau}$, upon receiving the shares for epoch $r+\tau$, immediately starts aggregating them, and sends the aggregated messages as well as the private messages to each node. By doing so, SPURT amortizes the leader's higher usage of computation and communication across τ epochs. As a result, during epoch $r+\tau$, $L_{r+\tau}$ only sends λ bits of data to every node, incurring a total bandwidth of $O(n\lambda)$ bits (instead of $O(n^2\lambda)$ bits), which is comparable to non-leader nodes.

Multi-exponentiation. We further reduce the computation cost using the multi-exponentiation technique [59]. For any given group $\mathbb{G}$, let $\mathbf{g} = [g_1, g_2, \cdots, g_m]$ be a vector of m elements in $\mathbb{G}$, and let $\mathbf{a} = [a_1, a_2, \cdots, a_m]$ be a vector of m scalars in $\mathbb{Z}_q$. Given $\mathbf{a}$ and $\mathbf{g}$, the multi-exponentiation technique computes more efficiently:

$$g' = \prod_{k=1}^{m} g_k^{a_k} \tag{15}$$

In SPURT, nodes need to compute an expression of this form to: (i) validate the polynomial commitments sent during commitment phase; (ii) validate the aggregated polynomial sent by the leader; and (iii) compute the beacon output from reconstruction shares.

VI. Analysis of SPURT

In this section, we prove that SPURT is unpredictable, bias-resistant, available and publicly verifiable. We also analyze the computation and communication complexity of SPURT.

A. Unpredictability and Bias-Resistance

Lemma 2. *If an honest node decides* digest *in epoch r, then every honest node outputs $e(h_0^a, h_1)$ for some $a \in \mathbb{Z}_q$ at epoch r and $a = \hat{p}(0)$.*

Proof. If an honest node decides digest, there must be $2t+1$ PREPARE, PRECOMMIT, COMMIT, and TERMINATE messages (cf. §V-C). $t+1$ of these must come from honest nodes. From §V-C, an honest node sends a PREPARE message only if it receives from L_r a private message that passes the check in equation (10). Here on, we will refer to honest nodes that sent PREPARE messages as *sender* nodes.

This means, except for negligible probability, the degree of $\hat{p}(\cdot)$ is at most t. This is because any polynomial of degree greater than t passes the check in equation (10) with probability only $1/q$; hence, the probability that it passes the check at $t+1$ honest nodes is merely $\binom{2t+1}{t+1}\frac{1}{q^{t+1}} \leq \frac{1}{q}$, which is negligible.

By the security guarantees of dleq every honest sender node j holds $h_0^{sk_j \cdot \hat{p}(j)}$. It can then compute the decrypted share $h_0^{\hat{p}(j)}$. This implies that during the reconstruction phase, each sender node will multi-cast valid decrypted shares and will be able to validate the decrypted shares they receive.

Say an honest sender i outputs h_0^a for some $a \in \mathbb{Z}_q$ by combining $t+1$ valid decrypted shares. Then, for every decrypted share $\tilde{s}_j = h_0^{a_j}$ for some $a_j \in \mathbb{Z}_q$ that i receives during the reconstruction phase, i accepts $\tilde{s}_j$ only if the discrete log equality check $e(\tilde{s}_j, g_1) = e(h_0, \hat{v}_j)$ is successful. A successful equality check implies $a_j = \hat{p}(j)$ as

$$e(h_0, \hat{v}_j) = e(h_0, g_1)^{\hat{p}(j)} = e(h_0^{\hat{p}(j)}, g_1) \tag{16}$$

Since equation (16) holds for every valid decrypted shares, upon Lagrange interpolation in the exponent using these decrypted shares, node i will compute $h_0^{\hat{p}(0)}$ and recover the beacon output $o_{\mathsf{ht}} = e(h_0^{\hat{p}(0)}, h_1)$.

Upon recovering the beacon output o_{ht}, each sender will multi-cast $\langle \texttt{BEACON}, o_{\mathsf{ht}}, r, \mathsf{ht} \rangle$ to every node. Since there are at least $t+1$ of them, every node will receive at least $t+1$ identical BEACON messages with o_{ht} in it. Furthermore, since honest nodes only sends BEACON message for correct beacon output, only o_{ht} will occur more that t times. ☐

We capture the unpredictability and bias-resistance property of SPURT based on the Indistinguishability game we define below. Briefly, the Indistinguishability game captures the fact that before any honest node starts a reconstruction phase of an epoch, the beacon output remains indistinguishable from a random element from a large set. This holds even after honest nodes decide on the cryptographic digest digest, which finalizes the beacon output for that epoch. Note that Indistinguishability is not sufficient to guarantee unpredictability and bias-resistance. We also need to argue that honest nodes learn the beacon output within a small constant delay from the time an adversary learns the beacon output.

Definition 3. (Indistinguishability) SPURT guarantees Indistinguishability if for any PPT adversary $\mathcal{A}$ corrupting at most t parties, in each epoch of SPURT, $\mathcal{A}$ has a negligible advantage in the following game played against a challenger $\mathcal{C}$.

1) The challenger $\mathcal{C}$ generates public parameters of SPURT and sends all public information to $\mathcal{A}$.
2) $\mathcal{A}$ selects a set of at most t nodes and corrupts them. $\mathcal{A}$ send the identities of the corrupt nodes to $\mathcal{C}$.
3) $\mathcal{C}$ creates secret and public keys for all honest nodes and sends the corresponding public keys to $\mathcal{A}$.
4) $\mathcal{A}$ sends the public keys of the corrupt nodes to the challenger. Note that these public keys of corrupt nodes do not have to be legimate public keys (they do not need to have correponding private keys) and they can be chosen after seeing the public keys of honest nodes.
5) $\mathcal{C}$ and $\mathcal{A}$ starts executing an epoch of SPURT as follows:
   - $\mathcal{C}$ chooses PVSS inputs of honest nodes, computes corresponding commitment phase messages and sends them to $\mathcal{A}$.
   - The $\mathcal{A}$ aggregates the PVSS messages and runs agreement phase with $\mathcal{C}$ where $\mathcal{A}$ can observe messages sent between every pair of nodes during the agreement phase and reorder them.
   - The challenger waits till at least one honest node decides on a digest.
6) $\mathcal{C}$ samples a bit $b \in \{0,1\}$ and sends either the beacon output corresponding to digest or a uniformly random element from the target group $\mathbb{G}_T$.
7) $\mathcal{A}$ makes a guess b'.

The advantage of $\mathcal{A}$ is defined as $|\Pr[b = b'] - 1/2|$.

Next, we state the Indistinguishability theorem and prove it in Appendix C. The proof will assume a *static* adversary. As of now we are unable to prove SPURT secure against an adaptive adversary. But we do not know of any concrete attack either.

Theorem 1 (Indistinguishability)**.** *Assuming hardness of Decisional Bilinear Diffie–Hellman,* SPURT *guarantees Indistinguishability in the programmable random oracle model.*

Next, we prove that SPURT guarantees *nearly simultaneous output*. To do so, we will first prove that our modified SMR protocol guarantees that all honest nodes decide within two message delays.

Lemma 3 (Nearly Simultaneous SMR)**.** *When an honest node decides on a* digest*, every honest node decides on* digest *decide within two message delays.*

Proof. Recall an honest node decides on digest upon receiving $2t+1$ TERMINATE message on digest. At least $t+1$ of these are sent by honest nodes. Hence, every honest node will receive at least $t+1$ TERMINATE message during the same round. As a result, all honest nodes will send TERMINATE message for digest in the next round. Thus, all honest node will receive at least $2t+1$ TERMINATE message for digest at the end of the next round and decide on digest. ☐

Lemma 4 (Nearly Simultaneous Beacon Output)**.** *For any given height* ht*, every honest node learns the output* o_{ht} *at most four message delay later from when the adversary learns the output.*

Proof. Using Theorem 1 we know that till an honest node starts reconstruction, the beacon output remains indistinguishable from a uniformly random element in the target group $\mathbb{G}_T$. Since honest nodes start reconstruction only upon SMR decision and all honest nodes decide within two message delays, every honest node will start reconstruction within two message delays. Thus, a subset of at least $t+1$ honest nodes will recover the beacon output at most three message delays later than $\mathcal{A}$, and all honest nodes will recover the beacon output at most four message delays later than $\mathcal{A}$. □

Theorem 2 (Bias-resistance and Unpredictability)**.** *Assuming hardness of DBDH,* SPURT *is unpredictable and bias-resistant.*

Proof. Follows from Lemma 2, Theorem 1, and Lemma 4. □

B. Availability and Public Verifiability

Theorem 3. (Availability) *During periods of synchrony, if the leader L_r of an epoch r is honest, every honest node will produce a beacon output.*

Proof. When L_r is honest, all the checks described in §V-C would be successful at every honest node. During periods of synchrony, due to the liveness property of SMR honest nodes will decide on the value proposed by L_r. Furthermore, as the leader is honest, every honest will have the data needed to validate the decrypted shares sent during the reconstruction phase. Thus, each honest node will successfully output a beacon value during that epoch. □

Public verifiability for beacon protocols producing true random numbers differs from beacon protocols producing pseudorandom numbers [2], [21], [23], [72]. In pseudorandom beacons, each beacon output is some deterministic function of the secret key generated during the initial setup phase. Hence, the output can be efficiently verified given only the verification/public key corresponding to the secret key used for beacon generation. Contrary to this, truly random beacons such as Scrape [25], Hydrand [68], and SPURT have to verified using the transcript of the interaction between nodes.

To verify the validity of o_{ht}, a client (not one of the nodes) simply needs to obtain $t+1$ `BEACON` message signed by distinct nodes, which we call a beacon certificate. The client can query a single node for such a certificate. Note that in SPURT every honest node will be able to produce a beacon certificate as it receives at least $t+1$ identical `BEACON` message.

We also mention that there are other ways to achieve public-verifiability. For example, the SMR decision certificate on `digest` along with aggregated message is an alternative way.

C. Performance

In this section, we analyze the communication cost of each epoch and the amortized cost of generating every beacon output. We will report the communication complexity in the number of bits of nodes need to send in every epoch. We then analyze each node's computation cost measured in the number of exponentiations and pairings each node needs to perform every epoch and report the amortized computation cost of each beacon output. Throughout this section, we assume that signatures are $O(\lambda)$ bits long.. Also, we assume that a node needs to perform $O(1)$ exponentiations to compute and validate a single signature. We summarize our performance analysis in Table III.

Table III: Summary of communication and computation cost of each epoch of SPURT. — indicate the no cost for the corresponding phase. In the latency column, the $(+k)$ denotes that in the worst case k additional rounds might be required for that phase to end.

Protocol Phase	Communication		Computation		Latency (# rounds)
	Leader	Non-leader	Leader	Non-leader	
Commitment	$O(\lambda n^2)$	$O(\lambda n)$	—	$O(n)$	1
Aggregation	—	—	$O(n^2)$	—	0
Agreement	$O(\lambda n^2)$	$O(\lambda n)$	—	$O(n)$	5(+1)
Reconstruction	—	$O(\lambda n^2)$	—	$O(n)$	1(+1)

Communication cost. During the commitment phase of an epoch r, each node sends $O(n)$ group elements to L_r. Thus, the commitment phase's total communication cost is $O(\lambda n^2)$. Next, during the agreement phase L_r sends back $O(n)$ group elements to every node. During the agreement phase, every node multi-casts the hash of the aggregated message. Hence, the agreement phase's total communication complexity is $O(\lambda n^2)$. Finally, during the reconstruction phase, each node multi-casts $O(1)$ group elements to all other nodes, hence incurs a cost of $O(\lambda n^2)$. Also observe that during periods of synchrony, for every n epochs, there will be at least $\lceil 2n/3 \rceil$ honest leaders. From Theorem 3, for every honest leader, SPURT will produce an output. Hence, in every sequence of n epochs, SPURT will produce at least $\lceil 2n/3 \rceil$ beacon outputs. This implies that the *amortized* communication complexity of each beacon output is $O(\lambda n^2)$.

Computation cost. During the commitment phase of an epoch r, each node performs $O(n)$ exponentiation to evaluate $\mathsf{PVSS.Share}$ for their chosen secret, and to sign the PVSS shares. In the aggregation phase, only L_r verifies the PVSS shares from all nodes. Since, verification of PVSS shares each node requires $O(n)$ exponentiations, L_r performs $O(n^2)$ exponentiations to verify all the PVSS shares. Computing the aggregated commitment and aggregated encryption requires $O(n^2)$ multiplications of group elements. Lastly, L_r hashes $O(n)$ group elements to compute the digest of the aggregated commitment and the aggregated ciphertexts. Overall, during the aggregation phase, L_r performs $O(n^2)$ exponentiations while the remaining nodes do not perform any computation.

During the agreement phase, each node performs $O(n)$ exponentiations to validate commitments, and the aggregated polynomial. Furthermore, as a part of the SMR step, each node performs $O(n)$ exponentiations to validate signatures on messages sent by other nodes. Finally, in the reconstruction phase, every node verifies decrypted shares using $O(n)$ pairings. Hence, the computation cost per node in both agreement and reconstruction phase is $O(n)$ exponentiations and pairings.

In summary, in every epoch, the leader of the epoch performs $O(n^2)$ exponentiations whereas every other node performs $O(n)$ exponentiations and $O(n)$ pairings. However, in a sequence of n epochs, each node becomes the leader only once, and due to pre-aggregation optimization (cf. §V-E), the leader gets $\tau = \Theta(n)$ rounds to compute $O(n^2)$ exponentiations. As a result, during periods of synchrony, the *amortized* computation cost of each beacon output is $O(n)$ exponentiations and pairings per node.

Public verification. Recall from §VI-B, to validate a beacon output, external clients need to download a beacon certificate. The beacon certificate consists of $O(n)$ `BEACON` messages and associated signatures. Hence, the communication cost of verifying a beacon output is $O(\lambda n)$. Verifying $O(n)$ signatures have a computation cost of $O(n)$. So the computation complexity of public verification is $O(n)$.

Latency. During periods of synchrony, when an honest node is chosen as the leader of an epoch, SPURT produces a beacon output. Thus, in the fault-free case, in practice, SPURT would only require seven message delays. However, there might be a sequence of t malicious leaders in the worst case, and all of them may decide to abort their epochs. In such cases, SPURT will take $O(t)$ message delay to produce the next output. Nevertheless, in a sequence of n epochs, SPURT will produce at least $2n/3$ beacon outputs, so the amortized latency is at most 1.5 epochs. Furthermore, SPURT is *responsive* [62], i.e., it can produce beacon outputs at the actual speed of the network, as opposed to any pre-determined conservative parameters. This is another advantage of a partially synchronous protocol over a synchronous one, besides the main advantage of being more robust to long network delays.

VII. Implementation & Evaluation

We have implemented a prototype of SPURT using the go programming language version `1.13.0`. Our implementation builds atop the open-source Quorum client version `2.4.0`. Quorum is a fork of Ethereum go client which we modify to implement the HotStuff SMR protocol. We disable the artificial delay between consecutive proposals and modify the underlying implementation such that the next leader proposes as soon as the previous beacon output is finalized.

Throughout our implementation, we have used the `bls12-381` elliptic curve as our pairing curve. In particular, we have used the implementation of `bls12-381` by `gnark-crypto` [1] for primitive elliptic curve operations. When transmitting elliptic curve group elements we use the standard point compression technique [51]. After point compression, an element of $\mathbb{G}_0$ and $\mathbb{G}_1$ is 48 bytes and 96 bytes, respectively. For multi-exponentiations, we have used the native implementation of [1] which implements the multi-exponentiation algorithm from [11, §4].

A. Experimental Setup

We evaluate our implementation of SPURT with varying nodes, i.e., 16, 32, 64, and 128. We run all nodes on Amazon Web Services (AWS) *t3a.medium* virtual machine (VM) with one node per VM. All VMs have two vCPUs and 4GB RAM. The operating system for each VM is Ubuntu 20.04.

Network. To simulate an execution over the internet, we pick eight different AWS regions, namely, Canada, Ireland, N. California, N. Virginia, Oregon, Ohio, Singapore, and Tokyo. For any choice of total number of nodes, we distribute the nodes evenly across all eight regions. We create an overlay network among nodes where all nodes are pair-wise connected, i.e., they form a complete graph.

Baselines. We compare our implementation with two state of the art publicly available implmentations: Hydrand [3] and Drand [2]. Note that Hydrand has imperfect predictability an Drand requires a DKG setup. Nevertheless, we chose Hydrand as it is most closely related to SPURT in terms of cryptographic and setup assumptions and Drand as it has been deployed.

B. Evaluation Results

All our evaluation results are averaged over three runs for each value of number of nodes.

Bandwidth usage. We report the bandwidth usage measured as the amount of bytes sent and received per node per beacon output in Figure 5. Recall from §VI that at every epoch, each node sends and receives a total of $O(\lambda n)$ bits of information to and from other nodes. Hence, with an increase in the number of nodes, we observe an approximately linear increase in the bandwidth usage per node per beacon output. For example, from 32 to 64 nodes, the average bandwidth usage per node per beacon output increases from 34 to 65 Kilobytes. This is about 55% of the bandwidth cost of Hydrand. For Drand, we expected a bandwidth cost of $96n$ Kilobytes as each node multi-casts one and receives n partial signatures which are 48 Bytes long each. However, in the Drand implementation, we observed a bandwidth cost of $2 \times 96n$ Kilobytes. Upon close inspection, we found that each node in Drand also multi-casts the previous beacon output, which doubles the bandwidth usage. Hence, for 32 and 64 nodes Drand has a communication cost of 6.2 and 12.3 Kilobytes, respectively. Although SPURT has $5\times$ higher bandwidth usage than Drand, we believe that this is a reasonable trade-off for removing DKG and handling partial synchrony.

Throughput. We report the throughput of SPURT as the number of beacon output generated per minute in Figure 6. Our evaluation results illustrate that with 16, 32, 64, and 128 nodes, SPURT on average can generate 145, 83, 42, and 12 beacon output per minute.

We will try to compare with Drand and Hydrand but there is a subtlety here. Since Drand and Hydrand assume synchronous networks, their throughput is directly decided by a hard-coded parameter, the estimated network delay upper bound. A higher estimate hurt throughput but is a safer choice since synchronous protocols lose security when the delay bound is violated. Thus, we first look for a smallest network delay parameter that do not break their implementations and then measure throughput with that delay parameter. For Hydrand, the throughput we found in our experiment is much lower than what was reported in [3], so we simply use Hydrand's reported throughput in Figure 6 in their favor. For Drand, the actual deployed Drand sets its throughput to be two beacon values per minute (one per 30 seconds). We are able to adopt much more aggressive parameters, making Drand's throughput

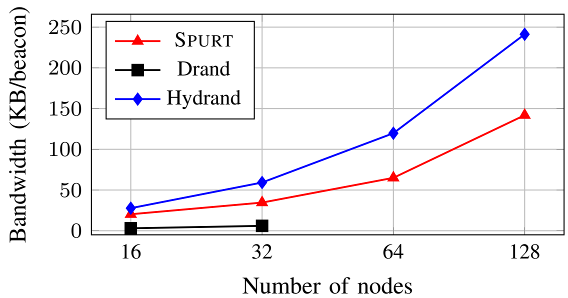

Figure 5: Average bandwidth usage (sent + received data) measured in Kilobytes per beacon output with varying number of nodes.

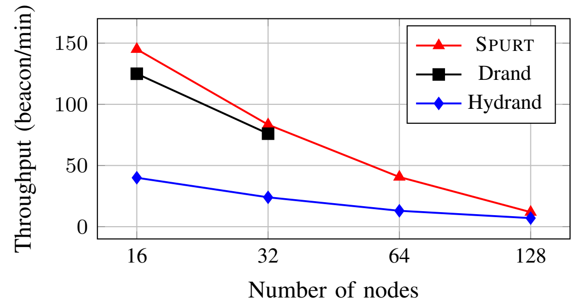

Figure 6: Average number of random beacon generated per minute with varying number of nodes.

	Time taken (in milliseconds)			
Protocol Phase	$n = 16$	$n = 32$	$n = 64$	$n = 128$
Commitment	9.72	19.53	39.00	78.89
Aggregation	108.08	371.5	1447.07	5728.5
Agreement	120.77	240.22	479.74	957.66
Reconstruction	188.91	359.39	702.38	1392.21

Table IV: Time taken (in milliseconds) to compute different cryptographic functions required in the different phases of SPURT.

in Figure 6 much higher than its deployed version, again in their favor.

Interestingly, even after favoring the baselines, SPURT achieves significantly better throughput than Hydrand despite having only slightly better bandwidth; furthermore, despite having higher communication cost than Drand, SPURT slightly outperforms Drand in terms of throughput. We believe this is in part because SPURT is partially synchronous, i.e., it does not require any network delay parameter and hence can make progress at the speed of true network delay. In contrast, Hydrand and Drand assume synchronous networks and need to run at the speed of a conservatively chosen network delay estimate. Of course, there may be other inefficiencies in the implementations of Drand and Hydrand that hindered their throughput. Also we could only report the throughput of Drand for up to 32 nodes, as in our experiments, the DKG step in Drand keeps aborting for 64 or more nodes, even when we choose very large estimates for the network delay.

Computation cost. Table IV presents the concrete time required for each of the four phases. Except for the aggregation phase, the computation times for the other three phases scale linearly with the number of nodes. The aggregation phase requires the leader to perform a quadratic amount of computation. But since we pipeline the aggregation phase by sending the commitments to the leader in advance (cf. §V-E), the aggregation phase is not the bottleneck in the critical path.

VIII. Related Work

Based on the setup assumption, existing distributed protocols can be classified into two categories; protocols with *trusted* setup and protocols with *transparent* setup. Protocols with trusted setup involve generation of public parameters that embed a secret trapdoor. These parameters can either be generated by a *trusted* third party (hence the name trusted setup) or by running a maliciously secure multi-party computation protocol, often a Distributed Key Generation (DKG) step. Note that protocols without a trusted setup assumption may also require a step to generate some uniformly random public parameters. The subtle difference is that these public parameters do not contain secrets or trapdoors (hence the name transparent setup) and it is hence a milder assumption.

Protocols in the trusted setup category include [2], [13], [21], [23], [29], [38], [72]. Most of them follow the paradigm of Cachin et al. where the random beacon's output at any given epoch is a unique threshold signature on the hash of the epoch number. Cachin et al. and Aleph can tolerate asynchronous network, Dfinity [23] tolerates partial asynchrony, while Drand assumes a synchronous network. Cachin et al., does not discuss details about the setup phase. Drand uses the DKG protocol of Gennaro et al. [39], which requires a Byzantine broadcast channel over which each node sends $O(\lambda n)$ bits to information. Hence, the overall communication complexity of Drand's setup phase is at least $O(\lambda n^3 \log n)$. Dfinity uses a non interactive distributed key generation protocol instead with at least an $O(\lambda n^3)$ communication cost [45]. After the DKG step, the communication complexity for generating one beacon output for all four protocols is $O(n^2\lambda)$. Aleph [38] presents their own asynchronous DKG protocol with a total communication complexity of $O(\lambda n^4 \log n)$.

Protocols in the transparent setup category include [8], [25], [32], [41], [53], [60], [68]. Most relevant to our work are Scrape [25] and Hydrand [68]. Both Scrape and Hydrand assume that the underlying network is synchronous. Scrape [25] improves the computation complexity of PVSS protocol of [69] from $O(n^2)$ exponentiation to $O(n)$ exponentiation per PVSS, and uses their PVSS over a broadcast channel to generate distributed randomness. In particular, for every beacon output in Scrape, each node uses the broadcast channel to share their secret. Once $t + 1$ nodes share their secret, nodes reconstruct the secrets using the reconstruction phase of the PVSS and combine them to produce the beacon output. In Scrape, each node uses the broadcast channel to share $O(n\lambda)$ size message per beacon output. Thus, the total communication cost per beacon output is at least $O(n^4\lambda)$. Also, for every beacon output, each node requires to perform

$O(n^2)$ exponentiations.

Hydrand [68] modifies the Scrape protocol to remove the broadcast channel. Unlike Scrape, in each epoch of Hydrand, only a leader node shares a secret using Scrape's PVSS scheme. Hydrand has an initialization step where each node shares a secret using PVSS scheme, which costs $O(\lambda n^3 \log n)$ communication. After the setup phase, for every beacon, Hydrand has a total communication and computation cost of $O(\lambda n^2)$ and $O(n)$, respectively. One major disadvantage of Hydrand is that it only provides imperfect unpredictability, i.e., at any epoch, an adversary can predict beacon output for up to t future epochs.

A concurrent work by Bhat et al. [13] presents GRandPiper, which improves upon Hydrand to have one-half fault tolerance, but requires a trusted setup to generate q-SDH parameters. To fix the unpredictability issue of Hydrand and GRandPiper, Bhat et al [13] further presents BRandPiper, where the leader shares n secrets in a single epoch and nodes reconstruct a random value accumulating secrets from $t+1$ nodes. Similar to GRandPiper, BRandPiper also requires a trusted setup to generate q-SDH parameters, and has a worst-case communication complexity of $O(\lambda n^3)$.

Randherd [72] uses Randhound in a one-time setup to partition nodes into smaller subgroups of size c, and additionally setup keys for threshold signatures. The total complexity of Randherd is $O(\lambda c^2 \log\ n)$. Randherd, as presented, is not bias-resistant as a malicious leader can abort the protocol after observing the beacon output, and will require additional mechanisms to make it bias-resistant. The committee sampling technique is orthogonal to random beacon designs and can be applied to most random beacon protocols. It is effective in improving scalability when there are a very large number of nodes at the cost of slightly reducing fault tolerance.

In addition to the above mentioned protocols, other beacon protocols include Bitcoin's Proof-of-Work (PoW) [60], Proof-of-Delay [20], Algorand [41], Ouroboros [53], Ouroboros Praos [32], etc. The Ouroboros [53] protocol requires every node to perform PVSS over a broadcast channel, and hence has high communication complexity. Bitcoin, Algorand and Ouroboros Praos are not bias-resistant as a malicious adversary can decide to discard undesirable beacon outputs (even though they are still secure as blockchain protocols). Protocol based on Proof-of-Delay rely on strong and new assumptions about verifiable time-lock puzzles [10], [26], [67] or Verifiable Delay Functions [16].

PVSS schemes without Random Oracle. PVSS schemes in the plain model, i.e., without random oracle, were first proposed in [66] and later improved in [25], [48]–[50]. These schemes either rely on non-standard assumption or have high computation cost. For example, the schemes due to Ruiz and Villar [66] and Jhanwar et al. [50] are based on the hardness of Decisional Composite Residuosity assumption [61] and the verifier in these schemes need to perform $O(n^2)$ exponentiations. The schemes from [48] and [25] rely on the Decisional Bilinear Square Assumption and require $2n$ pairing operations for each verifier. Jhanwar [49] reduces the number of pairing operation needed during verification to 4 using the even less standard (n, t)-multi-sequence of exponents Diffie-Hellman assumption [49]. Our new PVSS scheme relies on the standard Decisional Bilinear Diffie-Hellman assumption and achieves similar performance as Scrape, which assumes the less standard hardness of Decisional Bilinear Squaring problem. Both in our PVSS scheme and Scrape's PVSS scheme, a verifier needs to perform n exponentiations and $2n$ pairings to validate shares for all the nodes.

Concurrently and independently, Gurkan et al. [46] proposes a modification to the PVSS scheme of Scrape and uses it to design a DKG protocol. The PVSS scheme of [46] assumes hardness of the Symmetric External Diffie-Hellman (SXDH) problem in Type-III pairing groups. The DKG protocol of [46] requires each node to broadcast $\log n$ messages of size $O(n)$ each, and n messages of $O(1)$ size each. Hence, it is not immediately suitable for improving SPURT.

IX. Conclusion and Future Directions

We have presented SPURT, an efficient distributed randomness beacon protocol with transparent setup, i.e., trapdoor-free public parameters. SPURT guarantees that each beacon output is unpredictable, bias-resistant and publicly verifiable, and provides these properties in a partially synchronous network against a malicious adversary controlling up to one third of the total nodes. SPURT has amortized total communication of $O(\lambda n^2)$. Computation wise, each node performs $O(n)$ group exponentiations per beacon output. While designing SPURT, we modify existing State Machine Replication (SMR) protocol to ensure nearly simultaneous decision at all nodes. We also design a publicly-verifiable secret sharing (PVSS) scheme whose security relies on the standard Decisional bilinear Diffie-Hellman assumption and does not require a Random oracle.

An interesting question for future work is whether it is possible to design a randomness beacon protocol with optimal fault tolerance and *sub-quadratic* communication complexity (possibly with a trusted setup). Note that protocols that sample subsets can be easily made sub-quadratic in the trusted setup phase. But such protocols come with reduced fault tolerance. It is interesting to study whether we can design a sub-quadratic protocol that does not resort to subset sampling. On the flip side, it would also be very interesting to show study communication lower bound for randomness beacon. Similar lower bounds for Byzantine agreement or multiparty computation may be good starting points towards that direction. One may also try to extend SPURT to fully asynchronous networks. The major hurdle we encounter is that consensus (SMR) protocols in fully asynchronous network require shared randomness [37], which creates a circularity.

Acknowledgment

The authors would like to thank Amit Agarwal, Adithya Bhat, Aniket Kate, Jong Chan Lee, Kartik Nayak, Nibesh

Shrestha, Zhuolun Xiang, and Tom Yurek for helpful discussions related to the paper.

REFERENCES

[1] "bls12381," 2020. [Online]. Available: https://github.com/ConsenSys/gnark-crypto/tree/master/ecc/bls12-381

[2] "Drand - a distributed randomness beacon daemon," 2020, https://github.com/drand/drand.

[3] "Hydrand," 2020. [Online]. Available: https://github.com/PhilippSchindler/HydRand

[4] "Proof of stake (pos)," 2020. [Online]. Available: https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/proof-of-stake/

[5] I. Abraham, T. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi, "Communication complexity of byzantine agreement, revisited," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 317–326.

[6] B. Adida, "Helios: Web-based open-audit voting." in *USENIX security symposium*, vol. 17, 2008, pp. 335–348.

[7] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," in *Proceedings of the 25th Annual Network and Distributed System Security Symposium*, 2017.

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

[9] T. Baigneres, C. Delerablée, M. Finiasz, L. Goubin, T. Lepoint, and M. Rivain, "Trap me if you can-million dollar curve." *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 1249, 2015.

[10] C. Baum, B. David, R. Dowsley, J. B. Nielsen, and S. Oechsner, "Craft: Composable randomness and almost fairness from time," 2020, https://eprint.iacr.org/2020/784.

[11] D. J. Bernstein, J. Doumen, T. Lange, and J.-J. Oosterwijk, "Faster batch forgery identification," in *International Conference on Cryptology in India*. Springer, 2012, pp. 454–473.

[12] D. J. Bernstein, T. Lange, and R. Niederhagen, "Dual ec: A standardized back door," in *The New Codebreakers*. Springer, 2016, pp. 256–281.

[13] A. Bhat, N. Shrestha, A. Kate, and K. Nayak, "Randpiper–reconfiguration-friendly random beacons with quadratic communication," in *(To appear) Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[14] G. R. Blakley, "Safeguarding cryptographic keys," in *1979 International Workshop on Managing Requirements Knowledge (MARK)*. IEEE, 1979, pp. 313–318.

[15] M. Blum, "Coin flipping by telephone a protocol for solving impossible problems," *ACM SIGACT News*, vol. 15, no. 1, pp. 23–27, 1983.

[16] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Annual international cryptology conference*. Springer, 2018, pp. 757–788.

[17] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *International conference on the theory and application of cryptology and information security*. Springer, 2001, pp. 514–532.

[18] J. Bonneau, J. Clark, and S. Goldfeder, "On bitcoin as a public randomness source." *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 1015, 2015.

[19] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

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

[21] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[22] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*. IEEE, 2005, pp. 191–201.

[23] J. Camenisch, M. Drijvers, T. Hanke, Y.-A. Pignolet, V. Shoup, and D. Williams, "Internet computer consensus," *https://eprint.iacr.org/2021/632*, 2021.

[24] R. Canetti and T. Rabin, "Fast asynchronous byzantine agreement with optimal resilience," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, pp. 42–51.

[25] I. Cascudo and B. David, "Scrape: Scalable randomness attested by public entities," in *International Conference on Applied Cryptography and Network Security*. Springer, 2017, pp. 537–556.

[26] ——, "Albatross: publicly attestable batched randomness based on secret sharing," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 311–341.

[27] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, 1999, p. 173–186.

[28] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Annual International Cryptology Conference*. Springer, 1992, pp. 89–105.

[29] A. Cherniaeva, I. Shirobokov, and O. Shlomovits, "Homomorphic encryption random beacon." 2019.

[30] R. Cohen and Y. Lindell, "Fairness versus guaranteed output delivery in secure multiparty computation," *Journal of Cryptology*, vol. 30, no. 4, pp. 1157–1186, 2017.

[31] S. Das, V. J. Ribeiro, and A. Anand, "Yoda: Enabling computationally intensive contracts on blockchains with byzantine and selfish nodes," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, 2019.

[32] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 66–98.

[33] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Naval Research Lab Washington DC, Tech. Rep., 2004.

[34] D. Dolev and R. Reischuk, "Bounds on information exchange for byzantine agreement," *Journal of the ACM (JACM)*, vol. 32, no. 1, pp. 191–204, 1985.

[35] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[36] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 186–194.

[37] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[38] A. Gągol, D. Leśniak, D. Straszak, and M. Świętek, "Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 214–228.

[39] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *Journal of Cryptology*, vol. 20, no. 1, pp. 51–83, 2007.

[40] M. Ghosh, M. Richardson, B. Ford, and R. Jansen, "A torpath to torcoin: Proof-of-bandwidth altcoins for compensating relays," NAVAL RESEARCH LAB WASHINGTON DC, Tech. Rep., 2014.

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

[42] S. Goel, M. Robson, M. Polte, and E. Sirer, "Herbivore: A scalable and efficient protocol for anonymous communication," Cornell University, Tech. Rep., 2003.

[43] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game, or a completeness theorem for protocols with honest majority," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 1987, pp. 218–229.

[44] D. Goulet and G. Kadianakis, "Random number generation during tor voting," *Tor's protocol specifications-Proposal*, vol. 250, 2015.

[45] J. Groth, "Non-interactive distributed key generation and key resharing." *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 339, 2021.

[46] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu, "Aggregatable distributed key generation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 147–176.

[47] M. Haahr, "Random. org: True random number service," *School of Computer Science and Statistics, Trinity College, Dublin, Ireland. Website (http://www. random. org). Accessed*, vol. 10, 2010.

[48] S. Heidarvand and J. L. Villar, "Public verifiability from pairings in secret sharing schemes," in *International Workshop on Selected Areas in Cryptography*. Springer, 2008, pp. 294–308.

[49] M. P. Jhanwar, "A practical (non-interactive) publicly verifiable secret sharing scheme," in *International Conference on Information Security Practice and Experience*. Springer, 2011, pp. 273–287.

[50] M. P. Jhanwar, A. Venkateswarlu, and R. Safavi-Naini, "Paillier-based publicly verifiable (non-interactive) secret sharing," *Designs, codes and Cryptography*, vol. 73, no. 2, pp. 529–546, 2014.

[51] A. Jivsov, "Compact representation of an elliptic curve point," *Internet Engineering Task Force*, 2014.

[52] J. Kelsey, L. T. Brandão, R. Peralta, and H. Booth, "A reference for randomness beacons: Format and protocol version 2," National Institute of Standards and Technology, Tech. Rep., 2019.

[53] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[54] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598.

[55] A. K. Lenstra and B. Wesolowski, "A random zoo: sloth, unicorn, and trx." 2015.

[56] B. Libert and D. Vergnaud, "Unidirectional chosen-ciphertext secure proxy re-encryption," in *International Workshop on Public Key Cryptography*. Springer, 2008, pp. 360–379.

[57] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 17–30.

[58] R. J. McEliece and D. V. Sarwate, "On sharing secrets and reed-solomon codes," *Communications of the ACM*, vol. 24, no. 9, pp. 583–584, 1981.

[59] B. Möller, "Algorithms for multi-exponentiation," in *International Workshop on Selected Areas in Cryptography*. Springer, 2001, pp. 165–180.

[60] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[61] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[62] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[63] D. Pointcheval and J. Stern, "Security proofs for signature schemes," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1996, pp. 387–398.

[64] 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*, 1989, pp. 73–85.

[65] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[66] A. Ruiz and J. L. Villar, "Publicly verifiable secret sharing from paillier's cryptosystem," in *WEWoRC 2005–Western European Workshop on Research in Cryptology*. Gesellschaft für Informatik eV, 2005.

[67] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. Weippl, "Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness," in *Proceedings of the 29th Annual Network and Distributed System Security Symposium*, 2021.

[68] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "Hydrand: Practical continuous distributed randomness," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2020.

[69] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic voting," in *Annual International Cryptology Conference*. Springer, 1999, pp. 148–164.

[70] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[71] N. Starr, "Nonrandom risk: The 1970 draft lottery," *Journal of Statistics Education*, vol. 5, no. 2, 1997.

[72] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy (SP)*. Ieee, 2017, pp. 444–460.

[73] 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*, 2015, pp. 137–152.

[74] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, "Dissent in numbers: Making strong anonymity scale," in *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, 2012, pp. 179–182.

[75] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, 2019, pp. 347–356.

[76] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *ACM Conference on Computer and Communications Security*, 2018, pp. 931–948.

Appendix A
Threshold Secret Sharing

A $(n, t+1)$ threshold secret sharing scheme allows a secret $s \in \mathbb{Z}_q$ to be shared among n nodes such that any $t+1$ of them can come together to reconstruct the original secret, but any subset of t shares cannot be used to reconstruct the original secret [14], [70]. We use the common Shamir secret sharing [70] scheme, where the secret is embedded in a random degree t polynomial in the field $\mathbb{Z}_q$ for some prime q. Specifically, to share a secret $s \in \mathbb{Z}_q$, a polynomial $p(\cdot)$ of degree t is chosen such that $s = p(0)$. The remaining coefficients of $p(\cdot)$, $a_1, a_2, \cdots, a_t$ are chosen uniformly randomly from $\mathbb{Z}_q$. The resulting polynomial $p(x)$ is defined as:

$$p(x) = s + a_1 x + a_2 x^2 + \cdots + a_t x^t$$

Each node is then given a single evaluation of $p(\cdot)$. In particular, the i^{th} node is given $p(i)$ i.e., the polynomial evaluated at i. Observe that given $t+1$ points on the polynomial $p(\cdot)$, one can efficiently reconstruct the polynomial using Lagrange Interpolation. Also note that when s is uniformly random in $\mathbb{Z}_q$, s is information theoretically hidden from an adversary that knows any subset of t or less evaluation points on the polynomial other than $p(0)$ [70].

Appendix B
Linear Error Correcting Code

Let C be a $[n, k, d]$ linear error correcting code over $\mathbb{Z}_q$ of length n and minimum distance d. Also, let $C^\perp$ be the dual code of C i.e., $C^\perp$ consists of vectors $\mathbf{y}^\perp \in \mathbb{Z}_q^n$ such that for all $\mathbf{x} \in C$, $\langle \mathbf{x}, \mathbf{y}^\perp \rangle = 0$. Here, $\langle \cdot, \cdot \rangle$ is the inner product operation. Our PVSS scheme uses the basic fact from coding theory. Refer to Lemma 1 of [25] for its proof.

Lemma 5. *If $\mathbf{x} \in \mathbb{Z}_q^n \setminus C$, and $\mathbf{y}^\perp$ is chosen uniformly at random from $C^\perp$, then the probability that $\langle \mathbf{x}, \mathbf{y}^\perp \rangle = 1$ is exactly $1/q$.*

Throughout this paper, we will use C to be the $[n, k, n-k+1]$ Reed-Solomon Code of the form

$$C = \{p(1), p(2), ..., p(n) : p(x) \in \mathbb{Z}_q[x]; \text{and} \\ \mathsf{deg}(p(\cdot)) \leq k-1\}$$

where $\mathsf{deg}(p(\cdot))$ is the degree of the polynomial $p(\cdot)$. Thus its $[n, n-k, k+1]$ dual code $C^\perp$ can be written as

$$C^\perp = \{(\mu_1 f(1), \mu_2 f(2), ..., \mu_n f(n); f(x) \in \mathbb{Z}_q[x]; \text{and} \\ \mathsf{deg}(f(\cdot)) \leq n-k+1\}$$

where the coefficients $\mu_i = \prod_{i=1, i\neq j}^{n} \frac{1}{i-j}$. This implies that random elements from $C^{\perp}$ of interest is efficiently samplable.

APPENDIX C
PROOF OF THERHOREM 1

We will prove Theorem 1 by showing if a PPT adversary $\mathcal{A}$ wins the Indistinguishability game with non-negligible advantage, then we can can use $\mathcal{A}$ to break the DBDH assumption.

Before we describe our reduction, we will define some properties and prove some basic lemmas.

Definition 4 (Parallel Knowledge Sound)**.** Let Π be a interactive proof protocol for a relation R such that Π is knowledge sound. Then, for any given $m \in \mathbb{N}$ with $m > 1$, let Π^m be the protocol where the prover and verifier interact in m parallel instances of Π. Then, we call Π^m *parallel knowledge sound* if there exists an efficient extractor $\mathcal{E}$ such that $\mathcal{E}$ can extract witness from each instance of Π^m.

Definition 5 (Indistinguishable Parallel Knowledge Sound)**.** A parallel knowledge sound protocol Π^m for $m > 1 \in \mathbb{N}$ is *indistinguishable* parallel knowledge sound if the view of the prover in Π^m is indistinguishable from its view when it interacts with the extractor $\mathcal{E}$.

Lemma 6. *Assuming existence of a Random Oracle, the Chaum-Pedersen protocol (cf. III-B) is indistinguishable parallel knowledge sound.*

Proof. We will prove this by constructing an explicit extractor $\mathcal{E}$. Let $c_1, c_2, \ldots, c_m$ be commitments generated by a PPT adversary $\mathcal{A}$. Let $e_1, e_2, \ldots, e_m$ the outputs of the random oracle, i.e., $e_i = \mathsf{hash}(c_i)$. Let z_i be the response of $\mathcal{A}$ on the challenge e_i. Then, $\mathcal{E}$ rewinds $\mathcal{A}$ to a point just before receiving the challenges. Then, using programmability of the random oracle, $\mathcal{E}$ changes the challenges to $e'_1, e'_2, \ldots, e'_m$. Let z'_i be the response of $\mathcal{A}$ on challenge e'_i. Now, using the tuple (e_i, e'_i, z_i, z'_i) $\mathcal{E}$ recovers each witness α_i as.

$$\alpha_i = \frac{z'_i - z_i}{e_i - e'_i} \tag{17}$$

It is easy to see that if the soundness error of one instance of Chaum-Pedersen protocol is ε, then by union bound the soundness error of m parallel instance of the Chaum-Pedersen protocol is at most $m\varepsilon$. Hence, Chaum-Pedersen protocol is parallel knowledge sound. Also, since the e'_i are uniform random, the view of $\mathcal{A}$ in the real-protocol and during its interaction with $\mathcal{E}$ is identical. Hence, Chaum-Pedersen protocol is indistinguishable parallel knowledge sound. □

We will next argue that for every SMR decision on a cryptographic digest, the corresponding leader (even if it malicious) must include PVSS messages from at least one honest node in the aggregated PVSS message.

Lemma 7. *For any epoch r, if honest nodes decide on* `digest` *and $\hat{p}(\cdot)$ be the underlying polynomial, then there exists an honest node i such that*

$$\hat{p}(x) = p_i(x) + q(x)$$

where $p_i(\cdot)$ is the polynomial chosen by node i and $q(\cdot)$ is a polynomial of degree t.

Proof. When an honest node decides `digest`, by the proof of Lemma 2 and the collision resistance property of the hash function, at least $t+1$ honest nodes validate that contributions from at least $t+1$ nodes are included in coordinates of the commitment vector $\hat{\mathbf{v}} = (\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_n)$. This implies contribution from at least one honest node is included in at least $t+1$ elements of $\hat{\mathbf{v}}$.

Without loss of generality, let i be the honest node whose contribution is included in $t+1$ elements of $\hat{\mathbf{v}}$ and $p_i(\cdot)$ be the polynomial chosen by node i. By construction $\mathsf{deg}(p_i(\cdot)) = t$. Thus the $t+1$ evaluation points validated by the $t+1$ honest nodes fix the polynomial $p_i(\cdot)$ and hence all other evaluation points of $p_i(\cdot)$. Due to knowledge soundness of dleq protocol, except for negligible probability, the values $\hat{p}(x) - p_i(x)$ at all these $t+1$ indices are independent of $p_i(x)$. Since, degree of $\hat{p}(\cdot)$ is at most t, evaluation of $p_i(\cdot)$ must be included at all other remaining elements $\hat{\mathbf{v}}$ as well. Hence, $\hat{p}(x) = p_i(x) + q(x)$. This immediately implies $\mathsf{deg}(q(\cdot)) \leq t$. □

Proof of Theorem 1. We show that, if there exists a PPT adversary $\mathcal{A}$ that distinguishes between the beacon output $o_{\mathtt{ht}}$ and a random element u, then there exists an adversary $\mathcal{A}_{\mathrm{DBDH}}$ that can use $\mathcal{A}$ to break the DBDH with a similar (polynomially related) advantage.

Let $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$ be the generators of the groups. Let $(g_0^{\alpha}, g_1^{\alpha}, g_0^{\beta}, g_1^{\gamma}, z)$ be an instance of the DBDH problem. If $\alpha = 0$ or $\beta = 0$ or $\gamma = 0$, then the problem is trivial, so we assume these values are non-zero. Now $\mathcal{A}_{\mathrm{DBDH}}$, upon given the DBDH instance, interacts with $\mathcal{A}$ to simulate SPURT for a epoch as follows. Without loss of generality $\mathcal{A}$ corrupts the first t nodes. Let $T = \{t+1, t+2, \ldots, n\}$.

1) $\mathcal{A}_{\mathrm{DBDH}}$ sets $h_0 = g_0^{\beta}, h_1 = g_1^{\gamma}$ and sends them to $\mathcal{A}$. For $t < i \leq n$, $\mathcal{A}_{\mathrm{DBDH}}$ selects uniformly random values $u_i \leftarrow \mathbb{Z}_p$ (these can be thought to implicitly define $sk_i = u_i/\beta$) and sends $pk_i = g_0^{u_i}$ to $\mathcal{A}$.
2) For $1 \leq i \leq t$, $\mathcal{A}$ sends the public keys pk_i to the $\mathcal{A}_{\mathrm{DBDH}}$.
3) $\mathcal{A}_{\mathrm{DBDH}}$ samples an index $a \leftarrow T$ and uses the DBDH challenge as the PVSS input of node a for that epoch. Let T_{-a} denote the set $T \backslash \{a\}$. For nodes T_{-a}, $\mathcal{A}_{\mathrm{DBDH}}$ samples random secrets (i.e., $x_j \leftarrow \mathbb{Z}_q$ for node j) and uses them as their PVSS inputs.
4) For nodes in T_{-a}, $\mathcal{A}_{\mathrm{DBDH}}$ computes the PVSS messages as per the honest protocol. For node a, $\mathcal{A}_{\mathrm{DBDH}}$ computes the PVSS message as follows:
   - For $1 \leq i \leq t$, $\mathcal{A}_{\mathrm{DBDH}}$ chooses uniformly random values $s_i \in \mathbb{Z}_q$ and set $v_i = g_1^{s_i}, w_i = g_0^{s_i}$ and $c_i = pk_i^{s_i}$.
   - For $t < i \leq n$, it generates values $v_i = g_1^{p(i)}$ and $w_i = g_0^{p(i)}$ where $p(x)$ is the unique polynomial of degree at most t determined by $p(0) = \alpha$ and $p(i) = s_i$ for $i = 1, ..., t$. Note that $\mathcal{A}_{\mathrm{DBDH}}$ does not know α, but it does know g_1^{α}, g_0^{α}, $g_1^{s_i}$, and $g_0^{s_i}$ for $1 \leq i \leq t$, so it can use the Lagrange interpolation in the exponent to compute the adequate v_i and w_i.

- For $t < i \leq n$, it creates the values $c_i = w_i^{u_i}$. Note that then $c_i = g_0^{u_i \cdot p(i)} = pk_i^{p(i)}$.

5) $\mathcal{A}_{\mathrm{DBDH}}$ sends all these information to $\mathcal{A}$.
6) Let I be the set of indices $\mathcal{A}$ chooses to aggregate such that the aggregation verification for every nodes in I is successful at $t+1$ honest nodes.
7) If $a \notin I$, $\mathcal{A}_{\mathrm{DBDH}}$ outputs 1 and aborts. Otherwise, $\mathcal{A}_{\mathrm{DBDH}}$ extracts the secrets for all nodes in $I \setminus T_{-a}$ using the Chaum-Pedersen parallel extractor $\mathcal{E}$ and Lagrange interpolation.
8) Let s be the sum of secrets of all nodes in I_{-a}. Send $z \cdot e(h_0, h_1)^s$ to $\mathcal{A}$.
9) Output whatever $\mathcal{A}$ outputs.

Note that the information $\mathcal{A}$ receives in step 5) is distributed exactly like a epoch in SPURT. Also, from Lemma 7, I must contain contribution from at least one honest node. If $a \in I$, which happens with probability at least $1/n$, the beacon output becomes $e(h_0, h_1)^{\beta+s}$ which is equal to $z \cdot e(h_0, h_1)^s$ if $z = e(h_0, h_1)^{\beta}$. ☐

APPENDIX D
PVSS: DEFITIONS AND SECURITY

We adopt the general model for PVSS from [69], and the security definitions from [56], [66]. Given a set of n nodes, a dealer L seeks to share a randomly chosen secret s among the nodes using an $(n, t+1)$ threshold access-structure. Informally, the property we seek from PVSS is that any subset of t or fewer shares do not reveal any information about the secret s but any subset of $t+1$ or more shares recover the secret s. Additionally, any external verifier $\mathcal{V}$ should be able to check that the dealer L acted honestly without learning any information about the shares or the secret, hence the name *publicly verifiable*.

As mentioned in §IV, a PVSS protocol has four steps: Setup, Share, Verify, and Reconstruct.

– *Setup:* The setup algorithm generates and publishes the parameters of the scheme. Every node i publishes a public key pk_i and keeps the corresponding secret key sk_i private.
– *Share:* The dealer L creates shares $s_1, ..., s_n$ for a randomly chosen secret s. It then encrypts each share s_i with the public key pk_i of node i to obtain c_i. It then publishes these c_i's along with proofs π_i's that these are indeed encryptions of valid shares of some secret.
– *Verify:* In this step, any external $\mathcal{V}$ (not necessarily a participant in the protocol) can verify non-interactively that c_i are encryptions of valid shares of some secret.
– *Reconstruct:* In this step, node i decrypts c_i using its secret key sk_i to get $\tilde{s}_i$ and publishes s_i together with a (non-interactive) zero-knowledge proof $\tilde{\pi}_i$ that $\tilde{s}_i$ is indeed a correct decryption of c_i. An external verifier $\mathcal{V}$ validates the decrypted shares. If there are at least $t+1$ valid decrypted shares, $\mathcal{V}$ applies a reconstruction procedure to recover the original secret s shared by the dealer.

A PVSS scheme must provide the following three security guarantees: *Correctness, Verifiability, and IND1-Secrecy*.

– *Correctness:* If the dealer is honest, then all verification checks in all steps pass and the secret can be reconstructed.
– *Verifiability:* If the checks in the verification step pass, then except for negligible probability, the values c_i are encryptions of a valid shares of some secret. If the check in the reconstruction step passes, then the communicated values s_i are the shares created by the dealer.
– *IND1-Secret:* Prior to the reconstruction step, the published information together with the secret keys of any t nodes gives no information about the secret. This can be formalized by the following indistinguishability definition adapted from [25], [56], [66].

Definition 6. (IND1-Secret) A $(n, t+1)$ PVSS is said to be IND1-secret if for any probabilistic polynomial time adversary $\mathcal{A}$ corrupting at most t parties, if $\mathcal{A}$ has negligible advantage in the following game played against an challenger.

1) The challenger runs the Setup step of the PVSS as the dealer and sends all public information to $\mathcal{A}$. Moreover, it creates secret and public keys for all honest nodes, and sends the corresponding public keys to $\mathcal{A}$.
2) $\mathcal{A}$ sends the public keys of the corrupted nodes to the challenger.
3) The challenger chooses values s_0 and s_1 at random in the space of secrets. It then chooses $b \leftarrow \{0, 1\}$ uniformly at random and runs the Sharing step of the protocol with s_0 as secret. It sends $\mathcal{A}$ all public information generated in the Sharing step, together with s_b.
4) $\mathcal{A}$ makes a guess b'.

The advantage of $\mathcal{A}$ is defined as $|\Pr[b = b'] - 1/2|$.

The correctness of Π_{DBDH} follows trivially from the properties of bilinear pairing and the fact that every code word u in a code C is orthogonal to all code words in $C^{\perp}$. The following theorem ensures that Π_{DBDH} guarantees verifiability. Recall from §IV, q is the order of the groups in our PVSS scheme.

Theorem 4 (Verifiability)**.** *If the checks in the verification step is successful, then except for probability $1/q$ the c_i are correct encryptions of shares of each node. Furthermore, during the reconstruction step, honest nodes only accept s_i that are correct decryption of c_i.*

Proof. From Lemma 1, except with probability $1/q$ the polynomial committed by the dealer is a degree t polynomial. Furthermore, since $e(c_i, g_1) = e(pk_i, v_i)$ holds for every $i \in \{1, 2 \ldots, n\}$, this implies $\log_{g_1} v_i = \log_{pk_i} c_i$ for each i. Otherwise, if $a = \log_{g_1} v_i \neq \log_{pk_i} c_i = b$ for some i, then $e(c_i, g_1) = e(pk_i, g_1)^b \neq e(pk_i, g_1^a)^a = e(pk_i, v_1)$ and the check would fail.

Furthermore, during the reconstruction step, if an honest node accepts s_i that is not the correct decryption of c_i, then the verification would fail with probability 1, because when $\tilde{s}_i = h_0^a$ for some $a \neq s_i$, then $e(h_0, v_i) = e(h_0, g_1)^{s_i} \neq e(\tilde{s}_i, g_1) = e(h_0^a, g_1)$. ☐

The IND1-Secret property of Π_{DBDH} assumes hardness of the Decisional Bilinear Diffie-Hellman (DBDH) assumption, which is given below.

Definition 7 (Decisional Bilinear Diffie-Hellman (DBDH))**.** Given pairing groups $\mathbb{G}_0, \mathbb{G}_1$, target group $\mathbb{G}_T$, each of size q, let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ be the efficient bilinear pairing map. For generators $g_0 \in \mathbb{G}_0$, $g_1 \in \mathbb{G}_1$, random values $\alpha, \beta, \gamma, \delta \leftarrow \mathbb{Z}_q$ and $u_0 \leftarrow g_0^{\alpha}, u_1 \leftarrow g_1^{\alpha}, v_0 \leftarrow g_0^{\beta}, w_1 \leftarrow g_1^{\gamma}$, the following distributions D_0 and D_1 are computationally indistinguishable

$$D_0 = \big(u_0, u_1, v_0, w_1, e(g_0, g_1)^{\alpha\beta\gamma}\big)$$
$$D_1 = \big(u_0, u_1, v_0, w_1, e(g_0, g_1)^{\delta}\big)$$

Theorem 5. *Under the Decisional Bilinear Diffie-Hellman assumption, the protocol* Π_{DBDH} *is IND1-secret against a static PPT adversary.*

Proof. We show that, if there exists an adversary $\mathcal{A}_{\mathrm{priv}}$ that can break the IND1-secrecy of the protocol Π_{DBDH} then there exists an adversary $\mathcal{A}_{\mathrm{DBDH}}$ which can use $\mathcal{A}_{\mathrm{priv}}$ to break Bilinear Decisional Diffie-Hellman assumption with the same advantage. Without loss of generality $\mathcal{A}_{\mathrm{priv}}$ corrupts the first t nodes.

Let $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$ be the generators of the groups. Let $(g_0^{\alpha}, g_1^{\alpha}, g_0^{\beta}, g_1^{\gamma}, z)$ be an instance of the DBDH problem. If $\alpha = 0$ or $\beta = 0$ or $\gamma = 0$, then the problem is trivial, so we assume these values are non-zero. Now $\mathcal{A}_{\mathrm{DBDH}}$, upon given the DBDH instance, plays the role of the challenger for $\mathcal{A}_{\mathrm{priv}}$ and simulates the IND1 game to $\mathcal{A}_{\mathrm{priv}}$ as follows.

1) The challenger sets $h_0 = g_0^{\beta}, h_1 = g_1^{\gamma}$ and runs the Setup step of Π_{DBDH}. For $t < i \leq n$, $\mathcal{A}_{\mathrm{DBDH}}$ selects uniformly random values $u_i \leftarrow \mathbb{Z}_p$ (these can be thought of implicitly defining sk_i as $sk_i = u_i/\beta$) and sends $pk_i = g_0^{u_i}$ to $\mathcal{A}_{\mathrm{priv}}$.
2) For $1 \leq i \leq t$, $\mathcal{A}_{\mathrm{priv}}$ sends the public keys pk_i to the challenger.
3) For $1 \leq i \leq t$, the challenger chooses uniformly random values $s_i \in \mathbb{Z}_q$ and set $v_i = g_1^{s_i}, w_i = g_0^{s_i}$ and $c_i = pk_i^{s_i}$. For $t < i \leq n$, it generates values $v_i = g_1^{p(i)}$ and $w_i = g_0^{p(i)}$ where $p(x)$ is the unique polynomial of degree at most t determined by $p(0) = \alpha$ and $p(i) = s_i$ for $i = 1, ..., t$. Note that $\mathcal{A}_{\mathrm{DBDH}}$ does not know α, but it does know g_1^{α}, g_0^{α}, $g_1^{s_i}$, and $g_0^{s_i}$ for $1 \leq i \leq t$, so it can use the Lagrange interpolation in the exponent to compute the adequate v_i and w_i. For $t < i \leq n$, it also creates the values $c_i = w_i^{u_i}$. Note that then $c_i = g_0^{u_i \cdot p(i)} = pk_i^{p(i)}$. Finally it sends all this information together with the value z (which plays the role of x_b in the IND game) to $\mathcal{A}_{\mathrm{priv}}$.
4) $\mathcal{A}_{\mathrm{priv}}$ makes a guess b'.

If $b' = 0$, $\mathcal{A}_{\mathrm{DBDH}}$ guesses that $z = e(g_0, g_1)^{\alpha\beta\gamma}$ otherwise $\mathcal{A}_{\mathrm{DBDH}}$ guesses that z is a random element in $\mathbb{G}_T$.

Note that the information $\mathcal{A}_{\mathrm{priv}}$ receives in step 3) is distributed exactly like a sharing phase of the value $e(h_0^{\alpha}, h_1)$ with the PVSS. Since $h_0 = g^{\beta}$ and $h_1 = g_1^{\gamma}$, $e(h_0^{\alpha}, h_1) = e(g_0, g_1)^{\alpha\beta\gamma}$. It is now easy to see that the guessing advantage of $\mathcal{A}_{\mathrm{DBDH}}$ is the same as the advantage of $\mathcal{A}_{\mathrm{priv}}$. $\square$