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 in the fault-free case and $O(\lambda n^2 \log n + n^3)$ in the worst case, where λ is the security parameter. While designing SPURT, 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. 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 beacon output about every second with 64 nodes and every four seconds with 128 nodes. This performance is comparable to systems with stronger assumptions or weaker security.**

I. INTRODUCTION

A reliable source of a continuous stream of shared randomness, also referred to as a *random beacon*, is crucial for many distributed protocols. Applications of random beacon include leader election in proof-of-stake based blockchains [6], [43], blockchain sharding [10], [53], [57], [74], scaling smart contracts [33], annonymous communications [8], [44], [72], [73], solving consensus under asynchrony [39], anonymous browsing [35], [42], [46], publicly auditable auctions and lottery [22], electronic voting [9], cryptographic parameter generations [12], [55], and so on.

The simplest approach is to rely on a single node or organization such as NIST random beacon or Random.org, to produce the required randomness. This is undesirable given incidents such as the backdoor of Dual elliptic curve pseudo-random number generator [15] and 1969 US conscription lottery [1]. It is also unreasonable in systems such as blockchains as a trusted party for randomness generation defeats the blockchain's main object of removing 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 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 [18], a long line of works have looked into the problem of generating shared randomness under different system models [4], [11], [18], [24], [26], [27], [34], [43], [52], [61], [68], [71]. Due to its use in practical blockchain systems, which typically involves a large number of nodes [33], [53], [57], [74], 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 are aimed at eliminating 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 still have scalability issues, require strong cryptographic or network assumptions, or do not provide the full suite of desired properties.

For example, protocols such as [27], [31], [52] 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 running the protocol. A recent work Hydrand [68] reduces communication $O(\lambda n^2)$ but does not provide perfect unpredictability, even in the presence of a semi-honest adversary. Very recently, a concurrent work Brandpiper [16] improves upon Hydrand to provide perfect unpredictibility and increased fault tolerance. As a trade-off, Brandpiper incurs higher worst-case communication and computation costs and makes the q-SDH assumption, which requires a trusted setup to generate the desired public parameters.

Regarding setup assumptions, many protocols [4], [16], [24], [47] assume an initial trusted setup, where a trusted party generates trapdoors based on public parameters and shares them with the nodes. Security of such protocols relies crucially on the adversary's inability to access the trapdoor. Some protocols replace the trusted setup with a Distributed Key Generation (DKG) procedure [40], [71]. But DKG comes with a high initial setup cost: the best-known DKG protocols in

Table I: Comparison of existing randomness beacon protocol.

	Network model	Fault Tolerance	Liveness / Availability	Unpredictability	Bias-resistance	Communication Cost (total)	Computation Complexity	Public Verification Complexity	Cryptographic Primitives	Setup Assumption
Cachin et al. [61]	async.	1/3	✓	✓	✓	$O(\lambda n^2)$	$O(n)$	$O(1)$	Uniq. th-sig.	DKG
RandHerd [71]*	async.	1/3♣	✓	✓	✓	$O(\lambda c^2 \log n)$♣	$O(c^2 \log n)$	$O(1)$	PVSS+CoSi	DKG
Dfinity [47]	sync.	1/2	✓	✓	✓	$O(\lambda n^3)$	$O(n)$	$O(1)$	Uniq. th-sig.	DKG
Drand [4]	sync.	1/2	✓	✓	✓	$O(\lambda n^2)$	$O(n)$	$O(1)$	Uniq. th-sig.	DKG
HERB [31]	sync.	1/3	✓	✓	✓	$O(\lambda n^4)^\ddagger$	$O(n)$	$O(n)$	Partial HE	DKG
Algorand [43]	partial sync.	1/3♣	✓	$\Omega(t)$	✗	$O(\lambda cn)$♣	$O(c)$	$O(1)$	VRF	CRS
Proof-of-Work [61]	sync.	1/2	✓	$\Omega(t)$	✗	$O(\lambda n)$	very high	$O(1)$	Hash func.	CRS
Ouroboros [52]	sync.	1/2	✓	✓	✓	$O(\lambda n^4)^\ddagger$	$O(n^3)$	$O(n^3)$	PVSS	CRS
Scrape [27]	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 [16]	sync.	1/2	✓	$t+1$	✓	$O(\lambda n^2)$	$O(n^2)$	$O(n^2)$	PVSS	$q-$SDH
BRandPiper [16]	sync.	1/2	✓	✓	✓	$O(\lambda n^3)$	$O(n^2)$	$O(n^2)$	VSS	$q-$SDH
SPURT	partial sync.	1/3	✓	✓	✓	$O(\lambda n^2 \log n + n^3)$	$O(n)$	$O(n)$	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.

synchronous and asynchronous networks has a communication cost of at $O(n^3 \log n)$ and $O(n^4 \log n)$, respectively [40], [41], [71]. Another limitation of using DKG, as observed in [16], is the inability/inefficiency to replace nodes. Whenever a participating node is to be replaced, we would need to run the expensive DKG procedure again. Thus, DKG-based solutions such as [4], [25], [40], [47] are efficient when members are fixed, but are not suitable in applications where members change frequently (e.g., proof-of-stake [43], [52]).

An orthogonal and effective approach is the sampling technique [43], [71], 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 if the committee changes frequently, no efficient solution exists.

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, an efficient distributed random beacon protocol that does not require any trusted or expensive setup. SPURT guarantees availability, unpredictability, unbiasibility, and public verifiability in a partially synchronous network [37] against a malicious adversary that controls up to one-third of the nodes.

In a network of n nodes, SPURT's computation cost per node per beacon output is only $O(n)$. SPURT's communication cost per beacon output (across all nodes) is $O(\lambda n^2)$ in the good case (with up to a constant number of malicious nodes) and $O(\lambda n^2 \log n + n^3)$ in the worst case. λ is a security parameter representing the size of group elements. We make the λ term explicit in communication complexity because not all terms depend on it. This is important because the n^3 term is actually not the bottleneck in practice since $n^3 < \lambda n^2 \log n$ for up to $n < 2950$ with a typical $\lambda = 256$.

With these costs, we believe SPURT has good scalability and is suitable for applications with a large number of nodes deployed globally across the internet (possibly after applying the sampling technique).

While designing SPURT, we also design a new publicly verifiable secret sharing (PVSS) scheme whose security relies on the standard Decisional bilinear Diffie-Helmann (DBDH) assumption [21] and does not require a random oracle. Our new scheme is inspired by the PVSS scheme of Scrape [27], which assumes a less standard Decisional Bilinear Squaring assumption [48]. Our new PVSS scheme has comparable efficiency as Scrape's PVSS.

Design overview. Existing protocols that do not rely on trusted setup address these challenges using publicly verifiable secret sharing (PVSS) schemes. 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 these remain 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 $\Omega(n^2)$ [36].

Therefore, a main design philosophy of SPURT is to minimize the amount of data sent via the broadcast channel. Firstly, we utilize the additive homomorphism of commitments and encrypted shares in PVSS to aggregate PVSS messages across nodes. This requires us to customize the use of PVSS procedures in a non-blackbox manner. Secondly, even after aggregation, we still have linear-size data, and we only send its Merkle root via the broadcast channel. Other pieces of data will be sent over pair-wise private channels. These techniques minimize the use of the broadcast channel and yield the quadratic complexity of SPURT.

Another major downside of 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 [27] 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* [32]. The use of broadcast channels then becomes natural as it is the standard technique in the multi-party computation literature [45], [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 its output. This allows us to use a state machine replication protocol (SMR) (cf. §II-E) instead of broadcast channels, which enables us to handle a partially synchronous network.

Other secondary techniques include co-designing PVSS procedures and SMR and achieve the optimal $t < n/3$ fault tolerance in partial synchrony, using pairing to verify correct decryption of shares, and using multi-signatures to forward SMR decision proofs to enforce unpredictability. We will elaborate on these in §V.

Evaluation. We implement SPURT in Golang atop the open-source Quorum codebase [7] for Istanbul BFT [60]. We evaluate our prototype for a network of up to 128 nodes running in geographically distributed AWS EC2 instances. We evaluate the throughput of SPURT, measured as the number of beacon outputs generated per minute, the network bandwidth usage, and computation time per node per beacon output.

We compare with Hydrand [68] and the deployed protocol Drand [4]. Note that Hydrand has imperfect predictability, and Drand requires a DKG setup. Our evaluation illustrates that SPURT can generate beacons at a rate comparable to or better than Drand and Hydrand. For example, for a network of size of 32 and 64, SPURT can generate about 90 and 45 beacon output every every minute, respectively. However, SPURT has a bandwidth cost (amount of data sent and received per node) of 35 Kilobytes and 71 Kilobytes for a network of 32 and 64 nodes, respectively, which is $6\times$ higher than Drand and is about 65% of the bandwidth cost of Hydrand.

Summary. In summary, we make the following contributions:

- We design SPURT, a distributed random beacon protocol with a communication cost of $O(\lambda n^2 \log n + n^3)$ ($O(\lambda n^2)$ in the common case) per beacon output and does not require any trusted or expensive setup. We formally prove SPURT achieves all desired properties against a malicious adversary controlling up to one-third of the nodes in a partially synchronous network.
- We design a new PVSS scheme whose security relies on the Decisional bilinear Diffie-Hellman assumption.
- We prototype SPURT and evaluate it in a network of up to 128 nodes. Our evaluation shows that SPURT can generate an output every few seconds, which is comparable to systems with stronger assumptions or weaker guarantees.

Paper organization. The rest of the paper is organized as follows. In §II, we give preliminaries and notations. We give our new PVSS scheme in §III. We describe the system model and an overview of SPURTin §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. PRELIMINARIES

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 bold face lowercase letters such as $\mathbf{x}$.

We next define the desired properties of a distributed random beacon protocol and then briefly discuss the tools we use in SPURT.

A. Randomness Beacon

The two most crucial property for a randomness beacon are *unpredictability* and *bias-resistance*. Unpredictability ensures no nodes are able to predict or compute any function on any future beacon outputs with non-negligible advantage. The bias-resistance property of the beacon protocol requires that every output is chosen uniformly randomly from the intended distribution and independently of other outputs. 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*, which states that each beacon output is efficiently verifiable even by users that do not directly participate in the beacon generation protocol. All properties should hold in the presence of a computationally bounded adversary controlling a threshold fraction of nodes in the system.

B. Bilinear Pairings

SPURT and our new PVSS scheme Π_{DBDH} make use of pairing. In particular, security of Π_{DBDH} relies on the the decisional version of the *bilinear Diffie-Hellman* assumption.

Definition 1 (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 $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ satisfying the following properties.

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

$$e(u \cdot u', v) = e(u, v) \cdot e(u', v), \text{ and} \tag{1}$$
$$e(u, v \cdot v') = e(u, v) \cdot e(u, v') \tag{2}$$

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.

C. 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 equality of discrete logarithm. The first protocol is the classic Chaum-Pedersen Σ-protcotol [30] in the random oracle model, and the second protocol uses bilinear pairings.

Chaum-Pedersen Σ-protocol. 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.

This protocol can be made non-interactive in the random oracle model using the Fiat-Shamir heuristic [38], [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 above 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 pairing based protocol for equality of discrete logarithm is rather straightforward and 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:

$$e(g_0, y) = e(x, g_1) \tag{3}$$

In case of an honest prover equation (3) will hold because

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

D. 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 [17], [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].

E. 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 v in height r, then for height r, no honest node decides on a value v' such that $v' \neq v$ for height r.
- **Validity/Liveness.** If an honest node broadcasts a value v, every honest node eventually decides v 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 [29] in our case only participating nodes propose values and all decided values must meet a certain external valid predicate M.

We will use Istanbul BFT (IBFT) [60]. It is a variant of the popular PBFT [29] protocol and tolerates up to one third malicious nodes in a partially synchronous network (which is optimal). IBFT is an *epoch* based protocol, where each epoch has a leader. In every epoch, the IBFT protocol finalizes a value in three steps: *Propose, Prepare*, and *Commit*. We present a simplified description of the IBFT protocol in Figure 1, and refer the reader to [60] 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 z to be finalized at height ht by sending $\langle propose, z, r, \mathsf{ht}, X\rangle$ message to all the nodes. Here X is the view change certificate (if any) that validates that the proposal is safe.

Prepare. Each node P_j, upon receiving the proposal checks whether the proposal is consistent with IBFT specifications using X, and $M(z)$ is true for an external predicate $M(\cdot)$. If both checks pass, P_j multi-casts $\langle prepare, z, r, \mathsf{ht}\rangle$ to all nodes.

Commit. Upon receiving $2f+1$ $prepare$ messages for the proposal z at height ht and epoch r, P_j multi-casts $\langle commit, z, r, \mathsf{ht}\rangle$ message to every node.

Figure 1: Steady state of Istanbul BFT [60] SMR protocol.

III. 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 [27], 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 B for formal definition and desired security properties of PVSS.

Our PVSS scheme allows a node (dealer) to share a *uniform* random 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: PVSS.Setup, PVSS.Share, PVSS.Verify, and PVSS.Recon. The 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 $pk_i = h_0^{sk_i}$. 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.Recon is used to recover $e(h_0^s, h_1)$. We describe them in detail in Figure 2.

$\mathsf{PVSS.Setup}(1^\lambda) \to (g_0, h_0, g_1, h_1, \{(sk_i, pk_i)\})$:
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 pki by posting it to the public ledger, for $1 \le i \le n$.

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 $j = 1, 2, \ldots, n$.

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

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.

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

1) Sample a random code word $\mathbf{y}^\perp \in C^\perp$ and check whether
$$\prod_{k=1}^{n} v_k^{x_k^\perp} = 1_{\mathbb{G}_1} \tag{4}$$
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 $\tilde{s}_j$.

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

1) Use Lagrange interpolation to compute
$$\prod_{k\in H} (\tilde{s}_k)^{\mu_k} = \prod_{k\in H} h_0^{\mu_k \cdot p(k)} = h_0^{p(0)} \tag{5}$$
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}.

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 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 [27, Lemma 1] for its proof, and Appendix A 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 A. Also in appendix B, we will define the required properties for PVSSsuch as correctness, verifiability, and IND1-Secrecy. We then prove that assuming DBDH hardness, Π_{DBDH} guarantees the desired correctness, verifiability, and IND1-Secrecy properties.

IV. System Model and Overview

A. System Model

We consider a network of n nodes connected via pairwise authenticated channels. 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. 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 also 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 §II.

We assume the network is partially synchronous, 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, messages can be delayed arbitrarily. (Theoretical works often state partial synchrony differently [37] (e.g., using an unknown Global Standardization Time, GST) for rigor or convenience, but the essence is to capture the above practical oscillating network model.) A 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.

B. Overview

We now give an overview of SPURT to describe our core ideas. For ease of exposition, we will first explain how SPURT generates a single beacon output assuming that nodes have access to a broadcast channel. Strictly speaking, a single-value broadcast channel [54] is impossible in a partially-synchronous network; we merely assume its existence to simplify this overview and aid intuitive understanding. Later in §V, we will replace it with a BFT SMR protocol while preserving the overall efficiency of the protocol. Throughout this paper, we assume that all messages exchanged between honest nodes are digitally signed by the sender, and recipients validate them before processing them further.

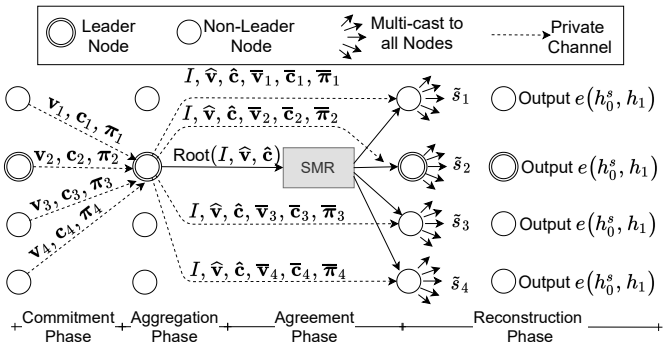

Figure 3: Messages sent during each phase of the SPURT. We describe contents of the messages i.e., the notations over the arrows in §V. We use Root$(\cdot)$ as the shorthand for Merkle root.

SPURT proceeds in epochs where 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. Every epoch has four phases: *Commitment, Aggregation, Agreement* and *Reconstruction* phase. We illustrate the communication pattern of in all four in Figure 3 and describe the message contents (symbols over the arrows) in §V.

Commitment phase. During the commitment phase, each node chooses a uniformly random secret and computes shares for the chosen secret using the PVSS.Share primitive described in §III. Each node then sends all these shares to L_r. Here on, we refer to the messages sent by nodes to L_r as the PVSS messages of epoch r. We remark that, despite having access to PVSS messages from all nodes, L_r can not break unpredictability of SPURT. This is because each share is encrypted using the public key of the intended recipient node, and a zero-knowledge scheme is used for proving consistency. We will give more details on this in §VI.

Aggregation phase. In the aggregation phase, L_r upon receiving PVSS tuple from a node, validates it using PVSS.Verify from §III. Upon receiving and validating PVSS messages from $t+1$ nodes, L_r aggregates them using the additive homomorphic property of the underlying polynomial commitment and encryption schemes. If $p_1, p_2, \ldots, p_{t+1}$ are the underlying polynomials from the $t+1$ valid polynomial commitments, L_r aggregates them to obtain the commitment to the *aggregated polynomial* $\hat{p}(x) = \sum_{j=1}^{t+1} p_j(x)$. Moreover, L_r aggregates the $t+1$ encrypted shares to obtain the encrypted shares corresponding to the aggregated polynomial $\hat{p}(\cdot)$.

Agreement phase. After aggregation, L_r computes a cryptographic digest of the commitment of the aggregated poly-

nomial $\hat{p}(\cdot)$, the identities (or indices) of nodes whose polynomials are aggregated into $\hat{p}(\cdot)$, and the encrypted shares of the secret embedded in $\hat{p}(\cdot)$. L_r then sends this cryptographic digest to all of the nodes via a broadcast channel. Note again that our actual protocol will use an SMR protocol instead.

Additionally, to each node i, L_r sends node i the entire commitment to the aggregated polynomial $\hat{p}(\cdot)$, and the encrypted shares corresponding to $\hat{p}(\cdot)$ using the pair-wise channel between i and L_r, Moreover, L_r also sends the encrypted shares for i of the original $t+1$ polynomials aggregated into $\hat{p}(\cdot)$, and the corresponding NIZK proofs. Note that these shares are encrypted under the public key of i. In total, during the agreement phase, L_r sends $O(\lambda)$ bits of data via the broadcast channel and $O(n\lambda)$ bits of data to each node using pair-wise private channels.

Each node i, upon receiving the cryptographic digest over the broadcast channel and private messages from L_r, validates them to ensure that L_r did the aggregation phase correctly. For this step, node i relies on the properties of linear error-correcting code and NIZK proofs forwarded by L_r. Upon successful validation, node i starts the reconstruction phase. Else, node i moves to the next epoch with the next leader and the cycle continues.

Reconstruction phase. When the agreement phase terminates, i.e., all honest nodes agree on the cryptographic digest broadcast by L_r, every honest node who received valid shares from L_r multicasts its aggregated share along with the NIZK proof of its correctness. As we show in §VI, if the agreement phase terminates successfully, then at least $t+1$ honest nodes hold valid shares of the aggregated polynomial $\hat{p}(\cdot)$. Also, all nodes will be able to prove the correctness of their aggregated shares. Moreover, all these nodes start the reconstruction process within three message transmission delays. Hence, during the reconstruction phase, every honest node will receive at least $t+1$ valid shares of $\hat{p}(\cdot)$, along with their correctness proofs. As a result, every honest node will be able to successfully reconstruct $h_0^{\hat{p}(\cdot)}$, and hence the output of the beacon for this epoch as $e(h_0^{\hat{p}(0)}, h_1)$.

V. Design and Optimizations

In this section, we present the detailed design of SPURT. As discussed in previous sections, SPURT proceeds in epochs and each epoch has four phases. We next describe each phase in detail. The notations are summarized in Table II.

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 the PVSS tuples using the PVSS.Share primitive described in §III:

$$\mathbf{v}_i, \mathbf{c}_i, \leftarrow \mathsf{PVSS.Share}(s_i, g_1, h_0, sk_i, \{pk\}_{j,j=1,2,\ldots,n}) \quad (6)$$

where $\mathbf{v}_i = \{v_{i,1}, \ldots, v_{i,n}\}$ and $\mathbf{c}_i = \{c_{i,1}, \ldots, c_{i,n}\}$.

Also, for each $j \in \{1, 2, \ldots, n\}$, node i computes π_j as

$$\pi_j = \mathsf{dleq.Prove}(g_1, v_j, pk_j, c_j, s_{i,j})$$

Table II: Notations used in the paper

Notation	Description
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 $pk_i^{s_{i,j}}$
$\mathsf{dleq}(\cdot)$	NIZK proof for equality of discrete logarithm
$C^{\perp}$	Dual of error correcting code C

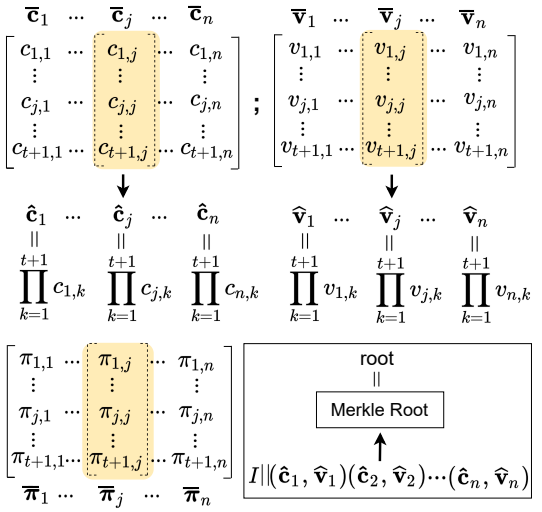

Figure 4: Aggregation phase at the leader.

where $s_{i,j}$ is the share of secret s_i for node j. Let $\boldsymbol{\pi}_i = \{\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,n}\}$. Node i then sends $(\mathbf{v}_i, \mathbf{c}_i, \boldsymbol{\pi}_i)$ to L_r.

B. Aggregation Phase

L_r on receiving a tuple $(\mathbf{v}_i, \mathbf{c}_i, \boldsymbol{\pi}_i)$, first validates $\mathbf{v}_i$ and $\mathbf{c}_i$ using $\mathsf{PVSS.Verify}(\mathbf{v}_i, \mathbf{c}_i, \{pk_j\}_{j=1,2,\ldots,n})$. Then, for each j, L_r checks $\pi_{i,j}$ using the dleq.Verify. We remark that since the leader anyway checks the equality of discrete 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 $\hat{p}(\cdot) = \sum_{i \in I} p_i(\cdot)$. L_r also aggregates the encrypted shares into $\hat{\mathbf{c}} = (\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_n)$, encrypted shares for the aggregated secret $\hat{p}(0)$.

$$\hat{v}_\ell = \prod_{i \in I} v_{i,\ell}, \qquad \hat{c}_\ell = \prod_{i \in I} c_{i,\ell} \quad (7)$$

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 $\bar{\boldsymbol{\pi}}_j = \{\pi_{1,j}, \pi_{2,j}, \ldots, \pi_{t+1,j}\}$. Then, $\hat{c}_j$ 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 root, the Merkle root that commits $I, \hat{\mathbf{v}}$, and $\hat{\mathbf{c}}$, also shown in Figure 4. In the agreement phase, root will be the only value that is sent via SMR. $I, \hat{\mathbf{v}}, \hat{\mathbf{c}}$ themselves and the original PVSS tuples will be sent privately to corresponding nodes.

C. Agreement Phase

Let ht be the height chosen by L_r according to SMR. Then, to each node j, L_r sends $(\texttt{root}, \hat{\mathbf{v}}, \hat{\mathbf{c}}, I, \bar{\mathbf{v}}_j, \bar{\mathbf{c}}_j, \bar{\boldsymbol{\pi}}_j, \texttt{ht})$ and proposes root 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 corresponds to the BFT SMR proposal on root for epoch r and height ht.

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

1) The proposal is safe according to SMR,
2) root is a valid Merkle root of $I, \hat{\mathbf{v}}$, and $\hat{\mathbf{c}}$; and
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 whether
$$\prod_{k=1}^{n} \hat{v}_k^{y_k^{\perp}} = 1_{\mathbb{G}_2}; \text{ and} \tag{8}$$
This check ensures that $\hat{\mathbf{v}}$ is a commitment to a polynomial if 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 valid dleq proof according to §II-C; and
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 mentioned checks pass, node j multi-casts the $\langle prepare, \texttt{root}, r, \texttt{ht}\rangle$ to all other nodes. Alternatively, 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. §II-E). An honest node upon receiving $2t+1$ $\langle prepare, \texttt{root}, r, \texttt{ht}\rangle$ messages multi-casts $\langle commit, \texttt{root}, r, \texttt{ht}\rangle$ to all other nodes. Next, each honest node upon receiving $2t+1$ $\langle commit, \texttt{root}, r, \texttt{ht}\rangle$ messages decides on root at height ht. We will use a multisignature scheme [20] to combine these $commit$ messages into a $(\lambda + n)$-bit proof for the SMR decision.

D. Reconstruction Phase

Every honest node that *decides* root (cf. §II-E) and receives valid messages from the leader during the agreement phase starts the reconstruction phase for the beacon at height ht. In particular, these nodes compute the reconstruction share $\tilde{s}_j$ as follows and multicasts $\tilde{s}_j$ to all other nodes.

$$\tilde{s}_j = \hat{c}_j^{\frac{1}{sk_j}} = h_0^{\sum_{i \in I} s_{i,j}} \tag{9}$$

Let H be the set of honest nodes and let $V \subseteq H$ be the set of honest nodes that received valid messages from L_r in SMR, but are yet to decide on root. Upon receiving a reconstruction message $\tilde{s}_j$ from a node j, nodes in V validate $\tilde{s}_j$ per equation 10. Upon successful validation, nodes in V request from node j the proof of the decision on root, to which node j responds with the multi-signature as the proof of SMR decision. Upon receiving the multi-signature, nodes in V validate it, and on successful validation multicast their reconstruction shares. This implies all honest nodes who received valid shares from L_r start reconstruction within three message delays from the instant an honest node decides on root.

Every node i, upon receiving a tuple $\tilde{s}_j$, validates it using the pairing-based discrete log equality check (cf. § II-C)

$$e(\tilde{s}_j, g_1) = e(h_0, \hat{v}_j) \tag{10}$$

Note that some honest nodes who did not receive valid messages from the leader may not have $\hat{\mathbf{v}}$ and/or $\hat{\mathbf{c}}$. In such a situation, these nodes, up receiving $\tilde{s}_j$ from node j, query node j for $(\hat{v}_j, \hat{c}_j)$ and the Merkle path from $(\hat{v}_j, \hat{c}_j)$ to root,

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 $|T| \geq t+1$, i outputs the beacon output 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:

$$\prod_{k \in T} (\tilde{s}_k)^{\mu_k} = \prod_{k \in T} h_0^{\mu_k \cdot \hat{p}(k)} = h_0^{\hat{p}(0)} \tag{11}$$

where $\mu_k = \prod_{j \neq k} \frac{j}{j-k}$ are the Lagrange coefficients.

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 usage 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:

$$g' = \prod_{k=1}^{m} g_k^{a_k} \tag{12}$$

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 will first argue that SPURT is available in a partially synchronous network. Next, we will prove that every output of SPURT is unpredictable, bias-resistant, and publicly-verifiable. We then analyze the computation and communication complexity of each epoch.

A. Reconstructability and Availability

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

Proof. If an honest node decides `root`, there must be $2t+1$ $prepare$ and $commit$ messages. $t+1$ 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 (8).

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 (8) 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, each of the $t+1$ honest nodes who sent $prepare$ (say 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, at least $t+1$ honest node will multi-cast valid decrypted shares, hence every honest node will receive at least $t+1$ valid decrypted shares which is sufficient for reconstruction.

Consider an honest node i that outputs h_0^a for some $a \in \mathbb{Z}_q$ during the reconstruction phase. 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. Observe that, a successful discrete equality check implies $a_j = s_j$ as

$$e(h_0, \hat{v}_j) = e(h_0, g_1)^{s_j} = e(h_0^{s_j}, g_1) \tag{13}$$

Since equation (13) holds for every valid decrypted shares, upon Lagrange interpolation in the exponent using these decrypted shares, node i will compute $h_0^{\hat{p}(0)}$. □

Next, we will argue that during periods of synchrony, when an honest node becomes leader SPURT is available.

Theorem 1. (Availability) *During periods of synchrony, if the leader L_r of an epoch r is honest,* SPURT *will produce an unique output and that output will be available at every honest node.*

Proof. When L_r is honest, all the checks described in §V-C will be successful at every honest node. Thus, the SMR will proceed normally. Hence, during periods of synchrony, due to the liveness property of the SMR honest nodes will decide on the value proposed by L_r. Next, from Lemma 2, we know that whenever the SMR decides, the corresponding beacon output is reconstructible. Moreover, from §V-D, we know that every node that received a valid message during the agreement phase will multicast their decrypted shares along with its proof of correctness to every other node. This implies that every honest node will receive at least $t+1$ valid decrypted shares to reconstruct the beacon output for epoch r. □

B. Unpredictability and Bias-Resistance

It follows from our Theorem 5, Proposition 1 of [52], and knowledge soundness of dleq protocol that except for negligible probability, the polynomials chosen by the adversarial nodes are independent of the polynomials chosen by the honest nodes. We will use this to argue that every beacon output includes an input of at least one honest node.

Proposition 1. *For any epoch r, if honest nodes decide on* `root` *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 independent of $p_i(\cdot)$. As a result, $\hat{p}(0)$ is uniformly random and independent of aggregated polynomial of any other epoch.

Proof. When an honest node decides `root`, 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 $\deg(p_i(\cdot)) = t$. 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 $p_i(\cdot)$. Moreover, since the polynomials chosen by adversarial nodes are independent of $p_i(\cdot)$, this implies that 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 $\deg(q(\cdot)) \leq t$. □

We will next use the safety property of the BFT SMR, unique reconstruction for every SMR decision, and Proposition 1 to prove the bias-resistance of every beacon output.

Theorem 2. (Bias-Resistant) *Every* SPURT *output is uniformly random and is independent of any other beacon output.*

Proof. For any epoch r and height ht, from Proposition 1, we know that for every SMR decision on digest $\mathtt{root}$ corresponding to aggregated polynomial $\hat{p}(\cdot)$, $\hat{p}(0)$ is uniformly random and independent of aggregated polynomials of any other height. Moreover, from Lemma 2, a beacon output corresponding to every digest finalized by the atomic broadcast is reconstructible. This implies that SPURT is bias resistant. □

Theorem 3. (Unpredictability) SPURT *ensures unpredictability in the sense that as soon an adversary $\mathcal{A}$ learns any function of a beacon output, every honest party learns the beacon output within three communication delays.*

Proof. From Proposition 1, every beacon output includes a secret from at least one honest party. Let s^* be the secret of the honest node. Then, from Theorem 5, till an honest node starts reconstruction of the aggregated secret, s^* is indistinguishable from a uniformly randomly chosen element in $\mathbb{Z}_q$. Whenever an honest node starts the reconstruction phase, after a round trip delay all honest nodes start the reconstruction phase. Hence, all honest nodes will reconstruct the corresponding beacon output at most three communication delays later since the time the adversary learns the output. This implies that SPURT ensures unpredictability. □

C. Public Verifiability

Public verifiability for beacon protocols producing true random numbers differs from beacon protocols producing pseudorandom numbers [4], [24], [47], [71]. In pseudorandom beacons, each beacon output is some deterministic function of the secret key generated during the initial setup phase. Hence, the output of such protocols can be efficiently verified given only the verification/public key corresponding to the secret key used for beacon generation. Contrary to this, truly random beacon protocols such as Scrape [27], Hydrand [68], and SPURT do not have a trusted setup phase, so their outputs are verified using the transcript of the interaction between nodes.

At any given height ht, let $\mathtt{root}$ be digest decided by honest nodes, and o_{ht} be the output of SPURT. Then, to verify the validity of o_{ht}, a user (need not be one of the nodes) queries up to $t+1$ nodes, either in sequence or in parallel, for the SMR decision certificate $\mathcal{C}_{\mathsf{ht}}$ for height ht, leaves of the Merkle tree corresponding to $\mathtt{root}$, and the $t+1$ valid reconstruction shares along with their proofs of correctness. By Theorem 1, every honest node will have these data at the end of the reconstruction phase. Upon receiving these data, an external client can validate them by checking that:

- $\mathcal{C}_{\mathsf{ht}}$ is a multi-signature of at least $2t+1$ nodes.
- $\mathtt{root}$ is a valid Merkle root of the received leaves.
- The dleq proofs of all the reconstruction shares are valid and consistent with $\mathtt{root}$ and the quorum certificate $\mathcal{C}_{\mathsf{ht}}$.

Table III: Summary of communication and computation cost of each epoch of SPURT. — indicate the no cost for the corresponding phase. The f in the communication cost of a non-leader during the reconstruction phase denotes the number of faulty nodes in the network.

	Communication		Computation	
Protocol Phase	Leader	Non-leader	Leader	Non-leader
Commitment	$O(\lambda n^2)$	$O(\lambda n)$	—	$O(n)$
Aggregation	—	—	$O(n^2)$	—
Agreement	$O(\lambda n^2)$	$O(\lambda n)$	—	$O(n)$
Reconstruction	—	$O(\lambda n + f(\lambda \log n + n))$	—	$O(n)$

D. 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 number of bits each node needs to send in every epoch. We then analyze the computation complexity of each node measured in of number of exponentiations and pairings each node needs to perform every epoch and report the amortized computation cost of each beacon output. Also, throughout this section, we will assume that signatures and multi-signatures are $O(\lambda)$ and $n + O(\lambda)$ bits long, respectively. Also, we assume that a node needs to perform $O(1)$ exponentiations and pairings to compute and validate a single signature, and $O(k)$ exponentiations,$O(1)$ pairings to create and validate a multi-signature of k nodes. We summarize our performance analysis in Table III.

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 a constant number of group elements to all other nodes. Hence, the agreement phase's total communication complexity is $O(\lambda n^2)$. Finally, during the reconstruction phase, when there are f malicious nodes, honest nodes may have to send the Merkle path ($O(\log n)$ group elements) corresponding to their share, the aggregated signature, and the list of signers ($O(n)$ bits) to f different nodes. Hence, we get a communication cost per node would be $O(n\lambda + f(\lambda \log n + n))$.

Also observe that during periods of synchrony, for every n epochs, there will be at least $\lceil 2n/3 \rceil$ honest leaders. From Theorem 1, for every honest leader, SPURT will produce an output. Hence, in every sequence of n epochs, SPURT will output at least $\lceil 2n/3 \rceil$ outputs. This implies that the *amortized* communication complexity of each beacon output is $O(\lambda n^2)$ in the good case (i.e., $f = O(1)$) and $O(\lambda n^2 \log n + n^3)$ in the worst case.

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 [27], 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 performs $O(n)$ hash computation to construct the required Merkle tree. 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 $O(n)$ pairing operations to validate signatures on messages sent by other nodes. Finally, in the reconstruction phase, every node verifies decrypted shares using $O(n)$ pairings and possibly reconstructs a Merkle tree of size $O(n)$. Moreover, nodes might also need to aggregate $O(n)$ signatures. 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-C, to validate a beacon output, external clients need to download the SMR BFT decision certificate, the leaves of the Merkle tree, the reconstruction shares, and the corresponding commitments. Each of these messages is $O(n)$ group elements. Hence, the communication cost of verifying a beacon output is $O(\lambda n)$. Moreover, verifying all these messages require $O(n)$ exponentiations and pairings. Verifying $O(n)$ signatures and commitments , constructing the Merkle tree, and reconstruction of the secret all 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 five 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.

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 and implements the Istanbul BFT protocol as one of its consensus protocol. Istanbul BFT [60] is a variant of PBFT protocol with a total quadratic communication complexity both during view change and steady-state. 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` [2] 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-signatures during the reconstruction phase, we implemented the multi-signature scheme of [20] using the `bn256` elliptic curve [3]. For multi-exponentiations, we have used the native implementation of [2] which implements the multi-exponentiation algorithm from [14, §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 [5] and Drand [4]. 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 $O(n\lambda)$ 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 35 to 71 Kilobytes. This is about 65% 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

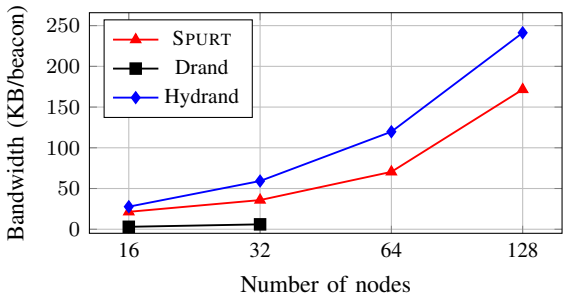

Figure 5: Average bandwidth usage (sent + received data) measured in Kilobytes per beacon output with varying number of nodes.

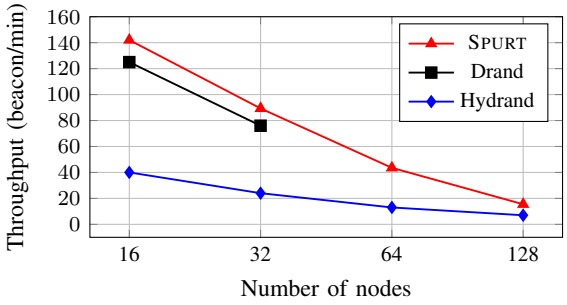

Figure 6: Average number of random beacon generated per minute with varying number of nodes.

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 multicasts 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 $6\times$ 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 140, 90, 45, and 15 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 [5], 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 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 V 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.

We also report in Table IV the number of exponentiation and pairing operations each node needs to perform per beacon output. These numbers are more useful for comparing computation cost with prior works because the concrete computation time depend upon implementation details. Note that the number of operations depend on f, the actual number of faulty nodes in the system. Concretely, the computation cost of SPURT is comparable to Hydrand but about 20x worse than Drand.

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 [4], [16], [24], [31], [40], [47], [71]. 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

Protocol	# exponentiations		Best case, i.e., $n = 64, f = 0$		Worst case, i.e., $n = 64, f = t$	
	Leader	Non-leader	Leader	Non-leader	Leader	Non-leader
Scrape [27]	—	$5n^2 + 7f$	—	20901	—	21125
Drand [4]	—	$n/2$	—	32	—	32
Hydrand [68]	$4n$	$\frac{32}{3}n + 17f$	563	694	901	1054
SPURT	$\frac{7}{3}n^2 + 8n + f(7n)$	$\frac{28}{3}n + 4f$	10387	613	19935	698

Table IV: Computation cost (amortized) measured in number of exponentiations and pairing operations per node per beacon output. We assume $n = 64$; $t = n/3$ for Hydrand and SPURT and $t = n/2$ for Scrape to calculate concrete worst case costs. The best case refers to $f = 0$. SPURT uses pairing based signature scheme from [20] for signing SMR messages. It also uses pairings for validating decrypted shares during the reconstruction phase. Overall, SPURT incurs a pairing cost of $8(n/3 + f)$ which translates to 342 pairing operations for 64 nodes. Drand uses BLS signatures and incurs a pairing cost of $2(n/2 + f)$ which translates to 128 pairing operations for 64 nodes.

Protocol Phase	Time taken (in milliseconds)			
	$n = 16$	$n = 32$	$n = 64$	$n = 128$
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 V: Time taken (in milliseconds) to compute different cryptographic functions required in the different phases of SPURT.

asynchronous network whereas Drand and Dfinity assume a synchronous network. Cachin et al., and Dfinity do not discuss details about the setup phase. Drand uses the DKG protocol of Gennaro et al. [41], 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 $O(\lambda n^3 \log n)$. After the DKG step, the communication complexity for generating one beacon output for all three protocol is $O(n^2\lambda)$. Aleph [40] 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 [11], [27], [34], [43], [52], [61], [68]. Most relevant to our work are Scrape [27] and Hydrand [68]. Both Scrape and Hydrand assume that the underlying network is synchronous. Scrape [27] 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. [16] 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 [16] 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 [71] 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) [61], Proof-of-Delay [23], Algorand [43], Ouroboros [52], Ouroboros Praos [34], etc. The Ouroboros [52] 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 [13], [28], [67] or Verifiable Delay Functions [19].

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 [27], [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 [62] and the verifier in these schemes need to perform $O(n^2)$ exponentiations. The schemes from [48] and [27] 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.

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. In the fault-free case of operation, SPURT has amortized total communication of $O(\lambda n^2)$. In the worst case, the amortized total communication cost is $O(\lambda n^2 \log n + n^3)$. (Note that for $\lambda = 256$ and $n < 2950$, $n^3 < \lambda n^2 \log n$.) Computation wise, each node performs $O(n)$ group exponentiations per beacon output. While designing SPURT, 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 [39], 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] "Draft lottery (1969)," 1969. [Online]. Available: https://en.wikipedia.org/wiki/Draft_lottery_(1969)
[2] "bls12381," 2020. [Online]. Available: https://github.com/ConsenSys/gnark-crypto/tree/master/ecc/bls12-381
[3] "bn256cloudflare," 2020. [Online]. Available: https://github.com/cloudflare/bn256
[4] "Drand - a distributed randomness beacon daemon," 2020, https://github.com/drand/drand.
[5] "Hydrand," 2020. [Online]. Available: https://github.com/PhilippSchindler/HydRand
[6] "Proof of stake (pos)," 2020. [Online]. Available: https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/proof-of-stake/
[7] "Quorum: A permissioned implementation of ethereum supporting data privacy," 2020. [Online]. Available: https://github.com/ConsenSys/quorum
[8] 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.
[9] B. Adida, "Helios: Web-based open-audit voting."
[10] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," 2017.
[11] S. Azouvi, P. McCorry, and S. Meiklejohn, "Winning the caucus race: Continuous leader election via public randomness," *arXiv preprint arXiv:1801.07965*, 2018.
[12] 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.
[13] C. Baum, B. David, R. Dowsley, J. B. Nielsen, and S. Oechsner, "Craft: Composable randomness and almost fairness from time," Cryptology ePrint Archive, Report 2020/784, 2020, https://eprint.iacr.org/2020/784.
[14] 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.
[15] D. J. Bernstein, T. Lange, and R. Niederhagen, "Dual ec: A standardized back door," in *The New Codebreakers*. Springer, 2016, pp. 256–281.
[16] A. Bhat, N. Shrestha, A. Kate, and K. Nayak, "Randpiper–reconfiguration-friendly random beacons with quadratic communication," 2020.
[17] G. R. Blakley, "Safeguarding cryptographic keys," in *1979 International Workshop on Managing Requirements Knowledge (MARK)*. IEEE, 1979, pp. 313–318.
[18] M. Blum, "Coin flipping by telephone a protocol for solving impossible problems," *ACM SIGACT News*, vol. 15, no. 1, pp. 23–27, 1983.
[19] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Annual international cryptology conference*. Springer, 2018, pp. 757–788.
[20] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2018, pp. 435–464.
[21] 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.
[22] J. Bonneau, J. Clark, and S. Goldfeder, "On bitcoin as a public randomness source." *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 1015, 2015.
[23] 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.
[24] 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.
[25] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*. IEEE, 2005, pp. 191–201.
[26] 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.
[27] 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.

[28] ——, "Albatross: publicly attestable batched randomness based on secret sharing," Cryptology ePrint Archive, Report 2020/644, 2020, https://eprint.iacr.org/2020/644.
[29] 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.
[30] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Annual International Cryptology Conference*. Springer, 1992, pp. 89–105.
[31] A. Cherniaeva, I. Shirobokov, and O. Shlomovits, "Homomorphic encryption random beacon." 2019.
[32] 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.
[33] 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.
[34] 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.
[35] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Naval Research Lab Washington DC, Tech. Rep., 2004.
[36] 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.
[37] 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.
[38] 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.
[39] 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.
[40] 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.
[41] 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.
[42] 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.
[43] 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.
[44] S. Goel, M. Robson, M. Polte, and E. Sirer, "Herbivore: A scalable and efficient protocol for anonymous communication," Cornell University, Tech. Rep., 2003.
[45] 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.
[46] D. Goulet and G. Kadianakis, "Random number generation during tor voting," *Tor's protocol specifications-Proposal*, vol. 250, 2015.
[47] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series, consensus system," *arXiv preprint arXiv:1805.04548*, 2018.
[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] 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.
[53] 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.
[54] L. LAMPORT, R. SHOSTAK, and M. PEASE, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.
[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] H. Moniz, "The istanbul bft consensus algorithm," *arXiv preprint arXiv:2002.03613*, 2020.
[61] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[62] 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.
[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," 2021.
[68] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "Hydrand: Practical continuous distributed randomness," 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] 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.
[72] 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.
[73] 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.
[74] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 931–948.

APPENDIX A
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 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 [27] for its proof.

Lemma 3. *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 B
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*.

A PVSS protocol has four steps described below:

- *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.
- *Sharing:* 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.
- *Verification:* 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.
- *Reconstruction:* During reconstruction, 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.

Any 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 and the secret can be reconstructed in the reconstruction step.
- *Verifiability:* If the checks in the verification step passes, 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 dealder.
- *IND1-Secrecy:* 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 [27], [56], [66].

Definition 2. (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_b 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 §III, 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-Secrecy of Π_{DBDH} assumes hardness of the Decisional bilinear Diffie-Helmann (DBDH) assumption, which is given below.

Definition 3 (Decision 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

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

Theorem 5. *Under the Bilinear Decisional 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 $\mathcal{A}_{\mathrm{priv}}$. □