OptRand: Optimistically Responsive Distributed Random Beacons

Adithya Bhat*[1], Aniket Kate[1], Kartik Nayak[2], and Nibesh Shrestha*[3]

[1]Purdue University, {bhat24, aniket}@purdue.edu
[2]Duke University, kartik@cs.duke.edu
[3]Rochester Institute of Technology, nxs4564@rit.edu

Abstract

Distributed random beacons publish random numbers at regular intervals, which anyone can obtain and verify. The design of public distributed random beacons has been an exciting research direction with significant implication to blockchains, voting and beyond. Random beacons, in addition to being bias-resistant and unpredictable, also need to have low communication cost, low latency, and ease of reconfigurability. Existing works on synchronous random beacons sacrifice one or more of these properties.

In this work, we design an efficient unpredictable synchronous random beacon protocol, OptRand, with quadratic (in the number n of system nodes) communication complexity per beacon output. First, we innovate by employing a novel combination of bilinear pairing based publicly verifiable secret sharing and non-interactive zero-knowledge proofs to build a linear (in n) sized publicly verifiable random sharing. Second, we develop a state machine replication protocol with linear-sized inputs that is also optimistically responsive, i.e., it can progress responsively at actual network speed during optimistic conditions, despite the synchrony assumption, and thus incur low latency. In addition, we present an efficient reconfiguration mechanism for OptRand that allows nodes to leave and join the system.

1 Introduction

The use of public random numbers is fundamental to many secure privacy-preserving systems where protocol parties have tamper-proof access to these random values (or common coins). Voting, lotteries, blockchains, and financial services depend on public randomness, and generating public randomness [42] has been an active area of research for the last four decades [49, 23, 44, 45, 11, 33, 32, 18, 31, 41, 15, 8]. Among these efforts, NIST's Randomness Beacons project [19] and Drand organization's beacon [24] have emerged in the last few years as real-world systems towards catering to this need for randomness beacons.

Informally, these systems offer random beacons, which are regular outputs of fresh public random numbers that are bias-resistant and unpredictable. Bias-resistance ensures that the adversary cannot affect any future beacon value, say, for instance, affect the beacon to win a lottery, and unpredictability ensures that the adversary cannot predict any future beacon value, say, for example, bet on a favorable number in a lottery.

To begin with, we can assume a computing system has access to a sound/unbiased source of randomness, and it may act as a random beacon. However, it becomes a single-point-of-failure in terms of (i) safety or problems with having consistent beacon values or having a bias, (ii) secrecy or problems related to its predictability, and (iii) liveness or availability.

Distributing trust across multiple nodes such that only a minority of those can be compromised allows us to mitigate this single-point-of-failure. Here, a distributed coin-tossing protocol combines randomness from multiple nodes to generate random beacons, which has been explored both in theoretical [25, 12, 11, 30] and systems fronts [49, 45, 24, 8, 44, 33].

A common approach used to design random beacons is to have a set of n parties share a random value and output a random value by combining a subset of these individual values. To share the value, protocols typically require a node

*Contributed equally, listed alphabetically (called a dealer or leader) to use a verifiable secret sharing (VSS) scheme where the node commits to a value with the guarantee that if it is successful, it can be reconstructed by honest nodes even if the dealer is malicious (Byzantine). By utilizing at least $t + 1$ such dealers, where t is the maximum number of compromised parties, we are guaranteed that the reconstructed value is uniformly random since the one honest node whose contribution is included is a uniformly random number. There are several ideal properties that protocols in the literature have aimed to optimize.

Latency and communication complexity. These are two key goals for achieving ideal performance in random beacons – both of them relate to outputting beacon values at a fast rate. In the literature, protocols such as Cachin et al. [11] and Drand [24] output beacon values using a single round of communication and with $O(\kappa n^2)$ communication complexity, where κ is the security parameter. However, they inherently require a threshold sharing [29, 48] as a part of the setup. Consequently, each time a node joins or leaves the protocol, they need to re-build their setup, which is inefficient.

Reconfiguration-friendliness. The ability to easily add or remove existing nodes is called reconfiguration-friendliness. Protocols such as HydRand [45] improved upon Cachin et al. [11] and Drand [24] and were designed to be reconfiguration-friendly while maintaining quadratic communication complexity. However, their resilience to faults, $t < n/3$, is sub-optimal for the synchronous setting.

Resilience to faults. To work under adversarial conditions, we need protocols with high-adversarial resilience. Protocols such as RandRunner [44] employ verifiable delay functions (VDFs) frequently to generate beacons and possibly tolerate a large number of faults. However, VDFs are computationally expensive (analogous to Proof-of-Work) and these protocols cannot offer low latency [23, 44].

Finally, protocols such as RandPiper [8] offer optimal resilience to faults, reconfiguration-friendliness and quadratic communication complexity in the best case (when there are $O(1)$ number of faults). However, they output a beacon value every 11Δ time (where Δ is the synchronous communication delay bound) and are communication-inefficient when the number of faults $t = O(n)$.

Thus, in a nutshell, in an n-node system tolerating t Byzantine faults, a secure random beacon protocol should aim for (i) bias-resistance, (ii) unpredictability, (iii) optimal resilience, (iv) low latency, (v) low communication complexity (or high scalability), and (vi) reconfiguration-friendliness. Unfortunately, all existing works fall short in one or more of these directions. We saw a few examples of these related works earlier; Table 1 presents a summary of many of the others.

1.1 Our Approach and Results

A natural question is whether we can achieve all the above-mentioned desirable properties simultaneously. Our work, OptRand, answers this question affirmatively. In particular, OptRand is a bias-resistant and unpredictable random beacon, with an $O(\kappa n^2)$ communication complexity and tolerating one-half Byzantine faults in a synchronous network. In fact, under optimistic conditions when the number of faults are $< n/4$ and a "leader" is honest, the protocol is responsive, i.e., it advances at the speed of the network, thus achieving low latency. Compared to the state-of-the-art, RandPiper [8], this protocol has better communication complexity (always $O(\kappa n^2)$) and optimistically-responsive latency. In the following, we elaborate on the key features of OptRand and our approach to achieving them.

Towards an always quadratic communication random beacon protocol. An approach to obtaining random values is to combine random secrets from more than t nodes, at least one of whom is honest. In OptRand, every node commits to a random number using a Publicly Verifiable Secret Sharing (PVSS) scheme [13], which, naïvely, requires sharing $O(\kappa n)$-sized information per node with every other node, making the communication complexity at least cubic. Thus, the crux of the challenge is to route this information through a "leader" such that the (i) communication complexity is quadratic while (ii) still allowing nodes to verify the correctness of the shares (iii) even when the number of faults is $t < n/2$. We achieve this using the homomorphic properties of PVSS [13] employing bilinear pairings and attaching a proof that proves that the combined PVSS contains contributions from $t + 1$ nodes.

While the approach of routing the PVSS through the leader has been considered recently [20] for tolerating $t < n/3$ faults, the higher resilience of $t < n/2$ forces us to solve the problem differently. In particular, to deal with Byzantine leaders efficiently without using threshold signatures, similar to RandPiper [8], we take a two-pronged approach: (i) we employ a pipelined state machine replication (SMR) protocol that piggybacks consecutive consensus instances, and

Table 1: **Comparison of related works on Random Beacon protocols.**

Protocol	**Net.**	**Res.**	**Comm. Compl.**		**Unpred.**	**Reusable Setup**	**Resp.**	**Assumption**
			Best	**Worst**				
Cachin et al./Drand [11, 24]	sync	50%	$O(\kappa n^2)$	$O(\kappa n^2)$	1	✗	✗	Threshold Secret/BLS
Dfinity [33, 2]	sync.	50%	$O(\kappa n^2)$	$O(\kappa n^3)^*$	$O(\kappa)$	✗	✗	Threshold Secret/BLS
HERB [18]	sync.	33%	$O(\kappa n^3)$	$O(\kappa n^3)$	1	✗	✗	Threshold ElGamal
RandHerd [49]	sync.	33%	$O(\kappa c^2 \log n)^\dagger$	$O(\kappa n^4)$	$O(\kappa)$	✗	✗	Threshold Schnorr
RandChain [32]	sync.	50%	$O(\kappa n^2)$	$O(\kappa n^2)$	$O(\kappa)$	✗	✗	PoW
RandHound [49]	sync.	33%	$O(\kappa c^2 n)^\P$	$O(\kappa c^2 n^2)^\P$	1	✓	✗	Client Based, PVSS
RandShare [49]	async.	33%	$O(\kappa n^3)$	$O(\kappa n^4)$	1	✓	✓	VSS
RandRunner [44]	sync.	50%	$O(\kappa n^2)$	$O(\kappa n^2)$	$O(\kappa)$	✓	✗	VDF
HydRand [45]	sync.	33%	$O(\kappa n^2)$	$O(\kappa n^3)$	$t+1$	✓	✗	PVSS
SPURT [20]	psync.	33%	$O(\kappa n^2)$	$O(\kappa n^2)$	1	✓	✓	PVSS, Multisig
GULL [14]	sync.	50%¶	$O(\kappa n^3)$	$O(\kappa n^3)$	1	✗	✗	PVSS, DDH
GRandPiper [8]	sync.	50%	$O(\kappa n^2)$	$O(\kappa n^2)$	$t+1$	✓	✗	PVSS, q-SDH
BRandPiper [8]	sync.	50%	$O(\kappa n^2)^\S$	$O(\kappa n^3)$	1	✓	✗	VSS, q-SDH
OptRand	sync.	50%	$O(\kappa n^2)$	$O(\kappa n^2)$	1	✓	✓‡	PVSS, q-SDH

Net. refers to the network assumption. **Res.** refers to the number of Byzantine faults tolerated in the system. **Unpred.** refers to the unpredictability of the random beacon, in terms of the number of future rounds a rushing adversary can predict. **Reusable Setup** refers to a setup that can be reused when a node is replaced in the system. **Resp.** refers to responsiveness, i.e., commit latency is a function of the network speed δ. *probabilistically $O(\kappa n^3)$ when $\Theta(n)$ consecutive leaders are bad. $^\dagger c$ is the average (constant) size of the groups of server nodes. ‡ optimistically responsive during optimistic conditions. §in conditions where the actual number of faults $f = O(1) \leq t$

(ii) we buffer secrets shared using PVSS for each party, and reconstruct the last shared secret/beacon for the Byzantine leader before removing it from subsequent proposals.

Towards optimistically responsive random beacons. While using the previous cryptographic approach and substituting it in RandPiper can offer us always quadratic communication complexity, RandPiper [8] still requires 11Δ latency in each epoch (a duration coordinated by a distinct leader) for beacons – even under optimistic conditions, their beacon protocol cannot progress at the network speed.

The key challenges to obtain responsiveness are (i) efficient propagation of large messages, and (ii) efficient synchronization of all the nodes when some nodes move to the next epoch. RandPiper [8] uses erasure coding and cryptographic accumulators along with waiting for $\Omega(\Delta)$ time to check for possible misbehavior from the current leader to efficiently propagate large messages. In OptRand, we design a new technique to efficiently propagate large messages without checking for misbehavior from the current leader; hence, do not require $\Omega(\Delta)$ wait to ensure propagation.

In synchronous protocols, synchronization refers to all the nodes starting the protocol within Δ of each other. When committing responsively at speeds independent of Δ, the nodes can easily go out-of-sync. Typically, such synchronization between all the nodes is performed by multicasting synchronization proofs to all other nodes [21, 1]; in the absence of threshold signatures, these proofs tend to be $O(n)$-sized, making the communication cubic again. In OptRand, we instead broadcast aggregated secrets opened in a verifiable manner in an epoch to synchronize all the nodes. The size of the random number is $O(\kappa)$ bits and thus, the communication complexity stays quadratic.

In particular, OptRand combines this ability to move responsively to the next epoch with responsive propagation of large messages to obtain an optimistically responsive random beacon. The resulting random beacon protocol can output beacon values responsively whenever more than $3n/4$ nodes and the leader of the epoch are honest, and otherwise, emits the next beacon value every 11Δ time.

Reconfiguration mechanism. We present a reconfiguration mechanism that allows a new node to enter the system with a latency of $t+1$ epochs, which is $2t+2$ epochs (explained later) faster than RandPiper [8]. The added benefit of responsiveness means the $t+1$ epochs can be responsive leading to even faster reconfiguration. A key improvement is in the synchronization mechanism [21, 1] to allow the new node to synchronize with all the existing honest nodes; while prior work required $2t+2$ epochs to perform this, we use a more efficient synchronization mechanism at the end of every epoch to synchronize the new node.

In summary,

✓ We present an efficient random beacon protocol assuming broadcast channels in Section 4.

✓ We present an optimistically responsive random beacon protocol with $O(\kappa n^2)$ communication in Section 5. The resulting protocol is reconfiguration-friendly and can be used as an optimistically responsive BFT SMR protocol.

✓ We present our reconfiguration scheme in Section 6.

2 Related Work

2.1 Related Works in the Random Beacon Literature

There has been a long line of work on distributed public randomness starting from Blum's two-node coin-tossing protocol [9]. Due to its practical application, the problem has been studied under various system models [49, 23, 14, 44, 45, 11, 33, 32, 18, 20]. We review the most recent and closely related works below. Compared to all of these protocols, OptRand has optimal resilience, perfect unpredictability, incurs $O(\kappa n^2)$ communication per beacon output and has a reusable setup. Moreover, OptRand is optimistically responsive i.e., it can make progress at the speed of actual network delay δ during optimistic conditions despite synchrony assumption.

The protocols by Cachin et al. [11], Drand [24] and Dfinity [33] require DKG to setup threshold keys among participating nodes. Although these protocols have optimal resilience, perfect unpredictability, and quadratic communication complexity per beacon output, these protocols do not have reusable setup i.e., replacing a single node in the system involves re-running DKG all over again which blows up communication.

HERB [18] and GULL [14] use partial homomorphic ElGamal encryption scheme to generate random numbers. HERB [18] tolerates only $t < n/3$ failures despite synchrony assumption and uses bulletin boards to post random shares. Mt. Random [14] uses PVSS and threshold ElGamal, protocols from Cachin et al. [11], VRG and assumes bulletin boards to realize their random beacons. Instantiating bulletin boards using Byzantine Consensus primitives trivially incurs $O(\kappa n^3)$. Moreover, both protocols use a variant of threshold setup and thus lack a reusable setup.

RandShare [49] assumes an asynchronous network and requires executing n concurrent instances of Byzantine Agreement with a worst case communication of $O(\kappa n^4)$. RandHerd [49] improves on RandShare by sampling the system into smaller groups of size c resulting in a communication complexity of $O(\kappa c^2 \log n)$ in the common case. RandHound [49] further improves on RandHerd by building tree-based hierarchy among the nodes and executes leader-based Byzantine Consensus among sub-trees. The resulting construction has a communication of $O(\kappa c^2 \log n)$ when all leaders are honest. With a sequence of Byzantine leaders, the communication worsens to $O(\kappa n^3)$.

RandChain [32] has optimal resilience of $t < n/2$, and incurs $O(\kappa n^2)$ per beacon output. However, they use computationally expensive sequential Proof-of-Work, and VDFs along with Nakamoto consensus for consistency and has high computation cost.

RandRunner [44] uses trapdoor Verifiable Delay Functions - VDFs with strong uniqueness properties that produces unique values efficiently for the node that has the trapdoor, but takes time T to produce an output for the nodes that do not have the trapdoor. This allows the beacon to output bias-resistant outputs in every round. While RandRunner has qudratic communication per round, it has worst case unpredictability of $t+1$ rounds.

Most relevant to our protocol are Hydrand [45], RandPiper [8] and SPURT [20]. HydRand [45] tolerates only $t < n/3$ faults despite assuming synchrony. While Hydrand has low computation overhead and a reusable setup due to its use of PVSS scheme, it incurs $O(\kappa n^3)$ communication in the worst case and an unpredictability of $t+1$ rounds in the worst case.

RandPiper [8] improved upon Hydrand by designing a communication efficient BFT SMR protocol. Using the SMR protocol, they obtain a random beacon protocol with optimal resilience, quadratic communication and reusable setup but with worst case unpredictability of $t+1$ rounds. To provide perfect unpredictability, they propose BRand-Piper [8] using VSS scheme to share n secret in each round. The resulting construction has $O(\kappa f n^2)$ communication where $f < t$ is the actual number of faults. However, when $f = \Theta(n)$, the communication is $O(\kappa n^3)$. Moreover, their construction incur large latency to generate random beacons.

Recently, Das et al. proposed SPURT [20] in the partial synchronous model with perfect unpredictability, reusable setup and responsiveness and an always $O(\kappa n^2)$ communication. They use aggregatable PVSS scheme to combine $t + 1$ PVSS vector in each round and provide perfect unpredictability with $O(\kappa n^2)$ communication.

2.2 Related Works in the BFT SMR Literature

There has been a long line of work in improving communication complexity of consensus protocols [35, 26, 1, 50, 4, 38] and round complexity of consensus protocols [22, 1, 7, 26, 28, 35, 42]. We review the most recent and closely related works below. Compared to all of these protocols, our protocol incurs $O(\kappa n^2)$ communication per consensus decision while avoiding the use of threshold signatures. Moreover, our protocol is optimistically responsive with a responsive commit latency of 4δ and synchronous commit latency of $4\Delta + 3\delta$ in common case (or 7Δ in the worst case). Our protocol follows rotating leader paradigm and can change leaders in optimistically responsive manner.

With respect to the communication complexity, the state-of-the-art synchronous BFT SMR protocols [1, 3, 47, 5, 6] incur quadratic communication per consensus decision while using threshold signatures. Without threshold signatures, they incur cubic communication per consensus decision. To the best of our knowledge, the only optimally resilient protocol to achieve $O(\kappa n^2)$ communication without threshold signature is BFT SMR protocol of RandPiper [8]. However, their protocol is not responsive even under optimistic conditions and commits a decision every 11Δ time.

With respect to optimistic responsiveness, protocols due to Thunderella [40] and Sync HotStuff [3] are presented in a back-and-forth slow-path–fast-path paradigm. If started in the wrong path, these protocol cannot commit responsively. Recent work such as PiLi [16], OptSync [47] and Hybrid-BFT [37] achieve simultaneity between responsive and synchronous modes. However, they incur cubic communication without the use of threshold signatures. Ours is the first work that achieves simultaneity under synchrony assumption with $O(\kappa n^2)$ communication while avoiding threshold signatures.

OptSync. OptSync [47] presents an optimistically responsive protocol with optimal 2δ latency during responsive commit and 2Δ synchronous latency. However, their protocol follow stable leader paradigm and incur synchronous delay of 2Δ while changing leaders. They also provide a separate protocol that support changing leaders in optimistically responsive manner in $O(\delta)$ time. Compared to their protocol, our protocol can change leaders responsively only when the new leader has highest ranked certificate; otherwise our protocol incurs 2Δ wait.

Hybrid-BFT. Hybrid-BFT [37] presents an optimistically responsive protocol with both responsive and synchronous commit paths existing simultaneously. They also follow rotating leader paradigm and has responsive commit latency of 2δ and synchronous commit latency of $2\Delta + 2\delta$. Similar to our work, their protocol can also change leaders in responsive manner only when the new leader has highest ranked certificate; otherwise the protocol waits for 2Δ time.

3 Model and Definitions

System and Threat model. We consider a system $\mathcal{P} := \{p_1, \ldots, p_n\}$ consisting of n nodes in a reliable, authenticated all-to-all network, where up to $t < n/2$ nodes can be Byzantine faulty. We assume static corruption and the Byzantine nodes can behave arbitrarily. A node that is not corrupted is considered to be honest and executes the protocol as specified.

Communications between nodes are synchronous. If an honest node p_i sends a message x to another node p_j at time τ, p_j receives the message by time $\tau + \delta$. The delay parameter δ is upper bounded by Δ. The upper bound Δ is known, but δ is unknown to the system. δ can be regarded as an actual delay in the real-world network. We assume all honest nodes have clocks moving at the same speed. They also start executing the protocol within Δ time from each other. This can be easily achieved by using the clock synchronization protocol [1] once at the beginning of the protocol.

We make use of digital signatures and a public-key infrastructure (PKI) to prevent spoofing and replays and to validate messages. Message x sent by a node p_i is digitally signed by p_i's private key and is denoted by $\langle x \rangle_i$. We use $H(x)$ to denote the invocation of the random oracle H on input x.

3.1 Definitions

Pairings. We assume a Type-III pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of prime order q. Let $\mathbb{Z}_q$ be its scalar field. We will use the multiplicative notation of groups for group operations. Let $g_1 \in \mathbb{G}_1$ and $g_2, g_2' \in \mathbb{G}_2$ be independent generators[1].

SMR. We consider a state machine replication protocol defined as follows:

Definition 3.1 (Byzantine Fault-tolerant State Machine Replication [46])**.** *A Byzantine fault-tolerant state machine replication protocol commits client requests as a linearizable log to provide a consistent view of the log akin to a single non-faulty server, providing the following two guarantees:*

1. ***Safety**. Honest nodes do not commit different values at the same log position.*
2. ***Liveness**. Each client request is eventually committed by all honest nodes.*

We use the following definition to capture the security properties of the random beacon.

Random beacon. We consider the following definition of a secure random beacon:

Definition 3.2 (Secure random beacon protocol [20])**.** *A random beacon protocol is secure if for any PPT adversary $\mathcal{A}$ corrupting at most t parties in an epoch, $\mathcal{A}$ has a negligible advantage in the following security game played against a challenger $\mathcal{C}$.*

1. *$\mathcal{C}$ sends the setup parameters of the system.*
2. *$\mathcal{A}$ compromises up to t nodes and notifies $\mathcal{C}$ of these nodes.*
3. *$\mathcal{C}$ creates the remaining public parameters (such as public keys) and sends it to $\mathcal{A}$.*
4. *$\mathcal{A}$ sends the remaining public parameters (such as public keys).*
5. *$\mathcal{C}$ and $\mathcal{A}$ execute the protocol per epoch:*
   - *$\mathcal{C}$ sends messages on behalf of the honest parties to $\mathcal{A}$*
   - *$\mathcal{A}$ decides on the delivery of messages, and sends (or does not send) its messages.*
   - *The above steps are run interactively until an epoch ends and an honest node outputs the protocol transcript* $\mathsf{transcript}$.
6. *$\mathcal{C}$ samples a random bit $b \in \{0, 1\}$ and sends either the beacon output based on* $\mathsf{transcript}$ *or a random $\mathbb{G}_T$ element.*
7. *$\mathcal{A}$ outputs a guess bit b'*

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

3.2 Primitives

In this section, we present primitives used in our protocol.

Linear erasure and error correcting codes. We use standard (n, b) Reed-Solomon (RS) codes [43]. This code encodes b data symbols into code words of n symbols using ENC function and can decode the b elements of code words to recover the original data using DEC function. More details on ENC and DEC are provided in Appendix A.

In our protocol, we instantiate the RS codes with n equal the number of all nodes, and b equal to $\lfloor n/4 \rfloor + 1$.

Cryptographic accumulators. A cryptographic accumulator scheme constructs a value called the accumulator to prove membership of elements using Eval function, and produces a witness for each value in the accumulator using $\mathsf{CreateWit}$ function. Given the accumulation value and a witness, any party can verify if a value is indeed in the set using Verify function. An example accumulator is Merkle trees, where the root is the accumulator, and the paths are witnesses to leaves. In this paper, we employ *collision-free bilinear accumulators* from Nguyen [39] which generates constant-sized witness and accumulators. The bilinear accumulators of Nguyen [39] requires q-SDH assumption. Merkle trees [36] can be used instead, at the expense of $O(\log n)$ multiplicative communication complexity. More details on accumulators are provided in Appendix A.

Discrete log proof of knowledge. Let $g \in \mathbb{G}$ be public generators with $\mathbb{Z}_q$ as the scalar field. Let $u \in \mathbb{G}$ be a public value. A prover P who knows the value s such that $u = g^s$, and wants to prove the knowledge of $s \in \mathbb{Z}_q$

[1]By independent generators, we mean that the adversary controlling t nodes does not know a value $x \in \mathbb{Z}_q$ such that $e(g_1, 1_{\mathbb{G}_2})^x = e(1_{\mathbb{G}_1}, g_2)$ and similarly a value $y \in \mathbb{Z}_q$ such that $g_2^y = g_2'$.

Table 2: **Summary of notation.**

Symbol	Meaning
$\mathcal{A}$	PPT Adversary
b	Number of data shards in RS
C	Linear Error Correcting Code
$\mathcal{C}$	Challenger for cryptographic games
$C^{\perp}$	Dual of the Linear Error Correcting Code space C
c	SCRAPE Ciphertexts/Encryptions
d	SCRAPE decryptions
$\mathcal{D}$	Distinguishing adversary
NIZKPK	Algorithm to prove knowledge of x for g, u such that $g^x = u$.
$\mathsf{Verify}()$	Algorithm that verifies the above relation
δ	The actual network speed
Δ	The worst case maximum network delay
e	Type III Bilinear Pairing function
f	Actual number of faults in the system
$\mathbb{G}_1$	The first pairing group of e
$\mathbb{G}_2$	The second pairing group of e
$\mathbb{G}_T$	The target pairing group of e

Symbol	Meaning
H	The random oracle
λ	The security parameter
κ	The normalized message size
$\langle m \rangle_{p_i}$	A message m along with a signature by node p_i
$\mathsf{negl}()$	A negligible function
$\mathcal{O}_r$	The beacon output for round r
pk	SCRAPE Public Keys
n	Total number of nodes
$\mathcal{P}$	The set of all the nodes in the system
p_i	The ith node of the system
r	Rounds of the Random Beacon Protocol
sk	SCRAPE Secret Keys
q	The prime order of all the pairing groups
t	Maximum number of faults tolerated in the system
τ	Time instants
v	SCRAPE Commitments
$\mathbb{Z}_q$	The scalar field of all the groups of e

runs the algorithm $\pi \leftarrow \mathsf{NIZKPK}(s, g, u)$ to generate a non-interactive zero-knowledge (NIZK) proof π. Using $\{0, 1\} \leftarrow \mathsf{Verify}(\pi, g, u)$ anyone can locally verify the NIZK proof. We will also use the same algorithm to prove knowledge of $s \in \mathbb{Z}_q$ for elements in two groups raised to the same exponent.

We provide a summary of the notations in Table 2.

3.3 SCRAPE PVSS

Setup. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be an efficient pairing group with independent generators $g_1 \in \mathbb{G}_1, g_2, g_2' \in \mathbb{G}_2$.

Public and Private Keys. SCRAPE assumes an additional form of PKI. Every node $p_i \in \mathcal{P}$ has a secret key $sk_i \leftarrow \mathbb{Z}_q$ and public keys $pk_i = g_1^{sk_i}$. We denote all the public keys of all the nodes as follows $\mathbf{pk} := \{pk_1, pk_2, \ldots, pk_n\}$.

Sharing Phase. In the sharing phase, the leader $L \in \mathcal{P}$ chooses a secret $s \in \mathbb{Z}_q$ and creates a polynomial $p(x)$ of degree t with the constraint that $p(0) = s$. Without loss of generality, the shares s_i for node p_i is defined as $p(i)$.

Commitment. In SCRAPE, the leader *commits* to the polynomial $p(x)$ by producing the following vector of elements: $\mathbf{v} = \{g_2^{s_1}, g_2^{s_2}, \ldots, g_2^{s_n}\}$. Here, $\mathbf{v}$ is a *commitment* to $p(x)$ because any node that observes this vector can check that it is indeed a t degree polynomial. Naively, one can try all possible combinations of $t+1$ elements in the commitment vector and check that using Lagrange interpolation they all form the same element g_2^s. SCRAPE [13, Section 2] observes that a (n, t) sharing, i.e., n elements containing an evaluation of a degree t polynomial, corresponds to choosing n elements from a code space $C = \mathbb{Z}_q^n$. This code space has an orthogonal code space $C^{\perp} = \mathbb{Z}_q^n$ such that the following holds true for any t degree polynomial $p(x)$ with high probability: $\prod_{i=1}^{n} p(i) \cdot c_i^{\perp} = 0,\ \{c_1^{\perp}, c_2^{\perp}, \ldots, c_n^{\perp}\} \in C^{\perp}$.

Using this coding check, we can easily ensure that the commitment vector $\mathbf{v}$ is also a *commitment* to a t degree polynomial by checking the following:

$$\prod_{i=1}^{n} v_i^{c_i^{\perp}} = 1_{\mathbb{G}_2},\ \{c_1^{\perp}, c_2^{\perp}, \ldots, c_n^{\perp}\} \in C^{\perp} \tag{1}$$

Let $L \in \mathcal{P}$ be a designated node with secret $s \in \mathbb{Z}_q$.

Sharing Phase

$\mathsf{Share}(s, \mathbf{pk}) \to (\mathbf{v}, \mathbf{c})$

– Build a degree t polynomial $p(x) \in \mathbb{Z}_q[x]$ with $p(0) = s$.

– Compute and output shares $s_i = p(i)$, commitments $v_i = g_2^{s_i}$, and encryptions $c_i = {pk_i}^{s_i}$, for $i \in [n]$.

$\mathsf{PVrfy}(\mathbf{v}, \mathbf{c}, \mathbf{pk}) \to \{0, 1\}$

– Parse $\mathbf{v} := \{v_i\}$, $\mathbf{c} := \{c_i\}$, and $\mathbf{pk} := \{pk_i\}$ for $i \in [n]$.

– Check if $e(c_i, g_2) = e(pk_i, v_i)$ for $i \in [n]$.

– Sample a random code word $\mathbf{y}^\perp \in C^\perp$ and parse it as $\mathbf{y}^\perp := \{y_1^\perp, \ldots, y_n^\perp\}$, and check if $\prod_{i \in [n]} v_i^{y_i^\perp} = 1_{\mathbb{G}_2}$, where $1_{\mathbb{G}_2}$ is the identity element of $\mathbb{G}_2$.

– If both the checks for commitments (Eq. (1)) and encryptions pass, then output 1, otherwise output 0.

Reconstruction Phase

$\mathsf{Dec}(sk_i, c_i) \to d_i$

– Decrypts the share by computing $d_i = c_i^{{sk_i}^{-1}}$

$\mathsf{ShVrfy}(v_j, c_j, d_j) \to \{0, 1\}$

– Check if $e(d_j, g_2) = e(g_1, v_j)$.

$\mathsf{Recon}(\mathbf{S} := \{d_j\}_{j \in [n]}) \to s$

– On receiving $t + 1$ valid shares in $\mathbf{S}$, reconstruct $B = g_1^s$ by computing $B = \prod_{d_j \in \mathbf{S}} d_j^{\ell_j} = \prod_{d_j \in \mathbf{S}} g_1^{p(j) \cdot \ell_j} = g_1^{p(0)} = g_1^s$, where ℓ_j is the j^{th} Lagrange coefficient.

– Reconstruct the secret $S \in \mathbb{G}_T$ as $S = e(B, g_2')$

Figure 1: **Recap of pairing based PVSS** in SCRAPE [13, Section 4]. Note that we only present the algorithms here and not the actual broadcast-channel based protocol. We also present the scheme for asymmetric pairings.

Encryption. Commitments alone are not sufficient to finish sharing, and the leader needs to provide the secret shares to all the nodes. Since this is a PVSS scheme, the public verifiability comes from the encryptions that will be given to all the nodes. The encryption vector is produced as follows and is posted on the broadcast channel: $\mathbf{c} = \{pk_1^{s_1}, pk_2^{s_2}, \ldots, pk_n^{s_n}\}$. Any node that observes the commitment vector $\mathbf{v}$ and the above encryption vector, can check that every node has received the correct encryption $c_i = pk_i^{s_i}$ using the following pairing check for all the nodes $p_i \in \mathcal{P}$: $e(pk_i, v_i) = e(c_i, g_2) = e(g_1, g_2)^{sk_i s_i}$

Reconstruction Phase. In the reconstruction phase, we will reconstruct $g_1^s = g_1^{p(0)}$ and compute $e(g_1^s, g_2')$ as the secret.

Decryption. Every node $p_i \in \mathcal{P}$ provides $g_1^{s_i} = c_i^{sk_i^{-1}}$ since every node knows its secret key sk_i. When a node p_i receives a share d_j from another node p_j, it can verify its correctness with the following check: $e(d_j, g_2) = e(g_1, v_j) = e(g_1, g_2)^{s_j}$.

4 Random Beacons with Broadcasts

In this section, we will describe a (warm up) random beacon protocol with the following security properties:

Definition 4.1 (Warm up Beacon)**.** *Let $L \in \mathcal{P}$ be the designated leader $L \in \mathcal{P}$ with the following properties:*

1. ***Weak Agreement.*** *Let $\mathcal{B}$ be the space of all beacon values. Then all honest nodes output the same value in $\mathcal{B} \cup \perp$.*
2. ***Value Validity.*** *If an honest node outputs $v \neq \perp$, then v is uniformly random in $\mathcal{B}$.*
3. ***Validity.*** *If L is honest, then the value v output by all the honest nodes satisfies $v \neq \perp$.*

We start with a protocol that satisfies Definition 4.1. We allow the honest nodes to output $\perp$ if the Byzantine leader decides to deviate from the protocol. If an honest node outputs a value $v \neq \perp$ then according to *value validity*, it must be the case that this value is uniformly random even if the leader is Byzantine, and from *weak agreement* all other honest nodes also output the same value.

Such a protocol is not yet a complete solution, as it misses two important factors. First, the honest nodes can output $\perp$ when L is Byzantine. Secondly, we also need to implement the broadcast channel using an SMR with the constraint that the communication complexity cannot exceed $O(n^2)$ in an epoch. We will address both of these factors in the next section.

4.1 Our Protocol

Let $L \in \mathcal{P}$ be the leader. Every node $p_i \in \mathcal{P}$ does the following:

Sharing Phase

Commitment

- Sample a random element $r_i \leftarrow \mathbb{Z}_q$ and create $(\mathbf{v}_i, \mathbf{c}_i)$ using SCRAPE $\mathsf{Share}(r_i, \mathbf{pk})$
- Compute knowledge proof $\widetilde{\pi}_i$ using Eq. (2)
- Send $(\mathbf{v}_i, \mathbf{c}_i, \widetilde{\pi}_i)$ to the leader L.

Aggregation

The leader does the following:

- Collect all valid $S_j = (\mathbf{v}_j, \mathbf{c}_j, \widetilde{\pi}_j)$ from all the nodes.
- Build the aggregated commitment vector $\mathbf{v}$ (Eq. (3)), the aggregated encryption vector $\mathbf{c}$ (Eq. (4)), and the aggregate proof vectors $\boldsymbol{\pi}_j$ (Eq. (5)) for all the nodes $p_j \in \mathcal{P}$ for $j \in [n]$, from all the received shares by nodes $p_k \in \mathcal{P}$.
- Post $(\mathbf{v}, \mathbf{c}, \boldsymbol{\pi})$ on the broadcast channel.

Commit

If there is a valid sharing on the broadcast channel, i.e., the $\mathbf{v}$ is a valid (n, t) sharing, $e(v_i, pk_i) = e(g_1, c_i)$ holds, all the constituent $\pi_i \in \boldsymbol{\pi}$ are valid, and finally interpolated values for $\mathbf{v}$ and $\boldsymbol{\pi}$ at $g_1^{p(0)}$ are the same, commit $(\mathbf{v}, \mathbf{c})$.

Reconstruction Phase

- Send d_i and π_i^{Dec} to all the nodes.
- Collect $t+1$ valid shares $(d_j, \pi_j^{\mathsf{Dec}})$ from node $p_j \in \mathcal{P}$.
- On collecting $t+1$ valid shares in the set $\mathbf{S}$, reconstruct the beacon value $S = \mathsf{Recon}(\mathbf{S})$.

Figure 2: **A warm up beacon protocol** using a modified pairing bases SCRAPE PVSS [13] and broadcast-channels.

Our protocol (Fig. 2) satisfies Definition 4.1 and consists of a designated leader $L \in \mathcal{P}$, who acts as the coordinator. When the leader is honest, all the honest nodes obtain a secret share, which when used for reconstruction outputs a unique and unpredictable element $S \in \mathbb{G}_T$. When the leader is Byzantine, either all the honest nodes commit shares for an element g_1^s from which we build a random unpredictable element $S \in \mathbb{G}_T$, or all the honest nodes abort, i.e., output $\perp$.

Decomposition proofs. When using a leader as the coordinator, if we use SCRAPE naively, we need a mechanism to prevent the possibility that a leader can simply cancel the contributions of honest nodes by proposing a PVSS vector where every element is raised to -1. We mitigate this by adding a simple cryptographic proof of knowledge of the shared secret $\widetilde{\pi}$ called *decomposition proofs*. In particular, we make a node prove that it knows the secret (or $p(0)$) using the discrete log proof of knowledge [17] algorithm NIZKPK discussed in Section 3.2. This proves to everyone non-interactively that the proposer knows the secret being shared without revealing it.

Our protocol forces the leader to propose the combined commitments and encryptions and produce at least $t+1$ proofs of knowledge $\widetilde{\pi}$ values. These proofs are $O(1)$ sized making the whole proposal $O(n)$ sized. Since $n - t > t$, we know that there must always be $t+1$ valid shares. So an honest leader will always have sufficient proofs to propose. Therefore, if a leader does not propose, it must be Byzantine. Since any valid proposal from a leader must include $t+1$ proofs of knowledge, at least one of them is a sharing for a random number, we can intuitively guarantee *Value validity* from 2.

Sharing phase. The sharing phase consists of three steps: (i) Commitment, (ii) Aggregation, and (iii) Commit steps. We detail the steps below.

Commitment. In this step, all the nodes generate a random polynomial $p_i(x)$. Then the nodes build commitment vectors $\mathbf{v}_i$ and encryption vectors $\mathbf{c}_i$, and proofs of knowledge $\widetilde{\pi}_i$. The node also signs and sends

$$\widetilde{\pi}_i := \langle g_2^{p_i(0)}, \mathsf{NIZKPK}(p_i(0), g_2, g_2^{p_i(0)}) \rangle_i \tag{2}$$

Finally, all the nodes send their commitments and encryptions to the leader L.
2. Aggregation. WLOG, assume that the leader receives valid $t+1$ PVSS vectors from nodes in $[t+1]$. In this step, the leader selects the valid $t+1$ PVSS vectors, and combines it to produce the final PVSS vector for the polynomial P. This consists of the combined commitments (Eq. (3)), combined encryptions (Eq. (4)) and aggregated proofs (Eq. (5)) for $j \in I$ and for all the nodes $p_i \in \mathcal{P}$.

$$\mathbf{v} = \mathbf{v}_1 + \ldots + \mathbf{v}_{t+1} \tag{3}$$

$$\mathbf{c} = \mathbf{c}_1 + \ldots + \mathbf{c}_{t+1} \tag{4}$$

$$\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_{t+1}\} \tag{5}$$

After combining the PVSS vectors, the leader broadcasts $(\mathbf{v}, \mathbf{c}, \widetilde{\pi})$. Note that this post has a size of $O(n)$.
3. Commit. In the commit step, all the nodes observe the sharing (or its lack thereof within sufficient time) and decide (to commit, or to abort) by checking (i) the $\mathbf{v}$ is a valid (n, t) sharing, (ii) $e(v_i, pk_i) = e(g_1, c_i)$ holds, (iii) all the constituent $\pi_i \in \boldsymbol{\pi}$ are valid, and finally (iv) the interpolated values for $\mathbf{v}$ and $\boldsymbol{\pi}$ at $g_1^{p(0)}$ are the same. If the leader was honest, then all the honest nodes commit. If the leader was Byzantine, the sharing will still include at least one honest node's contribution and if committed by all the honest nodes, is secure.

Reconstruction phase. In this phase, the nodes decrypt their shares by unmasking the encrypting share, i.e., by removing the secret key sk_i from c_i. The share value d_i is then sent to all the nodes who can non-interactively verify the validity. On collecting $t+1$ valid decryptions, the nodes can reconstruct $B = g_1^s$ using Lagrange interpolation from which the final secret $S = e(B, g_2')$ is derived.

Verifiable Beacon. The pairing-based PVSS scheme allows for any party to verify the correctness of the reconstructed unpredictable secret. In particular, on reconstructing $B = g_1^s$, any node can confirm that this is the correct beacon value against a sharing, by checking $e(g_1^s, g_2) = e(g_1, g_2^s)$, where g_2^s can be generated using Lagrange interpolation on $\mathbf{v}$. This property is critical to obtain *optimistic responsiveness* in the next section.

Security Analysis. We defer the security analysis to Appendix B and Appendix C.

Note that the protocol described in Fig. 2, can result in nodes aborting the beacon generation when the leader is Byzantine. This can make the protocol violate guaranteed output delivery. In the next section, we overcome the problem while maintaining quadratic communication complexity using pipelined SMRs [50, 3] and pre-processing.

4.2 Crowd-sourcing Randomness

Existing distributed random beacon protocols and implementations [11, 24, 20, 45, 8] rely on the nodes for the quality of the random beacons that they produce. The 30-odd year-old computer security history filed with numerous examples of badly generated local randomness, and it can be nice if we do not entirely rely on the nodes or the random oracle assumption for unbiased output.

In that direction, our decomposition proofs can be used to crowd-sourcing the input randomness. A NIST client can submit a sharing to any node in the system, and the node will provide a decomposition proof proving that its share is included in its beacons. Now all applications that trust only NIST's source of randomness can select blocks which include sharings from NIST only. This feature is made possible with the introduction of decomposition proofs. This can be recursively used to form large trees of sharing where in every level, the node proves to its children that its share was included.

5 Optimistically Responsive Random Beacon

In this section, we present OptRand, an optimistically responsive random beacon protocol. Our protocol is a novel combination of a state machine replication (SMR) protocol and a random beacon protocol to achieve an optimistically responsive random beacon. Our protocol uses the generated random beacons to achieve responsiveness. In particular, we use aggregated secrets to synchronize between honest nodes and achieve responsiveness.

The underlying SMR protocol includes an optimistic path that can make progress at the network speed i.e., in $O(\delta)$ time during optimistic condition when the leader and $> 3n/4$ nodes behave honestly. Under standard conditions, i.e., when only $> n/2$ nodes behave honestly, the SMR protocol makes progress in $\Omega(\Delta)$ time. We follow the optimistic responsive paradigm of OptSync [47], i.e., our protocol does not require explicit back-and-forth switching between slow synchronous mode and fast optimistic mode employed in [40, 3]. Similar to the optimistically responsive view-change protocol in OptSync, our protocol changes leaders in an optimistically responsive manner.

Epochs. Our protocol progresses through a series of numbered *epochs* with epoch r coordinated by a distinct leader L_r rotated in a round-robin manner every epoch. During optimistic conditions, the system progresses through epochs responsively, i.e., in $O(\delta)$ time; otherwise each epoch lasts for $O(\Delta)$ time.

Blocks and block format. We represent a proposal in an epoch in the form of a *block*. Each block references its predecessor to form a blockchain with the exception of the genesis block which has no predecessor. We call a block's position in the blockchain as its height. A block B_h at height h has the format, $B_h := (b_h, H(B_{h-1}))$ where b_h denotes the proposed payload at height h and $H(B_{h-1})$ is the hash digest of B_{h-1}. The predecessor for the genesis block is $\perp$. In our protocol, the payload b_h is set to the aggregated PVSS commitment and encryption. A block B_h is said to be *valid* if (1) its predecessor block is valid, or if $h = 1$, predecessor is $\perp$, and (2) the payload in the block is a valid PVSS vector, i.e., the *verification* algorithm outputs a 1 (discussed in Vote step in Fig. 2). A block B_h *extends* a block B_l ($h \geq l$) if B_l is an ancestor of B_h.

Certified blocks, and locked blocks. A *block certificate* represents a set of signatures on a block in an epoch by a quorum of nodes. We use two types of signed vote messages: a responsive vote resp-vote and a synchronous vote sync-vote. Accordingly, we consider two *types* of block certificates. A *responsive certificate* $\mathcal{C}_r^{3/4}(B_h)$ for a block B_h consists of $\lfloor 3n/4 \rfloor + 1$ distinct resp-vote on B_h in epoch r. Similarly, a *synchronous certificate* $\mathcal{C}_r^{1/2}(B_h)$ consists of $t + 1$ distinct sync-vote on B_h in epoch r. Whenever the distinction is not important, we will represent the certificates by $\mathcal{C}_r(B_h)$.

Certified blocks are ranked by epochs, i.e., blocks certified in a higher epoch have a higher rank. We do not rank between responsive and synchronous certificate from the same epoch. During the protocol execution, each node keeps track of all certified blocks and keeps updating the highest ranked certified block to its knowledge. Nodes will lock on highest ranked certified blocks and do not vote for blocks that do not extend highest ranked block certificates to ensure safety of a commit.

Equivocation. Two or more messages of the same *type* but with different payload sent by an epoch leader is considered an equivocation. In this protocol, the leader of an epoch e sends propose, resp-cert, and sync-cert messages (explained later) to all other nodes. In order to facilitate efficient equivocation checks, the leader sends the payload along with signed hash of the payload. When an equivocation is detected, broadcasting the signed hash suffices to prove equivocation by L_r.

Background: Dissecting BFT SMR protocol of RandPiper [8]. A key component of RandPiper is a communication efficient BFT SMR protocol that incurs $O(\kappa n^2)$ communication per decision to decide on $O(n)$-sized input without using threshold signatures. The efficient communication was achieved by making use of erasure coding schemes, cryptographic accumulators and broadcast of equivocating hashes (if any). In their protocol, they use $(n, t + 1)$ RS codes to encode large messages. When a node receives a valid proposal from the leader, they use RS codes to encode the proposal into n code words $(s_1, \ldots, s_n)$ and compute corresponding cryptographic witnesses $(w_1, \ldots, w_n)$, and send each code word and witness pair (s_i, w_i) to node p_i $\forall p_i \in \mathcal{P}$. A node votes for the proposed block only if it does not detect any equivocation for 2Δ time. The 2Δ wait before voting ensures (i) no honest node received an equivocating proposal and conflicting (s_i', w_i') before receiving (s_i, w_i) (ii) all honest nodes receive at least $t + 1$ code words for the proposed block sufficient to reconstruct the proposal.

To ensure safety of a committed block, in general, SMR protocols ensure that all honest nodes receive and lock a certificate for the proposed block. A certificate consisting of $t + 1$ signatures for the proposed block is linear in size in the absence of threshold signatures. Thus, an all-to-all broadcast of the certificate trivially incurs cubic communication. The BFT SMR protocol of RandPiper solves the issue using following technique. First, nodes send their vote only to

$\mathsf{Deliver}(\mathsf{mtype}, m, z_r, r)$:

1. Partition input m into $\lfloor n/4 \rfloor + 1$ data symbols. Encode the $\lfloor n/4 \rfloor + 1$ data symbols into n code words $(s_1, \ldots, s_n)$ using ENC function. Compute witness $w_j \; \forall s_j \in (s_1, \ldots, s_n)$ using CreateWit function. Send $\langle \mathsf{codeword}, \mathsf{mtype}, s_j, w_j, z_r, r \rangle_{p_i}$ to node $j \; \forall j \in [n]$.
2. If j^{th} node receives the first valid code word $\langle \mathsf{codeword}, \mathsf{mtype}, s_j, w_j, z_r, r \rangle$ for the accumulator z_r, forward the code word to all the nodes.
3. Upon receiving $\lfloor n/4 \rfloor + 1$ valid code words for a common accumulator z_r, decode m using DEC function.

Figure 3: **Deliver function**

the leader. The leader is expected to collect $t+1$ votes, form a single certificate and send it to all nodes. Second, in order to ensure the certificate is propagated among all honest nodes, instead of broadcasting it to all nodes, they use RS codes to encode the certificate, send the code word and witnesses and wait for 2Δ to check for an equivocation before making a commit.

Achieving optimistic responsiveness. The techniques employed by the BFT SMR protocol enables communication efficient consensus on $O(n)$-sized input. However, their technique requires waiting for $\Omega(\Delta)$ time to detect equivocation before making a decision.

In this paper, we propose a new technique that allows us to responsively make decision and change leaders without relying on equivocation detection. We modify RandPiper in the following manner: First, we use $(n, \lfloor n/4 \rfloor + 1)$ RS codes to encode large messages (in the Deliver primitive in Fig. 3). This allows decoding with $\lfloor n/4 \rfloor + 1$ code words at the expense of doubled code word size. Second, a node sends a responsive vote to the leader as soon as it receives a valid block proposal. The node also sends the RS coded code words and witnesses to all other nodes. The leader collects $\lfloor 3n/4 \rfloor + 1$ votes to form a responsive certificate and sends the responsive certificate to all nodes. The nodes broadcast an ack message in response to the responsive certificate and commit on receiving $> 3n/4$ distinct ack messages. In addition, they also send RS coded code words and witnesses for the responsive certificate. The existence of $> 3n/4$ ack messages ensures that all honest parties can decode the proposed blocks and the responsive certificate. In particular, at least $\lfloor n/4 \rfloor + 1$ honest nodes must have received the block proposal and the responsive certificate for the committed block and they have forwarded their code words to all nodes. Thus, all honest node must receive $\lfloor n/4 \rfloor + 1$ code words sufficient to decode the proposed blocks and the responsive certificate.

Responsively changing epochs. The above technique allows an honest node to responsively commit a decision. In order to responsively change epochs, a synchronization primitive is required to signal all honest nodes to move to a higher epoch. Prior works [3, 47, 5] perform an all-to-all broadcast of certificates to synchronize between epochs which incurs cubic communication without threshold signatures. In this protocol, we broadcast aggregated secret opened in an epoch to synchronize all the nodes. The size of aggregated secret is $O(1)$ bits and all-to-all broadcast of $O(1)$-sized aggregated secret does not blow up communication.

In cases when optimistic conditions are not met, the underlying consensus mechanism works similar to the BFT SMR in RandPiper except we use $(n, \lfloor n/4 \rfloor + 1)$ RS codes.

5.1 Protocol Details

Deliver function. We first present a Deliver function (refer Fig. 3) that is used by an honest node to propagate long messages received from the epoch leader. This function is similar to that in RandPiper [8] except that we use $(n, \lfloor n/4 \rfloor + 1)$ RS codes instead of $(n, t+1)$ RS codes used in [8]. As a result, the size of code word is doubled and the communication is increased by a factor of 2. However, this does not linearly blow up the communication complexity and the communication complexity still remains $O(\kappa n^2)$ (more details in Lemma 11).

Our beacon protocol is described in Fig. 4. Nodes maintain a chain of blocks to add blocks proposed by leaders, a queue $\mathcal{Q}()$ to store a recently committed PVSS vector proposed by an epoch leader and set $\mathcal{P}_r$ to keep track of removed nodes. Before the start of the beacon protocol execution, a setup phase is executed where we establish PVSS parameters (namely $g_1 \in \mathbb{G}_1$, $g_2, h_2 \in \mathbb{G}_2$), and public keys pk_i for every node $p_i \in \mathcal{P}$. We also buffer one secret

Let r be the current epoch, L_r be the leader of epoch r and $\mathcal{P}_r$ be the set of removed nodes. For each epoch r, node $p_i \in \mathcal{P}$ performs following operations:

1. **Epoch advancement.** Node p_i advances to epoch r using following rules:
   (a) When $\mathsf{epoch\text{-}timer}_{r-1}$ reaches 0, enter epoch r.
   (b) On receiving aggregated secret R_{r-1}, broadcast R_{r-1}. Wait until $\mathcal{C}_{r-1}(B_l)$ is received and enter epoch r.

   Upon entering epoch r, send PVSS tuple $(\mathbf{v}_i, \mathbf{c}_i, \widetilde{\pi}_{K,i})$ and highest ranked certificate $\mathcal{C}_{r'}(B_l)$ to L_r. Set $\mathsf{epoch\text{-}timer}_r$ to 11Δ and start counting down.
2. **Propose.** Wait for $t+1$ PVSS tuples and either $\mathcal{C}_{r-1}(B_l)$ or 2Δ time after entering epoch r. Upon receiving $t+1$ valid PVSS tuples, L_r aggregates them to obtain $(\mathbf{v}, \mathbf{c}, \widetilde{\pi}_K)$ (refer Aggregation Step in Fig. 2). Set $b_h := (\mathbf{v}, \mathbf{c}, \widetilde{\pi}_K)$ and send $\langle \mathsf{propose}, B_h, \mathcal{C}_{r'}(B_l), z_{pa}, r\rangle_{L_r}$ to node p_j $\forall p_j \in \mathcal{P}$ where B_h extends B_l and $\mathcal{C}_{r'}(B_l)$ is the highest ranked certificate known to L_r.
3. **Vote.** If $\mathsf{epoch\text{-}timer}_r \geq 7\Delta$ and node p_i receives the first proposal $p_r := \langle \mathsf{propose}, B_h, \mathcal{C}_{r'}(B_l), z_{pa}, r\rangle_{L_r}$, check the validity of the aggregated PVSS tuple (refer Commit Step in Fig. 2). If valid and B_h extends a highest ranked certificate, invoke $\mathsf{Deliver}(\mathsf{propose}, p_r, z_{pa}, r)$ and send $\langle \mathsf{resp\text{-}vote}, H(B_h), r\rangle_{p_i}$ to L_r. Set $\mathsf{vote\text{-}timer}_r$ to 2Δ and start counting down. When $\mathsf{vote\text{-}timer}_r$ reaches 0, send $\langle \mathsf{sync\text{-}vote}, H(B_h), r\rangle_{p_i}$ to L_r.
4. **Resp cert.** On receiving $\lfloor 3n/4 \rfloor + 1$ $\mathsf{resp\text{-}vote}$ for B_h, L_r broadcasts $\langle \mathsf{resp\text{-}cert}, \mathcal{C}_r^{3/4}(B_h), z_{ra}, r\rangle_{L_r}$.
5. **Sync cert.** On receiving $t+1$ $\mathsf{sync\text{-}vote}$ for B_h, L_r broadcasts $\langle \mathsf{sync\text{-}cert}, \mathcal{C}_r^{1/2}(B_h), z_{sa}, r\rangle_{L_r}$.
6. **Ack.** Upon receiving the first responsive certificate $rc := \langle \mathsf{resp\text{-}cert}, \mathcal{C}_r^{3/4}(B_h), z_{ra}, r\rangle_{L_r}$, invoke $\mathsf{Deliver}(\mathsf{resp\text{-}cert}, rc, z_{ra}, r)$ and broadcast $\langle \mathsf{ack}, H(B_h), z_{ra}, r\rangle_{p_i}$.
7. **Commit.** Node p_i commits using one of the following rules:
   (a) *Responsive.* If $\mathsf{epoch\text{-}timer}_r \geq 2\Delta$ and node p_i receives $\langle \mathsf{ack}, H(B_h), z_{ra}, r\rangle$ from $\lfloor 3n/4 \rfloor + 1$ distinct nodes and detects no equivocation, commit B_h and all its ancestors.
   (b) *Synchronous.* If $\mathsf{epoch\text{-}timer}_r \geq 3\Delta$ and node p_i receives the first certificate (either responsive or synchronous), set $\mathsf{commit\text{-}timer}_r$ to 2Δ and start counting down. If the received certificate is synchronous i.e., $sc := \langle \mathsf{sync\text{-}cert}, \mathcal{C}_r^{1/2}(B_h), z_{sa}, r\rangle_{L_e}$, invoke $\mathsf{Deliver}(\mathsf{sync\text{-}cert}, sc, z_{sa}, r)$. When $\mathsf{commit\text{-}timer}_r$ reaches 0, if no epoch-r equivocation has been detected, commit B_h and all its ancestors.
8. **Update, reconstruct and output.** When node p_i commits or when epoch r ends, perform following operations:
   (a) Commit block B_ℓ proposed in epoch $r-t$ if the highest ranked chain extends B_ℓ (if B_ℓ has not been committed).
   (b) If block B_ℓ proposed by L_{r-t} has been committed by epoch r, update $\mathcal{Q}(L_{r-t})$ with $(\mathbf{v}, \mathbf{c}, \widetilde{\pi}_K)$ shared in b_ℓ. Otherwise, remove L_{r-t} from future proposals, i.e., $\mathcal{P}_r \leftarrow \mathcal{P}_r \cup \{L_{r-t}\}$.
   (c) Obtain $(\mathbf{v}, \mathbf{c}, \widetilde{\pi}_K)$ corresponding to block committed in $\mathtt{Dequeue}(\mathcal{Q}(L_r))$. Broadcast decrypted share d_i. On receiving share d_j from another node p_j, ensure that $\mathsf{ShVrfy}(pk_j, c_j, d_j) = 1$. On receiving $t+1$ valid shares in $\mathbf{S}$, reconstruct B and $R_r \leftarrow \mathsf{Recon}(\mathbf{S})$. Broadcast (B, R_r). On receiving (B, R_r) from others, accept R_r if $R_r = e(B, h_2)$ and $e(B, g_2) = e(g_1, g_2^s)$.
   (d) Compute and output $\mathcal{O}_r \leftarrow H(R_r)$.
9. **(Non-blocking) Equivocation.** Broadcast equivocating hashes signed by L_r and stop performing epoch r operations, except Step 8. If $\mathsf{epoch\text{-}timer}_r > 2\Delta$, reset $\mathsf{epoch\text{-}timer}_r$ to 2Δ and start counting down.

Figure 4: Optimistically responsive random beacon protocol with $O(\kappa n^2)$ bits communication per epoch.

share for aggregated PVSS tuples for every node p_i, i.e., fill $\mathcal{Q}(p_i)$ for $p_i \in \mathcal{P}$. The nodes in $\mathcal{P} \setminus \mathcal{P}_r$ are selected as leaders in a round-robin manner.

After the setup phase, the nodes execute following steps in each epoch r.

Epoch advancement. Each node keeps track of epoch duration $\mathsf{epoch\text{-}timer}_r$ for epoch r. A node p_i enters epoch r (i) when its $\mathsf{epoch\text{-}timer}_{r-1}$ expires, or (ii) when it receives a round $r-1$ aggregated secret R_{r-1} and a round $r-1$ block certificate $\mathcal{C}_{r-1}(B_l)$. Upon entering epoch r, node p_i generates PVSS vector $(\mathbf{v}_i, \mathbf{c}_i, \widetilde{\pi}_{K,i})$ (defined in *Commitment phase* of Fig. 2) and sends the PVSS tuple and its highest ranked certificate to the leader L_r. In addition, it aborts all timers below epoch r and sets $\mathsf{epoch\text{-}timer}_r$ to 11Δ and starts counting down.

Propose. Upon entering epoch r, if Leader L_r has $\mathcal{C}_{r-1}(B_l)$, it proposes as soon as it receives $t+1$ PVSS tuples; otherwise, it waits for 2Δ time to ensure it can receive the highest ranked certificate from all honest nodes. Upon receiving $t+1$ PVSS tuples from $I \subset [n]$, it aggregates the PVSS tuples to obtain aggregated PVSS committments $\mathbf{v}$, aggregated encrypted secret shares $\mathbf{c}$ and NIZK proofs $\{\widetilde{\pi}_{K,i}\}_{i \in I}$ denoted as $\widetilde{\pi}_K$. The leader L_r constructs a block B_h by extending on the highest ranked certificate $\mathcal{C}_{r'}(B_l)$ known to L_r with payload b_h set to $(\mathbf{v}, \mathbf{c}, \widetilde{\pi}_K)$ and sends proposal $p_r := \langle \mathsf{propose}, B_h, \mathcal{C}_{r'}(B_l), z_{pa}, r\rangle_{L_r}$ to node p_j, $\forall p_j \in \mathcal{P}$. Here, z_{pa} is the accumulation value for the pair $(B_h, \mathcal{C}_{r'}(B_l))$. The proposal for B_h is common to all nodes. While conceptually, the leader is sending $\langle \mathsf{propose}, B_h, \mathcal{C}_{r'}(B_l), z_{pa}, r\rangle_{L_r}$, to facilitate equivocation checks it instead sends $\langle \mathsf{propose}, H(B_h, \mathcal{C}_{r'}(B_l)), z_{pa}, r\rangle_{L_r}$ with B_h and $\mathcal{C}_{r'}(B_l)$ sent separately. The size of the signed message is $O(1)$ and hence can be broadcast during equivocation or while delivering proposal p_r without incurring cubic communication overhead.

Vote. If node p_i receives a proposal $p_r := \langle \mathsf{propose}, B_h, \mathcal{C}_{r'}(B_l), z_{pa}, r\rangle_{L_r}$ it first checks PVSS verification for $(\mathbf{v}, \mathbf{c}, \widetilde{\pi}_K)$ is valid (refer Commit step in Fig. 2). We call such a proposal valid. If node p_i receives the valid proposal and the proposed block B_h extends the highest ranked certificate known to the node such that its $\mathsf{epoch\text{-}timer}_r \geq 7\Delta$, then it invokes $\mathsf{Deliver}(\mathsf{propose}, p_r, z_{pa}, r)$ and sends a responsive vote $\langle \mathsf{resp\text{-}vote}, H(B_h), r\rangle_{p_i}$ immediately to L_r. In addition, the node sets its $\mathsf{vote\text{-}timer}_r$ to 2Δ and starts counting down. When $\mathsf{vote\text{-}timer}_r$ reaches 0 and detects no epoch r equivocation, the node sends a synchronous vote $\langle \mathsf{sync\text{-}vote}, H(B_h), r\rangle_{p_i}$ to L_r. If block B_h does not extend the highest ranked certificate known to the node or receives proposal p_r when its $\mathsf{epoch\text{-}timer}_r < 7\Delta$, the node simply ignores the proposal and does not vote for B_h.

Resp cert. When the leader L_r receives $\lfloor 3n/4 \rfloor + 1$ distinct $\mathsf{resp\text{-}vote}$ messages for the proposed block B_h in epoch r, denoted by $\mathcal{C}_r^{3/4}(B_h)$, L_r broadcasts $\langle \mathsf{resp\text{-}cert}, \mathcal{C}_r^{3/4}(B_h), z_{ra}, r\rangle_{L_r}$ to all nodes where z_{ra} is the accumulation value of $\mathcal{C}_r^{3/4}(B_h)$. Similar to the proposal, the hash of the certificate $\mathcal{C}_r^{3/4}(B_h)$ is signed to allow for efficient equivocation checks. Since our protocol requires the certificate to be *delivered* to all parties in case of a commit, we require two different certificates for the same block shared by a leader to be considered an equivocation.

Sync cert. When leader L_r receives $t+1$ distinct $\mathsf{sync\text{-}vote}$ messages for the proposed block B_h in epoch r, denoted by $\mathcal{C}_r^{1/2}(B_h)$, L_r broadcasts $\langle \mathsf{sync\text{-}cert}, \mathcal{C}_r^{1/2}(B_h), z_{ra}, r\rangle_{L_r}$ to all nodes where z_{ra} is the accumulation value of $\mathcal{C}_r^{1/2}(B_h)$. Again, the hash of the certificate $\mathcal{C}_r^{1/2}(B_h)$ is signed to allow for efficient equivocation checks.

Ack. When a node p_i receives a responsive certificate $rc := \langle \mathsf{resp\text{-}cert}, \mathcal{C}_r^{3/4}(B_h), z_{ra}, r\rangle_{L_r}$ while in epoch r, it invokes $\mathsf{Deliver}(\mathsf{resp\text{-}cert}, rc, z_{ra}, r)$ to deliver rc and broadcasts $\langle \mathsf{ack}, H(B_h), z_{ra}, r\rangle_{p_i}$ to all nodes. If $\mathsf{epoch\text{-}timer}_r \leq 3\Delta$, node p_i sets $\mathsf{commit\text{-}timer}_r$ to 2Δ and starts counting down.

Commit. The protocol includes two commit rules that commits proposals made in the same epoch and an additional commit rule that commits proposals made $t+1$ epochs earlier. A replica commits using the rule that is triggered first. In responsive commit, a replica commits block B_h and all its ancestors immediately when it receives at least $\lfloor 3n/4 \rfloor + 1$ ack messages for a responsive certificate $\mathcal{C}_r^{3/4}(B_h)$ with a common accumulation value z_{ra} such that its $\mathsf{epoch\text{-}timer}_r$ is large enough (2Δ). Note that a responsive commit happens at the actual speed of the network (δ).

In synchronous commit, when node p_i receives the valid epoch r certificate when its $\mathsf{epoch\text{-}timer}_r$ is large enough (3Δ), it sets $\mathsf{commit\text{-}timer}_r$ to 2Δ and starts counting down. If the received certificate is synchronous i.e., $sc := \langle \mathsf{sync\text{-}cert}, \mathcal{C}_r^{1/2}(B_h), z_{sa}, r\rangle_{L_r}$, it invokes $\mathsf{Deliver}(\mathsf{sync\text{-}cert}, sc, z_{sa}, r)$ and sets $\mathsf{commit\text{-}timer}_r$ to 2Δ. When $\mathsf{commit\text{-}timer}_r$ reaches 0, if no epoch-r equivocation has been detected, node p_i commits B_h and all its ancestors. Invoking $\mathsf{Deliver}()$ on the $\mathsf{sync\text{-}cert}$ ensures that all honest nodes have received $\mathcal{C}_r(B_h)$ before quitting epoch r.

In addition to above commit rules, we include an additional commit rule. We consider a block B_ℓ proposed in epoch $r-t$ proposed by L_{r-t} committed if the highest ranked chain at the end of epoch r extends B_ℓ even though none of the blocks that extends B_ℓ proposed after epoch $r-t$ have been committed using either of the above commit rules. This commit rule helps in committing safe blocks possibly uncommitted due to responsively moving to higher epoch.

We note that if an honest node commits a block B_h in epoch r using one of the commit rules, it is not necessary that all honest nodes commit B_h in epoch r using the same rule, or commit B_h at all. Depending on how Byzantine nodes behave, only some honest nodes may receive $> 3n/4$ ack messages and commit using responsive commit rule while some other honest nodes may commit using synchronous commit rule. It is also possible that only some honest node commits B_h while no commit rules are triggered for rest of the honest nodes. For example, an honest node commits a block B_h responsively but all other nodes detect equivocation in the epoch. In such a case, we ensure that

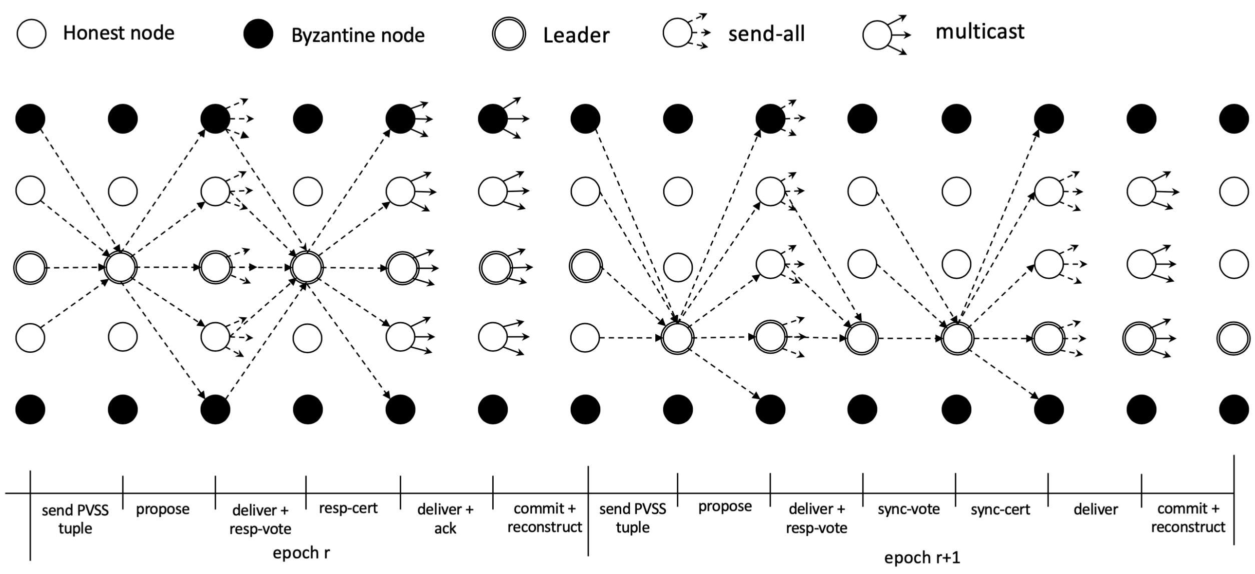

Figure 5: An example execution of two epochs of OptRand. Here, epoch r is responsive and epoch $r+1$ is synchronous. *send-all* implies a node sending different messages to different parties. This differs from multicast as multicast involves sending same message to all nodes.

all honest nodes receive and lock on a certificate for B_h, i.e., $\mathcal{C}_r(B_h)$, to ensure safety of a commit. Eventually after $t+1$ epochs, all honest nodes will commit B_h using our third commit rule.

Equivocation. At any time in epoch r, if a node p_i detects an equivocation, it broadcasts equivocating hashes signed by leader L_r. Node p_i also stops performing epoch r operations except update, reconstruct and output steps described below. In addition, if $\mathsf{epoch\text{-}timer}_r > 2\Delta$, node p_i resets $\mathsf{epoch\text{-}timer}_r$ to 2Δ to assist in terminating a faulty epoch faster.

Update. The update step ensures that the leaders failing to commit a block in $t+1$ epochs are removed the active set of nodes, i.e., if the leader L_{r-t} of epoch $r-t$ fails to add a new block by the end of epoch r, L_{r-t} is removed from future proposals, i.e., $\mathcal{P}_r \leftarrow \mathcal{P}_r \cup \{L_{r-t}\}$. On the other hand, if block B_ℓ proposed by L_{r-t-1} has been committed by epoch r, update $\mathcal{Q}(L_{r-t})$ with $(\mathbf{v}, \mathbf{c}, \widetilde{\pi}_K)$ shared in b_ℓ. This step ensures that our protocol produces a random beacon in each epoch.

Reconstruct and output. Node p_i starts to reconstruct aggregated secret R_r when node p_i commits or when its $\mathsf{epoch\text{-}timer}_r$ expires. It obtains $(\mathbf{v}, \mathbf{c}, \widetilde{\pi}_K)$ corresponding to block committed in $\mathtt{Dequeue}(\mathcal{Q}(L_r))$ and decrypts the share by computing $d_i = c_i^{sk_i^{-1}}$. It then broadcasts d_i to all other nodes. On receiving share d_j from another node p_j, it verifies it using $\mathsf{ShVrfy}(g_1, g_2, e_j, d_j)$. On receiving a set $\mathbf{S}$ of $t{+}1$ valid shares, it reconstructs $R_r \leftarrow \mathsf{Recon}(g_1, g_2, \mathbf{S})$. In addition, it also broadcasts the aggregated secret R_r. Any node can verify the correctness of beacon value without reconstruction by checking $e(g_1, R_r) = e(g_1^s, g_2)$. An epoch r beacon output $\mathcal{O}_r$ is the hash of the aggregated secret R_r, i.e., $\mathcal{O}_r \leftarrow H(R_r)$.

Observe that the size of aggregated secret R_r is $O(1)$ and all-to-all broadcast of the aggregated secret does not blow up communication. Moreover, the aggregated secret R_r cannot be reconstructed without an honest node sending its secret share. Thus, we use the aggregated secret R_r to synchronize all other nodes and responsively change epochs.

Latency and communication complexity. When the epoch leader is Byzantine, not all honest nodes may be locked on a certificate for a common block at the end of the epoch. When the epoch leader is honest, at least one honest node commits block B_h proposed by an honest epoch leader and all honest nodes lock on a certificate for common block B_h and do not act on block proposals that do not extend B_h afterwards. Thus, block B_h and all its ancestors are finalized in an honest epoch. Due to round-robin leader selection, there will be at least one honest leader every $t+1$ epochs and all honest nodes finalize on common blocks up to the honest epoch. Thus, our protocol has a commit latency of $t+1$

epochs. Our protocol has communication cost of $O((\kappa + w)n^2)$ bits per epoch.
Why is it safe to commit a block B_ℓ proposed $t+1$ epochs earlier if the highest ranked chain extends B_ℓ? The round robin leader selection policy ensures that there will be at least one honest leader in last $t+1$ epochs. An honest epoch leader L_r ensures it extends the highest ranked block certificate from all honest nodes. Our protocol ensures that the block B_h proposed by the leader L_r is committed by at least one honest node in epoch r and all honest nodes receive and lock on a certificate for block B_h. Thus, no honest node acts on the future block proposals that do not extend B_h and the highest ranked chain after epoch r always extends B_h, and all its ancestors. This concludes that if block B_ℓ proposed $t+1$ epochs earlier is extended by the highest ranked chain, there will never be an equivocating chain that does not extend B_ℓ and it is safe commit a block B_ℓ.

Optimistically responsive BFT SMR for free. While our protocol designs an optimistically responsive random beacon protocol, the same protocol can be used as optimistically responsive rotating leader BFT SMR protocol by simply adding additional payload that meets application level validity conditions to b_h. Rotating leader protocols provide better *fairness* and *censorship resistance* compared to stable leader protocols [3, 47]. Compared to prior optimistically responsive schemes, our BFT SMR protocol has a communication cost of $O(\kappa n^2)$ *without* using threshold signatures.

5.2 Security Analysis

We say a block B_h is committed directly in epoch r if an honest node successfully runs one of the following commit rules (i) responsive commit rule (Step 7a), or (ii) synchronous commit rule (Step 7b) in Fig. 4. We say a block B_h is committed indirectly if it is a result of directly committing a block B_ℓ ($\ell > h$) that extends B_h.

Fact 1. *If an honest node sends* sync-vote *for block B_h in epoch r, then no equivocating block certificate exists in epoch r.*

Proof. Suppose an honest node p_i sends a sync-vote for block B_h in epoch r at time τ. Node p_i must have invoked $\mathsf{Deliver}(\mathsf{propose}, p_r, z_{pa}, r)$ to deliver proposal p_r for B_h at time $\tau - 2\Delta$ and did not detect an epoch r equivocation by time τ. Observe that no honest node invoked Deliver and sent resp-vote nor sync-vote for equivocating block proposals before time $\tau - \Delta$; otherwise node p_i must have received a code word for equivocating proposal i.e., an epoch r equivocation by time τ. In addition, all honest nodes receive their code word for proposal p_r by time $\tau - \Delta$ and will neither send resp-vote nor sync-vote for equivocating block proposals after time $\tau - \Delta$. Thus, no equivocating block certificate exists in epoch r. □

Fact 2. *If a responsive certificate for block B_h exists in epoch r, then no equivocating block certificate exists in epoch r.*

Proof. A responsive certificate for block B_h in epoch r, i.e., $\mathcal{C}_r^{3/4}(B_h)$ requires resp-vote from $\lfloor 3n/4 \rfloor + 1$ nodes in epoch r. A simple quorum intersection argument shows that a responsive certificate for an equivocating block $B'_{h'}$ cannot exist in epoch r.

Suppose for the sake of contradiction, an equivocating synchronous block certificate $\mathcal{C}_r^{1/2}(B'_{h'})$ for block $B'_{h'}$ exists in epoch r. At least one honest node, say node p_i, must have sent sync-vote for $B'_{h'}$ in epoch r. By Fact 1, an equivocating block certificate i.e., $\mathcal{C}_r^{3/4}(B_h)$ cannot exists. However, since $\mathcal{C}_r^{3/4}(B_h)$ exists, node p_i must not have sent sync-vote for $B'_{h'}$ in epoch r. A contradiction. □

Lemma 3. *If an honest node directly commits a block B_h in epoch r using the responsive commit rule (Step 7b), then (i) no equivocating block certificate exists in epoch r, and (ii) all honest nodes receive a block certificate for B_h before entering epoch $r+1$.*

Proof. Suppose an honest node p_i directly commits a block B_h in epoch r using responsive commit rule at time τ. Node p_i must have received $\langle \mathsf{ack}, H(B_h), z_{ra}, r \rangle$ for B_h from a set R of $\lfloor 3n/4 \rfloor + 1$ nodes when its epoch-timer$_r \geq 2\Delta$ and detected no epoch r equivocation by time τ. At least $\lfloor n/4 \rfloor + 1$ of them are honest and have received $\mathcal{C}_r^{3/4}(B_h)$ (corresponding to accumulation value z_{ra}). By Fact 2, there does not exist an equivocating block certificate in epoch r. This proves part(i) of the Lemma.

For part (ii), observe that node p_i has its epoch-timer$_r$ $\geq 2\Delta$ at time τ. Since, honest nodes are synchronized within Δ time, honest nodes that are still in epoch r at time τ must have epoch-timer$_r$ $\geq \Delta$ at time τ. In addition, since node p_i did not detect an epoch r equivocation at time τ, no honest node detected epoch r equivocation before time $\tau - \Delta$ and did not reset their epoch-timer$_r$ to 2Δ before time $\tau - \Delta$. Thus, honest nodes that are still in epoch r at time τ must have epoch-timer$_r$ $\geq \Delta$ at time τ. Since, node p_i received $\langle \mathsf{ack}, H(B_h), z_{ra}, r\rangle$ from a set R' of at least $\lfloor n/4 \rfloor + 1$ honest nodes at time τ, nodes in R' must have invoked Deliver(resp-cert, rc, z_{ra}, r) for $rc := \mathcal{C}_r^{3/4}(B_h)$ by time τ and their code words for $\mathcal{C}_r^{3/4}(B_h)$ arrives all honest nodes by time $\tau + \Delta$.

Suppose for the sake of contradiction, some honest node p_j did not receive an epoch r block certificate before entering epoch $r+1$. If node p_j entered epoch $r+1$ when its epoch-timer$_r$ expired, node p_j must be in epoch r at time $\tau + \Delta$ (since, its epoch-timer$_r$ $\geq \Delta$ at time τ) and must have received $\lfloor n/4 \rfloor + 1$ valid code words for $\mathcal{C}_r^{3/4}(B_h)$ sufficient to reconstruct $\mathcal{C}_r^{3/4}(B_h)$ by time $\tau + \Delta$. A contradiction. On the other hand, if node p_j enters epoch $r+1$ by receiving a aggregated secret R_r before its epoch-timer$_r$ expired, it waits for an epoch r block certificate. By Fact 2, there does not exist an equivocating block certificate in epoch r. Thus, the only block certificate node p_j can receive is $\mathcal{C}_r^{1/2}(B_h)$ or $\mathcal{C}_r^{3/4}(B_h)$. If node p_j has not received any block certificate, node p_j receives $\lfloor n/4 \rfloor + 1$ valid code words for $\mathcal{C}_r^{3/4}(B_h)$ by time $\tau + \Delta$ sufficient to reconstruct $\mathcal{C}_r^{3/4}(B_h)$. Again a contradiction. This proves part(ii) of the Lemma. □

Lemma 4. *If an honest node directly commits a block B_h in epoch r using synchronous commit rule (Step 7b), then (i) no equivocating block certificate exists in epoch r, and (ii) all honest nodes receive a block certificate for B_h before entering epoch $r+1$.*

Proof. Suppose an honest node p_i directly commits a block B_h in epoch r at time τ. Node p_i must have received either a synchronous certificate i.e., $\mathcal{C}_r^{1/2}(B_h)$ (or a responsive certificate i.e., $\mathcal{C}_r^{3/4}(B_h)$) at time $\tau - 2\Delta$ when its epoch-timer$_r$ $\geq 3\Delta$ and did not detect an epoch r equivocation by time τ. If node p_i received $\mathcal{C}_r^{1/2}(B_h)$, at least one honest node must have sent sync-vote for B_h and by Fact 1, no equivocating block certificate exists in epoch r. Similarly, if node p_i received $\mathcal{C}_r^{3/4}(B_h)$, by Fact 2, there does not exist an equivocating block certificate in epoch r. This proves part(i) of the Lemma.

For part (ii), observe that node p_i must have invoked Deliver(sync-cert, sc, z_{ve}, e) for $sc = \mathcal{C}_r^{1/2}(B_h)$ (or Deliver(resp-cert, rc, z_{ve}, e) for $rc = \mathcal{C}_r^{3/4}(B_h)$) at time $\tau - 2\Delta$ and did not detect an epoch r equivocation by time τ. Moreover, no honest node received aggregated secret R_r along with $\mathcal{C}_r(B_h)$ before time $\tau - \Delta$; otherwise node p_i must have received aggregated secret R_r before time τ and having already received $\mathcal{C}_r(B_h)$, would not commit using synchronous commit rule. Observe that no honest node detected an epoch r equivocation by time $\tau - \Delta$; otherwise, node p_i must have detected the equivocation by time τ and would not commit. Thus, all honest nodes will receive and forward their code words for $\mathcal{C}_r^{1/2}(B_h)$ (or $\mathcal{C}_r^{3/4}(B_h)$) by time $\tau - \Delta$ and all honest nodes will receive at least $t+1$ code words sufficient to decode $\mathcal{C}_r^{1/2}(B_h)$ (or $\mathcal{C}_r^{3/4}(B_h)$) by time τ.

Since, all honest nodes are synchronized within Δ time, all other honest nodes must have epoch-timer$_r$ $\geq 2\Delta$ at time $\tau - 2\Delta$. Thus, honest nodes that quit epoch r when their epoch-timer$_r$ expired must still be in epoch r at time τ and receive $\mathcal{C}_r^{1/2}(B_h)$ (or $\mathcal{C}_r^{3/4}(B_h)$) before entering epoch $r+1$. If some honest node, say node p_j, enters epoch $r+1$ by receiving aggregated secret R_r, it waits for an epoch r block certificate. By part(i) of the Lemma, there does not exist an equivocating block certificate in epoch r. Thus, the block certificate must be either $\mathcal{C}_r^{3/4}(B_h)$ or $\mathcal{C}_r^{1/2}(B_h)$. This proves part(ii) of the Lemma. □

Lemma 5. *If an honest node directly commits a block B_h in epoch r, then (i) no equivocating block certificate exists in epoch r, and (ii) all honest nodes receive and lock on $\mathcal{C}_r(B_h)$ before entering epoch $r+1$.*

Proof. Straight forward from Lemma 3 and Lemma 4. □

Lemma 6 (Unique Extensibility). *If an honest node directly commits a block B_h in epoch r, then any certified blocks that ranks higher than $\mathcal{C}_r(B_h)$ must extend B_h.*

Proof. The proof is by induction on epochs $r' > r$. For an epoch r', we prove that if a $\mathcal{C}_{r'}(B_{h'})$ exists then it must extend B_h.

For the base case, where $r' = r + 1$, the proof that $\mathcal{C}_{r'}(B_{h'})$ extends B_h follows from Lemma 5. The only way $\mathcal{C}_{r'}(B_{h'})$ for $B_{h'}$ forms is if some honest node votes for $B_{h'}$. However, by Lemma 5, there does not exist any equivocating block certificate in epoch r and all honest nodes receive and lock on $\mathcal{C}_r(B_h)$ before quitting epoch r. Thus, a block certificate cannot form for a block that does not extend B_h.

Given that the statement is true for all epochs below r', the proof that $\mathcal{C}_{r'}(B_{h'})$ extends B_h follows from the induction hypothesis because the only way such a block certificate forms is if some honest node votes for it. An honest node votes in epoch r' only if $B_{h'}$ extends a valid certificate $\mathcal{C}_{r''}(B_{h''})$. Due to Lemma 5 and the induction hypothesis on all block certificates of epoch $r < r'' < r'$, $\mathcal{C}_{r'}(B_{h'})$ must extend B_h. □

Lemma 7. *Let B_h be a block proposed in epoch r. If the leader of an epoch r is honest, then at least one honest node commits block B_h and all honest nodes lock on $\mathcal{C}_r(B_h)$ before entering epoch $r + 1$.*

Proof. Suppose leader L_r of an epoch r is honest. Let τ be the earliest time when an honest node p_i enters epoch r. Due to Δ delay between honest nodes, all honest nodes enter epoch r by time $\tau + \Delta$. Some honest nodes might have received a higher ranked certificate than leader L_r before entering epoch r; thus, they send their highest ranked certificate to leader L_r.

Leader L_r needs to ensure that it has the highest ranked certificate before proposing in epoch r. If L_r has $\mathcal{C}_{r-1}(B_l)$, $\mathcal{C}_{r-1}(B_l)$ is already the highest ranked certificate and L_r proposes immediately. Otherwise, it waits for 2Δ time to ensure it can receive highest ranked certificates $\mathcal{C}_{r'}(B_l)$ from all honest nodes. If L_r entered epoch r Δ time after node p_i, L_r sends a valid proposal $p_r = \langle \mathsf{propose}, B_h, r, \mathcal{C}_{r'}(B_l), z_{pa}\rangle_{L_r}$ by time $\tau + 3\Delta$ which arrives all honest nodes by time $\tau + 4\Delta$.

In any case, all honest nodes receive a valid proposal B_h that extends the highest ranked certificate while satisfying the constraint $\mathsf{epoch\text{-}timer}_r \geq 7\Delta$. Thus, all honest nodes will send resp-vote to leader L_r. In addition, all honest nodes will invoke $\mathsf{Deliver}(\mathsf{propose}, p_r, z_{pa}, r)$ and set $\mathsf{vote\text{-}timer}_r$ to 2Δ which expires by time $\tau + 6\Delta$. If $\lfloor 3n/4 \rfloor + 1$ nodes send resp-vote to L_r, leader L_r can immediately broadcast $\langle \mathsf{resp\text{-}cert}, \mathcal{C}_r^{3/4}(B_h), z_{ra}, r\rangle_{L_r}$ to all other nodes. All honest nodes will then broadcast ack to all other nodes. If an honest node, say node p_i, receives $\lfloor 3n/4 \rfloor + 1$ ack messages, it commits responsively. Observe that all honest nodes receive $\mathcal{C}_r^{3/4}(B_h)$ from the leader before epoch r ends. On the other hand, if no honest node received $\lfloor 3n/4 \rfloor + 1$ ack messages, all honest nodes will set their $\mathsf{commit\text{-}timer}_r$ to 2Δ on receiving $\langle \mathsf{resp\text{-}cert}, \mathcal{C}_r^{3/4}(B_h), z_{ra}, r\rangle_{L_r}$. Since, no equivocation exists in epoch r, at least one honest node (the earliest honest node), say node p_i, will commit in epoch r. All other nodes will either commit when their $\mathsf{commit\text{-}timer}_r$ expires or move to epoch $r + 1$ on receiving aggregated secret R_r. In either case, they lock on $\mathcal{C}_r^{3/4}(B_h)$ and satisfies the Lemma.

Next we discuss the case when optimistic conditions are not met, i.e., $\lfloor 3n/4 \rfloor + 1$ nodes do not send resp-vote for B_h or do not send ack messages. In this case, all honest nodes will at least send sync-vote for B_h to L_r which arrives L_r by time $\tau + 7\Delta$. Leader L_r forwards $\mathcal{C}_r(B_h)$ which arrives all honest nodes by time $\tau + 8\Delta$. Note that all honest nodes satisfy the constraint $\mathsf{epoch\text{-}timer}_r \geq 3\Delta$ and honest nodes set their $\mathsf{commit\text{-}timer}_r$ to 2Δ which expires by time $\tau + 10\Delta$. Moreover, no equivocation exists in epoch r. Thus, the earliest honest node, say node p_j that sets $\mathsf{commit\text{-}timer}_r$ will commit. All other nodes will either commit when their $\mathsf{commit\text{-}timer}_r$ expires or move to epoch $r + 1$ on receiving aggregated secret R_r. In either case, they lock on $\mathcal{C}_r^{1/2}(B_h)$ and satisfies the Lemma.

□

Lemma 8. *If an honest node commits a block B_ℓ proposed in epoch $r - t$ at the end of epoch r such that the highest ranked chain in epoch r extends B_ℓ, then any certified blocks in epoch r or higher must extend B_ℓ.*

Proof. Due to round robin leader election, there will at least one honest leader between epoch $r - t$ and r, say epoch r'. By Lemma 7, at least one honest node directly commits a block B_h proposed in epoch r'. By Lemma 5 all honest nodes lock on $\mathcal{C}_{r'}(B_h)$ and do not vote for blocks that do not extend B_h. By Lemma 6, any certified blocks that ranks higher than $\mathcal{C}_{r'}(B_h)$ must extend B_h. Thus, the highest ranked chain at the end of epoch r must extend $\mathcal{C}_{r'}(B_h)$.

Since, the highest ranked chain at the end of epoch r extends B_ℓ and B_ℓ was proposed at epoch $r-t < r'$, B_h extend B_ℓ. By Lemma 6, any certified blocks that ranks higher than $\mathcal{C}_{r'}(B_h)$ must extend B_h. Thus, any certified blocks in epoch r or higher must extend B_ℓ. □

Theorem 9 (Safety)**.** *Honest nodes do not commit conflicting blocks for any epoch r.*

Proof. Suppose for the sake of contradiction two distinct blocks B_h and B'_h are committed in epoch r. Suppose B_h is committed as a result of $B_{h'}$ being directly committed in epoch r' and B'_h is committed as a result of $B'_{h''}$ being directly committed in epoch r''. Without loss of generality, assume $h' < h''$. Note that all directly committed blocks are certified. By Lemma 6 and Lemma 8, $B'_{h''}$ extends $B_{h'}$. Therefore, $B_h = B'_h$. □

Theorem 10 (Liveness)**.** *All honest nodes keep committing new blocks.*

Proof. For any epoch r, if the leader L_r is Byzantine, it may not propose any blocks or propose equivocating blocks. Whenever an honest leader is elected in epoch r, by Lemma 7, at least one honest node commits block B_h proposed in epoch r and all other honest nodes lock on $\mathcal{C}_r(B_h)$ proposed in epoch r i.e., all honest nodes add block B_h proposed in epoch r. Since we assume a round-robin leader rotation policy, there will be an honest leader every $t+1$ epochs, and every time an honest leader is selected, all honest nodes keep committing new blocks. □

Lemma 11 (Communication complexity)**.** *Let κ be the size of accumulator and w be the size of witness. The communication complexity of the protocol is $O((\kappa+w)n^2)$ bits per epoch.*

Proof. At the start of an epoch r, each node sends a highest ranked certificate and $O(\kappa n)$-sized PVSS tuple to leader L_r. Since, size of each certificate and PVSS tuple is $O(\kappa n)$, this step incurs $O(\kappa n^2)$ bits communication. A proposal for a block B_h consists of $O(\kappa n)$-sized aggregated PVSS tuple, and a block certificate of size $O(\kappa n)$. Proposing $O(\kappa n)$-sized block to n nodes incurs $O(\kappa n^2)$. Delivering $O(\kappa n)$-sized message has a cost $O((\kappa+w)n^2)$, since each node broadcasts a code word of size $O((\kappa n)/n)$, a witness of size w and an accumulator of size κ.

In Resp cert step, the leader broadcasts a responsive certificate for block B_h, i.e, $\mathcal{C}_r^{3/4}(B_h)$ which incurs $O(\kappa n^2)$ communication. Delivering $O(\kappa n)$-sized $\mathcal{C}_r^{3/4}(B_h)$ incurs $O((\kappa+w)n^2)$ bits. Again, in Sync cert step, the leader broadcasts a synchronous certificate for block B_h, i.e, $\mathcal{C}_r^{1/2}(B_h)$ which incurs $O(\kappa n^2)$ communication. Delivering $O(\kappa n)$-sized $\mathcal{C}_r^{1/2}(B_h)$ incurs $O((\kappa+w)n^2)$ bits. In Ack step, nodes perform an all-to-all broadcast of κ-sized accumulator which incurs $O(\kappa n^2)$ bits communication.

During reconstruction, nodes broadcast κ-sized secret shares and w-sized witness. All-to-all broadcast of secret shares and witness incur $O((\kappa+w)n^2)$ bits communication. In addition, nodes broadcast $O(\kappa)$-sized aggregated secret R_r at the end of epoch r which incurs $O(\kappa n^2)$. Hence, the total cost is $O((\kappa+w)n^2)$ bits. □

Theorem 12 (Consistent Beacon)**.** *For any epoch r, all honest nodes reconstruct the same aggregated secret R_r and output the same beacon $\mathcal{O}_r$.*

Proof. Honest nodes reconstruct the same aggregated secret R_r and output the same beacon $\mathcal{O}_r$ in epoch r if all honest nodes receive $t+1$ valid homomorphic secret shares for the same PVSS tuple. This condition is satisfied if all honest nodes have consistent $\mathcal{Q}(p_i), \forall p_i \in \mathcal{P}$ and consistent $\mathcal{P}_r$ in each epoch.

Consider epochs 1 through t. During setup phase, each node is assigned same PVSS tuple for each $\mathcal{Q}(p_i), \forall p_i \in \mathcal{P}$ and $\mathcal{P}_r$ is set to $\emptyset$. No honest node updates $\mathcal{Q}(p_i)$ during epochs 1 to t. Thus, for epochs 1 to t, all honest nodes have consistent $\mathcal{Q}(p_i), \forall p_i \in \mathcal{P}$ and $\mathcal{P}_r$.

Consider an epoch $r > t$. In epoch r, all honest nodes update only $\mathcal{Q}(L_{r-t})$. If L_{r-t} proposed a valid block B_l (with $b_l = (\mathbf{v}, \mathbf{c})$) and B_l is committed by epoch r, all honest nodes update $\mathcal{Q}(L_{r-t})$ with $(\mathbf{v}, \mathbf{c})$. Otherwise, all honest nodes update $\mathcal{P}_r$ to exclude L_{r-t} i.e., $\mathcal{P}_r \leftarrow \mathcal{P}_r \cup \{L_{r-t}\}$. Thus, all honest nodes should have consistent $\mathcal{Q}(L_{r-t})$ by epoch r. Since honest nodes do not update $\mathcal{Q}(p_i \neq L_{r-t})$ and do not add p_i into $\mathcal{P}_r$ in epoch r, all honest nodes should have consistent $\mathcal{Q}(p_i)$ $\forall p_i \in \mathcal{P}$ and consistent $\mathcal{P}_r$ in epoch r. Since, all honest nodes have a consistent $\mathcal{Q}(p_i)$ $\forall p_i \in \mathcal{P}$ and consistent $\mathcal{P}_r$, all honest nodes perform $\{\texttt{Dequeue}(\mathcal{Q}(p_i)), \forall p_i \in \mathcal{P} \setminus \mathcal{P}_r\}$ for common PVSS tuples in epoch r.

Next, we show $\{\texttt{Dequeue}(\mathcal{Q}(p_i)) \neq \bot, \forall p_i \in \mathcal{P} \setminus \mathcal{P}_r\}$ in epoch r. Suppose for the sake of contradiction, $\texttt{Dequeue}(\mathcal{Q}(p_i)) = \bot$ and $p_i \notin \mathcal{P}_r$ in epoch r. Observe that, honest nodes update $\mathcal{Q}(p_i)$ or include p_i in $\mathcal{P}_r$ $t+1$

epochs after node p_i becomes an epoch leader. Let r' be the last epoch in which node p_i last proposed with $r' < r - t$. However, if node p_i did not propose in r', all honest nodes would have removed p_i by epoch $r' + t < r$ and $p_i \in \mathcal{P}_r$ by epoch r. A contradiction.

Thus, all nodes will dequeue common secret shares and will receive at least $t + 1$ valid aggregated secret shares for a common PVSS tuple and reconstruct the same aggregated secret R_r and output the same beacon $\mathcal{O}_r$. □

Lemma 13 (Guaranteed Beacon Output)**.** *For each epoch r, all honest nodes output a new beacon output $\mathcal{O}_r$.*

Proof. In each epoch r, honest nodes perform $\texttt{Dequeue}(\mathcal{Q}(L_r))$ which contains PVSS tuples proposed by leader L_r when L_r was an epoch leader in some epoch r' (with $r' + t < r$). Since the leader L_r is the current leader, $L_r \notin \mathcal{P}_r$ in epoch $r' + t$. Suppose for the sake of contradiction, $\texttt{Dequeue}(\mathcal{Q}(L_r)) = \perp$ and $L_r \notin \mathcal{P}_r$ in epoch $r' + t$. Observe that, honest nodes update $\mathcal{Q}(L_r)$ or include L_r in $\mathcal{P}_r$ $t + 1$ epochs after node p_i becomes an epoch leader. However, if node L_r did not propose in r', all honest nodes would have removed L_r by epoch $r' + t < r$ and $L_r \in \mathcal{P}_r$ in epoch $r' + t$. Thus, L_r would not have been a leader in epoch r. A contradiction.

Thus, at the end of each epoch, $\texttt{Dequeue}(\mathcal{Q}(L_r)) \notin \perp$ and all honest nodes dequeue common secret shares and send homomorphic secret shares to all other nodes. Thus, honest nodes will have $t + 1$ valid homomorphic secret shares and will output new beacon outputs in every epochs. □

Lemma 14 (Bias-resistance)**.** *For any epoch r, the output $\mathcal{O}_r$ is uniformly random.*

Proof. The beacon output $\mathcal{O}_r$ is the function of $t+1$ PVSS polynomials one of which is from at least one honest node. A polynomial from an honest node is chosen uniformly random and one uniformly random polynomial is sufficient to ensure the combined polynomial is uniformly random. This implies OptRand is bias-resistant. □

Lemma 15 (Unpredictability)**.** *For any epoch r, an adversary can reconstruct output $\mathcal{O}_r$ at most 6Δ time before honest nodes.*

Proof. An honest node sends its homomorphic secret shares in epoch r when it commits in epoch r or its epoch-timer$_r$ expires. The worst case is when some honest node p_i commits responsively and sends its homomorphic secret share while all other honest nodes send their homomorphic secret shares only when their epoch-timer$_r$ expires. Suppose an honest node p_i responsively commits block B_h proposed in epoch r at time τ and sends its homomorphic secret share at time τ. An adversary may instantaneously receive the secret share and reconstruct the output O_r at time τ.

By time $\tau + \Delta$, all honest nodes will receive a responsive certificate for B_h and start their commit-timer$_r$ which should expire by time $\tau + 3\Delta$. However, before their commit-timer$_r$ expires, they observe equivocation and reset their epoch-timer$_r$ to 2Δ which should expire by time $\tau + 5\Delta$. Thus, these honest nodes will sends their homomorphic secret share at time $\tau + 5\Delta$ which arrives all honest nodes by time $\tau + 6\Delta$. Thus, all honest nodes will output O_r only at time $\tau + 6\Delta$ giving 6Δ advantage to an adversary. □

6 Reconfiguration

In this section, we present a reconfiguration scheme for our beacon protocol to restore the resilience of the protocol after some Byzantine nodes have been removed. We adapt the reconfiguration scheme of RandPiper [8] and make efficiency improvements in terms of the number of epochs before a new joining node becomes an active participant in the system. We make following modifications to obtain this efficiency.

A reconfiguration scheme for a synchronous protocol requires new joining nodes to synchronize with all other nodes such that the clocks of all honest nodes differ by at most Δ time. In RandPiper, they designed a clock synchronization primitive to sychronize the joining nodes. In the clock synchronization primitive, they first secret shared $t + 1$ secrets from distinct leaders using verifiable secret sharing (VSS) scheme. To ensure all the nodes agree on the shared secret, the clock synchronization protocol had to be executed for $2t + 2$ epochs. The agreed upon $t + 1$ secrets were homomorphically combined to obtain a aggregated secret that is used to synchronize new joining nodes. In our protocol, the reconstructed secret is already a aggregated secret combined using $t + 1$ secrets from different nodes. Moreover, we are generating and using the aggregated secret to synchronize in every epoch. Thus, the same aggregated secret can be used to synchronize the new joining nodes and avoid the need to execute a separate clock synchronization

A new node p_k that intends to join the system uses following procedure to join the system.

1. **Inquire.** Node p_k inquires all nodes in the system to send the set of active nodes, i.e., $\mathcal{P} \setminus \mathcal{P}_r$. Upon receiving the inquire request, an honest node p_i responds to the request only if $n_t > 0$. Node p_i sends set $\mathcal{P} \setminus \mathcal{P}_r$ along with PVSS tuple $(\mathbf{v}_i, \mathbf{c}_i, \widetilde{\pi}_{K,i})$ at the end of some epoch r' in which the inquire request was received. Node p_k waits for at least $t+1$ consistent responses from the same epoch r' and forms an inquire certificate. An inquire certificate is valid if it contains $t+1$ inquire responses that belong to the same epoch r' and contains the same set of active nodes. In addition, node p_k aggregates $t+1$ PVSS tuples to obtain $(\mathbf{v}, \mathbf{c}, \widetilde{\pi}_K)$ (refer Aggregation Step in Fig. 2).
2. **Join.** Node p_k sends a join request to all nodes $\mathcal{P} \setminus \mathcal{P}_r$ with the inquire certificate and aggregated PVSS tuple $(\mathbf{v}, \mathbf{c}, \widetilde{\pi}_K)$ to all nodes $\mathcal{P} \setminus \mathcal{P}_r$ to node $p_j \; \forall p_j \in \mathcal{P} \setminus \mathcal{P}_r$.
3. **Accept.** Upon receiving the join request with valid inquire certificate and aggregated PVSS tuple $(\mathbf{v}, \mathbf{c}, \widetilde{\pi}_K)$, node p_i checks the validity of the received PVSS tuple (refer Commit Step in Fig. 2). If valid, send $\langle \mathsf{accept}, H(\mathbf{v}, \mathbf{c}), r\rangle_{p_i}$ to node p_k.
4. **Accept Cert.** Upon receiving $t+1$ accept messages, node p_k broadcasts the accept certificate to all nodes $\mathcal{P} \setminus \mathcal{P}_r$.
5. **Propose.** Upon receiving the join request with valid inquire certificate, aggregated PVSS tuple and accept certificate, the leader L_r of current epoch r adds the join request containing inquire certificate, PVSS tuple and accept certificate in its block proposal B_h if (i) L_r does not observe a block proposal with a join request in last $t+1$ epochs in its highest ranked chain and (ii) no new node has been added since epoch r'.
6. **Update.** If the block B_h with the join request from node p_k proposed in epoch r gets committed by epoch $r+t$, update $n_t \leftarrow n_t - 1$ in epoch $r+t$, update $\mathcal{Q}(p_j)$ with aggregated PVSS tuple $(\mathbf{v}, \mathbf{c})$ and send set $\mathcal{P} \setminus \mathcal{P}_r$ to node p_k. Henceforth, node p_k becomes a *passive* node and receives all protocol messages from active nodes.
7. **Synchronize.** The first time node p_i receives a valid aggregated secret R_{r+t}, it
   - resets its epoch-timer$_{r+t+1}$ to the beginning of epoch $r+t+1$.
   - broadcasts aggregated secret R_{r+t} to all other nodes.

If node p_k fails to join the system, it restarts reconfiguration process again after some time.

Figure 6: Reconfiguration protocol

primitive. In the process, the new joining nodes can become active $2t+2$ epochs earlier than RandPiper. In addition, due to optimistic responsiveness, the length of each epoch is considerably shorter during optimistic conditions and new nodes can join the system much quicker. In this regard, our reconfiguration scheme is strictly better compared to RandPiper.

Observe that in OptRand, nodes in $\mathcal{P} \setminus \mathcal{P}_r$ are rotated in round robin manner and when some node p_j becomes an epoch leader in an epoch, the secrets node p_j shared the last time it became an epoch leader is used. To be specific, the secrets in $\mathcal{Q}(p_j)$ is used. Thus, our reconfiguration scheme needs to ensure that when some node p_k joins the system, all nodes $\mathcal{P} \setminus \mathcal{P}_r$ have $\mathcal{Q}(p_k)$ filled with aggregated PVSS tuple before p_k becomes an epoch leader. We accomplish this by having the joining node aggregate $t+1$ PVSS tuple and send it to all nodes $\mathcal{P} \setminus \mathcal{P}_r$ before it can join the system.

The reconfiguration protocol is presented in Fig. 6. Each node maintains a variable n_t that records the number of additional nodes that can been added to the system. Variable n_t is incremented each time a Byzantine node is added to set $\mathcal{P}_r$ and is decremented by one when a new node joins the system. The value of n_t can be at most t.

Node p_k that intends to join the system uses the reconfiguration protocol to join the system. All nodes update $\mathcal{Q}(p_k)$ with the aggregated PVSS tuple provided by node p_k once the join request from node p_k get committed.

Node p_k becomes an *active* node when it has $\mathcal{Q}(p_j) \neq \bot \forall p_j \in \mathcal{P} \setminus \mathcal{P}_r$. This happens when all nodes in $\mathcal{P} \setminus \mathcal{P}_r$ becomes a leader at least once after node p_k joins the system. Due to round robin leader election, node p_k will have a full queue after $n+t+1$ epochs.

Lemma 16 (Liveness). *If $n_t > 0$ at some epoch r^* and there are new nodes intending to join the system in epochs $\geq r^*$, then eventually a new node will be added to the system.*

Proof. Suppose $n_t > 0$ and a new node p_k intends to join the system. Suppose for the sake of contradiction, no new node including p_k is added to the system. However, since node p_k intends to join the system, it must have sent inquire requests to all nodes in the system and at least $t+1$ honest nodes will respond to the inquire request since $n_t > 0$ at the end of some epoch $r' \geq r^*$.

Let node p_k send join request along with an inquire certificate and nodes receive the request in epoch $r \geq r'$. The first honest leader $L_{r''}$ of epoch $r'' \geq r$ will include the join request in its block proposal if no new node has been added to the system since epoch r' and there does not exist any block proposal with a join request in the last $t+1$ epochs in its highest ranked chain and by Lemma 7, the block proposal with join request will be committed. A contradiction.

If some node has already been added to the system since epoch r', this trivially satisfies the statement and we obtain a contradiction. If there exists a block proposal B_h with a join request for some node p_j in last t epochs in the highest ranked chain for $L_{r''}$, B_h will be committed since honest node $L_{r''}$ extends it. The lemma holds and we obtain a contradiction. □

Theorem 17. *OptRand maintains safety and liveness after reconfiguration.*

Proof. Let node p_i be the new joining node. OptRand is safe and live before reconfiguration. Since there are $t+1$ honest nodes and honest nodes are never removed, there will always be $t+1$ honest nodes that have full queue, i.e., $\mathcal{Q}(p_j) \neq \perp \forall p_j \in \mathcal{P} \setminus \mathcal{P}_r$. Thus, there will always be $t+1$ honest nodes with correct secret shares. Hence, the protocol maintains safety and liveness after reconfiguration. □

7 Conclusion

We present a decomposition proof technique that allows a node to aggregate $t+1$ shares and prove that it combined $t+1$ individual PVSS shares whose proof size is $O(\kappa n)$. The aggregated information is also verifiable, i.e., given the reconstructed value, any node can check that it is indeed the correct reconstructed value, with the verifiability proof size being $O(\kappa)$. These two features along with careful protocol design allows OptRand to be optimistically responsive with commit latencies of $O(\delta)$ when the leader of a round and $> 3n/4$ nodes are honest, while simultaneously having commit latencies of 11Δ otherwise explicitly switching within the protocol. OptRand has a communication complexity of $O(\kappa n^2)$ always, while also allowing new nodes to join the system within $t+1$ epochs using the novel reconfiguration protocol.

8 Acknowledgements

We thank Sourav Das and Ling Ren for helpful discussions on SPURT [20]. This research was supported in part by grants by VMware Research and Novi Research. This work has been partially supported by the Army Research Laboratory (ARL) under grant W911NF-20-2-0026, the National Institute of Food and Agriculture (NIFA) under grant 2021-67021-34251, and the National Science Foundation (NSF) under grant CNS-1846316.

References

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected $O(1)$ rounds, expected $O(n^2)$ communication, and optimal resilience. *Financial Cryptography and Data Security (FC)*, 2019.

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, and Ling Ren. Dfinity consensus, explored. *IACR Cryptol. ePrint Arch.*, 2018:1153, 2018.

[3] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 654–667, 2020.

[4] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2019.

[5] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Optimal good-case latency for byzantine broadcast and state machine replication. *arXiv preprint arXiv:2003.13155*, 2020.

[6] Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. Optimal good-case latency for rotating leader synchronous bft. In *25nd International Conference on Principles of Distributed Systems (OPODIS 2025)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2021.

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

[8] Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. Randpiper – reconfiguration-friendly random beacons with quadratic communication. ACM SIGSAC CCS 2021, 2021.

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

[10] Dan Boneh and Victor Shoup. A graduate course in applied cryptography, 2017. `http://toc.cryptobook.us/book.pdf`.

[11] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

[12] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *ACM Symposium on Theory of computing (STOC)*, pages 42–51, 1993.

[13] Ignacio Cascudo and Bernardo David. Scrape: Scalable randomness attested by public entities. In *International Conference on Applied Cryptography and Network Security*, pages 537–556. Springer, 2017.

[14] Ignacio Cascudo, Bernardo David, Omer Shlomovits, and Denis Varlakov. Mt. random: Multi-tiered randomness beacons. Cryptology ePrint Archive, Report 2021/1096, 2021. `https://ia.cr/2021/1096`.

[15] Generate random numbers for smart contracts using chainlink vrf. `https://docs.chain.link/docs/chainlink-vrf`.

[16] T-H Hubert Chan, Rafael Pass, and Elaine Shi. Pili: An extremely simple synchronous blockchain. *IACR Cryptology ePrint Archive*, 2018:980, 2018.

[17] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Annual international cryptology conference*, pages 89–105. Springer, 1992.

[18] Alisa Cherniaeva, Ilia Shirobokov, and Omer Shlomovits. Homomorphic encryption random beacon. *IACR Cryptol. ePrint Arch.*, 2019:1320, 2019.

[19] Information Technology Laboratory Computer Security Division. Interoperable randomness beacons: Csrc. `https://csrc.nist.gov/projects/interoperable-randomness-beacons`.

[20] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. Spurt: Scalable distributed randomness beacon with transparent setup. Technical report, Cryptology ePrint Archive, Report 2021/100. https://eprint. iacr. org/2021/100, 2021.

[21] Danny Dolev, Joseph Y Halpern, Barbara Simons, and Ray Strong. Dynamic fault-tolerant clock synchronization. *Journal of the ACM (JACM)*, 42(1):143–185, 1995.

[22] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[23] J Drake. Minimal vdf randomness beacon. ethereum research post (2018). `https://ethresear.ch/t/minimal-vdf-randomness-beacon/3566`.

[24] Drand. Drand - a distributed randomness beacon daemon. `https://github.com/drand/drand`.

[25] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

[26] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

[27] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[28] Matthias Fitzi and Juan A Garay. Efficient player-optimal protocols for strong and differential consensus. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 211–220, 2003.

[29] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[30] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. Cryptology ePrint Archive, Report 2021/005, 2021. `https://eprint.iacr.org/2021/005`, Appearing in EUROCRYPT '21.

[31] Mads Haahr. True random number service. `https://www.random.org/`.

[32] Runchao Han, Haoyu Lin, and Jiangshan Yu. Randchain: Decentralised randomness beacon from sequential proof-of-work. Cryptology ePrint Archive, Report 2020/1033, 2020. `https://eprint.iacr.org/2020/1033`.

[33] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[34] Somayeh Heidarvand and Jorge L Villar. Public verifiability from pairings in secret sharing schemes. In *International Workshop on Selected Areas in Cryptography*, pages 294–308. Springer, 2008.

[35] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In *Annual International Cryptology Conference*, pages 445–462. Springer, 2006.

[36] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[37] Atsuki Momose, Jason Paul Cruz, and Yuichi Kaji. Hybrid-bft: Optimistically responsive synchronous consensus with optimal latency or resilience. *IACR Cryptol. ePrint Arch.*, 2020:406, 2020.

[38] Atsuki Momose and Ling Ren. Optimal communication complexity of byzantine consensus under honest majority. *arXiv preprint arXiv:2007.13175*, 2020.

[39] Lan Nguyen. Accumulators from bilinear pairings and applications. In *Cryptographers' track at the RSA conference*, pages 275–292. Springer, 2005.

[40] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018.

[41] blockchain oracle service, enabling data-rich smart contracts. `https://provable.xyz/`.

[42] Michael O Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409. IEEE, 1983.

[43] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[44] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar Weippl. Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness. Technical report, Cryptology ePrint Archive, Report 2020/942, https://eprint. iacr. org/2020/942, 2020.

[45] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. Hydrand: Practical continuous distributed randomness. In *2020 IEEE Symposium on Security and Privacy (SP). IEEE*, 2020.

[46] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[47] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the Optimality of Optimistic Responsiveness. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 839–857, 2020.

[48] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. Synchronous distributed key generation without broadcasts. *Cryptology ePrint Archive*, 2021:1635, 2021.

[49] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 444–460. Ieee, 2017.

[50] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

A Extended Preliminaries

Definition A.1 (Pairings [10]). *Let $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ be three cyclic groups of prime order q where $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ are generators. A pairing is an efficiently computable function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfying the following properties:*

– *Bilinear. For all $u_1, v_1 \in \mathbb{G}_1$ and $u_2, v_2 \in \mathbb{G}_2$, we have $e(u_1 \cdot v_1, u_2) = e(u_1, u_2) \cdot e(v_1, u_2)$ and $e(u_1, u_2 \cdot v_2) = e(u_1, u_2) \cdot e(u_1, v_2)$,*

– *Non-degenerate. $g_T := e(g_1, g_2)$ is a generator of $\mathbb{G}_T$*

Linear erasure and error correcting codes. A coding scheme has the following interfaces:

- ENC. Given inputs $m_1, \dots, m_b$, an encoding function ENC computes $(s_1, \dots, s_n) = \mathsf{ENC}(m_1, \dots, m_b)$, where $(s_1, \dots, s_n)$ are code words of length n. A combination of any b elements of the code word uniquely determines the input message and the remaining of the code word.
- DEC. The function DEC computes $(m_1, \dots, m_b) = \mathsf{DEC}(s_1, ..., s_n)$, and is capable of tolerating up to c errors and d erasures in code words $(s_1, \dots, s_n)$, if and only if $n - b \geq 2c + d$.

Cryptographic accumulators. Formally, given a parameter κ, and a set D of n values $d_1, \dots, d_n$, an accumulator has the following interface:

- Gen($1^\kappa, n$): takes a parameter κ and an accumulation threshold n (an upper bound on the number of values that can be accumulated securely), returns an accumulator key ak. The accumulator key ak is part of the trusted setup and therefore is public to all nodes.
- Eval($ak, \mathcal{D}$): takes an accumulator key ak and a set $\mathcal{D}$ of values to be accumulated, returns an accumulation value z for the value set $\mathcal{D}$.

- $\mathsf{CreateWit}(ak, z, d_i, \mathcal{D})$: takes an accumulator key ak, an accumulation value z for $\mathcal{D}$ and a value d_i, returns $\perp$ if $d_i \in \mathcal{D}$,and a witness w_i if $d_i \in \mathcal{D}$.
- $\mathsf{Verify}(ak, z, w_i, d_i)$: takes an accumulator key ak, an accumulation value z for $\mathcal{D}$, a witness w_i and a value d_i, returns true if w_i is the witness for $d_i \in \mathcal{D}$, and false otherwise.

The bilinear accumulator satisfies the following property:

Lemma 18 (Collision-free accumulator [39])**.** *The bilinear accumulator is collision-free. That is, for any set of size n and a probabilistic polynomial-time adversary $\mathcal{A}$, the following function is negligible in κ:*

$$\mathsf{P}r\left[\begin{array}{c|c} ak \leftarrow \mathsf{G}en(1^\kappa, n), & \\ (\{d_1, \dots, d_n\}, d', w') \leftarrow & (d' \notin \{d_1, ..., d_n\})\wedge \\ \mathcal{A}(1^\kappa, n, ak), & (\mathsf{V}erify(ak, z, w', d') = 1) \\ z \leftarrow \mathsf{E}val(ak, \{d_1, \dots, d_n\}) & \end{array}\right]$$

co-Decisional Bilinear Squaring assumption. This is a modified version of the symmetric pairing based assumption in SCRAPE and related works [13, 34]. We formally show that this is the correct generalization of the DBS assumption for Type-III pairings by showing that it implies the *Decisional Bilinear Diffie Hellman (DBDH)* assumption in Definition A.3.

Definition A.2 (co-Decisional Bilinear Squaring (co-DBS) Assumption)**.** *Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be an efficient pairing scheme, with $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ being two independent generators. We say that the co-DBS assumption holds if the following is true for any PPT adversary $\mathcal{A}$:*

$$\Pr\left[\begin{array}{c|c} \alpha, \beta, \gamma \leftarrow \mathbb{Z}_q, b \leftarrow \{0,1\}, & \\ u_1 = g_1^\alpha, u_2 = g_2^\alpha, & \\ v_1 = g_1^\beta, v_2 = g_2^\beta, & \\ T_0 = e(g_1, g_2)^{\alpha^2\beta}, & b' = b \\ T_1 = e(g_1, g_2)^\gamma, & \\ b' \leftarrow \mathcal{A}(g_1, g_2, u_1, u_2, v_1, v_2, T_b) & \end{array}\right] \leq \mathsf{negl}(\kappa)$$

Prior works such as SCRAPE [13] define the problem in the symmetric pairing setting where $\mathbb{G}_1 = \mathbb{G}_2$. However, no known explicit construction exists for the asymmetric pairing definitions, although most of these works argue that the generalization is easy. SCRAPE and its sources [34] mention that it is easy to generalize it to the general model. In this work, we make explicit the generic model assumption and show reduction to the known *Decisional Bilinear Diffie Hellman* assumption.

Definition A.3 (Decisional Bilinear Diffie Hellman (DBDH) Assumption [10])**.** *Let $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be an efficient pairing. Let $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be two independent generators. The assumption is said to hold if for any PPT adversary $\mathcal{A}$ the following is true:*

$$\Pr\left[\begin{array}{c|c} \alpha, \beta, \gamma, \delta \leftarrow \mathbb{Z}_q, b \leftarrow \{0,1\} & \\ u_1 \leftarrow g_1^\alpha, u_2 \leftarrow g_2^\alpha & \\ v_1 \leftarrow g_1^\beta, v_2 \leftarrow g_2^\beta & \\ w_2 \leftarrow g_2^\gamma & b' = b \\ T_0 = e(g_1, g_2)^{\alpha\beta\gamma} \in \mathbb{G}_T & \\ T_1 = e(g_1, g_2)^\delta \in \mathbb{G}_T & \\ b' \leftarrow \mathcal{A}(g_1, g_2, u_1, u_2, v_1, v_2, w_2, T_b) & \end{array}\right] \leq \mathsf{negl}(\kappa)$$

Lemma 19. *The co-DBS assumption implies the DBDH assumption.*

Proof. Given an instance of co-DBS $(g_1, g_2, u_1, u_2, v_1, v_2, T_b)$ we can construct a correct DBDH instance using $(g_1, g_2, u_1, u_2, v_1, v_2, w_2 = g_2^{\gamma}, T'_b = T_b^{\gamma})$ for a randomly chosen $\gamma \in \mathbb{Z}_q$. If $b = 0$, then it is the correct instance of DBDH with $T'_0 = e(g_1, g_2)^{\alpha\beta\gamma}$. If $b = 1$, then it is the correct instance of DBDH with $T_1 = e(g_1, g_2)^{\gamma'\gamma}$ where γ' originated from the co-DBS instance. □

Chaum-Pedersen Scheme for NIZKPK**.** Concretely,

Let (g, u) be public values with $u = g^s$. A prover P runs the following interactive protocol:

1. P first sends to V, the values $a = g^w$ for a randomly drawn $w \in \mathbb{Z}_q$.
2. The verifier V chooses a random $c \in \mathbb{Z}_q$, and sends c to the prover P.
3. The prover sends $r = s + cw$ to V.
4. The verifier checks if $g^r = ua^c$ and outputs the result.

Figure 7: Interactive discrete log Proof of Knowledge protocol for NIZKPK

Using Fiat-Shamir heuristic [27], we transform this into a non-interactive proof (assuming Random Oracle Model) by setting $c = H(u, a)$ and proof $\pi := (a, r)$.

Note. This can be easily extended to prove that given g_1, g_2, u_1, u_2, a prover knows x such that $g_1^x = u_1$ and $g_2^x = u_2$ by duplicating all the steps except generating the challenge c. The challenge can be generated together as $c \leftarrow H(u_1, u_2, a_1, a_2)$.

B Security Analysis, IND1-Secrecy and Proof

Indistinguishability of secrets (IND1-Secrecy) refers to notion that when the dealer of a PVSS scheme is honest, the (computational) adversary does not learn any information about the secret. Formally, it is defined by Heidarvand et al. [34] in the game defined in Definition B.1. It assumes that a PVSS scheme consists of the following four phases: (i) Setup, (ii) Distribution, (iii) Verification, and (iv) Reconstruction.

Definition B.1 ((IND1-secrecy) Indistinguishability of secrets [34])**.** *We say that an (n, t) threshold PVSS scheme is IND1-secret if any PPT adversary $\mathcal{A}$ has a negligible advantage in the following game played against a challenger $\mathcal{C}$. During the game, $\mathcal{A}$ can corrupt a new node at any time, but up to t nodes in total. When $\mathcal{A}$ corrupts a node, he receives his secret key (only after the setup). A list of corrupted nodes is maintained during the game.*

1. *$\mathcal{C}$ runs the setup subprotocol and sends the public parameters to $\mathcal{A}$ along with the public keys of still uncorrupted nodes. $\mathcal{C}$ stores the secret keys of those nodes.*
2. *$\mathcal{A}$ sends the public keys of already corrupted nodes.*
3. *$\mathcal{C}$ picks two random secrets x_0, x_1 and a random bit $b \in \{0, 1\}$. Then $\mathcal{C}$ runs the distribution subprotocol for secret x_0 and sends all the resulting information to $\mathcal{A}$, along with x_b.*
4. *$\mathcal{C}$ runs reconstruction subprotocol for the set of all uncorrupted nodes and sends all the messages exchanged via public channels (if any) to $\mathcal{A}$. No new corruptions are allowed from this point.*
5. *$\mathcal{A}$ outputs a guess bit b'.*

The advantage of the adversary $\mathcal{A}$ in this game is defined as $|\mathsf{Prob}\,[b' = b] - \frac{1}{2}|$.

Note. In this work, we will assume a static variant of this game where the adversary $\mathcal{A}$ corrupts up to t nodes before Step 1. of the game.

Proof of IND1-Secrecy of our modified PVSS

Theorem 20 (IND1-Secrecy for sharing)**.** *Assuming that the hash function H is random oracle and that co-DBS assumption holds, the protocol in Fig. 2 achieves IND1-Secrecy for sharing against any $t-$bounded PPT adversary $\mathcal{A}$.*

Proof. We show that if a t-bounded static PPT adversary $\mathcal{A}$ has a non-negligible advantage $\varepsilon_{\mathcal{A}}$ in breaking the IND1-secrecy of our protocol in Fig. 2, then there exists a PPT adversary $\mathcal{A}_{DBS}$ that has a non-negligible advantage in breaking the co-DBS assumption.

The $\mathcal{A}_{DBS}$ simulates our modified sharing to $\mathcal{A}$ when given an instance of co-DBS $(g_1, g_2, u_1 = g_1^{\alpha}, u_2 = g_2^{\alpha}, v_1 = g_1^{\beta}, v_2 = g_2^{\beta}, T_b)$ as follows:

1. $\mathcal{A}_{DBS}$ sends $u_1 \in \mathbb{G}_1$ and $g_2, u_2 \in \mathbb{G}_2$ as the generators for the group.
2. The static adversary $\mathcal{A}$ corrupts up to t nodes and sends their public keys. WLOG, we assume that the corrupted nodes have indices $1 \leq j \leq t$.
3. $\mathcal{A}_{DBS}$ sends the public keys for the honest nodes $pk_i = g_1^{r_i}$ for $t+1 \leq i \leq n$, where $r_i \leftarrow \mathbb{Z}_q$. (This is equivalent to setting the secret key $sk_i = \alpha/r_i$.)
4. For $1 \leq i \leq t$, $\mathcal{A}_{DBS}$ sets $v_i = g_2^{r_i}$ and $c_i = pk_i^{r_i}$ without knowing β since it knows $v_2 = g_2^{\beta}$. For $t+1 \leq i \leq n$, $\mathcal{A}_{DBS}$ sets $\mathbf{v}$ using Lagrange interpolation of a polynomial using $v_2 = g_2^{\beta}$; and sets $c_i = v_i^{r_i}$.
5. For the NIZK proof NIZKPK, $\mathcal{A}_{DBS}$ chooses $r, c \leftarrow \mathbb{Z}_q$, sets $a_1 = c_i^c u_1^r$ and $a_2 = v_i^c g_2^r$ by simulating the random oracle and setting $H(u_1, u_2, c_i, v_i) = c$.
6. Finally, $\mathcal{A}_{DBS}$ sends T_b to $\mathcal{A}$.
7. $\mathcal{A}$ guesses a bit b'.

If $b' = 0$, $\mathcal{A}_{DBS}$ guesses that $b = 0$, i.e., that $T_b = T_1$ a random element in $\mathbb{G}_T$.

Observe that this is a secret sharing of $e(u_1^{\beta}, u_2) = e(g_1, g_2)^{\alpha^2\beta}$ and the information sent by $\mathcal{A}_{DBS}$ is distributed exactly like an actual secret sharing instance. When given T_0, the $\mathcal{A}$ can detect the correct sharing with non-negligible probability $\varepsilon_{\mathcal{A}}$, and with the same probability, $\mathcal{A}_{DBS}$ can make a correct guess with probability $\varepsilon_{\mathcal{A}}$.

Thus, if our scheme is not IND1-secret then we can break the co-DBS assumption, leading to a contradiction. □

C Postponed Cryptographic Proofs

C.1 Proof for Warm-up Beacon Security

We capture the security of our protocol in the following theorem:

Theorem 21 (Warm-up Beacon)**.** *Assuming RO and the co-DBS assumptions hold, the protocol in Fig. 2 is secure as per Definition 4.1.*

Proof. Weak Agreement. follows trivially due to the guarantees of the broadcast channel. The coding check can introduce a negligible probability that an honest party may accept an invalid share.

Validity. also follows from the construction, and correctness follows from existing works [13, 17].

Value-Validity Assume an adversary exists that can violate this property, i.e., it can make an honest node output a value that is not uniform. From the decomposition proof, we know that the final polynomial being shared (from which $\mathcal{B}$ is derived) contains contributions from at least one honest node whose input is uniformly random.

Any t-bounded adversary must select $t+1$ valid secret sharings. The final vector (from which an honest node outputs the beacon) must satisfy:

(i) *Discrete Log Equality.* For any $j \in [n]$, the combined commitment $v_j \in \mathbf{v}$ with respect to g_2 has the same discrete log as the combined encryption $c_j \in \mathbf{c}$ with respect to g_1.

(ii) *Unique degree-t polynomial.* With high probability ($1 - 1/q$, where q is the order of the groups), due to coding scheme used from SCRAPE [13], we know that the polynomial P encoded in $\mathbf{c}$ when reconstructed using any $t+1$ decryptions will reconstruct to a unique secret $S = e(g_1^s, g_2')$.

The only way an adversary can bias this distribution is by learning some information about the value shared by some honest node.

Let P be the polynomial in the commitments. We know that the (n,t) sharing is a valid degree $t+1$ polynomial, and that the product of $g_2^{p_j(0)}$ and t other values produce $g_2^{P(0)}$. We know that $P(0) = p_j(0) + X$, where X can be known by the adversary if it picks its own t sharings.

Let some adversary $\mathcal{A}$ construct $P(0)$ such that all the checks pass but $P(0)$ is not a function of some s_i, and $g_2^{s_i}$ knowledge proof along with t others are included. We know that $g_2^{s_i} \cdot g_2^{s_j} \cdots = g_2^{P(0)}$. So we get a contradiction that $P(0)$ is not a function of $g_2^{s_i}$. Intuitively, the only way to remove $g_2^{s_i}$ is by knowledge of it.

Since no $t-$bounded adversary can know $P(0)$, P which is a function of s_i is also unpredictable for any $t-$bounded adversary.

From Theorem 20, we know that this is not the case, leading to a contradiction. □

C.2 Proof for Beacon Unpredictability

Theorem 22 (Beacon Unpredictability)**.** *Assuming RO and the co-DBS assumptions hold, the protocol in Fig. 2 produces an unpredictable beacon.*

Proof. From Theorem 21, we know that in the broadcast channel world, it guarantees *Value validity*. In OptRand, we realize this and counter the weak-agreement property by making sure the beacon is constructed from a random sharing proposed by the same leader last time.

Formally, to prove security via reduction in Definition 3.2, any adversary that breaks the security of OptRand must violate synchrony, can only do so by breaking the cryptography, which remains secure with high probability.

Concretely, any adversary that wins the game defined in Definition 3.2 can be simulated to by a simulator to either break Theorem 20. When an adversary outputs b' it must be the case that it has learnt a sharing of one of the honest nodes, which we sue to win the co-DBS game (similar to the IND1-secrecy) simulation. □