RandPiper – Reconfiguration-Friendly Random Beacons with Quadratic Communication

Adithya Bhat *[1], Nibesh Shrestha *[2], Aniket Kate[3], and Kartik Nayak[4]

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

Abstract

Random beacon protocols provide a continuous public source of randomness and their applications range from public lotteries to zero-knowledge proofs. Existing random beacon protocols in the bounded synchronous model sacrifice either the fault tolerance or the communication complexity for security, or ease of reconfigurability. This work overcomes the challenges with the existing works through a novel communication efficient combination of state machine replication and (publicly) verifiable secret sharing (PVSS/VSS) protocols.

We first design a new Byzantine fault-tolerant state machine replication protocol with $O(\kappa n^2)$ bits communication per consensus decision without using threshold signatures. Next, we design GRandPiper (Good Pipelined Random beacon), a random beacon protocol with bias-resistance and unpredictability, that uses PVSS and has a communication complexity of $O(\kappa n^2)$ *always* (best and worst cases), for a static adversary. However, GRandPiper allows an adaptive adversary to predict beacon values up to $t+1$ epochs into the future. Therefore, we design BRandPiper (Better RandPiper), that uses VSS and has a communication complexity of $O(\kappa f n^2)$, where f is the *actual* number of faults, while offering a strong unpredictability with an advantage of only a single round even for an adaptive adversary.

1 Introduction

Public digital randomness is essential across a large spectrum of security applications ranging from e-voting to blockchains. Its practical relevance further evident from NIST's Randomness Beacons project [17] and from the recent emergence of Drand Organization [21]. Both these projects as well as several other proposals [13, 16, 20, 26, 27, 35, 36, 39] and implementations [1, 2, 25] offer *random beacons* [33].

A random beacon emits a new *random* value at intermittent intervals such that the emitted values are bias-resistant, i.e., no entity can influence a future random beacon value, and unpredictable, i.e., no entity can predict future beacon value. Clearly, we cannot trust a single node to offer such a service — the node can easily affect both bias-resistance and unpredictability of the beacon. A series of recent work [13, 27, 36, 39] has instead relied on distributing the trust across multiple nodes such that even if a subset of nodes gets compromised, the beacon is still secure.

In a system consisting of n nodes, tolerating t Byzantine faults, with security parameter κ, an ideal distributed randomness beacon protocol, in addition to being bias-resistant and unpredictable, should have the following properties: (i) optimal resilience, (ii) low communication overhead, (iii) reconfiguration friendly (allowing efficient addition and removal of nodes), and (iv) use efficient cryptographic schemes (as opposed to computationally expensive schemes such as Proof-of-Work (PoW) or Verifiable Delay Functions (VDFs)).

*Contributed equally and listed alphabetically

Existing works trade one or the other of the above features expected from a random beacon. For instance, HydRand [36] sacrifices optimal resilience ($t < n/3$) for better communication complexity ($O(\kappa n^2)$ in the best case and $O(\kappa n^3)$ in the worst case) with minimal setup assumptions. Cachin et al. [13] provides a communication complexity of $O(\kappa n^2)$, but it requires a threshold (cryptographic) setup and hence cannot support a reconfiguration of the system without changing the trusted setup through proactive secret sharing techniques. Several other solutions [20, 35] use computationally expensive mechanisms such as VDFs[1]where nodes compute VDF function constantly to ensure security of the beacon.

In this paper, we ask whether we can design an optimally resilient random beacon protocol that achieves good communication complexity while using efficient cryptographic schemes and a re-usable setup, i.e., avoiding setups such as those of threshold signatures where the entire setup needs to re-generated when a participating node is replaced. To answer this question, we first design an optimally resilient Byzantine fault-tolerant (BFT) state machine replication (SMR) protocol with $O(\kappa n^2)$ communication complexity per consensus decision while avoiding setups that do not allow reconfiguration. Next, we present two random beacon protocols GRandPiper (Good Pipelined Random beacon) and BRandPiper (Better Pipelined Random beacon) using our BFT SMR protocol as a building block and provide similar guarantees. GRandPiper is communication efficient ($O(\kappa n^2)$ in the best and worst case) but allows an adaptive adversary to predict $t + 1$ epochs into the future. BRandPiper offers stronger unpredictability guarantees (only 1 epoch into the future), but has a communication complexity of $O(\kappa f n^2)$ where f is the actual number of faults.

1.1 Efficient State Machine Replication without Threshold Signatures

There has been a long sequence of work in improving communication complexity of consensus protocols [3, 6, 11, 22, 29, 30, 40]. In the synchronous SMR setting, the optimal communication complexity per consensus decision of an SMR protocol is $O(\kappa n^2)$ bits [3, 5, 30, 38]. However, all of these solutions use threshold signatures. Our first result improves upon the communication complexity in the absence of threshold signatures. Specifically, we show the following:

Theorem 1.1. *There exists a state machine replication protocol with $O(\kappa n^2)$ communication complexity per consensus decision tolerating $t < n/2$ Byzantine faults without assuming threshold signatures.*

Our first result presents an optimally resilient BFT SMR protocol with quadratic communication per consensus decision without using threshold signatures in the synchronous setting. To be precise, the protocol incurs $O(\kappa n^2)$ communication complexity under q-SDH assumption (whose parameters can be generated using distributed protocols) or $O(\kappa n^2 \log n)$ without it. Getting rid of threshold signatures allows for efficient reconfiguration of the participating nodes and does not require generating threshold keys each time a new node joins the system. Thus, an efficient BFT protocol in this setting is of independent interest. We reduce communication by making use of efficient erasure coding schemes [34] and cryptographic accumulators [10] to efficiently broadcast large messages at the expense of increase in latency of SMR protocol. As we will see, the increase in latency does not affect our random beacon protocols adversely.

1.2 RandPiper – Random Beacon Protocols

RandPiper is a suite of random beacon protocols that use our SMR protocol as a building block. We present two protocols GRandPiper (Good Pipelined Random beacon) and BRandPiper (Better Pipelined Random beacon) which differ in unpredictability and communication complexity. In both protocols, we use secret sharing schemes to privately commit random numbers ahead of time. This ensures bias-resistance as the random number once shared cannot be changed. For unpredictability, we ensure that the beacon outputs are generated using inputs from $t + 1$ nodes (where t is the threshold of Byzantine nodes) at least one of which is truly random, and therefore the output is truly random.

[1]VDFs require nodes to always compute the VDF operation such as squarings, which are energy intensive. For importantly, VDFs require a precise estimation of how fast the fastest machine can compute the VDF operation (by CPUs, FPGAs, or ASICs), which is generally a hard thing to do [20]. A variation in this estimation gives an adversary using the faster hardware a better advantage.

GRandPiper protocol. In GRandPiper, we explore how to build a communication optimal random beacon protocol with bias-resistance and strong unpredictability, allowing a static adversary to predict up to a security parameter number of epochs into the future. In particular we show the following:

Theorem 1.2. *With n nodes and $t < n/2$ static corruptions, there exists a reconfiguration friendly, bias-resistant, and $O(\kappa)$-epoch unpredictable random beacon protocol with a communication complexity of $O(\kappa n^2)$ always.*

With n nodes out of which $t < n/2$ can be Byzantine, our GRandPiper protocol outputs a random beacon with $O(\kappa n^2)$ communication complexity per beacon output, where κ is the security parameter. The output of the beacon protocol is bias-resistant and it satisfies strong unpredictability against a static adversary, i.e., the probability of a static adversary predicting c rounds into the future is less than 2^{-c} (in expectation this is 2 rounds into the future). We also do not need any threshold setups, which allows nodes to join and leave the system easily without stopping our protocol.

At a high-level, our protocol uses Publicly Verifiable Secret Sharing (PVSS) schemes, and allows a leader to input an $O(\kappa n)$-sized PVSS encryption vector into the SMR to share a single secret per epoch. Our key insight in GRandPiper is to use extension techniques to deliver the $O(\kappa n)$-sized PVSS encryption vector (along with additional "votes") to *all the honest nodes* with communication complexity $O(\kappa n^2)$. Trivially, this would have incurred $O(\kappa n^3)$ where every node simply echoes the message to all the nodes. The secret shared by the leader will be reconstructed when the same node is chosen as the leader again. To ensure that eventually there is an honest leader, a leader does not repeat again for the next t epochs. This ensures that our BFT SMR protocol decides on the proposed shares once we get an honest leader. Our construction ensures that we *always* have a communication complexity of $O(\kappa n^2)$ for the beacon, as the beacon keeps outputting values based on buffered PVSS shares, and we remove Byzantine nodes to avoid the buffer from ever being empty.

However, an adaptive adversary can predict $t + 1$ rounds into the future in GRandPiper by simply corrupting the next t leaders and learning their committed secrets. At this point, continuing to use PVSS to improve the unpredictability leads to a loss of our nice property of optimal communication complexity. Hence, we look in a different direction to improve the unpredictability.

BRandPiper protocol. In BRandPiper protocol, we explore how to achieve the best possible unpredictability (1 epoch into the future), while having the best possible communication complexity and also supporting reconfiguration. In particular, we show the following result:

Theorem 1.3. *With n nodes and $t < n/2$ adaptive corruptions and with κ as the security parameter, there exists a reconfiguration-friendly, bias-resistant and 1-epoch unpredictable random beacon protocol with a communication complexity of $O(\kappa f n^2)$, where $f \leq t$ is the actual number of faults.*

With n nodes out of which $t < n/2$ allowed to be Byzantine, our second protocol BRandPiper outputs a random beacon with $O(\kappa f n^2)$ communication complexity per output, and guarantees bias-resistance and strong unpredictability. Here, f is the actual number of faults; when $f = O(1)$, the protocol enjoys $O(\kappa n^2)$ communication complexity. BRandPiper protocol uses random inputs from $> t$ nodes in every epoch, ensuring strong unpredictability of only 1 epoch into the future.

As a building block, we first construct an improved VSS (iVSS) protocol by modifying the state-of-the-art VSS scheme eVSS [28]. Compared to eVSS, which requires $O(\kappa f n)$ information on the bulletin board, iVSS posts only $O(\kappa n)$ bits of information on the bulletin board which in effect improves the amortized communication complexity of the VSS scheme to $O(\kappa f n^2)$ where f is the *actual* number of faults and κ is the security parameter. This may be of independent interest to applications requiring batched VSS sharings.

At a high level, we use round-robin leaders and iVSS in point-to-point channels to secret share n random numbers in an epoch. Since we are producing n shares every epoch, we can now consume n shares in every epoch. Thus, in every epoch, using the homomorphic properties of VSS secret shares, we reconstruct a homomorphic sum of n shares in every epoch, thus eliminating the $t+1$ epoch advantage held by the adaptive adversary and reducing it to just 1 epoch. We carefully design the protocol so that we have a communication complexity of $O(\kappa f n^2)$. Our key insight in BRandPiper is that a leader can secret share n shares at once instead of one. These shares are buffered by all nodes, and it ensures that there are always sufficient shares available for reconstruction in the next n epochs so far as leaders are chosen in a round-robin manner. The buffering helps prevent a Byzantine node from biasing by refusing to share new blocks, when the outputs are unfavorable. Without our techniques, while assuming threshold signatures, existing VSS protocols have an optimistic communication complexity of $O(\kappa n^2)$ and a worst case communication complexity of $O(\kappa n^3)$ to perform 1 secret sharing. BRandPiper protocol shows how to perform $O(n)$ VSS secret sharings with a communication complexity of $O(\kappa f n^2)$ which is quadratic when $f = O(1)$.

1.3 Related Work

In this section, we explore random beacon protocols, sometimes also referred to as coin tossing protocols, in the synchronous setting. Some of the works were originally designed for the asynchronous settings, but in this section, we evaluate them in the bounded synchronous setting.

Cachin et al. [13] use a threshold shared secret to generate beacons by creating a share of the beacon by binding the shared secret to an agreed upon beacon value (for instance, a counter). When $> t$ such shares are obtained, all honest nodes obtain the same beacon value. Drand [21] uses a similar approach replacing the threshold secret with a threshold BLS key and using signatures on the beacon value (say, the counter). This incurs a communication complexity of $O(\kappa n^2)$ always, but comes with the drawback that it does not support reconfiguration, i.e., if a single node is replaced or joins the system, the threshold shared secret must be regenerated.

Homomorphic Encryption Random Beacon (HERB) [16] uses homomorphic ElGamal encyrption scheme to generate random numbers. The system tolerates $n > 3t$ faults. Each node in the system encrypts a random share and posts it on the bulletin board. The protocol uses $t + 1$ such encryptions to produce the final beacon output. The work requires the use of a Byzantine Agreement protocol whose inputs are $O(\kappa n)$ sized, and therefore has a communication complexity of $O(\kappa n^3)$ in the best and worst cases. It also uses a variant of threshold setup, thereby not permitting a re-usable setup.

RandChain [26] builds a DRB - Decentralized Random Beacon that assumes sequential Proof-of-Work (Seq-PoW), and VDFs along with Nakamoto consensus for consistency. Constructions using these assumptions are not energy-efficient. In PoW, an adversary with more hash power can neglect unfavorable random numbers by forking, and to avoid this requires the total honest hash power to be greater than $1/2$. The work uses existing Byzantine Agreement techniques which makes the protocol have a communication complexity of $O(\kappa n^2)$ in the best case, while inheriting the $O(\kappa n^3)$ communication complexity from BA [18] in the worst case.

Drake *et al.* [20] proposed a minimal bias-resistant VDF-based random beacon scheme, that assumes the existence of a VDF and that the adversary has an advantage A_{max} in terms of speed over the honest nodes, in computing the VDF. The VDF is used to determine the beacon output for a round, and sufficiently old beacon outputs are used to select leaders for the Ethereum Proof-of-Stake protocol. The system tolerates $n > 3t$ faults, and is designed for partial synchrony.

RandRunner [35] builds a random beacon protocol using VDFs. Therefore, it has a setup that can be re-used. It 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. It is not immediately unpredictable as an adaptive adversary can corrupt the next $t < n/2$ leaders to know the outputs for the next t epochs. RandRunner also requires the usage of reliable broadcasts which incurs a communication complexity of $O(\kappa n^2)$ for every round.

RandShare [39] is a strawman protocol for where the beacon output is generated using n independent Byzantine Agreement instances are run in a system tolerating $n > 3t$ faults. Running n BA instances incurs a communication complexity of $O(\kappa n^3)$ in the best case and $O(\kappa n^4)$ in the worst case. RandHerd [39] is an improved version of RandShare, driven by a client seeking a random beacon value. The client splits the system into groups of size c which internally use RandShare, leading to a communication complexity of $O(\kappa c^2 n)$, even in the worst case. However, even though c is a constant, it depends on n as the randomness of the beacon output is determined by c. RandHound [39] goes beyond RandHerd by using a stable-leader

Table 1: **Comparison of related works on Random Beacon protocols in standard synchrony**

Protocol		**Res.(t)**	**Unpred.**	**Comm. Compl**		**Adp. Adv.**	**Re-usable Setup**	**No DKG?**	**Special Assumption**
				Best	**Worst**				
Cachin et al./Drand	[13, 21]	49%	1	$O(\kappa n^2)$	$O(\kappa n^2)$	✗	✗	✗	Threshold Secret/BLS
Dfinity	[4, 27]	49%	$O(\kappa)$	$O(\kappa n^2)$	$O(\kappa n^3)^*$	✓	✗	✗	Threshold BLS
HERB	[16]	33%	1	$O(\kappa n^3)$	$O(\kappa n^3)$	✓	✗	✗	-
HydRand	[36]	33%	$O(\min(\kappa, t))^\dagger$	$O(\kappa n^2)$	$O(\kappa n^3)$	✗	✓	✓	-
HydRand	[36]	33%	$t+1$	$O(\kappa n^2)$	$O(\kappa n^3)$	✓	✓	✓	-
RandChain	[26]	33%	$O(\kappa)$	$O(\kappa n^2)$	$O(\kappa n^3)$	✓	✗	✗	PoW
RandHerd	[39]	33%	$O(\kappa)$	$O(\kappa c \log n)^\P$	$O(\kappa n^4)$	✓	✗	✗	Threshold Schnorr
RandHound	[39]	33%	1	$O(\kappa c^2 n)^\P$	$O(\kappa c^2 n^2)^\P$	✓	✓	✓	Client based
RandRunner	[35]	49%	$t+1$	$O(\kappa n^2)$	$O(\kappa n^2)$	✓	✓	✓	VDF
RandShare	[39]	33%	1	$O(\kappa n^3)$	$O(\kappa n^4)$	✓	✓	✓	-
GRandPiper (Single)		49%	$O(\min(\kappa, t))^\dagger$	$O(\kappa n^2)$	$O(\kappa n^2)$	✗	✓	✓	-
GRandPiper (Single)		49%	$t+1$	$O(\kappa n^2)$	$O(\kappa n^2)$	✓	✓	✓	-
GRandPiper (Multiple)‡		49%	$o(\log(t/p))$	$O(p\kappa n^2)$	$O(p\kappa n^2)$	✓	✓	✓	-
BRandPiper		49%	1	$O(\kappa n^2)^\S$	$O(\kappa n^3)$	✓	✓	✓	-

κ is the security parameter denoting maximum of sizes of signatures, hashes, and other components used in the protocol. **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 an adaptive rushing adversary can predict. A rushing adversary can always obtain outputs before correct nodes, and hence, the minimum is 1. **Adp. Adv** refers to *Adaptive Adversary* whether the adversary can pick its t corruptions at any point in the protocol. **Reusable Setup** refers to a setup that can be reused when a node is replaced in the system. *probabilistically $O(\kappa n^3)$ when $O(n)$ consecutive leaders are bad. $^\P c$ is the average (constant) size of the groups of server nodes. †In expectation it is 2 rounds, the probability of an adversary predicting c epochs into the future is 2^{-c}, with a security parameter κ it is $\min(\kappa, t) + 1$ epochs. ‡Run p instances of GRandPiper in parallel. §In the optimistic case, when the leader is honest and $f = O(1)$ nodes are Byzantine.

approach and dividing the system into groups of nodes with group leaders in a tree structure during the setup. This incurs RandHound, a communication complexity of $O(\kappa c^2 \log n)$ when the leader and the group leaders are honest. However, when the leader is bad, it uses a *view-change* protocol which is analogous to Byzantine Agreement, and incurs a cost of $O(\kappa n^3)$ communication complexity when t consecutive leaders are Byzantine.

HydRand [36] is a random beacon protocol in the bounded synchrony model which tolerates t faults, with $n > 3t$, with a communication complexity of $O(\kappa n^2)$ in bits, where κ is the security parameter, and $O(\kappa n^3)$ communication complexity in the worst case. It uses PVSS schemes (in particular SCRAPE [14]) and tolerates an adaptive adversary which can predict upto $t + 1$ epochs into the future.

Summary of limitations. Before, we set out to show what we achieve in this work, we show what the current works show what is achievable and what is not currently possible. The state of the art SMR protocols [3,5,7,40] have a lower bound of $O(\kappa n^2)$ on the communication complexity in the best case, which hints that we cannot do better than $O(\kappa n^2)$ without improving SMR first. State of the art random beacon protocols [13,16,39] show that we cannot achieve an unpredictability advantage better than 1 epoch, since a rushing adversary can always know one epoch output before the rest of the honest nodes. State of the art Random beacon protocols [13,16,39] also show that a random beacon not in lock-step cannot avoid giving a time advantage of less than 2Δ to a rushing adversary. Our work aims to bridge these gaps in existing random beacon protocols.

Insights from existing works. RandPiper uses some insights from HydRand [36] and non-trivially improves upon them for optimal fault tolerance ($t < n/2$ unlike $t < n/3$ from HydRand) and better communication complexity (recall that HydRand has a communication complexity of $O(\kappa n^2)$ in the best case and $O(\kappa n^3)$ in the worst case). We first observe that HydRand secret shares one value and uses this shared value the next time the same node becomes a leader again. We observe that this is buffering of shares, and that this buffering can be done for more than one share, i.e., every time a node becomes a leader, we can use the value from the last d^{th} time it was a leader to buffer d shares. Next, we observe that HydRand cannot tolerate more than $t > n/3$ because it fails to deliver the PVSS vector to all correct nodes, if the leaders fails to send it to them. We solve this concern using extension techniques from recent works [31]. However, these works assume threshold signatures which we avoid in our protocol. Thus, in RandPiper, we achieve an optimal fault tolerance of $t < n/2$ and also improved communication complexity.

2 Model and Definitions

We consider a system $\mathcal{P} := \{p_1, \ldots, p_n\}$ consisting of n nodes out of which at most $t < n/2$ nodes can be Byzantine. The Byzantine nodes may behave arbitrarily. When we assume an adaptive adversary $\mathcal{A}$, the nodes can be corrupted to behave arbitrarily at any time during execution. When we assume a static adversary $\mathcal{A}$, the nodes to be corrupted must be chosen by the adversary before the start of the protocol. A node that is not faulty throughout the execution is considered to be honest and executes the protocol as specified.

We assume the network between nodes consists of point-to-point secure (authenticated and confidential) bounded synchronous channels. Messages between nodes may take at most Δ time before they arrive, where Δ is a known maximum network delay. To provide safety under adversarial conditions, we assume that the adversary is capable of delaying the message for an arbitrary time upper bounded by Δ. In addition, 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 [3] 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 is digitally signed by p's private key and is denoted by $\langle x \rangle_p$. In addition, we use $H(x)$ to denote the invocation of the random oracle H on input x.

2.1 Definitions

We consider a state machine replication protocol defined as follows:

Definition 2.1 (Byzantine Fault-tolerant State Machine Replication [37]). *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.*

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

We define d-absolute unpredictability as follows:

Definition 2.2 (d-absolute unpredictability). *Consider an epoch based protocol. Let the honest nodes be at epoch e. The protocol is said to be unpredictable with absolute bound d for $d \geq 1$, if the the probability of an adversary $\mathcal{A}$ predicting the honest output for epoch $e + d$ is a negligible function $\mathsf{negl}(\kappa)$.*

We define the security requirements for a random beacon protocol $\mathcal{RB}$ as follows:

Definition 2.3 (Secure random beacon protocol). *An epoch based protocol $\mathcal{RB}$ is said to be a d-secure random beacon protocol if it satisfies the following conditions:*

1. ***Bias-resistance.*** *Let $\mathcal{O}$ be the output of the beacon for some epoch e. No t-bounded adversary $\mathcal{A}$ can bias the output of the beacon, i.e., fix c bits of $\mathcal{O}$ for any epoch $e > 1$ with probability better than $\frac{1}{2^c} + \mathsf{negl}(\kappa)$.*
2. ***Unpredictability.*** *The protocol satisfies d-unpredictability.*
3. ***Guaranteed Output Delivery.*** *For every epoch $e \geq 1$, the protocol outputs a value.*

2.2 Primitives

In this section, we present several primitives used in our protocols.

Linear erasure and error correcting codes. We use standard $(t + 1, n)$ Reed-Solomon (RS) codes [34]. This code encodes $t + 1$ data symbols into codewords of n symbols and can decode the $t + 1$ elements of codewords to recover the original data.

- ENC. Given inputs $m_1, \ldots, m_{t+1}$, an encoding function ENC computes $(s_1, \ldots, s_n) = \mathsf{ENC}(m_1, \ldots, m_{t+1})$, where $(s_1, \ldots, s_n)$ are codewords of length n. A combination of any $t + 1$ elements of n codewords uniquely determines the input message and the remaining of the codeword.
- DEC. The function DEC computes $(m_1, \ldots, m_{t+1}) = \mathsf{DEC}(s1, ..., s_n)$, and is capable of tolerating up to c errors and d erasures in codewords $(s_1, \ldots, s_n)$, if and only if $t \geq c + d$.

Cryptographic accumulators. The cryptographic accumulator constructs an accumulation value for a set of values and produces witness for each value in the set. Given the accumulation value and a witness, any node can verify if a value is indeed in the set. Formally, given a parameter k, and a set D of n values $d_1, \ldots, d_n$, an accumulator has the following components:

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

- $\mathsf{CreateWit}(a_k, z, d_i)$: This algorithm takes an accumulator key a_k, 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 $d_i \in \mathcal{D}$.
- $\mathsf{Verify}(a_k, z, w_i, d_i)$: This algorithm takes an accumulator key a_k, 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.

We use cryptographic accumulators proposed by Barić and Pfitzmann [10].

Lemma 2.1 (Collision-free accumulator [32])**.** *The bilinear accumulator is collision-free. That is, for any set size n, there is only a probability negligible in k for a probabilistic polynomial-time adversary to find an accumulator key a_k, a set $\mathcal{D} = \{d_1, \ldots, d_n\}$, an accumulation value z for $\mathcal{D}$, a value $d' \notin \mathcal{D}$, and a witness w' such that* $\mathsf{Verify}(a_k, z, w', d') = \mathit{true}$*.*

Publicly Verifiable Secret Sharing — PVSS. PVSS schemes consist of communication such as broadcasts, posts on the bulletin board, as well as computational components such as share generation, encryption, etc. We separate the two components and present interfaces to computational algorithms that we will use in our protocols. We use the interfaces to a secure PVSS scheme $\mathcal{PVSS}$ as described in Table 2.

Table 2: **PVSS scheme algorithm interface**

Interface	**Description**
$\mathcal{PVSS}.\mathsf{pp} \leftarrow \mathcal{PVSS}.\mathsf{Setup}(\kappa, \mathsf{aux})$	Generate the scheme parameters $\mathcal{PVSS}.\mathsf{pp}$. $\mathcal{PVSS}.\mathsf{pp}$ is an implicit input to all other algorithms.
$(\mathcal{PVSS}.\mathsf{pk}, \mathcal{PVSS}.\mathsf{sk}) \leftarrow \mathcal{PVSS}.\mathsf{KGen}(\kappa)$	Generate PVSS key-pair $(\mathcal{PVSS}.\mathsf{pk}, \mathcal{PVSS}.\mathsf{sk})$ used for share encryption and decryption
$c \leftarrow \mathcal{PVSS}.\mathsf{Enc}(\mathcal{PVSS}.\mathsf{pk}, m)$ $m \leftarrow \mathcal{PVSS}.\mathsf{Dec}(\mathcal{PVSS}.\mathsf{sk}, c)$	The encryption and decryption algorithms used to send shares to all, and obtain private share respectively. The invariant $\Pr[\mathcal{PVSS}.\mathsf{Dec}(\mathcal{PVSS}.\mathsf{Enc}(m)) = m] = 1$ must always hold true for all m in the message domain of $\mathcal{PVSS}.\mathsf{Enc}$.
$(\mathcal{PVSS}.\vec{S}, \mathcal{PVSS}.\vec{E}, \mathcal{PVSS}.\pi) \leftarrow \mathcal{PVSS}.\mathsf{ShGen}(s)$	Typically, executed by the dealer L with secret s to generate secret share vector $\mathcal{PVSS}.\vec{S} := \{\mathcal{PVSS}.\mathsf{s}_1, \ldots, \mathcal{PVSS}.\mathsf{s}_n\}$ and encryption vector of shares $\mathcal{PVSS}.\vec{E} := \{\mathcal{PVSS}.\mathsf{Enc}(\mathcal{PVSS}.\mathsf{s}_1), \ldots, \mathcal{PVSS}.\mathsf{Enc}(\mathcal{PVSS}.\mathsf{s}_n)\}$ for all nodes $\mathcal{P}$, and a cryptographic proof $\mathcal{PVSS}.\pi$ committing to s which guarantees any node with $> t$ shares reconstruct a unique s.
$\{0,1\} \leftarrow \mathcal{PVSS}.\mathsf{ShVrfy}(\mathcal{PVSS}.\vec{E}, \mathcal{PVSS}.\pi)$	Verify if the sharing is correct. A successful verification guarantees that its share is correct and $t+1$ nodes reconstruct a unique s. 0 indicates a failure and 1 indicates a success.
$s \leftarrow \mathcal{PVSS}.\mathsf{Recon}(\mathcal{PVSS}.\vec{S})$	Reconstruct the shared secret s from the collection of shares $\mathcal{PVSS}.\vec{S} \subseteq \{\mathcal{PVSS}.\mathsf{s}_1, \ldots, \mathcal{PVSS}.\mathsf{s}_n\}^{t+1}$

We assume the existence of a secure PVSS algorithm $\mathcal{PVSS}$ as defined in Definition 2.4.

Definition 2.4 (PVSS security)**.** *Let $L \in \mathcal{P}$ be the dealer with secret s and κ be the security parameter. A PVSS scheme $\mathcal{PVSS}$ must provide the following guarantees:*

1. ***Secrecy.*** *If the dealer L is honest, then the adversary's view during the sharing phase reveals no information about the dealer's secret s with probability better than* $\mathsf{negl}(\kappa)$*.*

2. ***Correctness.*** *If L is honest, then the honest nodes output the secret s at the end of the reconstruction phase with high probability $1 - \mathsf{negl}(\kappa)$.*

3. ***Commitment.*** *If L is Byzantine, then at the end of the sharing phase there exists a value $s^\star$ in the input space including $\perp$, such that at the end of the reconstruction phase all honest nodes output $s^\star$ with high probability $1 - \mathsf{negl}(\kappa)$.*

4. ***(Public) Verifiability.*** *If the check in verification algorithm returns 1, i.e., succeeds, then with high probability $1 - \mathsf{negl}(\kappa)$, the encryptions are valid shares of some secret.*

Verifiable Secret Sharing and Commitments. Similar to the PVSS schemes, we use the interfaces to a secure VSS scheme $\mathcal{VSS}$ as described in Table 3.

Table 3: **VSS scheme algorithm interface**

Interface	**Description**
$\mathcal{VSS}.\mathsf{pp} \leftarrow \mathcal{VSS}.\mathsf{Setup}(\kappa, \mathsf{aux})$	Generate the scheme parameters $\mathcal{VSS}.\mathsf{pp}$. $\mathcal{VSS}.\mathsf{pp}$ is an implicit input to all other algorithms.
$(\mathcal{VSS}.\vec{S}, \mathcal{VSS}.\vec{W}, \mathcal{VSS}.\mathsf{C}) \leftarrow \mathcal{VSS}.\mathsf{ShGen}(s)$	Typically executed by the dealer L with secret s to generate a secret share vector $\mathcal{VSS}.\vec{S} := \{\mathcal{VSS}.\mathsf{s}_1, \ldots, \mathcal{VSS}.\mathsf{s}_n\}$, witness vector $\mathcal{VSS}.\vec{W} := \{\mathcal{VSS}.\pi_1, \ldots, \mathcal{VSS}.\pi_n\}$, and a constant size commitment $\mathcal{VSS}.\mathsf{C}$ to a Shamir polynomial.
$\{0,1\} \leftarrow \mathcal{VSS}.\mathsf{ShVrfy}(\mathcal{VSS}.\mathsf{s}, \mathcal{VSS}.\pi, \mathcal{VSS}.\mathsf{C})$	Verify if the share $\mathcal{VSS}.\mathsf{s}$ and witness $\mathcal{VSS}.\pi$ form a correct share for $\mathcal{VSS}.\mathsf{C}$. 0 indicates a failure and 1 indicates a success.
$s \leftarrow \mathcal{VSS}.\mathsf{Recon}(\mathcal{VSS}.\vec{S})$	Reconstruct the shared secret s from the collection of shares $\mathcal{VSS}.\vec{S} \subseteq \{\mathcal{VSS}.\mathsf{s}_1, \ldots, \mathcal{VSS}.\mathsf{s}_n\}^{t+1}$.
$\{0,1\} \leftarrow \mathcal{VSS}.\mathsf{ComVrfy}(\mathcal{VSS}.\mathsf{C}, s)$	Check if s is the correct opening for the commitment $\mathcal{VSS}.\mathsf{C}$. 0 indicates a failure, and 1 indicates a success.

We assume the existence of a secure Verifiable secret sharing scheme with commitments $\mathcal{VSS}$ with the following security properties:

Definition 2.5 (VSS Security). *We call an $n-$node VSS protocol, with $t-$bounded adversary $\mathcal{A}$ and security parameter κ, an $(n-t)$-VSS protocol if it satisfies the following conditions:*

1. ***Secrecy.*** *If the dealer L is honest, then the probability of $\mathcal{A}$ learning any information about the dealer's secret s in the sharing phase is $\mathsf{negl}(\kappa)$.*

2. ***Correctness.*** *If L is honest, then the honest nodes output the secret s at the end of the reconstruction phase with a high probability of $1 - \mathsf{negl}(\kappa)$.*

3. ***Commitment.*** *If L is Byzantine, then at the end of the sharing phase there exists a value $s^\star$ in the input space including $\perp$, such that at the end of the reconstruction phase all honest nodes output $s^\star$ with high probability $1 - \mathsf{negl}(\kappa)$.*

In our work, we implicitly assume that the VSS scheme used is $(n - n/2)$-secure.

Normalizing the length of cryptographic building blocks. Let λ denote the security parameter, $\kappa_h = \kappa_h(\lambda)$ denote the hash size, $\kappa_a = \kappa_a(\lambda)$ denote the size of the accumulation value and witness of the accumulator and $\kappa_v = \kappa_v(\lambda)$ denote the size of secret share and witness of a secret. Further let $\kappa = \mathsf{max}(\kappa_h, \kappa_a, \kappa_v)$; we assume $\kappa = \Theta(\kappa_h) = \Theta(\kappa_v) = \Theta(\kappa_a) = \Theta(\lambda)$. Throughout the paper, we can use the same parameter κ to denote the hash size, signature size, accumulator size and secret share size for convenience.

3 BFT SMR Protocol

In this section, we present a simple BFT SMR protocol as a basic building block for the beacon protocols discussed in following sections. Our protocol achieves $O(\kappa n^2)$ bits communication complexity with a trusted setup under q-SDH assumption, or $O(\kappa n^2 \log n)$ bits communication complexity without a trusted setup assumption. In particular, we do not use threshold signatures, and therefore do not require any distributed key generation during the setup. We note that prior synchronous BFT SMR protocols [5,15,38] with honest majority incur $O(\kappa n^3)$ communication complexity for committing each block without threshold signatures.

Before we explain how our protocol achieves $O(\kappa n^2)$ (or $O(\kappa n^2 \log n)$) communication complexity, we first present some preliminaries pertaining to our SMR protocol.

Epochs. Our protocol progresses through a series of numbered *epochs* with each epoch coordinated by a distinct leader. Epochs are numbered by non-negative integers starting with 1. The leaders for each epoch are rotated irrespective of the progress made in each epoch. For simplicity, we use round-robin leader election in this section and the leader of epoch e, represented as L_e, is determined by $e \bmod n$. Later in the beacon protocols, we introduce different leader election rules. Each epoch lasts for 11Δ time.

Blocks and block format. The goal of the leader in each epoch is to add a block to the blockchain. Each block references its predecessor with the exception of the genesis block which has no predecessor. We call a block's position in the chain 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, B_{h-1} is the block at height $h-1$ and $H(B_{h-1})$ is the hash digest of B_{h-1}. The predecessor for the genesis block is $\perp$. 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 meets the application-level validity conditions. 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 on a block B_h consists of $t+1$ distinct signatures in an epoch e and is represented by $\mathcal{C}_e(B_h)$. Block certificates are ranked by epochs, i.e., blocks certified in a higher epoch has a higher rank. During the protocol execution, each node keeps track of all certified blocks and keeps updating the highest 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 and vote-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_e.

How does our protocol achieve $O(\kappa n^2)$ (or $O(\kappa n^2 \log n)$) communication complexity per epoch? Prior BFT SMR protocols perform all-to-all broadcast of certificates to ensure *totality* between nodes or to synchronize all nodes across epochs. In general, a certificate consists of a quorum of $n-t$ signatures which cannot be compressed without the use of threshold signatures. A naïve approach of performing all-to-all broadcast of $O(\kappa n)$-sized certificates trivially incurs $O(\kappa n^3)$ signatures of communication.

We make use of a combination of erasure-coding schemes and cryptographic accumulators to efficiently broadcast a message. To broadcast a message of size ℓ, this technique incurs $O(n\ell + (\kappa + w)n^2)$ bits where κ is the size of accumulator and w is the size of the accumulator *witness*. The size of witness is $O(\kappa)$ when bilinear accumulators are used. If one wishes to avoid bilinear accumulators, we can use Merkle trees as witness. The size of Merkle witness is logarithmic in the size of the number of witnesses (n). Thus, the total communication complexity to broadcast a single message of size ℓ is $O(n\ell + \kappa n^2)$ bits, or $O(n\ell + \kappa n^2 \log n)$

bits without trusted setup assumption.

As mentioned before, the size of a certificate is linear and broadcasting a certificate using the above technique yields a communication complexity of $O(kn^2)$ bits (or $O(kn^2 \log n)$ bits). However, in general, there can be exponentially many combinations of $\geq n-t$ signatures forming a certificate depending on the set of signers, and each node may broadcast a different combination. This will trivially blowup communication by a factor of n.

In order to address the issue, we need to ensure there is a single certificate for a block. We ensure this by using leader as a relay to collect signatures and form a single certificate. The leader forwards this certificate to all nodes. The next challenge is the relay can be Byzantine and forward the certificate only to a subset of nodes. We resolve this issue by having honest nodes forward their shares along with the witness. As a result, if an honest node receives a valid certificate, all honest nodes receive the corresponding share. Another challenge is the relay can form multiple certificates using different sets of signatures and send different certificate to different nodes. Our protocol treats such behaviors as faulty; we require the leader to send a signed hash, and in case of equivocation, we broadcast equivocating hashes signed by the leader. All-to-all broadcasting of size κ hashes incurs only $O(\kappa n^2)$ in communication. equivocating certificates are formed before making a decision. Broadcasting a single certificate in this way reduces the complexity to $O(\kappa n^2)$ (or $O(n^2 \kappa \log n)$) bits. Several prior works [24, 40] have used relay based message-distribution along with threshold-signatures to reduce the message complexity. Here, we reduce the communication complexity with extension techniques [31].

3.1 Protocol Details

We first describe a simple function that is used by an honest node to forward a message received from the epoch leader.

Deliver function. The Deliver() function (refer Figure 2) implements efficient broadcast of long messages using erasure coding techniques and cryptographic accumulators. The input parameters to the function are a keyword key, long message b, accumulation value z_e corresponding to object b and epoch e in which deliver function is invoked. The input keyword key corresponds to message *type* containing large object b sent by leader L_e of epoch e. In order to facilitate efficient leader equivocation, the input keyword key, hash of object b, accumulation value z_e and epoch e are signed by leader L_e.

When the function is invoked using above input parameters, the message b is first divided into n coded shares $(s_1, \ldots, s_n)$ using RS codes and a cryptographic witness w_j is computed for each value s_j. Then, the share (s_j, w_j) is sent to the node $p_j \in \mathcal{P}$ along with the accumulation value z_e, keyword key, L_e's signature on the message. When a node p_j receives the first valid shares for an accumulation value z_e such that the witness w_j verifies the share s_j, it forwards the share (s_j, w_j) to all nodes. When a node p_i receives $n-t$ valid shares corresponding to the first accumulation value z_e it receives, it reconstructs the object b. Note that node p_i reconstructs object b for the first valid share even though it detects equivocation in an epoch.

Our protocol (refer Figure 1) executes following steps in each epoch e.

Epoch advancement. When epoch-timer_{e-1} expires, node p_i enters epoch e. After entering epoch e, node p_i sends its highest ranked certificate $\mathcal{C}_{e'}(B_l)$ to leader L_e. In addition, it also sets epoch-timer_e to 11Δ and starts counting down.

Propose. After entering epoch e, Leader L_e waits for 2Δ to collect highest ranked certificates from all honest nodes. The 2Δ wait before a proposal ensures that leader L_e can collect highest ranked certificates from all honest nodes when leader L_e enters epoch e Δ time before other honest nodes.[2] After the 2Δ wait, leader L_e proposes a block B_h by extending the highest ranked block certificate $\mathcal{C}_e(B_l)$ known to L_e. The proposal for B_h, conceptually, has the form $\langle \text{propose}, B_h, e, \mathcal{C}_{e'}(B_l), z_{pe} \rangle_{L_e}$ where z_{pe} is the accumulation value for the pair $(B_h, \mathcal{C}_{e'}(B_l))$. In order to facilitate efficient equivocation checks, the leader signs the tuple $\langle \text{propose}, H(B_h, \mathcal{C}_{e'}(B_l)), e, z_{pe} \rangle_{L_e}$ and sends B_h and $\mathcal{C}_{e'}(B_l)$ separately. This size of the signed message

[2] We assume that all honest nodes enter the epoch and perform all steps within Δ time of each other. This can be achieved using a clock synchronization protocol [3] only once at the start of the execution.

Let e be the current epoch and L_e be the leader of epoch e. For each epoch e, node p_i performs following operations:

1. **Epoch advancement.** When $\textsf{epoch-timer}_{e-1}$ reaches 0, enter epoch e. Upon entering epoch e, send highest ranked certificate $\mathcal{C}_{e'}(B_l)$ to L_e. Set $\textsf{epoch-timer}_e$ to 11Δ and start counting down.

2. **Propose.** L_e waits for 2Δ time after entering epoch e and broadcasts $\langle \textsf{propose}, B_h, e, \mathcal{C}_{e'}(B_l), z_{pe}\rangle_{L_e}$ where B_h extends B_l. $\mathcal{C}_{e'}(B_l)$ is the highest ranked certificate known to L_e.

3. **Vote.** If $\textsf{epoch-timer}_e \geq 7\Delta$ and node p_i receives the first proposal $p_e = \langle \textsf{propose}, B_h, e, \mathcal{C}_{e'}(B_l), z_{pe}\rangle_{L_e}$ where B_h extends a highest ranked certificate, invoke $\textsf{Deliver}(\textsf{propose}, p_e, z_{pe}, e)$. Set $\textsf{vote-timer}_e$ to 2Δ and start counting down. When $\textsf{vote-timer}_e$ reaches 0, send $\langle \textsf{vote}, H(B_h), e\rangle_i$ to L_e.

4. **Vote cert.** Upon receiving $t+1$ votes for B_h, L_e broadcast $\langle \textsf{vote-cert}, \mathcal{C}_e(B_h), e, z_{ve}\rangle_{L_e}$.

5. **Commit.** If $\textsf{epoch-timer}_e \geq 3\Delta$ and node p_i receives the first $v_e = \langle \textsf{vote-cert}, \mathcal{C}_e(B_h), e, z_{ve}\rangle_{L_e}$, invoke $\textsf{Deliver}(\textsf{vote-cert}, v_e, z_{ve}, e)$. Set $\textsf{commit-timer}_e$ to 2Δ and start counting down. When $\textsf{commit-timer}_e$ reaches 0, if no equivocation for epoch-e has been detected, commit B_h and all its ancestors.

6. **(Non-blocking) Equivocation.** Broadcast equivocating hashes signed by L_e and stop performing epoch e operations.

Figure 1: **BFT SMR Protocol** with $O(kn^2)$ bits communication per epoch and optimal resilience.

$\textsf{Deliver}(\textsf{key}, b, z_e, e)$:

- Send $\langle \textsf{share}, \textsf{key}, s_j, w_j, z_e, e\rangle_i$ with shares computed from b to nodes p_j $\forall j \in [n]$.
- If node p_j receives the first valid share $\langle \textsf{share}, \textsf{key}, s_j, w_j, z_e, e\rangle_*$ for the accumulator z_e, send the share to all the nodes.
- Upon receiving $n-t$ valid shares for the accumulator z_e, decode b.

Figure 2: **Deliver function**

is $O(\kappa)$ and hence can be broadcast during equivocation or while delivering p_e without incurring cubic communication overhead.

Vote. If a node receives the first valid proposal $p_e = \langle \mathsf{propose}, B_h, e, \mathcal{C}_{e'}(B_l), z_{pe} \rangle_{L_e}$ when $\mathsf{epoch\text{-}timer}_e \geq 7\Delta$ and block B_h extends the highest ranked certificate known to the node, it invokes $\mathsf{Deliver}(\mathsf{propose}, p_e, z_{pe}, e)$. In addition, the node sets its $\mathsf{vote\text{-}timer}_e$ to 2Δ and starts counting down. When $\mathsf{vote\text{-}timer}_e$ reaches 0 and detects no epoch e equivocation, the node sends $\langle \mathsf{vote}, H(B_h), e \rangle_i$ to L_e. If block B_h does not extend the highest ranked certificate known to the node or receives proposal p_e when its $\mathsf{epoch\text{-}timer}_e < 7\Delta$, the node simply ignores the proposal and does not vote for B_h.

Vote certificate. When leader L_e receives $t+1$ vote messages for the proposed block B_h, denoted by $\mathcal{C}_e(B_h)$, L_e broadcasts $\langle \mathsf{vote\text{-}cert}, \mathcal{C}_e(B_h), e, z_{ve} \rangle_{L_e}$ to all nodes where z_{ve} is the accumulation value of $\mathcal{C}_e(B_h)$. Similar to the proposal, the hash of the certificate is signed to allow for efficient equivocation checks. It is important to note that two different certificates for the same value is still considered an equivocation in this step.

Commit. When node p_i receives the block certificate $v_e = \langle \mathsf{vote\text{-}cert}, \mathcal{C}_e(B_h), e, z_{ve} \rangle_{L_e}$ when epoch timer is large enough (3Δ), it invokes $\mathsf{Deliver}(\mathsf{vote\text{-}cert}, v_e, z_{ve}, e)$ and sets $\mathsf{commit\text{-}timer}_e$ to 2Δ. When $\mathsf{commit\text{-}timer}_e$ reaches 0, if no equivocation for epoch-e has been detected, node p_i commits B_h and all its ancestors. The $\mathsf{Deliver}()$ message ensures that all honest nodes have received $\mathcal{C}_e(B_h)$ before quitting epoch e. We note that it is not necessary for node p_i to vote for B_h to commit it. A block certificate on B_h implies at least one honest node voted for B_h and if node p_i had a different highest ranked certificate due to which it did not vote, then the corresponding block was not committed.

Equivocation. At any time in epoch e, if a node p_i detects an equivocation, it broadcasts equivocating message signed by L_e. As mentioned earlier, the message signed by L_e are $O(1)$ sized and does not incur large communication. Node p_i also stops participating in epoch e at that time.

Latency and communication complexity. We note that all honest nodes commit in the same epoch when the epoch leader is honest. However, when the epoch leader is Byzantine, some honest nodes may commit in the epoch while some honest nodes may not commit. Due to round robin leader selection, there will be at least one honest leader every $t+1$ epochs and all honest nodes commit common blocks upto to the honest epoch. Thus, our protocol has a commit latency of $t+1$ epochs. Our protocol has communication cost of $O(n\ell + (\kappa + w)n^2)$ bits per epoch (more details on Lemma 3.8).

3.2 Safety and Liveness

We say a block B_h is committed directly in epoch e if it is committed as a result of its own $\mathsf{commit\text{-}timer}_e$ expiring. We say a block B_h is committed indirectly if it is a result of directly committing a proposal B_ℓ ($\ell > h$) that extends B_h.

Fact 3.1. *If an honest node delivers an object b at time τ in epoch e and no honest node has detected an epoch e equivocation by time $\tau + \Delta$, then all honest nodes will receive object b by time $\tau + 2\Delta$ in epoch e.*

Proof. Suppose an honest node p_i delivers an object b at time τ in epoch e. Node p_i must have sent valid shares $\langle \mathsf{share}, \mathsf{key}, s_j, w_j, z_e, e \rangle_i$ computed from b to every $p_j \in \mathcal{P}$ at time τ. The shares arrive at all honest nodes by time $\tau + \Delta$.

Since no honest node has detected an epoch e equivocation by time $\tau + \Delta$, it must be that either honest nodes will forward their shares $\langle \mathsf{share}, \mathsf{key}, s_j, w_j, z_e, e \rangle$ when they receive the shares sent by node p_i or they already sent the corresponding share when they either delivered object b or received the share from some other node p_j. In any case, all honest nodes will forward their epoch e shares corresponding to object b by time $\tau + \Delta$. Thus, all honest nodes will have received $n - t$ valid shares for a common accumulation value z_e by time $\tau + 2\Delta$ and receive object b by time $\tau + 2\Delta$. □

Fact 3.2. *If an honest node votes for a block B_h at time τ in epoch e, then all honest nodes receive B_h by time τ.*

Proof. Suppose an honest node p_i votes for a block B_h at time τ in epoch e. Node p_i must have received proposal p_e for B_h by time $\tau - 2\Delta$ and detected no epoch e equivocation by time τ. This implies no honest node detected an epoch e equivocation by time $\tau - \Delta$. Node p_i must have invoked $\mathsf{Deliver}(\mathsf{propose}, p_e, z_{pe}, e)$ at time $\tau - 2\Delta$. By Fact 3.1, all honest nodes receive p_e by time τ. Thus, all honest nodes must have received B_h by time τ. □

Lemma 3.3. *If an honest node directly commits a block B_h in epoch e, then (i) no equivocating block certificate exists in epoch e, and (ii) all honest nodes receive $\mathcal{C}_e(B_h)$ before quitting epoch e.*

Proof. Suppose an honest node p_i commits a block B_h in epoch e at time τ. Node p_i must have received a $\mathsf{vote\text{-}cert}$ for B_h at time $\tau - 2\Delta$ such that its $\mathsf{epoch\text{-}timer}_e \geq 3\Delta$ and did not detect an equivocation by time τ. This implies no honest node detected an epoch e equivocation by time $\tau - \Delta$. In addition, some honest node p_j must have voted for B_h by time $\tau - 2\Delta$. By Fact 3.2, all honest nodes would receive B_h by time $\tau - 2\Delta$.

For part (i), observe that no honest node received an equivocating proposal by time $\tau - 2\Delta$; otherwise, all honest nodes would have received a share for equivocating proposal by time $\tau - \Delta$ and node p_i would not commit. And, no honest node would vote for an equivocating block after time $\tau - 2\Delta$ (since they have received B_h by time $\tau - 2\Delta$). Thus, an equivocating block certificate does not exist in epoch e.

For part (ii), observe that node p_i must have invoked $\mathsf{Deliver}(\mathsf{vote\text{-}cert}, v_e, z_{ve}, e)$ for $v_e = \mathcal{C}_e(B_h)$ at time $\tau - 2\Delta$ and did not detect epoch e equivocation by time τ. By Fact 3.1, all honest nodes receive v_e by time τ. Note that node p_i must have its $\mathsf{epoch\text{-}timer}_e \geq 3\Delta$ at time $\tau - 2\Delta$. Since, all honest nodes are synchronized within Δ time, all other honest nodes must have $\mathsf{epoch\text{-}timer}_e \geq 2\Delta$ at time $\tau - 2\Delta$. Thus, all nodes are still in epoch e at time τ and receive $\mathcal{C}_e(B_h)$ before quitting epoch e. □

Lemma 3.4 (Unique Extensibility)**.** *If an honest node directly commits a block B_h in epoch e, then any certified blocks that ranks higher than $\mathcal{C}_e(B_h)$ must extend B_h.*

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

For the base case, where $e' = e + 1$, the proof that $\mathcal{C}_{e'}(B_{h'})$ extends B_h follows from Lemma 3.3. The only way $\mathcal{C}_{e'}(B_{h'})$ for $B_{h'}$ forms is if some honest node votes for $B_{h'}$. However, by Lemma 3.3, there does not exist any equivocating block certificate in epoch e and all honest nodes receive and lock on $\mathcal{C}_e(B_h)$ before quitting epoch e. 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 e', the proof that $\mathcal{C}_{e'}(B_{h'})$ 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 e' only if $B_{h'}$ extends a valid certificate $\mathcal{C}_{e''}(B_{h''})$. Due to Lemma 3.3 and the induction hypothesis on all block certificates of epoch $e < e'' < e'$, $\mathcal{C}_{e'}(B_{h'})$ must extend B_h. □

Theorem 3.5 (Safety)**.** *Honest nodes do not commit conflicting blocks for any epoch e.*

Proof. Suppose for the sake of contradiction two distinct blocks B_h and B'_h are committed in epoch e. Suppose B_h is committed as a result of $B_{h'}$ being directly committed in epoch e' and B'_h is committed as a result of $B'_{h''}$ being directly committed in epoch e''. Without loss of generality, assume $h' < h''$. Note that all directly committed blocks are certified. By Lemma 3.4, $B'_{h''}$ extends $B_{h'}$. Therefore, $B_h = B'_h$. □

Fact 3.6. *Let B_h be a block proposed in epoch e. If the leader of an epoch e is honest, then all honest nodes commit B_h and all its ancestors in epoch e.*

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

Leader L_e might have entered epoch e at time τ while some honest nodes enter epoch e only at time $\tau + \Delta$. The 2Δ wait in the Propose step ensures that the leader can receive highest ranked certificates from all honest nodes. However, leader L_e may enter epoch e Δ time after the earliest honest nodes. Due to 2Δ wait after entering epoch e, leader L_e collects the highest ranked certificate $\mathcal{C}_{e'}(B_l)$ by time $\tau + 3\Delta$ and sends a valid proposal $p_e = \langle \mathsf{propose}, B_h, e, \mathcal{C}_{e'}(B_l), z_{pe} \rangle_{L_e}$ for a block B_h that extends $\mathcal{C}_{e'}(B_l)$ which arrives all honest nodes by time $\tau + 4\Delta$.

Thus, all honest nodes satisfy the constraint $\mathsf{epoch\text{-}timer}_e \geq 7\Delta$. In addition, B_h extends the highest ranked certificate. So, all honest nodes will invoke $\mathsf{Deliver}(\mathsf{propose}, p_e, z_{pe}, e)$ and set $\mathsf{vote\text{-}timer}_e$ to 2Δ which expires by time $\tau + 6\Delta$. All honest nodes send vote for B_h to L_e which arrives L_e by time $\tau + 7\Delta$. Leader L_e forwards $\mathcal{C}_e(B_h)$ which arrives all honest nodes by time $\tau + 8\Delta$. Note that all honest nodes satisfy the constraint $\mathsf{epoch\text{-}timer}_e \geq 3\Delta$ and honest nodes set their $\mathsf{commit\text{-}timer}_e$ to 2Δ which expires by time $\tau + 10\Delta$. Moreover, no equivocation exists in epoch e. Thus, all honest nodes will commit B_h and its ancestors in epoch e before their $\mathsf{epoch\text{-}timer}_e$ expires. □

Theorem 3.7 (Liveness)**.** *All honest nodes keep committing new blocks.*

Proof. For any epoch e, if the leader L_e is Byzantine, it may not propose any blocks or propose equivocating blocks. Whenever an honest leader is elected in epoch e, by Fact 3.6, all honest nodes commit in epoch e. Since we assume a round-robin leader rotation policy, there will be an honest leader every $t + 1$ epochs, and thus the protocol has a commit latency of $t + 1$ epochs.

□

Lemma 3.8 (Communication complexity)**.** *Let ℓ be the size of block B_h, κ be the size of accumulator and w be the size of witness. The communication complexity of the protocol is $O(n\ell + (\kappa + w)n^2)$ bits per epoch.*

Proof. At the start of an epoch e, each node sends a highest ranked certificate to leader L_e. Since, size of each certificate is $O(\kappa n)$, this step incurs $O(\kappa n^2)$ bits communication. A proposal consists of a block of size ℓ and block certificate of size $O(\kappa n)$. Proposing $O(n + \ell)$-sized object to n nodes incurs $O(\kappa n^2 + n\ell)$. Delivering $O(\kappa n + \ell)$-sized object has a cost $O(n\ell + (\kappa + w)n^2)$, since each node broadcasts a share of size $O((n + \ell)/n)$, a witness of size w and an accumulator of size κ.

In Vote cert step, the leader broadcasts a certificate for block B_h which incurs $O(\kappa n^2)$ communication. Delivering $O(\kappa n)$-sized $\mathcal{C}_e(B_h)$ incurs $O((\kappa + w)n^2)$ bits. Hence, the total cost is $O(n\ell + (\kappa + w)n^2)$ bits.

□

4 Random Beacon Protocols

In this section, we present two random beacon protocols while tolerating $f \leq t < n/2$ Byzantine faults. The first protocol GRandPiper outputs a random beacon with $O(\kappa n^2)$ communication complexity *always* per beacon output where κ is the security parameter, guarantees bias-resistance, and $O(\min(\kappa, t))$-absolute unpredictability against a static adversary, but $t+1$-absolute unpredictability against an adaptive adversary. The second protocol BRandPiper outputs a random beacon with $O(\kappa f n^2)$ communication complexity per output after amortization where κ is the security parameter, and guarantees bias-resistance and 1-absolute unpredictability. When the actual number of faults $f = O(1)$, the communication complexity is quadratic.

A key aspect of both of our protocols is their reconfiguration-friendliness. A protocol is said to be reconfiguration-friendly if it allows changing protocol parameters such as the scheme and nodes, without stopping the old instance, and starting a new one. Such reconfiguration is possible if the setup used for the protocol does not bind to the system, as such a binding will force a new setup to change any parameter in the system. This is true, for instance, when using threshold signatures in a protocol which is used by many existing permissioned systems [9, 11, 12, 23]. Neither of our protocols use a trusted setup for threshold signatures, but a setup based on the q-Strong Diffie Hellman assumption. This allows for easy reconfiguration.[3]

[3]We can use Merkle trees instead of q-SDH at the expense of $O(\log n)$ multiplicative communication complexity.

4.1 RandPiper – GRandPiper Protocol

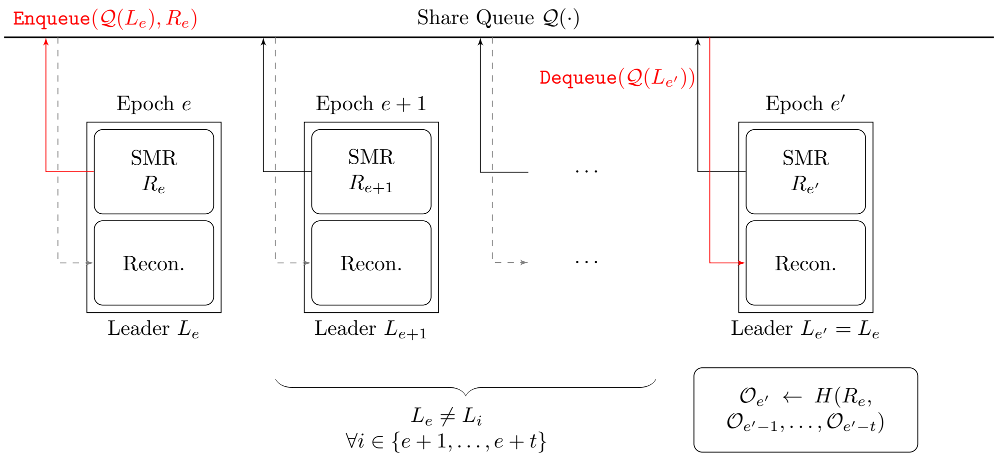
Enqueue($\mathcal{Q}(L_e), R_e$)
Share Queue $\mathcal{Q}(\cdot)$
Dequeue($\mathcal{Q}(L_{e'})$)
Epoch e
Epoch $e+1$
Epoch e'
SMR R_e
SMR R_{e+1}
SMR $R_{e'}$
Recon.
Recon.
Recon.
Leader L_e
Leader L_{e+1}
Leader $L_{e'} = L_e$
$L_e \neq L_i$ $\forall i \in \{e+1, \ldots, e+t\}$
$\mathcal{O}_{e'} \leftarrow H(R_e, \mathcal{O}_{e'-1}, \ldots, \mathcal{O}_{e'-t})$

Figure 3: **Overview of RandPiper – GRandPiper Random beacon protocol.** In every epoch, a PVSS encryption vector is used to secret share some random value. At the same time, a reconstruction protocol is used to reconstruct the random value committed by the leader of this epoch, the last time it was a leader. $\mathcal{O}_{e'}$ is generated using the random value R_e, shared in epoch e, reconstructed in epoch $e' > e + t$, and outputs $\{\mathcal{O}_{e'-1}, \ldots, \mathcal{O}_{e'-t}\}$ from previous epochs by using them as inputs to the random oracle H.

We use the SMR protocol (refer Figure 1) described in the Section 3 as a building block. At a high-level, consider using the SMR protocol such that the leader outputs a number chosen uniformly at random in each epoch. The random beacon output can be a function of the outputs of the last $t + 1$ epochs, allowing for the presence of at least one honest input (chosen uniformly at random) which is potentially sufficient to obtain a *random* output. This argument holds *only if* each leader chooses their input in the SMR protocol independently of other inputs. Otherwise, if a Byzantine leader can choose an input after knowing the outputs of the previous t instances then it can bias the output. A separate concern with using the SMR protocol as is, is that in an epoch with a Byzantine leader, honest nodes may not all output the same value or output at all.

To fix both of these concerns, we require each node to send a commitment of a random value more than t epochs before it will be reconstructed and used in the beacon protocol. To ensure the secrecy of this value (for unpredictability and bias-resistance), the values are shared with the nodes using a publicly verifiable secret sharing (PVSS) scheme. Committing a secretly chosen value ahead of time helps us solve both of our previous concerns. First, if the same leader is not chosen twice in any span of $t + 1$ epochs, it ensures that the $t + 1$ values that will be used to construct the beacon protocol are chosen independently of one another. Thus, when nodes reconstruct a value in an epoch, it corresponds to a value committed more than t epochs before. Moreover, the nodes can reconstruct this value independent of the participation of the leader in this epoch. Second, waiting for $t + 1$ epochs before opening allows for the value to be committed by the SMR protocol. Thus, all honest nodes will open the same value in an epoch.

A graphical description of this approach is presented in Figure 3. In epoch e, a leader L_e inputs PVSS shares corresponding to a random value R_e to the SMR protocol. Conceptually, when the block is committed, this value is added to a queue $\mathcal{Q}(L_e)$ corresponding to this leader. When the same node is chosen the next time as a leader, say in epoch e', the committed shares of R_e is dequeued and reconstructed by all honest nodes to obtain R_e. The output $\mathcal{O}_{e'}$ of epoch e' can be computed as $H(R_e, \mathcal{O}_{e'-1}, \ldots, \mathcal{O}_{e'-t})$. To allow for unpredictability in leader selection while disallowing repetition within $t + 1$ epochs, the leader for the next epoch $e' + 1$ is chosen based on $\mathcal{O}_{e'}$ and by removing the leaders $L_{e'}, \ldots, L_{e'-t}$.

A remaining concern is when no values are added to the chain at epoch e. Observe that the reconstruction in epoch e is not affected, since nodes reconstruct values previously committed. However, nodes may not have shares in epoch $e' > e + t$ where e' is the first epoch where L_e is chosen as the leader again. To fix this concern, we ensure that such a malicious leader who does not commit in epoch e can be removed by all nodes by $e + t < e'$. Subsequently, we can ensure that L_e is never chosen as the leader again. To allow for reconstruction the first time a node is chosen as the leader, we ensure a setup where each node has an agreed upon share buffered for every other node.

4.1.1 Protocol details

All nodes $p_i \in \mathcal{P}$ running the SMR protocol do the following:

- **Setup.** Set $e = 1$. All nodes agree upon seed random values for R_e and $\{\mathcal{O}_{e-1}, \ldots, \mathcal{O}_{e-t}\}$. Set $\mathcal{L}_{Last} \leftarrow \{p_n, \ldots, p_{n-t}\}$, $\mathcal{P}_r \leftarrow \emptyset$. Run $\mathcal{PVSS}.\mathsf{Setup}$ and agree on the public parameters $\mathcal{PVSS}.\mathsf{pp}$. Then every node generates a key pair $(\mathcal{PVSS}.\mathsf{sk}, \mathcal{PVSS}.\mathsf{pk}) \leftarrow \mathcal{PVSS}.\mathsf{KGen}(\kappa)$, and all nodes agree on each others public keys.

- **Leaders.** Choose leaders for an epoch e using Definition 4.1 instead of a round robin order.

- **Blocks.** The leader L_e of an epoch e, creates a PVSS sharing $(\mathcal{PVSS}.\vec{S}, \mathcal{PVSS}.\vec{E}, \mathcal{PVSS}.\pi) \leftarrow \mathcal{PVSS}.\mathsf{ShGen}(R)$ of a random value chosen from the input space of $\mathcal{PVSS}$, and creates a block B_h with block contents b_h as $b_h := (\mathcal{PVSS}.\vec{E}, \mathcal{PVSS}.\pi) \leftarrow \mathcal{PVSS}.\mathsf{ShGen}(R)$. (We drop the individual shares in $\mathcal{PVSS}.\vec{S}$.)

- **Block validity.** After regular checks for block validity, with the block content parsed as $b_h := (\mathcal{PVSS}.\vec{E}, \mathcal{PVSS}.\pi)$, ensure that $\mathcal{PVSS}.\mathsf{ShVrfy}(\mathcal{PVSS}.\vec{E}, \mathcal{PVSS}.\pi) = 1$.

- **Update.** When committing a block B_h sent by leader $L_{e'}$ for some epoch e', $\mathtt{Enqueue}(\mathcal{Q}(L_{e'}), b_h)$. At the end of epoch e, if no block was committed for epoch $e - t$ by L_{e-t}, then remove L_{e-t} from future proposals, i.e., $\mathcal{P}_r \leftarrow \mathcal{P}_r \cup \{L_{e-t}\}$ from epoch $e + 1$.

- **Reconstruct.** When the epoch timer $\mathsf{epoch\text{-}timer}_{e-1}$ for epoch $e - 1$ ends, obtain the $(\mathcal{PVSS}.\vec{E}, \mathcal{PVSS}.\pi)$ corresponding to the committed block in $\mathtt{Dequeue}(\mathcal{Q}(L_e))$. Send $s \leftarrow \mathcal{PVSS}.\mathsf{Dec}(\mathcal{PVSS}.\mathsf{sk}, \mathcal{PVSS}.\vec{E}_i)$ to all the nodes in the system. On receiving share s' from another node p_j, ensure that $\mathcal{PVSS}.\mathsf{Enc}(\mathcal{PVSS}.\mathsf{pk}_j, s') = \mathcal{PVSS}.\vec{E}_j$. On receiving $t+1$ valid shares in $\mathcal{PVSS}.\vec{S}$, reconstruct $R_e \leftarrow \mathcal{PVSS}.\mathsf{Recon}(\mathcal{PVSS}.\vec{S})$.

- **Output.** After reconstructing R_e for epoch e, output the beacon value $\mathcal{O}_e$ by computing,

$$\mathcal{O}_e \leftarrow H(R_e, \mathcal{O}_{e-1}, \ldots, \mathcal{O}_{e-t})$$

Figure 4: **RandPiper – GRandPiper beacon protocol description.** We describe the modifications required to the SMR protocol described in Figure 1 to build GRandPiper beacon protocol.

We now explain the protocol in detail (described in Figure 4). We use a Publicly Verifiable Secret Sharing (PVSS) scheme $\mathcal{PVSS}$ with threshold t to generate encrypted shares and an associated proof that guarantees that any $> t$ nodes will reconstruct a unique secret.

Setup. We establish PVSS parameters $\mathcal{PVSS}.\mathsf{pp}$, and public keys $\mathcal{PVSS}.\mathsf{pk}_i$ for every node $p_i \in \mathcal{P}$. We also buffer shares for one random value for every node p_i, i.e., fill $\mathcal{Q}(p_i)$ for $p_i \in \mathcal{P}$. We start with epoch $e = 1$, and use seed random values for R_e and $\{\mathcal{O}_{e-1}, \ldots, \mathcal{O}_{e-t}\}$. We also arbitrarily assign $\mathcal{L}_{Last} \leftarrow \{p_n, \ldots, p_{n-t}\}$ and set $\mathcal{P}_r \leftarrow \emptyset$.

Leader selection. The leader for epoch e is chosen based on the following rule:

Definition 4.1 (Leader selection rule)**.** *Let e be the current epoch, $\mathcal{L}_{Last} := \{L_{e-1}, \ldots, L_{e-t}\}$ be the leaders of the last t epochs, $\mathcal{P}_r$ be the set of nodes that are removed (due to misbehavior), and $\mathcal{L}_e = (\mathcal{P} \setminus \mathcal{L}_{Last}) \setminus \mathcal{P}_r := \{l_0, \ldots, l_{w-1}\}$, be a set of candidate leaders for epoch e ordered canonically, with $0 < w < n - t$ and $\mathcal{L}_e \subseteq \mathcal{P}$. Then the leader L_e of epoch e, is derived from output $\mathcal{O}_{e-1}$, as*

$$L_e \leftarrow l_{(\mathcal{O}_{e-1} \mod w)}$$

Blocks. The leader of an epoch chooses R uniformly at random from the input space of the $\mathcal{PVSS}$ algorithm (which could be a cyclic additive/multiplicative group, or pairing groups). The leader uses the $\mathcal{PVSS}.\mathsf{ShGen}$ algorithm to generate share $\mathcal{PVSS}.\mathsf{s}_i$ for node p_i which are encrypted using $\mathcal{PVSS}.\mathsf{pk}_i$, and all shares for the nodes are stored in the encryption vector $\mathcal{PVSS}.\vec{E}$. The $\mathcal{PVSS}.\mathsf{ShGen}$ algorithm also outputs the proof $\mathcal{PVSS}.\pi$ that any $> t$ shares will reconstruct a unique secret, which implies that the degree of the polynomial cannot be more than t. Finally, the block in our SMR protocol consists of the outputs of the $\mathcal{PVSS}.\mathsf{ShGen}$ algorithm, i.e., $b_h := (\mathcal{PVSS}.\vec{E}, \mathcal{PVSS}.\pi) \leftarrow \mathcal{PVSS}.\mathsf{ShGen}(R_e)$. Note that despite the blocks being $O(\kappa n)$ sized, due to our usage of $\mathsf{Deliver}$ primitive, we retain a communication complexity of $O(\kappa n^2)$ per epoch.

Commit, reconstruct, and output beacon value. In each epoch, nodes commit the shares sent by the leader. They also reconstruct the block last sent by the leader at the start of the epoch. Note that each node can separately maintain the last time a node was elected as the leader, and thus, be able to appropriately invoke $\texttt{Dequeue}(\mathcal{Q}(L_e))$. Moreover, since a leader does not repeat in any consecutive $t + 1$ epochs, and we ensure that the set of leaders are consistently known to all honest nodes (as will be shown in the next subsection), the reconstructed block is agreed upon by all honest nodes. When the nodes reconstruct R_e, they already have access to $\{\mathcal{O}_{e-1}, \ldots, \mathcal{O}_1\}$. Hence, they can compute a consistent output $\mathcal{O}_e$. Observe that since all nodes enter epoch e within a delay of Δ, they also output the beacon value within Δ time of each other.

Remove misbehaving leaders. Finally, at the end of an epoch e, if no block was committed in epoch $e-t$ by L_{e-t}, L_{e-t} is removed from all future proposals. Since this operation is performed after $t+1$ epochs, all nodes will perform this action consistently.

4.1.2 Security Analysis

Theorem 4.1 (Consistent beacon)**.** *Let $L = L_e$ denote the leader of epoch e. Then the following properties hold:*

(i) Block consistency*: if an honest node commits a block B proposed in epoch $e' \leq e - t$, then all honest nodes commit block B by epoch e.*

(ii) Leader validity*: all nodes have a block in $\mathcal{Q}(L_e)$.*

(iii) Output consistency*: all honest nodes output the same randomness R_e, output $\mathcal{O}_e$, and*

(iv) Leader consistency*: all honest nodes choose the same leader for epoch $e + 1$.*

Proof. We prove the theorem by induction on epochs.

Base case for epoch $e = 1$ **to** $e = t + 1$**.** (ii) should hold for the first $t + 1$ leaders because we fill $\mathcal{Q}(p_i)$ with $m = 1$ values for all nodes $p_i \in \mathcal{P}$ during the setup phase. Additionally, from Definition 4.1, no leaders repeat in the first $t + 1$ epochs, thereby proving (ii) for the base case. (iii) and (iv) hold since the first $t + 1$ outputs only use the seed values, and pre-agreed upon shares from $\mathcal{Q}$ from the setup phase. At epoch $e = t + 1$, from the proof for (iv) for the base cases, we know that all nodes agree on the leaders for epochs $1 \leq e' \leq t+1$. Therefore, if some honest node commits block B_1 from epoch $e = 1$, then all honest nodes will commit B_1 by epoch $e = t+1$, because at least one leader in some epoch $1 \leq e' \leq t+1$ must be honest, and from Fact 3.6 all honest nodes commit the block proposed in e' and therefore directly or indirectly commit B_1. Therefore, by epoch $e = t + 1$ all honest nodes commit B_1, thereby proving (i) for the base cases.

Induction hypothesis. The statements hold until epoch $e - 1$.

Induction step. *Proof for (i).* From the induction hypothesis for (iv), we have that all the leaders until epoch e are consistent and at epoch $e - 1$, and from the induction hypothesis for (i) all honest nodes would have committed all the blocks for epoch $e' < e - 1 - t$ by consistent leaders upto epoch $e - 1$. Now, at epoch e all honest nodes need to decide on the block B proposed in epoch $e - t - 1$. In epochs $e - t - 1 \leq e' < e$, there is one epoch e' whose leader $L_{e'}$ is honest, from Fact 3.6, all honest nodes commit B in epoch e', thus proving the hypothesis for (i).

Proof for (ii). By the induction hypothesis for (iv), the leader of epoch e and all previous epochs is agreed upon. Let L_e be the leader for epoch e. Then L_e must have committed a block in some epoch $< e - t$, or during the setup. If L_e was never the leader, then the hypothesis (ii) is trivially satisfied. Let $e' < e - t$ be the latest epoch in which L_e was the leader last. If L_e proposed a block in some epoch $e' < e - f$, then from the proof for (i) for epoch e, all nodes agree on the same block for epoch e'. If no block proposed in epoch e' is added to the chain by epoch $e' + t < e$, then from our leader selection rule (in Definition 4.1), no honest node will derive L_e as the leader as $L_e \in \mathcal{P}_r$. Therefore, (ii) also holds for epoch e.

Proof for (iii). The randomness R_e depends on choosing a committed value to be reconstructed. The output $\mathcal{O}_e$ depends on R_e and $\{\mathcal{O}_{e-1}, \ldots, \mathcal{O}_{e-t}\}$. By the induction hypothesis for (iii), all honest nodes agree on $\mathcal{O}_{e-1}, \ldots, \mathcal{O}_{e-f}$. Moreover, by the induction hypothesis for (iv), they also choose the same leader L_e for epoch e. Thus, if we can prove that all honest nodes agree upon the value from L_e that is reconstructed in epoch e, then agreement on $\mathcal{O}_e$ is trivial. From the proof of (i) and (ii), we know that there is a block b_h that is enqueued in the queue $\mathcal{Q}(L_e)$ for L_e, which all honest nodes agree on, and therefore obtain the same R_e for epoch e. Thus, (iii) holds true for epoch e.

Proof for (iv). The leader derivation depends on the candidate set $\mathcal{L}_e$, outputs of the last t iterations $\{\mathcal{O}_{e-1}, \ldots, \mathcal{O}_{e-t}\}$, and the reconstructed randomness R_e in epoch e. By the induction hypothesis for (iv), and proof of (iii), the output of the last f iterations and that of epoch e is agreed upon. From the proof of (i), (ii), and the induction hypothesis for (iv), all honest nodes have the same $\mathcal{L}_{Last}$ and $\mathcal{P}_r$. From (iii), all honest nodes derive the same leader for epoch $e + 1$. □

Next, we show that the SMR protocol is a secure PVSS protocol, satisfying the requirements specified in Definition 2.4.

Theorem 4.2 (Secure PVSS)**.** *The GRandPiper protocol is a secure publicly verifiable secret sharing protocol with the dealer as the leader of an epoch, and the rest of the nodes as the verifiers.*

Proof. We will individually prove all the requirements stated in Definition 2.4.

1. *Secrecy.* Let L_e be a honest leader for epoch e. If an adversary $\mathcal{A}$ has access to the internal state of any t nodes in $\mathcal{P} \setminus \{L_e\}$, it cannot get any information with probability better than $\mathsf{negl}(\kappa)$ from the security of the underlying scheme $\mathcal{PVSS}$.

2. *Correctness.* Let L_e be an honest leader for epoch e. Then its proposed block that shares R_e is always committed (from Fact 3.6). Thus when the secret is reconstructed in the beacon protocol (Figure 4) all honest nodes will output R_e with a high probability of $1 - \mathsf{negl}(\kappa)$ (from the underlying $\mathcal{PVSS}$ algorithm).

3. *Commitment.* If L_e is Byzantine, then either all nodes commit to one of the blocks B_h proposed by it or $\perp$ by epoch $e + t + 1$. Therefore, the *commitment* property is satisfied by our protocol. From the underlying scheme $\mathcal{PVSS}$, there is a negligible probability $\mathsf{negl}(\kappa)$ for two correct nodes p_i and p_j to output different $s_i^* \neq s_j^* \neq \perp$.

4. *(Public) Verifiability.* This property holds true with high probability from the underlying PVSS scheme $\mathcal{PVSS}$.

□

Now, we show that GRandPiper is a secure random beacon protocol.

Lemma 4.3 (Rushing Adversary Advantage). *For any epoch e, a rushing adversary can reconstruct output $\mathcal{O}_e$ at most 2Δ time before honest nodes.*

Proof. An honest node sends its secret shares in epoch e when its $\mathsf{epoch\text{-}timer}_{e-1}$ expires. Let node p_i be the earliest honest node whose $\mathsf{epoch\text{-}timer}_{e-1}$ expires and node p_i sends its secret share at time τ. A rushing adversary may instantaneously receive the share and reconstruct the output O_e at time τ.

Due to Δ delay in honest nodes entering into an epoch, other honest nodes may send their secret shares only at time $\tau + \Delta$ which arrives all honest nodes by time $\tau + 2\Delta$. In the worst case, honest nodes can reconstruct only at time $\tau + 2\Delta$. Thus, a rushing adversary can reconstruct output $\mathcal{O}_e$ at most 2Δ time before honest nodes. □

Lemma 4.4 (Guaranteed Beacon Output). *For each epoch e, all honest nodes output a new beacon output $\mathcal{O}_e$.*

Proof. By Theorem 4.1 part (iv), all honest nodes have consistent leaders. Let node p_i be the leader of epoch e. Honest nodes output a new beacon output in each epoch e if $\mathcal{Q}(p_i) \neq \perp$. Suppose for the sake of contradiction $\mathcal{Q}(p_i) = \perp$ in epoch e. Observe that nodes update $\mathcal{Q}(p_i)$ with secret proposed in epoch e' (with $e' < e - t$) when p_i was an epoch leader in epoch e' by epoch e and node p_i did not propose any secrets in epoch e'. However, if p_i did not propose in epoch e', p_i would have been removed from the candidate leader set for epoch e and would not be epoch leader for epoch e and honest nodes would not use $\mathcal{Q}(p_i)$ in epoch e. A contradiction.

Thus, all honest nodes send secret shares for secret shared in $\mathcal{Q}(p_i)$ and all honest nodes will receive $t+1$ valid shares to reconstruct a common randomness R_e and output $\mathcal{O}_e$.

□

Lemma 4.5 (Bias-Resistance). *For any epoch $e \geq 1$, the probability that a t bounded adversary $\mathcal{A}$ can fix any c bits of the GRandPiper beacon output $\mathcal{O}_e$ is $\frac{1}{2^c} + \mathsf{negl}(\kappa)$.*

Proof. The output in any epoch e is $\mathcal{O}_e \leftarrow H(R_e, \mathcal{O}_{e-1}, \ldots, \mathcal{O}_{e-t})$. Assume that an adversary $\mathcal{A}$ wants to bias some c bits of the output. Now, there is at least one honest node who is a leader in epoch e' where $e - t \leq e' \leq e$. WLOG, assume that the leader at epoch $e' = e - t$ is honest. Then the output of epoch e' is known only in epoch e' within 2Δ time of entering the epoch e' (from Lemma 4.4). Therefore, a rushing adversary $\mathcal{A}$ can know the $\mathcal{O}_{e'}$ at max 2Δ before an honest node enters epoch e'. But the adversary has to choose all $R_{e''}$ before epoch e', where $e - t < e'' < e' \leq e$, so that it can bias $\mathcal{O}_e$. But all blocks containing $R_{e''}$ are committed before the epoch e', since $R_{e''}$ comes from the blocks previously proposed by the leaders before epoch e' at the start (or during the setup). Thus all blocks containing $R_{e''}$ are proposed before observing $R_{e'}$, which is guaranteed to be secret for a honest node against $\mathcal{A}$ (from the *secrecy* property of Theorem 4.2) except with negligible probability $\mathsf{negl}(\kappa)$. Thus, an adversary $\mathcal{A}$ can do no better than $\frac{1}{2^c} + \mathsf{negl}(\kappa)$ to fix c bits. □

Lemma 4.6 (GRandPiper $O(\min(\kappa, t))$-absolute unpredictability). *The GRandPiper random beacon protocol is an $O(\min(\kappa, t))$-unpredictable random beacon against a static adversary, when a single instance is run.*

Proof. Since the leaders are chosen using the beacon outputs, the probability that the adversary's nodes are chosen in an epoch e is $t/n < 1/2$. The probability that c consecutive leaders are Byzantine is therefore $\binom{t}{c}/(n-t)^c < 2^{-c}$ for $3 < c < t$ and is exponentially decreasing in c. The expected value is $\lceil \log 2 \rceil = 2$. If $c = t + 1$, the probability is already $\mathsf{negl}(\kappa)$. Thus, for a given security parameter κ, we can achieve a probability of a static adversary predicting the output with better than κ in $\min(\kappa, t) + 1$ epochs. □

Better unpredictability using multiple instances. Running a single instance of the GRandPiper beacon, a static adversary can predict c rounds into the future with probability $< 2^{-c}$ for $c > 3$ (Lemma 4.6). However, an adaptive adversary can choose to corrupt the leaders of the next t epochs at any epoch e, reducing the unpredictability to $t+1$-absolute unpredictability. We can improve this to obtain statistical security in parameter κ, by running p parallel instances of the beacon protocol whose leaders are chosen independently and randomly. This protocol has a communication complexity $O(p\kappa n^2)$ complexity per epoch. In any epoch, the probability of the next leader being Byzantine is $t/n < 1/2$. When p independent beacon instances run in parallel, the beacon protocol becomes $o(\log(t/p))$-absolute unpredictable.

Lemma 4.7 (Multiple-instance Unpredictability)**.** *GRandPiper is an $o(\log(t/p))$-absolute unpredictable random beacon protocol against an adaptive adversary, when p instances are run in parallel.*

Proof. We are running p instances of GRandPiper beacon protocol. Let an adaptive adversary $\mathcal{A}$ have $0 \le c \le t$ choices to corrupt nodes. In order to predict the output of epoch e, $\mathcal{A}$ has to corrupt *all* the leaders of the p instances.

Since the adversary wants to predict continuous beacon values, let us start with $c(0) \leftarrow 0$ initially. For the first epoch, $\mathcal{A}$ has to corrupt p leaders. For the next epoch, $c \leftarrow p$, i.e., the adversary has $t - c = t - p$ choices left. However, the c corrupted leaders from the first round can become leaders in other instances in this epoch. Therefore, let us first analyze the expected number of corruptions needed in this epoch.

Let X be the random variable defining the number of new corruptions. The probability of $\mathcal{A}$ needing x new corruptions is the probability that in $x \le p$ instances one of the c nodes do not become leaders, which is:

$$\Pr[X = x] = \left(\frac{c}{n-t}\right)^{p-x}\left(1 - \frac{c}{n-t}\right)^{x}$$

$$E[X] = p\left(\frac{c}{n-t}\right)$$

Hence, in every epoch from 1, with $c_0 = p$, c_i grows by $pc_{i-1}/(n-t)$, i.e., $c_i = c_{i-1} + pc_{i-1}/(n-t)$ whose solution is

$$c_{i+1} = p\left(\left(\frac{p}{n-t}\right)^{i}\right) + p\left(\frac{1 - \left(\frac{p}{n-t}\right)^{i}}{1 - \frac{p}{n-t}}\right)$$

. Now all we need to do is to ensure that c_i always stays less than t, whose solution gives us:

$$c_i \le t$$

$$\implies i = o\left(\log\left(\frac{t}{p}\right)\right)$$

Thus $o(\log(t/p))$ gives us a strict upper bound on the prediction power of an adaptive adversary. □

Theorem 4.8 (GRandPiper secure random beacon)**.** *GRandPiper protocol is a:*

1. $O(\min(\kappa, t))$*-secure random beacon protocol assuming a static adversary.*
2. $t+1$*-secure random beacon protocol assuming an adaptive adversary using a single instance.*
3. $o(\log(t/p))$*-secure random beacon protocol assuming an adaptive adversary using p parallel instances.*

Proof. 1. This follows from Lemma 4.5, Lemma 4.4, and Lemma 4.6.

2. The bias-resistance and guaranteed output delivery follow from Lemma 4.5 and Lemma 4.4. $t+1$-absolute unpredictability follows directly from Lemma 4.6 by allowing an adaptive adversary to corrupt the next t leaders after knowing the output for the current epoch before the other honest nodes (from Lemma 4.3).

3. This follows from Lemma 4.5, Lemma 4.4, and Lemma 4.7.

□

4.2 BRandPiper Protocol

In this section, we present a randomness beacon protocol with 1-absolute unpredictability with $O(\kappa f n^2)$ bits communication complexity where $f \leq t$ is the actual number of faults. Thus, in the optimistic case when $f = O(1)$, our communication complexity is quadratic.

4.2.1 Improved VSS

We will first describe an improved VSS scheme that achieves better communication complexity in the optimistic case which will then be used for our randomness beacon protocol.

Efficient VSS (eVSS). eVSS [28] (refer Figure 5) presents the state-of-the-art VSS scheme for synchronous network setting. The protocol is described assuming the presence of a bulletin board (or broadcast channels) [8, 14, 16, 28] where there exists a public bulletin board, in which messages posted by any node are available instantly, and the bulletin board provides a consistent view to all the nodes. We can realize such message delivery guarantees by invoking Byzantine Broadcast (BB) protocols.

Let $\mathcal{VSS}$ be the VSS scheme being used. Let $\mathcal{VSS}.\mathsf{pp}$ be the public VSS parameters. Let L be a dealer with secret s. Assuming the existence of a bulletin board, each node $p_i \in \mathcal{P}$ does the following:

1. **Post commitment.** If p_i is L, then generate shares for every node by running $(\mathcal{VSS}.\vec{S}, \mathcal{VSS}.\vec{W}, \mathcal{VSS}.\mathsf{C}) \leftarrow \mathcal{VSS}.\mathsf{ShGen}(s)$, and post the commitment $\mathcal{VSS}.\mathsf{C}$ to the secret s on the bulletin board.

2. **Send shares.** If p_i is L, then send shares $\mathcal{VSS}.\mathsf{s}_j \in \mathcal{VSS}.\vec{S}$ and witness $\mathcal{VSS}.\pi_j \in \mathcal{VSS}.\vec{W}$ over the confidential channel to all nodes $p_j \in \mathcal{P}$.

3. **Send blames.** Post complaints $\langle \mathsf{blame}, L \rangle_i$ on the bulletin board, if no valid share is received privately or if $\mathcal{VSS}.\mathsf{ShVrfy}(\mathcal{VSS}.\mathsf{s}_i, \mathcal{VSS}.\pi_i, \mathcal{VSS}.\mathsf{C}) = 0$.

4. **Open shares.** For all blames $\langle \mathsf{blame}, L \rangle_i$, if p_i is L, post their shares $\mathcal{VSS}.\mathsf{s}_j$ and witnesses $\mathcal{VSS}.\pi_j$ on the bulletin board.

5. **Decide.** If the published share and witness satisfies $\mathcal{VSS}.\mathsf{ShVrfy}(\mathcal{VSS}.\mathsf{s}_k, \mathcal{VSS}.\pi_k, \mathcal{VSS}.\mathsf{C}) = 1$ for every blame, and there are only up to $f \leq t$ blames on the bulletin board, then commit $\mathcal{VSS}.\mathsf{s}_j$. Otherwise, abort, i.e., output $\perp$.

Figure 5: **eVSS [28] protocol description.** This scheme is to secret share one secret.

In this scheme, a dealer L creates a commitment $\mathcal{VSS}.\mathsf{C}$ to a random polynomial whose constant term is the secret, and posts the commitment on the bulletin board (Step 1), while *privately* sending individual shares $\mathcal{VSS}.\mathsf{s}_j$ along with witnesses $\mathcal{VSS}.\pi_j$ to every node $p_j \in \mathcal{P}$ (Step 2). Nodes post complaints on the bulletin board in the form of blame message if they do not receive valid shares (Step 3) in a timely manner. The dealer then opens the secret shares on the bulletin board corresponding the nodes that blamed (Step

4). If there are $> t$ complaints, the nodes abort (Step 5). Otherwise, the honest nodes commit their shares (Step 5), with the guarantee that all honest nodes will be able to reconstruct the shared secret.

Note that $f \leq t$ Byzantine nodes can always blame regardless of the dealer being honest or not. This forces an honest dealer to post $O(fn)$ shares on the bulletin board when secret sharing $O(n)$ secrets. In general, the amount of information posted on the bulletin board is $O(\kappa n + \kappa f + \kappa f n)$ corresponding to $O(n)$ commitments, f blame messages and $O(fn)$ opened secret shares. A naïve approach of using BB protocols (extension BB protocols [31] for larger inputs can be used) to instantiate the bulletin board involves following steps:

1. *Commitment and sharing.* Dealer L invokes BB to broadcast n commitments Step 1, while privately sharing individual shares Step 2.

2. *Blame.* Nodes invoke n parallel instances of BB to broadcast blame messages Step 3.

3. *Open shares.* Dealer L invokes an instance of BB with secret shares corresponding to the blames received.

We note that state-of-the-art honest majority BB protocols, without threshold signatures, incur $O(\kappa n^3)$ bits communication cost to achieve consensus on a single decision [3, 19, 29]. Thus, invoking n parallel instances of BB trivially incurs $O(\kappa n^4)$ communication cost. In addition, running BB on inputs of size $O(fn)$ incurs $O(\kappa f n^3)$ without threshold signatures and extension techniques. Thus, the total communication complexity is $O(\kappa n^4)$ bits.

Improved eVSS (iVSS). In order to reduce the large communication overhead, we first present an improved VSS scheme, that reduces (i) the number of posts to the bulletin board, and (ii) the amount of information posted on the bulletin board.

In iVSS (refer Figure 6), the dealer posts commitments on the bulletin board, privately sends the secret shares and corresponding witnesses similar to eVSS. However, unlike eVSS, nodes send the blame messages to all nodes. In addition, nodes forward the received blame messages to the dealer to request for missing shares. The dealer *privately* sends missing shares to the nodes that forwarded the blame message instead of posting on the bulletin board. If an honest node receives missing shares for all blame messages it forwarded, it sends an ack to the dealer. The dealer collects $t + 1$ ack messages and posts the ack certificate on the bulletin board. An honest node commits the proposed commitment if it observes an ack certificate on the bulletin board.

Later, during the reconstruction phase, honest nodes forward the missing shares if the dealer sent the missing shares before they send an ack. A key correctness argument for our scheme is the following: if an honest node $p_i \in \mathcal{P}$ does not receive commitments and secret shares, it must have sent blame messages to all honest nodes. If some honest node $p_j \in \mathcal{P}$ sends an ack message, it must have received missing shares corresponding to the blame messages it received and forwarded (which includes share for p_i). Thus, honest nodes together have sufficient shares to reconstruct the proposed secrets.

We note that both eVSS and iVSS schemes guarantee secrecy (see Definition 2.5) only when the dealer is honest. If t Byzantine nodes send a blame message, then an honest but curious node can violate secrecy, however this can be easily solved by assuming an additional honest node, i.e., $n > 2t + 1$.

Theorem 4.9 (Security of iVSS)**.** *The verifiable secret sharing scheme proposed in Figure 6 is a secure verifiable secret sharing scheme assuming a bulletin board.*

Proof. We prove the security of iVSS protocol by proving that the individual requirements from Definition 2.5.

1. *Secrecy*: If the dealer L is honest, then no honest node will blame and the maximum number of blames is at most t. Thus, only up to t blames will be opened privately by the leader. Therefore, the adversary $\mathcal{A}$ does not learn any new share from which it can learn the secret s, except with $\mathsf{negl}(\kappa)$ probability.

2. *Correctness*: If the dealer L is honest, then all honest nodes have their shares for the secret s, and similar to eVSS, will output the same secret s except with $\mathsf{negl}(\kappa)$ probability.

Let $\mathcal{VSS}$ be the VSS scheme being used. Let $\mathcal{VSS}.\mathsf{pp}$ be the public VSS parameters. Let L be a dealer with n secrets $S := \{s_1, \dots, s_n\}$ it wishes to secret share with nodes $\mathcal{P}$. Assuming the existence of a bulletin board, each node $p_i \in \mathcal{P}$ does the following:

1. **Post commitment.** If p_i is L, run $(\mathcal{VSS}.\vec{S}_i, \mathcal{VSS}.\vec{W}_i, \mathcal{VSS}.\mathsf{C}_i) \leftarrow \mathcal{VSS}.\mathsf{ShGen}(s_i)$ for all $s_i \in S$. Build the commitment vector $\mathcal{VSS}.\vec{C} := \{\mathcal{VSS}.\mathsf{C}_1, \dots, \mathcal{VSS}.\mathsf{C}_n\}$ which contain commitments $\mathcal{VSS}.\mathsf{C}_i$ for s_i. Post $\mathcal{VSS}.\vec{C}$ on the bulletin board.

2. **Send shares.** If p_i is L, collect shares and witnesses $(\mathcal{VSS}.\mathsf{s}_j, \mathcal{VSS}.\pi_j)$ for every node $p_j \in \mathcal{P}$, and secret $s_i \in S$, and build $\mathcal{VSS}.\vec{S}_j, \mathcal{VSS}.\vec{W}_j$. Send $(\mathcal{VSS}.\vec{S}_j, \mathcal{VSS}.\vec{W}_j)$ to node $p_j \in \mathcal{P}$.

   If $p_i \in \mathcal{P}$ is not the dealer L, then wait to obtain $(\mathcal{VSS}.\vec{S}_i, \mathcal{VSS}.\vec{W}_i)$ from the dealer L, and ensure that $\mathcal{VSS}.\mathsf{ShVrfy}(\mathcal{VSS}.\mathsf{s}_j, \mathcal{VSS}.\pi_j, \mathcal{VSS}.\vec{C}_j) = 1$ holds for $\mathcal{VSS}.\mathsf{s}_j \in \mathcal{VSS}.\vec{S}_i$, and $\mathcal{VSS}.\pi_j \in \mathcal{VSS}.\vec{W}_i$.

3. **Send blames.** If invalid/no shares are received from the dealer L, then send $\langle \mathsf{blame}, L\rangle_i$ to *all* the nodes. Collect similar blames from other nodes.

4. **Private open.** Send all the collected blames to the dealer L. If p_i is the leader, then for every blame $\langle \mathsf{blame}, L\rangle_k$ received from node p_j, send $(\mathcal{VSS}.\vec{S}_k, \mathcal{VSS}.\vec{W}_k)$ to node p_j.

   If p_i is not L, then ensure that $\mathcal{VSS}.\mathsf{ShVrfy}(\mathcal{VSS}.\mathsf{s}_j, \mathcal{VSS}.\pi_j, \mathcal{VSS}.\vec{C}_j) = 1$ for every $\langle \mathsf{blame}, L\rangle_j$, $\mathcal{VSS}.\mathsf{s}_j \in \mathcal{VSS}.\vec{S}_j$, and $\mathcal{VSS}.\pi_j \in \mathcal{VSS}.\vec{W}_j$.

5. **Ack and decide.** If p_i received $\leq t$ blames and the leader responded with valid shares $(\mathcal{VSS}.\vec{S}_j, \mathcal{VSS}.\vec{W}_j)$ for every $\langle \mathsf{blame}, L\rangle_j$ it forwarded, then send an ack message to the dealer L.

   If p_i is L, then post ack certificate (denoted by $\mathcal{AC}(\mathcal{VSS}.\vec{C})$) on the bulletin board.

   If there is an ack certificate $\mathcal{AC}(\mathcal{VSS}.\vec{C})$ on the bulletin board, commit $\mathcal{VSS}.\vec{C}$, and send $(\mathcal{VSS}.\vec{S}_j, \mathcal{VSS}.\vec{W}_j)$, if received from L.

Reconstruction. Each node $p_i \in \mathcal{P}$ does the following:

1. If there is a share $\mathcal{VSS}.\mathsf{s}_i, \mathcal{VSS}.\pi_i$, send the share and witness to all the nodes.

   On receiving a share and witness $(\mathcal{VSS}.\mathsf{s}_j, \mathcal{VSS}.\pi_j)$ from p_j, ensure that $\mathcal{VSS}.\mathsf{ShVrfy}(\mathcal{VSS}.\mathsf{s}_j, \mathcal{VSS}.\pi_j, \mathcal{VSS}.\mathsf{C}) = 1$.

2. On receiving $t+1$ valid shares in $\mathcal{VSS}.\vec{S}$, reconstruct the secret s using $s \leftarrow \mathcal{VSS}.\mathsf{Recon}(\mathcal{VSS}.\vec{S})$. Send s to all the nodes.

3. On receiving an opened secret s, ensure that $\mathcal{VSS}.\mathsf{ComVrfy}(\mathcal{VSS}.\mathsf{C}, s) = 1$ and output s.

Figure 6: **iVSS - Improved eVSS protocol description**

3. *Commitment*: If the dealer L is Byzantine, then either a vote certificate is formed, or not. If a vote certificate is formed, then at least one honest node has not observed $\geq t+1$ blames, and has received valid shares for every blame. This honest node, say p_i has all the shares for every honest node that does not have a share. Therefore, all honest nodes together have $t+1$ shares, which guarantees reconstruction to the unique secret s that was committed except with $\mathsf{negl}(\kappa)$ probability. If no vote certificate is formed, then all the honest nodes, agree on $\perp$, thus satisfying the *Commitment* requirement with high probability of $1 - \mathsf{negl}(\kappa)$.

□

4.2.2 Randomness Beacon for BRandPiper Protocol

In this section, we instantiate bulletin boards using our SMR protocol (Section 3) and present a random beacon protocol, we call BRandPiper, using the iVSS scheme. When we use our SMR protocol with rotating leaders, a proposed block gets committed within $t+1$ epochs and incurs $O(\kappa n^2)$ bits of communications per epoch for $O(\kappa n)$-sized blocks. To obtain 1-absolute unpredictability, we need reconstruction of at least $t+1$ secrets from distinct nodes in each epoch. For simplicity, we reconstruct one secret from all nodes that have not been removed. Due to round-robin leader election policy, a leader can share secrets at least every n epochs. Thus, an epoch leader proposes commitments to n secrets using SMR protocol. These secrets can be used for n epochs for reconstruction. Our beacon output step takes advantage of the homomorphic properties of the underlying VSS scheme $\mathcal{VSS}$ to combine secret shares for multiple secrets into $O(\kappa)$-sized share which can be efficiently broadcast to all nodes. Honest nodes collect $t+1$ homomorphic shares to reconstruct the common randomness R_e. Such reconstructed randomness is guaranteed to be unbiasable since an adversary cannot know the secrets of honest nodes until reconstructed, and an adversary cannot prevent reconstruction. For the same reason, BRandPiper protocol ensures 1-absolute unpredictability, even for a rushing adaptive adversary.

Protocol Details

Leader selection. We employ a round robin leader selection policy. If an epoch leader p_i fails to commit within $t+1$ epochs, it is added to the set of removed nodes $\mathcal{P}_r$ and is prevented from being a future leader. Remaining nodes $\mathcal{P} \setminus \mathcal{P}_r$ propose in a round robin manner.

Setup. During the *setup* phase, all nodes are provided with VSS parameters $\mathcal{VSS}.\mathsf{pp}$ required for using the VSS scheme $\mathcal{VSS}$. Each node maintains n queues $\mathcal{Q}(p_i)\ \forall p_i \in \mathcal{P}$. Each queue $\mathcal{Q}(p_i)$ holds tuples with each tuple containing a secret share, its witness and commitment proposed by node p_i when node p_i was an epoch leader. During the setup phase, each queue $\mathcal{Q}(p_i)$ is filled with $m = n+t$ tuples containing secret shares, witnesses and commitments for m secrets. This ensures that all honest nodes have a common secret shares in $\mathcal{Q}(p_n)$ and can perform $\mathtt{Dequeue}(\mathcal{Q}(p_n))$ upto epoch $n+t$ even if p_n does not propose in epoch n. This is because honest nodes perform $\mathtt{Dequeue}(\mathcal{Q}(p_n))$ (explained later) in each epoch unless node p_n has been removed. If node p_n does not propose in epoch n, it is removed only in epoch $n+t$.

Block validation protocol. BRandPiper protocol consists of a commitment validation phase, called Block validation protocol, responsible for sharing commitments to secrets along with secret shares, and creating a certificate to prove sufficient secret shares have been shared among honest nodes. The commitment and certificate is then input to the SMR protocol to ensure that all honest nodes agree on the commitment. When the commitment is committed, all honest nodes use corresponding secret shares to reconstruct the common randomness.

The Block validation protocol (refer Figure 7) is executed in parallel with SMR protocol. The leader L_e of epoch e executes the block validation protocol while in epoch $e-1$ to generate an ack certificate for commitments to be proposed in epoch e. The protocol consists of following steps:

<u>Distribute.</u> Leader L_e creates n commitments $\mathcal{VSS}.\vec{C}$ corresponding to n secrets $\{s_1, \ldots, s_n\}$ it wishes to share using $\mathcal{VSS}.\mathsf{ShGen}$ algorithm for secrets $\{s_i | \forall 1 \leq i \leq n\}$, along with share vector $\mathcal{VSS}.\vec{S}_j :=$

The protocol is executed in parallel with BFT SMR protocol in Figure 1 using round robin leader selection. Let L_e be the leader of epoch e and the current epoch be $e-1$. Node p_i performs following operations while in epoch $e-1$:

1. **Distribute.** L_e waits for Δ time after entering epoch $e-1$ and then does the following:
   - Let $\{s_1, \ldots, s_n\}$ be n random numbers chosen uniformly from the input space of $\mathcal{VSS}$.
   - Build $SB := \langle \texttt{Commitment}, \mathcal{VSS}.\vec{C}, e, z_{se} \rangle_{L_e}$, the sharing block which consists of commitments $\mathcal{VSS}.\vec{C} := \{\mathcal{VSS}.\mathsf{C}_1, \ldots, \mathcal{VSS}.\mathsf{C}_n\}$ to the n random numbers generated by running $(\mathcal{VSS}.\vec{S}_i, \mathcal{VSS}.\vec{W}_i, \mathcal{VSS}.\mathsf{C}_i) \leftarrow \mathcal{VSS}.\mathsf{ShGen}(s_i)$ for $i \in \{1, \ldots, n\}$, where $\mathcal{VSS}.\vec{S}_i := \{\mathcal{VSS}.\mathsf{s}_{i,1}, \ldots, \mathcal{VSS}.\mathsf{s}_{i,n}\}$, and $\mathcal{VSS}.\vec{W}_i := \{\mathcal{VSS}.\pi_{i,1}, \ldots, \mathcal{VSS}.\pi_{i,n}\}$.
   - Build the share vector $\mathcal{VSS}.\vec{S}_j := \{\mathcal{VSS}.\mathsf{s}_1 \leftarrow \mathcal{VSS}.\vec{S}_{1,j}, \ldots, \mathcal{VSS}.\mathsf{s}_n \leftarrow \mathcal{VSS}.\vec{S}_{n,j}\}$ and the witness vector $\mathcal{VSS}.\vec{W}_j := \{\mathcal{VSS}.\pi_1 \leftarrow \mathcal{VSS}.\vec{W}_{1,j}, \ldots, \mathcal{VSS}.\pi_n \leftarrow \mathcal{VSS}.\vec{W}_{n,j}\}$ for node p_j using j^{th} share and witness from $\mathcal{VSS}.\vec{S}_i$ and $\mathcal{VSS}.\vec{W}_i$ for random number s_i.
   - Send $\mathcal{VSS}.\vec{S}_j$, $\mathcal{VSS}.\vec{W}_j$, and SB to every node $p_j \in \mathcal{P}$.

2. **Blame/Forward.** If $\mathsf{epoch\text{-}timer}_{e-1} \geq 8\Delta$ and node p_i receives valid share vector $\mathcal{VSS}.\vec{S}_i$, witness vector $\mathcal{VSS}.\vec{W}_i$ and commitment $SB := \langle \texttt{Commitment}, \mathcal{VSS}.\vec{C}, e, z_{se} \rangle_{L_e}$, then invoke $\mathsf{Deliver}(\texttt{Commitment}, SB, z_{se}, e)$. If no shares has been received within 3Δ time while in epoch $e-1$, broadcast a blame $\langle \mathsf{blame}, e \rangle_i$ to all nodes.

3. **Request open.** Wait until $\mathsf{epoch\text{-}timer}_{e-1} \geq 5\Delta$. Collect all blames received so far. If upto t blames are received so far, forward the blames to L_e. If no blames or equivocation by L_e has been detected, send $\langle \mathsf{ack}, H(SB), e \rangle_i$ to L_e.

4. **Private open.** L_e sends valid share $\mathcal{VSS}.\vec{S}_j$ and witness $\mathcal{VSS}.\vec{W}_j$ to node p_i, for every blame $\langle \mathsf{blame}, e \rangle_j$ received from node p_i.

5. **Ack.** Upon receiving valid share $\mathcal{VSS}.\vec{S}_j$ and witness $\mathcal{VSS}.\vec{W}_j$ for every $\langle \mathsf{blame}, e \rangle_j$ it forwarded and detects no equivocation, send $\langle \mathsf{ack}, H(SB), e \rangle_i$ to L_e. Forward share $\mathcal{VSS}.\vec{S}_j$ and witness $\mathcal{VSS}.\vec{W}_j$ to node p_j for every $\langle \mathsf{blame}, e \rangle_j$ it received.

6. **(Non-blocking) Equivocation.** Broadcast equivocating hashes signed by L_e and stop performing any operations.

Figure 7: **Block validation protocol**.

$\{\mathcal{VSS}.\mathsf{s}_{1,j}, \ldots, \mathcal{VSS}.\mathsf{s}_{n,j}\}$ and witness vector $\mathcal{VSS}.\vec{W}_j := \{\mathcal{VSS}.\pi_{1,j}, \ldots, \mathcal{VSS}.\pi_{n,j}\}\ \forall p_j \in \mathcal{P}$. We define a share block $SB := \langle \texttt{Commitment}, \mathcal{VSS}.\vec{C}, e, z_{se}\rangle_{L_e}$ the sharing block to share the commitment $\mathcal{VSS}.\vec{C}$. Leader L_e sends $\mathcal{VSS}.\vec{S}_j$, $\mathcal{VSS}.\vec{W}_j$, and SB to node $p_j\ \forall p_j \in \mathcal{P}$. Similar to the SMR protocol, the leader signs the tuple $\langle \texttt{Commitment}, H(\mathcal{VSS}.\vec{C})), e, z_{se}\rangle_{L_e}$ and sends $\mathcal{VSS}.\vec{C}$ separately to facilitate efficient equivocation checks. It is important to note that commitment $\mathcal{VSS}.\vec{C}$, shares $\mathcal{VSS}.\vec{S}_j$, and witness $\mathcal{VSS}.\vec{W}_j$ are $O(n)$-sized and the shares $\mathcal{VSS}.\mathsf{s}_j$ are only sent to node p_j. Sending only required shares to designated nodes reduces communication complexity.

Blame/Forward. If a node p_i receives a valid secret share $\mathcal{VSS}.\vec{S}_i$, witness $\mathcal{VSS}.\vec{W}_i$, and sharing block $SB := \langle \texttt{Commitment}, \mathcal{VSS}.\vec{C}, e, z_{se}\rangle_{L_e}$ within 3Δ time in epoch $e-1$, it invokes $\mathsf{Deliver}(\texttt{Commitment}, SB, z_{se}, e)$. A valid share satisfies the constraint $\mathcal{VSS}.\mathsf{ShVrfy}(\mathcal{VSS}.\mathsf{s}_j, \mathcal{VSS}.\pi_j, \mathcal{VSS}.\vec{C}_j) = 1\ \forall j \in [n]$. Otherwise, node p_i broadcasts $\langle \mathsf{blame}, e\rangle_i$ to all nodes.

Request open. Node p_i waits for 6Δ time in epoch $e-1$ to collect any blames sent by other nodes. If no blames or equivocation by L_e has been detected within that time, p_i sends $\langle \mathsf{ack}, H(SB), e\rangle_i$ to L_e. If upto t blames are received, p_i forwards the blames to L_e.

Private open. If L_e receives any blames from node p_i, it sends valid $\mathcal{VSS}.\vec{S}_j$, witness $\mathcal{VSS}.\vec{W}_j$ for every blame $\langle \mathsf{blame}, e\rangle_j$ received from node p_i.

Ack. If node p_i forwarded any blames and received valid secret shares $\mathcal{VSS}.\vec{S}_j$ and witness $\mathcal{VSS}.\vec{W}_j$ for every blame $\langle \mathsf{blame}, e\rangle_j$ it forwarded and detects no equivocation, node p_i sends $\langle \mathsf{ack}, H(SB), e\rangle_i$ to L_e. In addition, node p_i forwards secret shares $\mathcal{VSS}.\vec{S}_j$ and witness $\mathcal{VSS}.\vec{W}_j$ for every blame $\langle \mathsf{blame}, e\rangle_j$ it received. Thus, if an honest node sends an ack for the sharing block SB, then all honest nodes have their respective secret shares corresponding to sharing block SB (more details in Lemma 4.10).

Equivocation. At any time in epoch $e-1$, if a node p_i detects an equivocation, it broadcasts equivocating hashes signed by L_e and stops participating in epoch $e-1$ block validation protocol.

Beacon protocol. We now present the beacon protocol (refer Figure 8) for the BRandPiper protocol. The beacon protocol consists of the following rules for an epoch e. Here, an epoch corresponds to an epoch in SMR protocol.

Generate Blocks. The leader L_e of an epoch e chooses n secrets uniformly at random and invokes block validation protocol while in epoch $e-1$ to obtain an ack certificate (denoted by $\mathcal{AC}_e(SB)$), for the sharing block SB corresponding to the n secrets. In epoch e, the leader proposes block B_h with $b_h := (H(SB), \mathcal{AC}_e(SB))$ where $\mathcal{AC}_e(SB)$ is an ack certificate for commitment SB using the SMR protocol. We redefine valid blocks for the SMR protocol with an additional constraint to contain an ack certificate created in epoch $e-1$[4] and all honest nodes vote in the SMR protocol as long as the proposed block meets this additional constraint. As mentioned before, an ack certificate for a sharing block SB implies all honest nodes have secret shares required to reconstruct the secrets corresponding to commitments in SB. Thus, it is safe for honest nodes to vote in the SMR protocol although they sent blame during block validation phase.

Update. At the end of epoch e, node p_i updates $\mathcal{Q}(L_{e-t})$ as follows. If L_{e-t} proposed a valid block B_l in epoch $e-t$ and B_l has been committed by epoch e, node p_i replaces the contents of $\mathcal{Q}(L_{e-t})$ with n tuples with each tuple containing secret shares, witnesses and commitments shared in epoch $e-t$. If no epoch $e-t$ block was committed, it removes L_{e-t} from future proposals, i.e., $\mathcal{P}_r \leftarrow \mathcal{P}_r \cup \{L_{e-t}\}$. It is important to note that the SMR protocol guarantees all honest nodes commit proposed blocks in $t+1$ epochs. Thus, all honest nodes either update $\mathcal{Q}(L_{e-t})$ or remove L_{e-t} in epoch e.

[4]For the first epoch, an ack certificate can be created during the setup phase.

Let $\mathcal{VSS}$ be the VSS scheme used, e be the current epoch and L_e be the leader of epoch e. Node $p_i \in \mathcal{P}$ augments SMR protocol in Figure 1 as follows:

- **Setup.** Set $e = 1$. All nodes agree upon and fill $\mathcal{Q}(p_i)$ with $m = n + t$ tuples $\forall p_i \in \mathcal{P}$. Set $\mathcal{P}_r \leftarrow \emptyset$. Run $\mathcal{VSS}.\mathsf{Setup}$ and agree on the public parameters $\mathcal{VSS}.\mathsf{pp}$. Set $L_e \leftarrow p_1$.

- **Blocks.** While in epoch $e - 1$, leader L_e starts the block validation protocol (refer Figure 7) with $\{s_1, \ldots, s_n\}$, where the secrets are chosen randomly $s_i \leftarrow_\$ \{0,1\}^\kappa$ for $1 \le i \le n$.

  In epoch e, L_e proposes block B_h with $b_h := (H(SB), \mathcal{AC}_e(SB))$ where $\mathcal{AC}_e(SB)$ is an ack certificate for commitment SB.

- **Update.** When $\mathsf{epoch\text{-}timer}_e$ expires, if L_{e-t} proposed a valid block B_l in epoch $e - t$ and B_l has been committed by epoch e, update $\mathcal{Q}(L_{e-t})$ with n tuples with each tuple containing secret shares, witnesses and commitments shared in epoch $e - t$. Otherwise, remove L_{e-t} from future proposals i.e., $\mathcal{P}_r \leftarrow \mathcal{P}_r \cup \{L_{e-t}\}$.

- **Reconstruct.** When $\mathsf{epoch\text{-}timer}_e$ expires, do the following:

  1. Get $(\mathcal{VSS}.\vec{S}, \mathcal{VSS}.\vec{W}, \mathcal{VSS}.\vec{C}) := \{\mathtt{Dequeue}(\mathcal{Q}(p_j)) | p_j \notin \mathcal{P}_r\}$.
  2. Build homomorphic sum share SV_i, witness $\mathcal{VSS}.\pi_i$, and commitment $\mathcal{VSS}.\mathsf{C}_e$ using all shares from $\mathcal{VSS}.\vec{C}$. Send SV_i and $\mathcal{VSS}.\pi_i$ to all the nodes.
  3. Upon receiving share SV_j and witness $\mathcal{VSS}.\pi_j$ for $\mathcal{VSS}.\mathsf{C}_e$, ensure that $\mathcal{VSS}.\mathsf{ShVrfy}(SV_j, \mathcal{VSS}.\pi_j, \mathcal{VSS}.\mathsf{C}_e) = 1$.
  4. Upon receiving $(t + 1)$ valid homomorphic sum shares in SV, obtain $R_e \leftarrow \mathcal{VSS}.\mathsf{Recon}(SV)$.

- **Output.** Compute and output $\mathcal{O}_e \leftarrow H(R_e)$.

Figure 8: **RandPiper – BRandPiper beacon protocol.**

Reconstruct. At the end of epoch e, each nodes perform $\{\mathtt{Dequeue}(\mathcal{Q}(p_j)) \forall p_j \in \mathcal{P} \setminus \mathcal{P}_r\}$ to fetch n secret shares (one from each node) and corresponding witnesses. It computes the homomorphic sum of shares and witnesses and broadcasts it to all nodes.

Output. From the above discussion, it is clear that all honest nodes send homomorphic sum of shares for common commitments and all honest nodes will receive at least $t + 1$ valid homomorphic shares. When a node p_i receives $t+1$ homomorphic shares, it reconstructs the randomness R_e using $\mathcal{VSS}.\mathsf{Recon}$ primitive and computes $\mathcal{O}_e \leftarrow H(R_e)$.

4.2.3 BRandPiper Security Analysis

Lemma 4.10. *If an honest node sends an* ack *for a sharing block SB in epoch e, then (i) all honest nodes receive the sharing block SB in epoch e, (ii) all honest nodes receive their respective secret shares corresponding to sharing block SB within Δ time of entering epoch $e + 1$.*

Proof. Suppose an honest node p_i sends an ack for sharing block $SB := \langle \mathtt{Commitment}, \mathcal{VSS}.\vec{C}, e, z_{se} \rangle_{L_e}$ at time τ in epoch e. Node p_i must have received upto t blame messages. This implies at least one honest node p_j received a valid share $\mathcal{VSS}.\mathsf{s}_i$ and sharing block SB within 3Δ time in epoch e and invoked $\mathsf{Deliver}(\mathtt{Commitment}, SB, z_{se}, e)$. Let τ' be the time when node p_j invoked $\mathsf{Deliver}(\mathtt{Commitment}, SB, z_{se}, e)$. The earliest node p_i sends an ack for SB is when it waits until $\mathsf{epoch\text{-}timer}_e \ge 5\Delta$ (i.e., 6Δ in epoch e) and does not detect any equivocation by L_e or any blame messages. Due to Δ delay between honest nodes entering into epoch e, this time corresponds to $\tau' + 2\Delta$ in the worst case. This implies no honest node received an epoch e equivocation by time $\tau' + \Delta$. By Fact 3.1, all honest nodes receives the sharing block SB. This proves part (i) of the Lemma.

For part (ii), node p_i can send ack on two occasions: (a) when it does not detect any equivocation or blame until its $\textsf{epoch-timer}_e \geq 5\Delta$, and (b) when leader L_{e+1} sent valid secret shares for every blame message it forwarded and does not detect any equivocation by time τ.

In case (a), node p_i did not detect any equivocation or blame messages until its $\textsf{epoch-timer}_e > 5\Delta$ at time τ. Observe that all honest nodes must have received valid shares corresponding to the sharing block SB within 3Δ time in epoch e; otherwise node p_i must have received blame message by time τ (since honest nodes may enter epoch e with Δ time difference and send blame message if no valid secret shares received within 3Δ time in epoch e). In addition, no honest node received an equivocating sharing block SB' within 3Δ time in epoch e; otherwise, node p_i must have received a share for SB' (via Deliver) by time τ. Thus, all honest nodes receive their respective secret shares corresponding to sharing block SB in epoch e (i.e., within Δ time of entering epoch $e+1$).

In case (b), node p_i receives valid secret shares from leader L_{e+1} for every blame (up to t blame) messages it forwarded and detected no equivocation by time τ. Observe that node p_i received $f \leq t$ blame messages and received valid shares for every blame message it forwarded. This implies at least $n - t - f$ honest nodes have received valid shares for sharing block SB from leader L_{e+1} within 3Δ in epoch e; otherwise, node p_i would have received more than f blame message by the time its $\textsf{epoch-timer}_e = 5\Delta$. Since, node p_i forwards f received secret shares corresponding to f received blame message in epoch e and honest nodes enter epoch $e+1$ within Δ time, all honest nodes receive their respective secret shares corresponding to sharing block SB within Δ time of entering epoch $e+1$.

□

Theorem 4.11 (Consistent Beacon)**.** *For any epoch e, all honest nodes reconstruct the same randomness R_e and output the same beacon $\mathcal{O}_e$.*

Proof. Honest nodes output the same randomness R_e and output the same beacon $\mathcal{O}_e$ in epoch e if all honest nodes receive $t+1$ valid homomorphic shares for the same set of secrets. This condition is satisfied if all honest nodes (i) have consistent $\mathcal{Q}(p_i)\ \forall p_i \in \mathcal{P}$ and consistent $\mathcal{P}_r$ in each epoch, (ii) $\{\texttt{Dequeue}(\mathcal{Q}(p_i)) \neq \perp \forall p_i \in \mathcal{P} \setminus \mathcal{P}_r\}$ in each epoch, and (iii) share valid homomorphic shares corresponding to dequeued secret shares.

For part(i), we show all honest nodes have consistent $\mathcal{Q}(p_i)\ \forall p_i \in \mathcal{P}$ and consistent $\mathcal{P}_r$ in every epoch.

We prove part (i) by induction on epochs. Consider the base case for epochs 1 to t. During setup phase, each node is assigned $m = n+t$ tuples (with each tuple containing secret shares, witnesses and commitments) for each $\mathcal{Q}(p_i)\ \forall p_i \in \mathcal{P}$ (i.e., $m * n$ secrets in total). Since, removing a Byzantine node requires $t+1$ epochs, all honest nodes have $\mathcal{P}_r = \emptyset$ for epochs 1 to t. In addition, no honest node update $\mathcal{Q}(p_i)$ 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 $\mathcal{P}_r$.

We assume part(i) holds until epoch $e-1$.

Consider an epoch $e > t$. In epoch e, all honest nodes update only $\mathcal{Q}(L_{e-t})$. If L_{e-t} proposed a valid block B_l (with $b_l = (H(SB), \textsf{ack-cert}(SB))$ for some commitment SB and B_l is committed by epoch e, all honest nodes update $\mathcal{Q}(L_{e-t})$ with n tuples containing secret shares, witnesses and commitments in SB (by Lemma 4.10, all honest nodes receive commitments and secret shares in SB before epoch e). Otherwise, all honest nodes update $\mathcal{P}_r$ to exclude L_{e-t} i.e., $\mathcal{P}_r \leftarrow \mathcal{P}_r \cup \{L_{e-t}\}$. Thus, all honest nodes should have consistent $\mathcal{Q}(L_{e-t})$ by epoch e. Since honest nodes do not update $\mathcal{Q}(p_i \neq L_{e-t})$ and do not add p_i into $\mathcal{P}_r$ in epoch e, by induction hypothesis, all honest nodes should have consistent $\mathcal{Q}(p_i)\ \forall p_i \in \mathcal{P}$ and consistent $\mathcal{P}_r$ in epoch e. This proves part(i). 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 secrets.

Next, we show $\{\texttt{Dequeue}(\mathcal{Q}(p_i)) \neq \perp \forall p_i \in \mathcal{P} \setminus \mathcal{P}_r\}$ in epoch e. Suppose for the sake of contradiction, $\texttt{Dequeue}(\mathcal{Q}(p_i)) = \perp$ and $p_i \notin \mathcal{P}_r$ in epoch e. 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 e' be the last epoch in which node p_i last proposed with $e' \leq e-t$. However, if node p_i did not propose in e', all honest nodes would have removed p_i by epoch $e'+t \leq e$ and $p_i \in \mathcal{P}_r$ in epoch e. A contradiction.

Finally, we show all honest nodes send valid homomorphic shares for the dequeued secret shares. Observe that honest nodes only dequeue secret shares corresponding to a committed block that contains a valid ack certificate. By Lemma 4.10 part(ii), all honest nodes receive valid secret shares before honest nodes update their queues. Thus, all nodes will dequeue common secret shares and will receive at least $t+1$ valid homomorphic shares for a common secrets and reconstruct the same randomness R_e and output the same beacon $\mathcal{O}_e$. □

Lemma 4.12 (Liveness)**.** *If the leader L_e of an epoch e is honest, then (i) an* ack *certificate for its sharing block SB will form in epoch $e-1$, and (ii) all honest nodes commit $(H(SB), \mathcal{AC}_e(SB))$ in epoch e.*

Proof. Consider an honest leader L_e for an epoch e. Let τ be the time when leader L_e enters epoch $e-1$. Leader L_e waits for Δ time after entering epoch $e-1$ and must have sent valid shares $\mathcal{VSS}.\mathsf{s}_i$ and sharing block SB containing commitments to node p_i $\forall p_i \in \mathcal{P}$ at time $\tau + \Delta$.

Since honest nodes enter epoch $e-1$ within Δ time, all honest nodes must have entered epoch $e-1$ by time $\tau+\Delta$. Leader L_e could have entered epoch $e-1$ Δ time before some honest nodes or Leader L_e could have entered epoch $e-1$ Δ time after some honest nodes. In any case, all honest nodes must have received valid secret shares and sharing block SB within 3Δ after entering epoch $e-1$. Thus, no honest nodes send blame in epoch $e-1$ and will receive no blame messages from honest nodes within 6Δ time in epoch $e-1$ (i.e., until $\mathsf{epoch\text{-}timer}_{e-1} > 5\Delta$).

Consider an honest node p_i. If node p_i receives no blame messages from Byzantine nodes, it will send an ack for sharing block SB to L_e. On the other hand, if node p_i receives upto t blame messages from Byzantine nodes, it forwards blame messages to L_e. Honest Leader L_e sends the shares corresponding to the blame messages to node p_i which node p_i receives within 8Δ in epoch $e-1$. Moreover, there is no equivocation from leader L_e. Thus, node p_i sends an ack for sharing block SB to L_e.

Thus, all honest nodes send ack for sharing block SB and leader L_e receives $t+1$ ack message for sharing block SB within 10Δ (L_e may start epoch $e-1$ Δ time before node p_i) in epoch $e-1$. This proves part (i) of the Lemma.

Since leader L_e proposes a valid proposal $(H(SB), \mathcal{AC}_e(SB))$ in epoch e, part(ii) follows immediately from Fact 3.6. □

Lemma 4.13 (Guaranteed Beacon Output)**.** *For each epoch e, all honest nodes output a new beacon output $\mathcal{O}_e$.*

Proof. Due to round-robin leader election, honest nodes propose in at least $n-t$ epochs out of n epochs. By Lemma 4.12, all honest nodes commit n new secret shares in every honest epoch and updates their queues after $t+1$ epochs. Thus, $\mathtt{Dequeue}(\mathcal{Q}(p_i)) \neq \perp \forall p_i \in \mathcal{P} \setminus \mathcal{P}_r$. where p_i is an honest node. From the proof of Theorem 4.11, all honest nodes have consistent queues and $\mathcal{P}_r$ in each epoch. At the end of each epoch, all honest nodes dequeue common secret shares and send homomorphic sums to all other nodes. Thus, honest nodes will have $t+1$ valid homomorphic sums and will output new beacon outputs in every epochs.

□

Lemma 4.14 (Communication Complexity)**.** *Let $f \leq t$ be the number of actual Byzantine faults, κ be the size of accumulator and w be the size of witness. The amortized communication complexity of the protocol is $O(\kappa f n^2 + (\kappa + w)n^2)$ bits per epoch.*

Proof. In the Block validation protocol, distributing $O(\kappa n)$-sized commitment has a cost of $O(\kappa n^2)$ communication. Sending corresponding $O(\kappa n)$-sized secret shares and $O(wn)$-sized witness incur $O((\kappa + w)n^2)$ communication. Next, f Byzantine nodes can always blame even when the epoch leader is honest. Thus, an epoch leader needs to send $O(\kappa f n)$-sized secret shares while privately opening the secret shares. The nodes also forward privately opened secret secrets to nodes that blamed. This step has a cost of $O(\kappa f n^2)$ in an honest epoch. When the leader is Byzantine, it can create a scenario when upto t nodes send blame and hence, this step has $O(\kappa t n^2)$ cost. Out of n consecutive epochs, there can be at most f Byzantine epochs and $n-f$ honest epochs. Hence, this step has amortized complexity of $O(\kappa f n^2)$.

By Lemma 3.8, SMR protocol has a cost $O((\kappa + w)n^2)$ bits for input of size $O(\kappa n)$. The homomorphic sum of secret shares is κ and homomorphic sum of witness is w. Thus, all-to-all broadcast of homomorphic sums incurs $O((\kappa + w)n^2)$. Thus, the amortized communication complexity is $O(\kappa f n^2 + (\kappa + w)n^2)$ bits per epoch. □

Now, we prove the beacon properties for BRandPiper protocol. We first show that BRandPiper is a secure VSS protocol as per Definition 2.5. The proof is similar to the proof in Theorem 4.9, but we present it here for completeness.

Theorem 4.15 (Secure VSS)**.** *The BRandPiper protocol is a secure verifiable secret sharing protocol with the dealer as the leader of an epoch, and the rest of the nodes as the verifiers.*

Proof. We prove the security of VSS by proving the individual properties:

1. *Secrecy*: When the leader L_e for an epoch e is honest, then no honest node will blame, and therefore an adversary $\mathcal{A}$ will only learn the t shares of its own corruption, and not learn any new share by blaming. Therefore the probability of $\mathcal{A}$ of violating the secrecy property is $\mathsf{negl}(\kappa)$ from the underlying $\mathcal{VSS}$ scheme.

2. *Correctness*: When the leader L_e of epoch e is honest, from Lemma 4.12, all honest nodes commit the SB with shares for the secret. During the reconstruction for the beacon, every honest node $p_i \in \mathcal{P}$ use the same share for $SV_{L_e,i}$ with a high probability of $1 - \mathsf{negl}(\kappa)$. A Byzantine node $p_j \in \mathcal{P}$ cannot provide a valid witness $\mathcal{VSS}.\pi_{L_e,j}$ for an incorrect share with probability better than $\mathsf{negl}(\kappa)$, thereby ensuring that the correctness property is maintained.

3. *Commitment*: If an honest node commits a valid block SB from a byzantine leader L_e in some epoch e, then all honest nodes commit SB, from the SMR property in Theorem 3.5. Therefore during reconstruction, a Byzantine node $p_j \in \mathcal{P}$ cannot provide incorrect shares as it cannot generate a valid witness $\mathcal{VSS}.\pi_{L_e,j}$ (except with $\mathsf{negl}(\kappa)$ probability). If a Byzantine leader does not propose any block, then all honest nodes agree on $\perp$, thereby ensuring the commitment property.

□

Next, we show that BRandPiper satisfies the bias-resistance requirement from Definition 2.3.

Lemma 4.16 (Bias-resistance)**.** *For any epoch $e \geq 1$, the probability that a t bounded adversary $\mathcal{A}$ can fix any c bits of the BRandPiper beacon output $\mathcal{O}_e$ is $\frac{1}{2^c} + \mathsf{negl}(\kappa)$.*

Proof. The output in any epoch e is $\mathcal{O}_e \leftarrow H(R_e)$, where R_e is the homomorphic sum of secrets from at least $t+1$ honest nodes. From the *secrecy* guarantee in Theorem 4.15, we know that no adversary $\mathcal{A}$ can predict the value of these honest nodes until reconstruction with probability better than $\mathsf{negl}(\kappa)$. At the same time, no adversary $\mathcal{A}$ can change the committed value for any p_i during reconstruction due to the *commitment* guarantee from Theorem 4.15 with probability better than $\mathsf{negl}(\kappa)$. Therefore, a t-bounded adversary cannot do better than guessing whose probability is $\frac{1}{2^c} + \mathsf{negl}(\kappa)$ to fix c bits in the output $\mathcal{O}_e$ for any epoch $e \geq 1$. □

Next, we show that BRandPiper satisfies the 1-absolute unpredictability requirement from Definition 2.3.

Lemma 4.17 (BRandPiper 1-absolute unpredictability)**.** *The BRandPiper random beacon protocol is an* 1*-absolute unpredictable random beacon.*

Proof. Let τ be some time at which all honest nodes are in an epoch $e \geq 1$. We show that an adversary $\mathcal{A}$ cannot predict $\mathcal{O}_{e+1}$. Due to the secrecy property in Theorem 4.15 and the fact that the beacon output $\mathcal{O}_{e+1}$ is derived from the reconstruction of R_{e+1}, which is a homomorphic sum of inputs from at least $n-t$ honest nodes. From Lemma 4.16 we know that no adversary $\mathcal{A}$ can bias $\mathcal{O}_{e+1}$. The values from the honest nodes are guaranteed to be truly random (by definition). Therefore, the output $\mathcal{O}_{e+1}$ is unpredictable for an adversary $\mathcal{A}$.

An adversary $\mathcal{A}$ can get a 1 epoch advantage since there can exists times τ where some honest nodes are in epoch e and others are in epoch $e-1$. At this point, a rushing adversary knows the output $\mathcal{O}_e$ before the honest nodes. □

Lemma 4.18 (Rushing Adversary Advantage)**.** *For any epoch e, a rushing adversary can reconstruct output $\mathcal{O}_e$ at most 2Δ time before honest nodes.*

The proof remains identical to Lemma 4.3.

Theorem 4.19 (BRandPiper Secure Random Beacon)**.** *BRandPiper protocol is a 1-secure random beacon.*

Proof. The proofs follow trivially from Lemma 4.16, Lemma 4.17, and Lemma 4.13. □

Acknowledgements

We would like to thank Alin Tomescu for helpful feedback on the q-SDH assumption. This work is supported in part by research gift grants from VMware Research and Novi, IARPA HECTOR program, the National Science Foundation under grant CNS-1846316, and ARL.

References

[1] blockchain oracle service, enabling data-rich smart contracts.

[2] Generate random numbers for smart contracts using chainlink vrf.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 590–609. Springer, 2011.

[9] Shehar Bano, Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, et al. State machine replication in the libra blockchain. *Avalaible at: https://developers. libra. org/docs/state-machine-replication-paper (Consulted on December 19, 2020)*, 2020.

[10] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *International conference on the theory and applications of cryptographic techniques*, pages 480–494. Springer, 1997.

[11] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains.* PhD thesis, 2016.

[12] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[13] 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.

[14] 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.

[15] TH Hubert Chan, Rafael Pass, and Elaine Shi. Pili: An extremely simple synchronous blockchain. 2018.

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

[17] Information Technology Laboratory Computer Security Division. Interoperable randomness beacons: Csrc.

[18] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.

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

[20] J Drake. Minimal vdf randomness beacon. ethereum research post (2018).

[21] Drand. Drand - a distributed randomness beacon daemon.

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

[23] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

[24] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: A scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 568–580. IEEE, 2019.

[25] Mads Haahr. True random number service.

[26] Runchao Han, Jiangshan Yu, and Haoyu Lin. Randchain: Decentralised randomness beacon from sequential proof-of-work.

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

[28] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *International conference on the theory and application of cryptology and information security*, pages 177–194. Springer, 2010.

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

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

[31] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H Vaidya, and Zhuolun Xiang. Improved extension protocols for byzantine broadcast and agreement. *arXiv preprint arXiv:2002.11321*, 2020.

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

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

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

[39] 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.

[40] 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.