Random Beacons in Monte Carlo: Efficient Asynchronous Random Beacon *without* Threshold Cryptography

Akhil Bandarupalli
Purdue University
abandaru@purdue.edu

Adithya Bhat
Visa Research
haxolotl.research@gmail.com

Saurabh Bagchi
Purdue University
sbagchi@purdue.edu

Aniket Kate
Purdue University / Supra Research
aniket@purdue.edu

Michael K. Reiter
Duke University / Chainlink Labs
michael.reiter@duke.edu

ABSTRACT

Regular access to unpredictable and bias-resistant randomness is important for applications such as blockchains, voting, and secure distributed computing. Distributed random beacon protocols address this need by distributing trust across multiple nodes, with the majority of them assumed to be honest. Numerous applications across the blockchain space have led to the proposal of several distributed random beacon protocols, with some already implemented. However, many current random beacon systems rely on threshold cryptographic setups or exhibit high computational costs, while others expect the network to be partial or bounded synchronous. To overcome these limitations, we propose HashRand, a computation and communication-efficient asynchronous random beacon protocol that only demands secure hash and pairwise secure channels to generate beacons. HashRand has a per-node amortized communication complexity of $O(\lambda n \log(n))$ bits per beacon. The computational efficiency of HashRand is attributed to the two orders of magnitude lower time of a one-way Hash computation compared to discrete log exponentiation. Interestingly, besides reduced overhead, HashRand achieves Post-Quantum security by leveraging the secure Hash function against quantum adversaries, setting it apart from other random beacon protocols that use discrete log cryptography. In a geo-distributed testbed of $n = 136$ nodes, HashRand produces 78 beacons per minute, which is at least 5x higher than Spurt [IEEE S&P'22]. We also demonstrate the practical utility of HashRand by implementing a Post-Quantum secure Asynchronous SMR protocol, which has a response rate of over 135k transactions per second at a latency of 2.3 seconds over a WAN for $n = 16$ nodes.

CCS CONCEPTS

• **Security and privacy → Systems security**; **Distributed systems security**;

KEYWORDS

Random Beacon, Byzantine Fault-Tolerance, Asynchronous, Hash functions, Approximate Agreement, Post-Quantum Security

ACM Reference Format:
Akhil Bandarupalli, Adithya Bhat, Saurabh Bagchi, Aniket Kate, and Michael K. Reiter. 2024. Random Beacons in Monte Carlo: Efficient Asynchronous Random Beacon *without* Threshold Cryptography . In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3658644.3670326

1 INTRODUCTION

Random beacon schemes [64, 76] emit random numbers at regular intervals. They are often used as a source of secure randomness in applications such as committee election in sharded and Proof-of-Stake blockchains [36, 49, 52, 71]; common coins in asynchronous Byzantine Agreement (BA) [3, 6, 66, 68]; State Machine Replication (SMR) [30, 46, 51, 52, 63, 67, 82]; asynchronous Multi-Party Computation (MPC) [28, 74]; and also in layer-2 blockchain applications in the DeFi and GameFi space [24]. The demand for secure randomness from these applications has recently surged and continues to grow further [24].

Contemporary random beacon protocols make a wide variety of trusted setup assumptions. Beacons from sources such as Random.org [54] or NIST's random beacon project [64] place complete trust in the source. The sources here become single points of failure for the system's liveness and safety as well as unpredictability and bias-resistance of beacon values. Approaches like DRand [76], Dfinity-DVRF [56], and Cachin et al. [19] mitigate these risks by distributing the trust across some n nodes such that the system remains safe, live, and secure as long as a subset of $t + 1$ or more nodes are honest.

Under the hood, these approaches employ a threshold cryptographic setup among n nodes that enables them to generate (n, t)-threshold unique signatures, which offer unpredictability and bias-resistance against an adversary corrupting $t < \frac{n}{3}$ nodes ($t < \frac{n}{2}$ in a synchronous network). Here, the trusted setup phase for threshold signatures can be replaced with a Distributed Key Generation (DKG) protocol [48, 60, 65]. However, DKG protocols are expensive in computation and communication and must be re-executed every time nodes' participation changes. This constraint makes threshold signatures costly for applications with churn, such as Proof-of-Stake blockchains [49].

As a result, a few recent random beacon proposals [15, 16, 31, 55, 72, 77] avoid the threshold setup and the threshold unique signature approach. Instead, they employ a Public Key Infrastructure (PKI) setup or broadcast setup, where digital signatures and secure channels can be used to realize Verifiable Secret Sharing (VSS) and State Machine Replication (SMR) primitives and build a beacon service on top of those. However, they depend on some form of network synchrony (bounded/partial) to achieve liveness and agreement amongst participant nodes, and cannot be used in the asynchronous communication setting. Indeed, as demonstrated by Miller et al. [67], synchronous protocols cannot take full advantage of the

network speed and out-of-order message delivery, thereby offering subpar performance under active adversarial scheduling.

Asynchronous randomness generation protocols that do not use threshold signatures are available: Some of those use a PKI setup [4, 5, 32, 34, 47], while other need a pairwise-channels setup [37, 65]. However, all of these protocols have a high computational cost due to discrete-log-based commitments used in their n-parallel Asynchronous VSS (AVSS) phases. For example, in protocols with PKI setup like Abraham et al. [5] and Bingo [4], the computational cost of n-parallel AVSS amounts to $\Omega(n^2)$ discrete log exponentiations and $O(n^2)$ bilinear pairing computations per node per beacon. As an exponentiation and a pairing respectively consume 100× and 1000× more compute time and energy than a cryptographic hash, there is significant scope for improvement. In protocols with only secure channels such as Kogias et al. [65] and Freitas et al. [37], the cost of n-parallel AVSS increases to $O(n^3)$ discrete log exponentiations per beacon per node. Overall, high computational cost becomes a scalability bottleneck for the existing asynchronous randomness generation protocols as n grows.

Asynchronous beacon protocols in the realm of Information-Theoretic and Statistical security utilize computation-efficient techniques that do not use any expensive cryptographic operations. For example, Huang et al. [59] use a technique called statistical fraud detection to detect misbehavior by Byzantine-faulty nodes. This approach has a communication complexity of $\Omega(n^6)$ bits per beacon. Although such approaches are computationally efficient, their high communication complexity hinders their practical use.

In this work, we ask the following question: *Is there an asynchronous random beacon protocol that does not use any threshold cryptographic setup, is computationally efficient to output a high throughput of beacons at high values of n, and has a practically scalable communication complexity?*

Our solution. We present HashRand, a computation-efficient asynchronous random beacon protocol that uses hash functions and a secure channel setup to output random beacons. HashRand has an amortized $O(\lambda n \log(n))$ communication complexity per beacon per node, where λ is the range of the hash function in bits, and requires amortized $O(\lambda n \log(n))$ Hash computations per beacon per node. HashRand requires only a pairwise secure channel setup, where nodes can send private, authenticated messages to each other. HashRand is computationally efficient because it only uses lightweight cryptographic primitives like hash functions and authenticated symmetric encryption. On top of computational efficiency and practical communication complexity, HashRand is also post-quantum secure assuming that cryptographic hash functions like SHA256 behave like a Random Oracle against a polynomial-time quantum adversary [18]. We offer a detailed comparison of HashRand with related works in Table 1.

Evaluation. We implement HashRand in Rust and evaluate it in a geo-distributed setting on AWS by measuring the throughput in terms of the number of beacons output per minute. We also compare HashRand with Dfinity-DVRF [56] based on BLS threshold signatures and Spurt [31], which require a trusted DKG setup and a PKI setup, respectively. Dfinity-DVRF's computation and communication complexities are equivalent to Cachin et al. [19]. At $n = 40$ nodes, HashRand outputs 1837 beacons per minute, against 1109 beacons pm for Dfinity-DVRF and 100 beacons pm for Spurt. The computational efficiency of Hash functions allows HashRand to produce beacons at a higher throughput at lower n.

With increasing n, HashRand's throughput drops faster than Dfinity-DVRF, with HashRand doing worse than Dfinity-DVRF at $n = 64$. This is because Dfinity-DVRF requires $O(n)$ discrete-log exponentiations per beacon whereas HashRand scales as $n^2 \log(n)$ hash computations per beacon at $n < 64$ (Refer to Table 1 for details). However, recall that Dfinity-DVRF is not post-quantum secure and needs DKG for a threshold signature setup, which is expensive for systems with churn. On the other hand, HashRand continues to scale much better than threshold-setup-free Spurt [31], outputting 5× the throughput of Spurt at $n = 136$ nodes. Hence, HashRand is the optimal choice in situations where threshold setup is infeasible.

An illustrative application. On top of implementing and evaluating HashRand as an asynchronous beacon, we demonstrate its practical utility by realizing a post-quantum secure asynchronous SMR protocol. We integrate HashRand with Tusk [30], the state-of-the-art asynchronous SMR protocol, to form PQ-Tusk, in which HashRand provides the randomness needed for wave leader election. We replace Tusk 's digital signatures with DiLithium [44], a post-quantum secure public key signature scheme. PQ-Tusk has an optimal communication complexity of $O(\lambda n)$ bits per transaction per node, which is much more practical than other post-quantum SMR protocols like WaterBear [82] with worst case $O(\exp(n))$ communication complexity. We also demonstrate the practicality of PQ-SMR by evaluating our SMR protocol PQ-Tusk in a geo-distributed setting. PQ-Tusk achieves a throughput of 135k transactions per second for $n = 16$ nodes with a latency of 2.3 seconds, compared to 150k transactions per second at a latency of 2.1 seconds for Tusk. We also propose a signature-free asynchronous SMR protocol based on DAG-Rider [63] for cross-chain consensus [78], an application where signatures are expensive and infeasible.

2 SOLUTION OVERVIEW

2.1 System Model

We assume a set of n nodes $\mathcal{N}$ connected by pairwise authenticated and secure channels. We consider a polynomial-time quantum adaptive adversary $\mathcal{A}$ defined according to Boneh et al. [18], who has access to a quantum computer and controls $t < \frac{n}{3}$ nodes. The channels connecting honest nodes are private and $\mathcal{A}$ cannot distinguish the contents of each message from a uniformly random bitstring within polynomial time. An efficient post-quantum symmetric encryption scheme like AES can satisfy this assumption. The network is asynchronous, where $\mathcal{A}$ can arbitrarily delay and reorder messages. However, it cannot drop messages between two honest nodes.

2.2 Asynchronous Random Beacons

We define a random beacon protocol and its properties based on Spurt's [31] definition and adapt it to asynchrony. Honest nodes participating in a beacon protocol $\mathcal{B}$ output tuples of the form $\langle i, b_i \rangle$, where $i \in \mathbb{N}$ is a unique beacon index and $b_i \in \mathcal{D}$ is an element from a predefined domain of numbers $\mathcal{D}$.

Definition 2.1. *A protocol $\mathcal{B}$ amongst n nodes $\mathcal{N}$ consists of two subprotocols $\mathcal{B}$.Prep(,,) and $\mathcal{B}$.Open(.), with outputs of the form*

Table 1: Comparison of relevant random beacon and asynchronous common coin protocols with HashRand.

Protocol	Network	Post Quantum Security	Termination Flavor	Adaptive Security	Communication Complexity (per node)	Computational Complexity (per node)	Setup	Cryptographic Assumption
Cachin et al. [19]	async.	✗	D	✓	$O(\lambda n)$	$O(n)$ DLE	DKG	pRO & CDH
RandShare [77]	async.	✗	LV	✗	$O(\lambda n^3)$	$O(n^3)$ DLE	DKG	DLog
Spurt [31]	partial sync.	✗	D	✗	$O(\lambda n^2)$	$O(n^2)$ DLE¶	CRS & PKI	pRO & DBDH
Kogias et al. [65]	async.	✗	LV	✗	$O(\lambda n^3)$	$O(n^3)$ DLE	Secure channels	pRO & DDH
Das et al. [34]	async.	✗	LV	✗	$O(\lambda n^2)$	$O(n^3)$ DLE	PKI	pRO & DDH
Gao et al. [47]	async.	✗	LV	✗	$O(\lambda n^2)$	$\Omega(n^2)$ DLE	PKI	pRO & SXDH
Abraham et al. [5]	async.	✗	LV	✗	$O(\lambda n^2)$	$\Omega(n^2)$ DLE	PKI	pRO & SXDH
Bingo [4]	async.	✗	LV	✓	$O(\lambda n^2)$	$\Omega(n^2)$ DLE	SRS & PKI	q−SDH
Freitas et al. [37]	async.	✓	MC	✗	$O(\lambda n^2 \log(n))$	$O(n^3)$ DLE	Secure channels	DLog
Huang et al. [59]	async.	✓	LV	✓	$\Omega(n^6)$	0	Auth channels	None
HashRand	async.	✓	MC	✓	$O(\lambda n \log(n))^{\dagger}$	$O(n \log(n))$ H*	Secure channels	pRO§

Termination Flavor: Monte-Carlo (MC) style protocols terminate in a deterministic number of rounds. But, even after termination, honest nodes can disagree on the beacon with a practically negligible probability $p = 1 - \delta$. A lower p implies a higher round complexity. Las Vegas (LV) style protocols terminate only upon agreement amongst honest nodes on the beacon value. These protocols can have infinitely many executions and we report the expected runtime complexities of these protocols. Deterministic (D) protocols offer agreement and a deterministic runtime. *__Computational complexity__: Concurrent beacon and coin protocols use discrete log exponentiations (denoted by DLE), which is the main reason for their high computational cost.HashRand instead uses $n^2 \log(n)$ Hash computations (100× cheaper than DLog expo and 1000× cheaper than a pairing computation) at low n and $O(n \log(n))$ Hashes at high n per beacon per node (denoted by H). †__AnyTrust Sampling__: HashRand has a per-node communication of $O(\lambda c n \log(n))$ bits per beacon. HashRand randomly samples a set of nodes C of size c and reconstructs secrets shared only by those nodes. We set the size of $c = |\mathrm{C}|$ to ensure at least one honest node is present in C with high probability. However, we note c is a small constant ($c < 70$ for $n = 1024$) at higher values of n, because of which we do not include it in our communication cost. §__Hash function__: HashRand assumes a Hash function to be a programmable Random Oracle (pRO) assumption to prove security against an adaptive and quantum adversary. The techniques in recent works [73, 74] can be used to replace the pRO in HashRand with a collision-resistant and linear hiding Hash function. ¶The leader performs $O(n^2)$ operations every round, while other nodes perform $O(n)$ operations each.

$\langle x, b_x \rangle : x \in \mathbb{N}, b_x \in \mathcal{D}$. *An honest node* $i \in \mathcal{N}$ *invokes* $\mathcal{B}$.Prep*(x,y)* $: x, y \in \mathbb{N}, x \leq y$ *to prepare beacons from index* x *to* y. *Upon terminating the preparation phase* $\mathcal{B}$.Prep*(x,y) until index* y, *and outputting* $\langle x, b_x \rangle \ \forall x \in [1, y)$, *an honest node* i *invokes* $\mathcal{B}$.Open*(y) to initiate generating* $\langle y, b_y \rangle$. *Such a protocol* $\mathcal{B}$ *is a random beacon protocol if and only if it satisfies the following properties.*

(1) ***Agreement****: For a given index* x, *if honest nodes* j *and* k *output* $\langle x, b_{xj} \rangle$ *and* $\langle x, b_{xk} \rangle$ *respectively, then* $b_{xj} = b_{xk}$, *except with a probability negligible in* λ.
(2) ***Liveness****: Every honest node* j *must eventually output a beacon* $\langle x, b_x \rangle$ *for all* $x \geq 1$.
(3) ***Bias-resistance and Unpredictability****: Given that no honest node invoked* $\mathcal{B}$.Open*(x) for an index* $x \geq 1$, *a polynomial time quantum adversary* $\mathcal{A}$ *should not have any advantage in predicting the beacon output* b_x. $\mathcal{D}$ *is the set of elements from which the protocol* $\mathcal{B}$ *outputs a value.*

$$Pr[\mathcal{A}(x) = b_x] = \frac{1}{|\mathcal{D}|}$$

Prior random beacon protocols [15, 16, 31] also guaranteed agreement with probability 1 and unpredictability with $\mathcal{A}$'s advantage negligible in λ. However, both properties cannot be achieved with probability 1 [37].

2.3 HashRand Key Insights

Conventional random beacon protocols (without the threshold setup) consist of two building blocks: VSS and consensus [16, 31, 72, 77]. In these protocols, each node uses VSS to share a uniformly random element from a given domain. Next, the nodes use some version of consensus to agree on a set of $t + 1$ nodes whose VSS instances have been successfully terminated by every node in the system. Finally, the nodes reconstruct the secrets shared by these $t + 1$ nodes and aggregate them to output the beacon. This template guarantees the contribution of at least one honest node to the beacon, which ensures its unpredictability. The liveness and agreement guarantees of such beacon protocols come from SMR's liveness and safety properties, respectively.

This design paradigm cannot be applied to an asynchronous beacon. This is because asynchronous SMR requires random beacons to terminate because of the FLP impossibility result [45]. This creates a circular dependency between the beacon and SMR in asynchrony [31]. A recent impossibility result by Freitas et al. [37] throws light on methods to resolve this circularity. It states that it is impossible for a deterministic asynchronous protocol to achieve agreement, deterministic liveness, and unpredictability. Hence, an asynchronous random beacon protocol must choose between the Monte-Carlo and Las Vegas flavors of termination.[1]

We propose HashRand, a Monte-Carlo style beacon protocol with deterministic termination and a fixed probability of successful execution denoted by δ. Mainly, HashRand breaks the beacon-SMR circularity in the conventional design paradigm by replacing the use of asynchronous SMR with the approximate agreement

[1]Monte-Carlo algorithms always terminate in a deterministic amount of time and return a correct result with some probability p and an incorrect result with a probability $1 - p$. In contrast, Las Vegas algorithms always return a correct result but have an indeterminate runtime.

(AA) primitive [39]. An AA protocol allows nodes to agree on values within a configured ϵ distance. Further, AA protocols offer deterministic termination and do not require access to random beacons. However, employing AA in HashRand requires us to trade off a small probability of agreement failure, where nodes output different beacons. This is because the approximation in the agreement property of AA induces a small, yet finite probability that nodes output different beacons.

HashRand **overview.** HashRand uses n-parallel AVSS [22], and approximate agreement [39] as building blocks. Prior AVSS protocols using discrete-log cryptography are communication efficient but have a high computational complexity, which is a performance bottleneck [33]. Although other AVSS protocols using Hash functions are computationally efficient [11], they have an $O(n)$ order increase in communication compared to discrete-log based AVSS schemes, which makes them practically inefficient. In HashRand, we overcome two challenges related to Hash-based AVSS using two key observations that enable HashRand to be computationally efficient with a practical communication complexity.

(1) We address the high communication cost of hash-based AVSS by making the following observation. AVSS's commitment property requires honest nodes to always reconstruct a valid element in the domain, even when the dealer is faulty. This commitment property is the major reason for prior AVSS protocols being communication intensive [11]. However, we note that this strong property is overkill for a random beacon protocol. As we show (Section 5.1), a weaker commitment property, where nodes can reconstruct a value $\perp$ designating an invalid sharing conducted by a malicious dealer, is enough to guarantee all the desired properties of a beacon. Using this weakened commitment property, we build a Batched Asynchronous weak VSS (BAwVSS) on top of Dolev et al.'s [41] AwVSS. For a batch size of $O(n)$, this BAwVSS scheme has an amortized sharing complexity of $O(\lambda \log(n))$ bits per node.

(2) The other source of high communication in HashRand's design is the requirement of reconstructing and aggregating at least $t+1$ secrets. This step requires $O(n^2)$ bits of communication per node ($O(n)$ bits per secret). We reduce this complexity by observing that reconstructing $t+1$ secrets is not necessary to guarantee the unpredictability of a beacon. We only need one honest node's contribution to ensure unpredictability. We propose an *AnyTrust* sampling procedure that randomly samples nodes to form a set C of size c. With high probability, this set contains at least one honest node i whose BAwVSS instance will be terminated by all honest nodes. All nodes later reconstruct the secrets proposed by nodes that are a part of this set, which brings down the communication complexity of the reconstruction phase to $O(\lambda n^2 \log(n))$ bits or $O(\lambda n \log(n))$ bits per beacon per node.

The first observation allows us to reduce computational complexity without increasing the communication complexity of the beacon. The second observation enables HashRand to reduce communication complexity by a $O(n)$ factor. Although conventional committee-based protocols only show performance effects at very large values of n, HashRand's AnyTrust sample set achieves a low statistical failure probability for much smaller c. Both these additions make HashRand computationally efficient while having a practical $O(\lambda n \log(n))$ per node communication complexity.

Table 2: Symbols and notations used in HashRand

Symbol	Description
δ	Statistical security parameter
λ	Computational security parameter
β	Batch size in BAwVSS
ϕ	Period of BAwVSS
$\mathcal{D}$	Range of beacons
r	Current round of HashRand
BAwVSS $_{r,i}$	BAwVSS instantiated by node i in round r
$S_{r,i}$	Set of β secrets shared by i through round r's BAwVSS instance
C_r	Set of nodes sampled for BAwVSS $_{r,.}$
$G_{r,i}$	Output from Gather.Term(r,I) for a BAwVSS $_{r,.}$
$\mathcal{E}_{r,i}$	AA instance corresponding to BAwVSS $_{r,i}$ instantiated in HashRand round r
B	List of all prepared beacons yet to be opened
B_c	List of all prepared beacons for AnyTrust Sampling

Security. HashRand is a Monte-Carlo beacon protocol with success probability $p \leq 1 - \delta + \text{negl}(\lambda)$. This corresponds to λ bits of cryptographic security and $\log(\frac{1}{1-\delta})$ bits of statistical security against a quantum adversary. We prove HashRand 's properties against a polynomial-time quantum adaptive adversary $\mathcal{A}$ under the assumption that the hash function H is a Random Oracle. We first prove HashRand is adaptively secure against a classical adversary and extend our proof to a quantum adversary using Boneh et al.'s [18] template for proving post-quantum security.

3 BUILDING BLOCKS

3.1 Asynchronous weak Verifiable Secret Sharing (AwVSS)

We describe the Asynchronous weak VSS primitive as defined by Dolev et al. [33].

Definition 3.1. *An* (n,t)-AwVSS *scheme among a set of n nodes with a dealer d consists of two subprotocols: the sharing protocol* AwVSS.Sh *and the reconstruction phase* AwVSS.Rec *protocol.*

- AwVSS$_d$.Sh*: A dealer d shares a secret* s_d *from a publicly known domain of values* $\mathcal{D}$*. At the end of* AwVSS.Sh*, at least* $t+1$ *honest nodes* $i \in \mathcal{N}$ *hold a secret share* $s_{d,i}$*.*
- AwVSS$_i$.Rec*: Every honest node* $i \in \mathcal{N}$ *reconstructs the secret* s_d *in a distributed fashion by broadcasting its secret share* $s_{d,i}$ *(if available). If the dealer d was honest, every honest i outputs* s_d *and* $\perp$ *otherwise.*

A protocol AwVSS *implements* (n,t)*-Asynchronous weak VSS if it satisfies the following properties.*

- ***Termination****:*
  - *(1) If the dealer d is honest, then each honest node will eventually terminate* AwVSS$_d$.Sh*.*
  - *(2) If an honest node terminates* AwVSS$_d$.Sh *protocol, then every honest node will eventually terminate* AwVSS$_d$.Sh*.*
  - *(3) If all honest nodes start* AwVSS.Rec*, then each honest node will eventually terminate* AwVSS.Rec*.*

- ***Correctness**: If d is honest, then each honest node upon terminating* AwVSS.Rec*, outputs the shared secret $s_d \in \mathcal{D}$ with probability $p \geq 1 - \mathsf{negl}(\lambda)$.*
- ***Secrecy**: If d is honest and no honest node has started* AwVSS.Rec*, then an adversary that corrupts up to t nodes has no information about s_d.*
- ***Weak Commitment**: Even if d is malicious, with probability at least $p \geq 1 - negl(\lambda)$, there exists a value $s^* \in \mathcal{D} \cup \{\perp\}$ at the end of* AwVSS$_d$.Sh*, such that all honest nodes output s^* at the end of* AwVSS.Rec *phase.*

This primitive is strictly weaker than AVSS because AVSS requires the honest nodes to hold shares of a secret $s \in \mathcal{D}$. In AwVSS, nodes can hold shares of a secret $s \in \{\mathcal{D}, \perp\}$. A malicious dealer who conducts an invalid sharing is detected in the sharing phase in AVSS and in the reconstruction phase in AwVSS.

We build on top of Dolev et al.'s [41] hash-based AwVSS protocol. In this protocol, a dealer d uses Shamir secret sharing to create secret shares for a secret s_d in domain $\mathcal{D}$. $\mathcal{D}$ can either be a ring or a finite field. The dealer d then creates a Merkle tree on top of the n secret shares. d then sends individual shares along with Merkle proofs to nodes and reliably broadcasts the Merkle root r. An honest node participates in the reliable broadcast of the root only upon verifying its Merkle proof. During the reconstruction phase, nodes that terminated with a share, broadcast their shares and Merkle proofs. Upon receiving $t + 1$ shares, every node reconstructs the entire share vector and the Merkle tree and verifies if the root broadcasted in the sharing phase r matches the reconstructed root r'. It outputs the shared secret if $r = r'$ and outputs $\perp$ otherwise.

Batched AwVSS. In order to reduce amortized communication complexity, we develop a batched version of AwVSS scheme. A protocol BAwVSS implementing Batched AwVSS has subprotocols BAwVSS$_d$.Sh(S_d) and BAwVSS$_d$.Rec(i). The dealer d shares a batch of multiple secrets S_d with BAwVSS$_d$.Sh(S_d) and honest nodes reconstruct each secret with BAwVSS$_d$.Rec(i): $\forall i \in \{1, \ldots, |S_d|\}$.

3.2 Gather

We also employ the Gather primitive [5].

Definition 3.2. *Let $\mathcal{T}$ be any protocol among nodes $\mathcal{N}$ satisfying the Totality property, which states that if an honest node j terminates $\mathcal{T}$, then every other honest node m terminates $\mathcal{T}$. Consider an n-parallel instantiation of the protocol $\mathcal{T}$ among $\mathcal{N}$, where node i started the instance $\mathcal{T}_i$. A protocol* Gather *implementing the Gather primitive is defined by two subprotocols –* Gather.Start *and* Gather.Term(). *Every honest node i starts by invoking* Gather.Start *(after invoking $\mathcal{T}_i$) and terminates by invoking* Gather.Term(G_i)*, where G_i is a set of node indices j such that i terminated $\mathcal{T}_j$.* Gather *satisfies the following properties.*

- ***Binding Common Core**: Once the first honest node i outputs set G_i, out of all possible sets $G' \subseteq G_i$ with size $|G'| = n - t$, there exists a unique core set G such that every other honest node j's output $G_j \cap G_i \supseteq G$. Moreover, the core set G is binding, meaning the adversary $\mathcal{A}$ cannot force any other honest node j to output a set $G_j \not\supseteq G$.*
- ***Termination**: If every honest node invokes* Gather.Start*, then every honest node i will eventually invoke* Gather.Term(G_i) *and output a set G_i: $G \subseteq G_i$.*

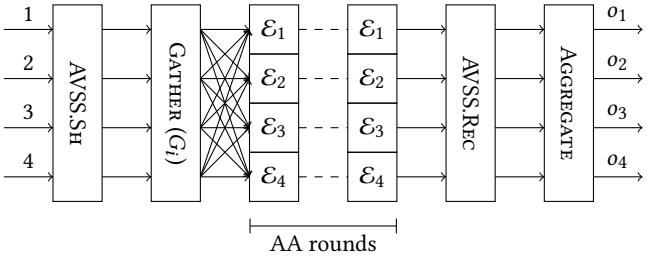

Figure 1: Approximate Common Coin: The structure of Freitas et al.'s [37] protocol with $n = 4$ nodes. Each node shares a random secret using AVSS, invokes Gather, and participates in n AA instances to agree on a weight for each node. Then nodes reconstruct and aggregate secrets to output o_i.

We use Abraham et al.'s [5] Gather protocol in HashRand. This protocol has a $O(n^3)$ communication complexity and requires two round trips to terminate.

3.3 Approximate agreement

We also use Approximate Agreement [40] (AA) primitive.

Definition 3.3. *A protocol $\mathcal{E}$ implementing the approximate agreement primitive among nodes $\mathcal{N}$ contains $\mathcal{E}$.Start() and $\mathcal{E}$.Term(), where every honest node i starts by invoking $\mathcal{E}$.Start(m_i) with a value $m_i \in \mathbb{R}$ and terminates by invoking $\mathcal{E}$.Term(o_i) with output o_i. $\mathcal{E}$ satisfies the following properties.*

- ***Termination**: If every honest node invokes $\mathcal{E}$.Start(), then every honest node i must eventually invoke $\mathcal{E}$.Term(o_i).*
- ***ϵ-agreement**: For $\epsilon > 0$, the outputs of any pair of honest nodes i and j satisfy $|o_i - o_j| < \epsilon$.*
- ***Validity**: Let $\mathcal{M}$ be the set of honest nodes' initial values m_i. The decision value of every honest node must be within the range of initial inputs of honest nodes $\mathcal{M}$, $\min \mathcal{M} \leq o_i \leq \max \mathcal{M}$.*

We use the Binary Approximate Agreement protocol [12] as the Approximate Agreement (AA) protocol $\mathcal{E}$ in HashRand. This protocol works under a binary input assumption, where honest nodes' inputs to $\mathcal{E}$ are publicly known binary values, for example, 0 and 1. This protocol proceeds in rounds, where after each round, the range of honest nodes reduces by a factor of $\frac{1}{2}$. A prior version of the protocol required $O(n^2 \log^2(\frac{1}{\epsilon}))$ bits of communication and $\log(\frac{1}{\epsilon})$ rounds for achieving approximate agreement. However, this communication can be improved to $O(n^2 \log(\frac{1}{\epsilon}) \log\log(\frac{1}{\epsilon}))$ bits using FIFO broadcast [1]. In this protocol [12], instead of broadcasting the entire value, a node broadcasts a constant-sized update of how its value changed in the previous round. Therefore, a node can construct another node's value in a round by applying updates over the value broadcast in the first round.

We invoke this protocol in a modular manner with the following methods: (a) $\mathcal{E}$.Start(v_i), using which node i starts an instance of the protocol with input $v_i \in \{0, 1\}$, (b) $\mathcal{E}$.StartRound(r) for starting round r of the protocol, (c) $\mathcal{E}$.EndRound(r), a callback function invoked after ending round r, and (d) $\mathcal{E}$.Term() for terminating the AA instance with output w_i.

3.4 Approximate Common Coin

We utilize the approximate common coin [37] primitive.

Definition 3.4. *A protocol Π_{CC} implementing the approximate common coin primitive among nodes $\mathcal{N}$ is composed of methods Π_{CC}.Start and Π_{CC}.End, where every honest node i starts by invoking Π_{CC}.Start and terminates by invoking Π_{CC}.End with output $o_i \in \mathcal{D}$, where $\mathcal{D}$ is a publicly defined set of values. Π_{CC} satisfies the following properties.*

- ***Termination**: If every honest node $i \in \mathcal{N}$ invokes Π_{CC}.Start, then every honest node i must invoke Π_{CC}.End with output o_i.*
- ***ϵ-agreement**: The outputs of any pair of honest nodes i and j satisfy $|o_i - o_j| < \epsilon$.*
- ***One process randomness**: The value output by at least one honest node must be uniformly distributed over the domain $\mathcal{D}$.*

We describe an overview of Freitas et al. [37] in Fig. 1.

4 DESIGN

We build HashRand from the Approximate Common Coin primitive. A strawman approach for building a beacon from this primitive is to start a brand new coin instance after terminating and reconstructing the previous coin. This approach has high computation, communication, and latency costs because of two main reasons. First, the n-parallel AVSS.Sh and AVSS.Rec phases cost $O(n^3)$ discrete log exponentiations per node if using discrete-log-based AVSS [33]. If we instead use Backes et al.'s [11] Hash-based AVSS, each beacon incurs $O(n^3)$ communication per node. Second, the AA phase has a high round complexity. Achieving a Monte-Carlo agreement probability of δ requires $\log(\frac{n}{1-\delta})$ rounds of AA. For example, generating a single beacon with $\delta = 1 - 2^{-40}$, equivalent to one expected failure in a trillion beacons, requires 200 round trips of communication at $n = 40$. We address these problems with two optimizations: (a) Batching and pipelining with Hash-based AwVSS for increasing throughput and reducing latency, and (b) AnyTrust sampling for reducing communication complexity.

4.1 Batching and Pipelining

AwVSS's weak commitment property enables every honest node to agree on the maliciousness of the dealer during the reconstruction phase and discard the contributions of these nodes. This observation enables us to use the AwVSS protocol proposed by Dolev et al. [41] and defined in Section 3.1. Directly using this AwVSS protocol in the strawman approach reduces the complexity of n-parallel AwVSS by a $O(n)$ factor to $O(\lambda n^2 \log(n))$ bits per node.

We further improve on this AwVSS protocol by proposing two crucial changes to develop BAwVSS, a Batched AwVSS protocol.

(1) We replace the computationally hiding commitments in Dolev et al. with unconditionally hiding commitments based on Backes et al. [11]. Along with the share polynomial $f(x)$ of degree t, the dealer also computes a nonce polynomial $R(x)$ of degree t with a secret R of size twice the range of hash function H (implies a 512-bit nonce for SHA256). The dealer then computes $H(R(i), f(i))$ as a commitment for the ith secret share, and builds a Merkle tree on this commitment vector. This change allows us to prove the security of HashRand against an adaptive adversary.

(2) We batch β secrets in each BAwVSS instance and reliably broadcast a vector of β Merkle roots. This amortizes the cost of RBC over β secrets. This change allows us to amplify the throughput of HashRand by sharing the expensive AA phase over all the secrets in the batch to produce β beacons at the end.

Replacing AVSS in the coin with BAwVSS drastically reduces the computation cost of each beacon by replacing DLog exponentiations with Hashes. It also amplifies the throughput by a $O(\beta)$ factor. Further, for a batch size $\beta = O(n)$ secrets,n-parallel BAwVSS has an amortized sharing communication of $O(\lambda \log(n))$ bits per node, which is an $O(\frac{\log(n)}{n})$ improvement over AVSS.

Pipelining. We use a pipeline in HashRand to ensure a consistent throughput of beacons. We leverage AA's deterministic termination property to create a pipeline of beacons with a pipelining period ϕ. For example, $\phi = 10$ implies one n-parallel BAwVSS instantiation for every 10 rounds of AA. Using pipelining, HashRand maintains a steady throughput of beacons with a constant latency by instantiating n-parallel BAwVSS every ϕ rounds and pipelining the corresponding AA phases. HashRand moves to $r + 1$ from round r only after terminating the Gather instance corresponding to round r, and round $r - r'$ of instance $\mathcal{E}_{r',j}$, $r \leq r' + r_t$. r_t is the number of rounds of AA required to achieve a success probability of δ.

The length of the pipeline is determined by the probability of agreement δ and n. For constant n, a higher δ implies a longer pipeline to fill, which increases the startup latency. However, once full, HashRand keeps up a continuous and constant throughput of $\frac{\beta}{\phi}$ Monte-Carlo beacons per round.

4.2 AnyTrust Sampling

The beacon opening phase requires HashRand to reconstruct and aggregate at least $t + 1$ secrets. Reconstructing a secret costs $O(\lambda n \log(n))$ bits of communication per node. Therefore, the cost of the beacon is $O(\lambda n^2 \log(n))$ bits per node and $O(\lambda n^3 \log(n))$ overall.

Ensuring the unpredictability of the beacon requires the contribution of only one honest node. However, the current design guarantees the contribution of at least $t + 1$ honest nodes, which is overkill and results in higher communication complexity. We address this issue by using a sampling procedure called AnyTrust sampling. In this procedure, honest nodes agree on a constant-sized set of nodes C and run an AA instance $\mathcal{E}_i \quad \forall i : i \in \mathsf{C}$ only for the node indices part of the sampled set, as opposed to running an $\mathcal{E}_i \quad \forall i : i \in \mathcal{N}$. Further, nodes sample C only after terminating Gather corresponding to an n-parallel BAwVSS $_{r,i}$ instantiation.

The sampled set C requires the presence of only one honest node that was part of the core set G to guarantee unpredictability. This condition is called the AnyTrust condition, which is satisfied at much smaller sizes of sampled sets than conventional supermajority or honest majority committees traditionally used in down sampled BFT systems [49, 53], and sharding systems [36], respectively. Hence, with AnyTrust sampling, HashRand needs to reconstruct secrets from at most $c = |\mathsf{C}|$ nodes.

However, this sampling procedure requires distributed randomness with agreement, where we run into the randomness-agreement circularity again. Moreover, the set must be randomly sampled with a secure beacon for every new n-parallel BAwVSS instantiation.

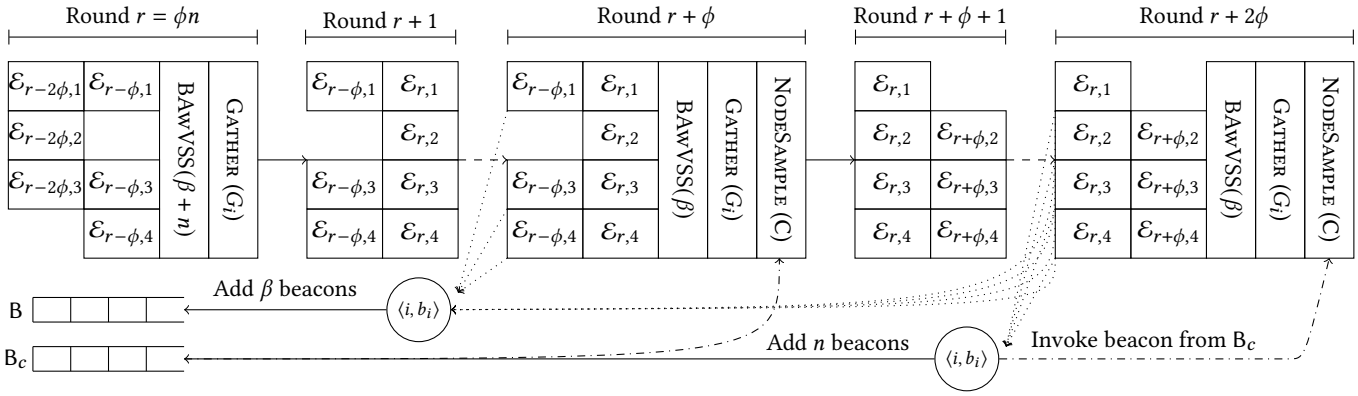

Figure 2: HashRand pipeline: We describe the pipeline of HashRand for $n = 4$ with batch size β period $\phi = \frac{r_t}{2}$. HashRand initiates a new BAwVSS instance once every ϕ rounds and pipelines the corresponding AA instances $\mathcal{E}_{r,i}$. HashRand also initiates preparation of n more beacons once every ϕn rounds. These beacons are prepared without node sampling, and therefore run $\mathcal{E}_{r,j}$ for all n nodes. Otherwise, nodes only run AA instances for sampled nodes (as in round $r+\phi+1$ for AA phase of round $r+\phi$). HashRand moves to the next round only after terminating BAwVSS,Gather, and round r of $\mathcal{E}_{\cdot}$ instances in the pipeline. After completing $r_t = 2\phi$ rounds, β beacons initiated before r_t rounds are added to B to be eventually reconstructed. Further, the extra n beacons prepared in round r are added into a separate queue B_c, used for AnyTrust sampling for future beacons in rounds $r : r \mod \phi = 0, r \mod n \neq 0$.

Otherwise, the adversary can bias future beacons by preventing the sampled members from being part of the Gather 's core set. To address this problem, we generate cn additional beacons once every ϕn rounds and use these beacons for sampling for the next n beacon preparation phases. We calculate the size of the sample set C based on the hypergeometric distribution.

4.3 Protocol description

We present the full protocol in Algorithm 1 and give a pictorial description in Fig. 2.

Beacon preparation phase. HashRand takes batch size β and instantiation period of BAwVSS ϕ, where $\phi \leq \log(\frac{n}{1-\delta})$, as configurable inputs. Once every ϕ rounds, nodes initiate BAwVSS by selecting β random secrets and secret sharing them by invoking BAwVSS$_{r,i}$.Sh(S_i) (Line 9). Once every ϕn rounds, nodes increase the batch size $\beta' = \beta + n$ to output enough beacons for AnyTrust sampling for the next n BAwVSS instances (Line 8). After instantiating BAwVSS, nodes also call Gather.Start and instantiate round r for each AA instance $\mathcal{E}_{r',j} \forall j \in \mathcal{N}$ started in the last $r' \in [r - r_t, r]$ rounds (Line 9).

After terminating Gather, node i calls Gather.Term(1) and outputs set G_i. Next, nodes sample the set C using NodeSample procedure, where they open the beacons prepared for committee election(Line 14). We note that nodes do not conduct sampling in rounds where randomness for future sampling procedures is being proposed. Once NodeSample terminates, the nodes instantiate AA $\mathcal{E}_{r,i} \forall i \in C_r$. For indices i in the committee C_r, nodes call $\mathcal{E}$.Start(r,i) (1) if $i \in G_i$ and calling $\mathcal{E}$.Start(r,i) (0) if $i \notin G_i \forall i \in C_r$(Line 14). Nodes terminate round r only after all AA instances $\mathcal{E}_{r',j} : r' \geq r - r_t, j \in C_{r'}$ terminate round r (Line 16). Once protocol reaches round r, the beacons initiated through BAwVSS in round $r - r_t$ are terminated and added to the beacon queue B (Line 19). These beacons are ready to be opened.

Open phase. i broadcasts its shares by calling BAwVSS$_{r',j}$.Rec(b) $\forall j \in C_{r'}$ (Line 26). It then waits until it reconstructs all secrets for which the approximate agreement instance $\mathcal{E}_{r',j}$ terminated with weight $w_{r',j} > 0$ (Line 31). Then, i outputs a weighted average of reconstructed secrets and rounds it off to the nearest checkpoint, which is a multiple of the domain size $|\mathcal{D}|$ (Line 33).

Verification. Any external client consuming beacons must receive $t+1$ equivalent authenticated beacon messages $\langle i, b_i \rangle$ from $t+1$ nodes. However, HashRand's beacon output can be made publicly verifiable, i.e. any client can verify the output of HashRand with a single message as opposed to $t+1$ messages, with a PKI and digital signatures. Such a scheme is also Post-Quantum secure with a PQ-secure digital signature scheme.

5 ANALYSIS

We analyze HashRand in this section. We start by proving HashRand 's security against an adversary corrupting at most $\frac{1}{3}$ faults. Then, we present the communication and computation complexity of HashRand. We give a rough proof sketch for brevity and refer the reader to the full version[13] for exhaustive proofs.

5.1 Security Analysis

We assume that the Hash function H is a Random Oracle (or ROM) for our proofs. We prove the liveness, correctness, and weak commitment properties using collision-resistance and preimage-resistance properties offered by H under the ROM. We prove the secrecy of BAwVSS against a PPT adaptive adversary $\mathcal{A}$. Our BAwVSS scheme offers computational correctness with $p = 1 - \mathsf{negl}(\lambda)$ and perfect secrecy in the ROM. We defer the adaptive security analysis of our BAwVSS scheme to the full version of the paper [13].

Algorithm 1 HashRand protocol

1: **INPUT**: $\mathcal{D}$,δ,β,ϕ // β: Batch size, ϕ: Pipelining period
▷ Choosing parameters
2: $\mathcal{D}'$: Domain of size at least $\lfloor \frac{4}{1-\delta} \rfloor |\mathcal{D}|$
3: $r \leftarrow 0$; $\epsilon \leftarrow \frac{1}{n\mathcal{D}'}$; $r_t \leftarrow \log(\frac{4n\mathcal{D}}{1-\delta})$
4: $c : p_c \geq \frac{2+\delta}{3}$ // Calculate sample size based on AnyTrust condition
5: B $\leftarrow \{\}$; $B_c \leftarrow \{\}$ // Prepared Beacons; Beacons for AnyTrust sampling

▷ BAwVSS phase
6: **upon** receiving (ID.i, Start,r) and $r \mod \phi = 0$:
7: Initiate generating beacons by using BAwVSS to share a batch of β random values, once every ϕ rounds
// More beacons for NodeSample every ϕn rounds
8: **if** $r \mod n = 0$, **then** $\beta' \leftarrow \beta + n$
9: **else** $\beta' \leftarrow \beta$
10: $S_{r,i}[j] \leftarrow \text{Random}(\mathcal{D}')\ \forall j \in \{1, \ldots, \beta'\}$ // β' random numbers
11: **Invoke** $\text{BAwVSS}_{r,i}$.Sh($S_{r,i}$)
12: **Invoke** Gather.Start () // Start Gather with BAwVSS

▷ Gather phase
13: **upon** invoking Gather.Term(G_i) for round r:
14: C $\leftarrow$NodeSample(r)
15: $\forall j \in \text{C} : \begin{cases} \mathcal{E}_{r,j}.\text{Start}(1), \textbf{if } j \in G_i \\ \mathcal{E}_{r,j}.\text{Start}(0), \textbf{otherwise} \end{cases}$ // Start AAA

▷ Approximate Agreement phase
// Wait until terminating round r for AA instances in the past r_t rounds
16: **upon** invoking $\mathcal{E}_{r',j}$.EndRound(r-r') $\forall r' : \{r' \in [r - r_t, r] \mid r' \mod \phi = 0\}$ and $\forall j \in \text{C}_{r'}$:
// Terminate AA instances that completed r_t rounds
17: **If** $r - r_t \mod \phi = 0$
18: $r' \leftarrow r - r_t$
// Add beacons generated to prepared beacons list
19: **If** $r' \mod n \neq 0$ **then** B $\leftarrow$ B $\cup \langle r', \langle 0, \beta \rangle \rangle$
// Set aside beacons for AnyTrust Sampling
20: **Else** B $\leftarrow \langle r', \langle n, n + \beta \rangle \rangle$; $B_c \leftarrow B_c \cup \langle r', \langle 0, n \rangle \rangle$
21: $r \leftarrow r + 1$ // Increment round
22: **for all** $\mathcal{E}_{r',j} : \{r' \in [r - r_t, r] \mid r' \mod \phi = 0\}$ and $j \in \text{C}_{r'}$ **do**
23: $\mathcal{E}_{r',j}$.StartRound(r-r') // Start next round of AAs
24: **If** $r \mod \phi = 0$ **then goto** Line 6

▷ Beacon reconstruction phase
25: **upon** receiving (ID.i, Reconstruct,k)
26: $r' \leftarrow \lfloor \frac{k}{\beta} \rfloor \phi$; $b \leftarrow k \mod \beta$
27: **If** $r' \mod n = 0$ **then** $b \leftarrow b + n$ // First n are for NodeSample
28: **for all** $j \in \text{C}_{r'}$ **do**
29: $w_{j,i} \leftarrow \mathcal{E}_{r',j}$.Term() // Output of $\mathcal{E}_{r',j}$
30: **Invoke** $\text{BAwVSS}_{r',j}$.Rec(b)
31: $x_j \leftarrow \begin{cases} \text{BAwVSS}_{r',j}.\text{Rec(b).Term()}, \text{if } w'_j \neq 0 \\ 0, \text{otherwise} \end{cases}$
32: $y_j \leftarrow \begin{cases} 0, \text{if } x_j = \perp \\ x_j, \text{otherwise} \end{cases}$ // Replace $\perp$ with 0
33: $o \leftarrow \left(\sum_{j \in \text{C}_{r'}} y_j \times w_{j,i}\right)$
34: **output** $\left\langle k, \left\lfloor \frac{o}{\lfloor \frac{4}{1-\delta} \rfloor} \right\rfloor \right\rangle$ and **return**

HashRand **security.** We start by proving the agreement, liveness, and unpredictability properties for rounds without AnyTrust sampling.

Theorem 5.1. *Given correctness and termination of the* BAwVSS *scheme, and secure Gather, and approximate agreement protocols,* HashRand *satisfies liveness for beacon indices* $i \in \{0, \ldots, \beta + n\}$.

Proof. From the termination property of BAwVSS, we know that BAwVSS satisfies totality, which ensures that every honest node i terminates Gather. Every honest node therefore participates in AA instance $\mathcal{E}_{0,i} \forall i \in \mathcal{N}$. Hence, every honest node terminates $\mathcal{B}$.Prep(0,$\beta + n$). In the opening phase $\mathcal{B}$.Open(l) $\forall l \in \{0, \beta + n\}$, an honest node j only waits to reconstruct secrets initiated by nodes i for which the output of $\mathcal{E}_{0,i}$, $w_{j,i} > 0$. If j's AA output $w_{j,i} > 0$, then from the Validity of AA, at least one honest node k must have input 1 to $\mathcal{E}_{0,i}$, meaning which k terminated i's BAwVSS instance. Therefore, from the termination property of BAwVSS, every honest node must eventually terminate i's BAwVSS instance and eventually reconstruct the secret shared by i. Therefore, j and all honest nodes terminate $\mathcal{B}$.Open(l) $\forall l \in \{0, \ldots, \beta + n\}$. □

Theorem 5.2. *Assuming secure* BAwVSS, *Gather and approximate agreement protocols,* HashRand *satisfies unpredictability property in Definition 2.1 for beacon indices* $i \in \{0, \ldots, \beta + n\}$.

Proof. We first show that at least $t + 1$ honest nodes' secrets will always contribute to beacons at indices $i \in \{0, \ldots, \beta + n\}$. From Gather's binding common core property, we know that at the time the first honest node i terminates Gather and outputs G_i, there is a core set G of size $n - t$ such that every other honest node j's output after Gather, denoted by G_j, will contain G. Every honest node terminates BAwVSS instances $BAwVSS_{0,j}$ of $t + 1$ honest nodes $j \in G$, and inputs 1 to the AA instance $\mathcal{E}_{0,j} \forall j \in G$. From the Validity property of AA, the output weights $w_{j,i} = 1$ for all honest nodes $j \in G$ and $i \in \mathcal{N}_{\text{hon}}$. Further, we note that the binding core property of Gather ensures that G is immutable by the time the first honest node begins AA. This immutability ensures that adversary $\mathcal{A}$ cannot prevent the inclusion of the sum $\sum_{j \in G} s_j$ in the beacon. Therefore, from the secrecy property of BAwVSS, until the first honest node invokes $\mathcal{B}$.Open(i), $\mathcal{A}$ has no information about the ith beacon.

Further, we also show that $\mathcal{A}$ cannot bias a beacon $\langle i, b_i \rangle$ after the first honest node invokes $\mathcal{B}$.Open(i). An honest node j invokes $\mathcal{B}$.Open(k) : $k \in \{1, \ldots, \beta + n\}$ until all $\mathcal{E}_{0,i} \forall i \in \{1, \ldots, n\}$ terminate at j. From the ϵ-agreement property of AA, we know that $|w_{j,i} - w_{j',i}| \leq \epsilon \forall j, j' \in \mathcal{N}_{\text{hon}}$ and $\forall i \in \mathcal{N}$. Therefore, the maximum difference between two honest nodes' weighted sums $|o_j - o_i| \leq n\epsilon\mathcal{D}' \leq 1, \forall i, j \in \mathcal{N}_{\text{hon}}$. Except for the cases where an honest weighted sum o_i is on the decision boundary of the rounding operation (which occurs with probability δ and agreement is already violated), $\mathcal{A}$ cannot bias the beacon's value after an honest node j starts reconstruction. □

Theorem 5.3. *Assuming secure* BAwVSS, Gather, *and approximate agreement,* HashRand *described in Algorithm 1 satisfies the agreement property in Definition 2.1 with probability* $p \geq \frac{2+\delta}{3} - \text{negl}(\lambda)$ *for beacon indices* $i \in \{0, \ldots, \beta + n\}$.

Proof. From the weak commitment property of BAwVSS, at the end of $\mathcal{B}$.Open(i) for an index i, honest nodes each have a weighted sum of secrets (Line 33) which correspond to a random range of $\epsilon = 2$ numbers in the domain $\lfloor \frac{4\mathcal{D}}{1-\delta} \rfloor$ with probability $p \geq 1-\mathsf{negl}(\lambda)$. When divided by $\frac{4\mathcal{D}}{1-\delta}$ and rounded off to the closest number, the probability that all honest nodes agree is $p = \frac{2+\delta}{3} - \mathsf{negl}(\lambda)$. □

Using the properties of the first $\beta + n$ beacons, we derive the security properties of all future beacons.

Theorem 5.4. *Assuming beacons at indices $i \in \{0, \ldots, \beta + n\}$ satisfy Liveness, Unpredictability, and Agreement with probability $p \geq \frac{2+\delta}{3} - \mathsf{negl}(\lambda)$,* HashRand *described in Algorithm 1 will output beacons $\langle i, b_i \rangle \forall i > 0$ that satisfy the properties in Definition 2.1 with probability $p \geq \delta - \mathsf{negl}(\lambda)$.*

Proof. Every batch of beacons instantiated in rounds $r > r_t; r \bmod n \neq 0$ samples a set of nodes using previously prepared beacons. From the security properties of first $\beta + n$ beacons, these beacons are secure with probability $p_c = \frac{2+\delta}{3} - \mathsf{negl}(\lambda)$, and is unpredictable. Therefore, every honest node starts AA instance $\mathcal{E}_{r,i} \forall i \in \mathsf{C}$ with probability p_c, and eventually terminates $\mathcal{E}_{r,i} \forall i \in \mathsf{C}$, guaranteeing liveness with probability p_c.

The sampled set C has at least one honest node j that is part of the core set G with probability $p_h = \frac{2+\delta}{3}$. Additionally, no honest node opens the beacon for sampling until it terminates Gather, which implies j's secret will have weight $w_{j,i} = 1 \forall i \in \mathcal{N}_{\mathsf{hon}}$. Therefore, the output beacon will always be the contribution of at least one honest node with probability $p_c p_h = (\frac{2+\delta}{3})^2 - \mathsf{negl}(\lambda) > \delta - \mathsf{negl}(\lambda)$.

Sampling with an adaptive adversary. We show that HashRand with AnyTrust sampling is adaptively secure under the secure erasure model [27, 35], where nodes can securely erase specified portions of their random execution tapes. In the BAwVSS phase, every honest node erases the sampled secrets and corresponding secret shares of nodes from its tape simultaneously while sending out the SHARE messages.

We utilize secure erasures to prove unpredictability. Consider BAwVSS and Gather instantiated in round r for which honest nodes use AnyTrust sampling. An honest node i invokes $\mathcal{B}$.Open(j) for sampling for round r only after terminating Gather of round r (Line 14). From the unpredictability property of the beacons used for sampling, we know that $\mathcal{A}$ cannot distinguish the sample members from uniformly random until i terminates Gather. Further, from the properties of Gather, the sampled node set C_r must contain at least one honest node j part of the core set G_r with probability $p_h = \frac{2+\delta}{3}$. Moreover, as node i already terminated j's BAwVSS instance, j would have already erased all data about its shared secret. Therefore, if $\mathcal{A}$ corrupts j after knowing its membership in the committee, it would only get j's share of the secret. $\mathcal{A}$ can corrupt at most t nodes and access t shares, which provides no information about j's secret and thereby ensures unpredictability.

In the no-erasures model of adaptive security [27], where the adversary gets access to the entire tape of a corrupted node, the above argument for adaptive security will not hold. The adversary can adaptively corrupt all committee members, gain access to the secrets shared by them, and know the beacons beforehand, which breaks unpredictability. HashRand needs to be run without AnyTrust sampling in this model.The per-node communication complexity in this model increases to $O(\lambda n^2 \log(n))$ bits per beacon.

After sampling a node set C_r with probability p_c and having at least one honest node $j \in G_r$ with probability p_h, AA and weighted average of random secrets enables honest nodes to agree on the beacon value with probability $p = p_c p_h p_a = (\frac{2+\delta}{3})^3 - \mathsf{negl}(\lambda) > \delta - \mathsf{negl}(\lambda)$, where p_a is the probability that the weighted average results in the same beacon at all nodes. Thus, beacons prepared with AnyTrust sampling are secure with probability $p > \delta - \mathsf{negl}(\lambda)$. □

Post Quantum security. We translate the proofs of BAwVSS and HashRand to be secure against a polynomial time quantum adversary. For this reduction, we require the Quantum Random Oracle Model(QROM) [18] to deduce that Grover's algorithm [50], the current best quantum algorithm to find collisions, needs at least $2^{\lambda/3}$ operations to find a collision with probability at least $\frac{1}{2}$. We also employ the history-free reduction technique proposed by Boneh et al. [18] to prove that HashRand is secure against a polynomial time quantum adversary. This technique states that a proof in the classical RO model can be translated to a proof in the quantum RO model, provided the proof simulates a RO in a *history-free* fashion. This implies that the RO's responses to a given query must not depend on its previous responses and query history. Our proofs (both static and adaptive) satisfy this condition because we do not program the RO based on prior query history. Hence, we also avoid the problem of recording a quantum adversary's queries and achieve post-quantum security in the Quantum RO model [18].

5.2 Complexity Analysis

We analyze the communication and computation complexity of HashRand. The n-parallel BAwVSS phase in round $r : r \mod \phi = 0$ with β secrets costs $O(\beta n \log(n) + \lambda n^2)$ bits of communication per node, with each node sending β Merkle proofs and reliably broadcasting a vector of β commitments. With Cachin-Tessaro's RBC [20], this complexity becomes $O(\beta\lambda n \log(n) + \lambda n^2 \log(n))$ per node for each n-parallel BAwVSS. The Gather primitive requires $O(n^2)$ bits per node. The AA phase for a round that does not use AnyTrust sampling ($r \mod n = 0$) requires n BinAA instances, with each round of one instance costing $O(n \log\log(\frac{\mathcal{D}}{1-\delta}))$ bits per node. Overall, $O(n)$ BinAA instances running for $\log(\frac{n\mathcal{D}}{1-\delta})$ rounds costs $O(n^2 \log(n) \log(\frac{\mathcal{D}}{1-\delta}) \log\log(\frac{\mathcal{D}}{1-\delta}))$ bits per node. For every beacon, the opening phase requires every node to broadcast secret shares of $O(n)$ secrets with evaluation proofs to all other nodes, which gives this phase a communication complexity of $O(\lambda n^2 \log(n))$ bits per node per beacon. Therefore, for β beacons, the overall communication per node without sampling is $O(\beta\lambda n \log(n) + \lambda n^2 \log(n) + n^2 \log(n) \log(\frac{\mathcal{D}}{1-\delta}) \log\log(\frac{\mathcal{D}}{1-\delta}) + \beta\lambda n^2 \log(n)) = O(\beta\lambda n^2 \log(n))$ bits per node for $\beta = O(n)$. The asymptotic communication complexity per beacon is dominated by the opening phase.

With AnyTrust sampling, each batch of β beacons requires honest nodes to open a beacon in $\mathsf{B_C}$ with $O(\lambda n^2 \log(n))$ per node. Then, the nodes run only $c = |\mathsf{C}_r|$ AA instances and require opening only c secrets. This results in a communication complexity of $O(\beta\lambda n \log(n) + \lambda n^2 \log(n) + cn \log(\frac{\mathcal{D}}{1-\delta}) \log\log(\frac{\mathcal{D}}{1-\delta}) + \beta\lambda cn \log(n))$ per β beacons. For $\beta = O(n)$, the communication complexity

per node comes to $O(\lambda n \log(n))$ bits. As beacon generation for sampling happens once every ϕn rounds, the average communication complexity per beacon totals $\frac{\lambda n^2 \log(n)}{n} + (1 - \frac{1}{n})\lambda n \log(n) = O(\lambda n \log(n))$ bits per beacon per node. The computation complexity can also be calculated in a similar fashion. For rounds $r : r \mod n = 0$, each node performs $O(n^2 \log(n))$ hash computations to verify $O(n)$ secret shares each for $O(n)$ secrets part of the beacon. With sampling, each node computes $O(cn \log(n))$ hashes for at most c secrets part of the beacon, and the average is $O(n \log(n))$ hashes per beacon.

6 APPLICATIONS

We present two asynchronous SMR protocols with HashRand providing common coins for liveness in asynchrony.

Post Quantum SMR. We demonstrate the utility of HashRand by implementing a Post-Quantum asynchronous SMR protocol using HashRand for Post-Quantum Liveness. There exist asynchronous SMR protocols like DAG-Rider [63] and FIN [43] that achieve post-quantum safety by using only symmetric key primitives like Message Authentication Codes (MACs), which are PQ-secure. However, these protocols depend on quantum-insecure random beacons like BLS threshold signatures [17] for liveness. Prior to HashRand, the PQ-secure random beacon protocol with the best communication complexity is Freitas et al. [37] with Dolev et al.'s [41] AwVSS. With this beacon, the communication complexity of PQ-secure asynchronous SMR is $O(n^2 \log(n))$ bits per block per node.

We present PQ-Tusk, an efficient post-quantum secure asynchronous SMR protocol built on top of Tusk [30]. Our choice of Tusk is motivated by its exceptional performance in a WAN. We create PQ-Tusk by using HashRand to provide the distributed randomness necessary for electing wave leaders. We tune HashRand's configuration parameters β and ϕ to ensure that HashRand prepares exactly one beacon per wave. Since revealing a prepared beacon in HashRand takes only one round trip in HashRand, PQ-Tusk can commit blocks at network speed. However, Tusk uses quantum-insecure `EdDSA` signatures for certificate generation. We replace this scheme with an appropriate PQ-secure signature scheme that offers maximum performance benefits in this context. Using PQ-secure signatures in Tusk does not induce any *asymptotic* overheads. However, PQ-secure signatures do induce a computation and communication overhead over traditional EdDSA signatures. We refer to Section 7.2 for a detailed discussion on PQ-secure signatures for SMR. Overall, PQ-Tusk has a communication complexity of $O(\lambda n \log(n))$ bits per block per node, which is an $O(n)$ factor better than the current best PQ-secure asynchronous SMR protocol, and only an $O(\log(n))$ factor higher than PQ-insecure Tusk.

Cross chain consensus. We also use HashRand to create the first asynchronous cross-chain consensus protocol without a PKI setup. A cross-chain consensus protocol is a key building block in creating a combined ledger with boosted trust from individual blockchains. Recent work TrustBoost [78] implements cross-chain consensus using a smart contract that runs on each participating blockchain. These contracts communicate using an inter-blockchain communication (IBC) protocol. A major challenge noted by TrustBoost is the computational expense of signature verification in prominent consensus protocols like HotStuff [79]. Further, TrustBoost also notes a limitation in the underlying IBC protocol, which prevents the use of public-key signatures in cross-chain consensus. These issues motivated TrustBoost to use Information-theoretic HotStuff (IT-HS) [7], which does not use any signatures. Moreover, these issues are also the main roadblock in running asynchronous cross-chain consensus. We propose a signature and PKI-free asynchronous cross-chain consensus protocol. We use DAG-Rider [63], a DAG-based atomic broadcast protocol with HashRand, to create a PKI-free asynchronous SMR protocol. This SMR protocol has a communication complexity of $O(\lambda n^2 \log(n))$ bits per transaction, which is $O(\log(n)/n)$ factor more efficient than IT-HotStuff in the worst case, and only $O(\log(n))$ factor higher than IT-HS under a stable leader.

7 EVALUATION

We describe the evaluation of HashRand in this section. We evaluate HashRand in a geo-distributed setting and compare it with Dfinity-DVRF [56] based on BLS threshold signatures [17]. We also evaluate PQ-Tusk with HashRand providing common coins.

Implementation. We implement HashRand in Rust with the `tokio` library as our asynchronous runtime. We use `SHA256` as our hash function H. We use Cachin-Tessaro's [20] computationally efficient Reliable Broadcast protocol in HashRand and use the Erasure Codes library. We then integrate HashRand with Tusk 's [30] codebase and use HashRand 's beacons to elect wave leaders in Tusk. Further, we implement PQ-Tusk by replacing the quantum-insecure `EdDSA` signatures in Tusk with PQ-secure `DiLithium` Signatures from the pqcrypto library.

Evaluation setup. We evaluate HashRand and its corresponding integrated asynchronous SMR protocols in a Geo-distributed testbed on Amazon Web Services (AWS). We evaluate HashRand with varying nodes $n = 16, 40, 64, 112$, and 136. We run our protocol on `t3a.medium` nodes, each with 2 cores and 4GB RAM. We also evaluate PQ-Tusk on a testbed with `c5.large` instances, each with 2 cores (but higher frequency than t3a.medium) and 4GB RAM, with $n = 16, 40$ nodes. We create a geo-distributed testbed of n nodes to simulate execution over the internet. We distribute the nodes equally across 8 regions: N. Virginia, Ohio, N. California, Oregon, Canada, Ireland, Singapore, and Tokyo.

Baselines. We compare HashRand with an asynchronous beacon protocol based on Dfinity-DVRF [56], which uses BLS threshold signatures [17]. Prior works such as Spurt [31] and OptRand [15] directly used DRand [76] as a benchmark for a beacon with threshold setup. However, we note that DRand's codebase does not produce beacons at network speed and thereby misrepresents the true pace of beacon generation with a threshold setup. To ensure a fair comparison, we implement the Dfinity-DVRF protocol using the Rust-based BLS signature library `blstrs`on the `bls12-381` curve, which internally uses the `blst`pairing library. An honest node constructs $\langle i, b_i \rangle$ immediately after it possesses $\langle i-1, b_{i-1} \rangle$, thus generating beacons at network speed. We also compare the numbers of HashRand to Spurt's [31] stated numbers. We ensure a fair comparison by running HashRand using the same geo distribution of nodes and configuration of machines as in Spurt. We compare the throughput of beacons produced by each protocol using the

metric beacons per minute. We also benchmark the performance of PQ-Tusk with PQ-security against a non PQ-secure Tusk [30].

7.1 HashRand evaluation

We configure HashRand to emit beacons for committee election in asynchronous SMR or blockchain sharding. For this application, the domain of beacons $\mathcal{D}$ is the total number of nodes in the system. We set a statistical success probability of $\delta = 1 - 2^{-38}$, which amounts to one failure in 10^{12} beacons. Even with an optimistic 10^6 beacons being invoked everyday, the average expected time for a failure is 2739 years. Assuming a domain $\mathcal{D}$ of size 2048, we choose the domain $\mathcal{D}'$ for BAwVSS to be a finite field of size $|\mathcal{D}'| > 2^{50}$. Nodes invoke $\mathcal{B}$.Open(i+1) immediately after they terminate $\mathcal{B}$.Open(i) to generate beacons at network speed.

Tuning β and ϕ. HashRand's runtime configuration parameters comprise the BAwVSS batch size β, and the period of BAwVSS instantiation ϕ. These parameters control the pace of beacon generation. In Fig. 3, we present the impact of these parameters on the HashRand's throughput using a heat map. For a given β, there is an optimal ϕ of BAwVSS instantiation that maximizes the use of the compute and network's resources. For example, in the row with $\beta = 100$, the throughput increases with reducing period until $\phi = 10$, after which it decreases again. The low throughput at a high period is resultant of idle time at the CPU, where nodes exhaust their available pool of prepared beacons faster than they can prepare new beacons. Similarly, at a lower period, the nodes saturate their network bandwidth by initiating BAwVSS in quick succession, which keeps increasing the pool of prepared beacons waiting to be reconstructed. Moreover, the period of maximum throughput ϕ is smaller for smaller β. Therefore, for any given n, there exists an optimal (β, ϕ) configuration that gives the maximum throughput of beacons. For $n = 40$ nodes, HashRand produces a maximum throughput of 1837 beacons pm at $(\beta, \phi) = (100, 10)$.

Scalability. We evaluate HashRand with increasing n in Fig. 4. At low n, the computational efficiency of hash computations outperforms threshold signing where for $n = 40$, HashRand outputs 66% more beacons than Dfinity. However, HashRand scales as $O(n^2 \log(n))$ Hash computations per node at small n, whereas threshold signatures scale linearly at all n. This rate of change is visible in Fig. 4 where Dfinity's slope is lower than HashRand. The cross-over point between HashRand and Dfinity-DVRF is between $n = 40$ and $n = 64$.

We observe the AnyTrust sampling procedure taking effect with increasing n. At small n, the set size c grows with n, resulting in a computation and communication complexity of $O(n^2 \log(n))$. However, with increasing n, c becomes constant and the complexities grow as $O(n \log(n))$. In Fig. 4, the rate of decrease of throughput of HashRand decreases with increasing n, where at higher n, HashRand scales more gracefully. Due to this improvement, HashRand outputs 78 beacons pm at $n = 136$ nodes without a threshold setup.

Hardware-accelerated Hash functions. The performance can further be improved using efficient hash functions accelerated by hardware. For example, cryptographic hash functions built from fixed-key block ciphers like AES [70] can be accelerated using the AES-NI hardware instruction set provided by many concurrent processors. These hash computations offer 20× speed up over conventional `SHA256` computations. With such specialized hash functions, we expect HashRand to outperform Dfinity at higher values of n.

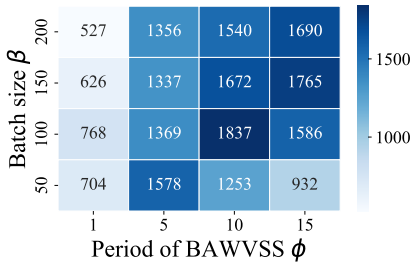

Figure 3: The rate of beacon generation (/min) vs β and ϕ. The figure shows the number of beacons emitted by HashRand per minute for $n = 40$ nodes with configuration parameters batch size β in each BAwVSS instance and the number of BAwVSS instances. For every value of n, there exists a sweet spot of β and ϕ that gives maximum throughput

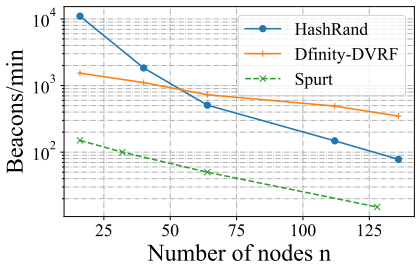

Figure 4: Scalability results: The figure shows the number of beacons produced per minute by HashRand, Dfinity-DVRF [56], and the numbers reported by Spurt [31] in the same geo-distributed setting. HashRand's $O(n^2 \log(n))$ Hash computational cost results in a steeper slope than Dfinity's $O(n)$ computational cost. Dfinity's cost doesn't include the cost for ADKG. HashRand's computational efficiency allows it to produce to produce 1 beacon per second at $n = 136$ without a threshold setup, which is 5x higher than Spurt at $n = 128$.

HashRand vastly outperforms Spurt at all values of n. Particularly, at $n = 128$, Spurt outputs only 15 beacons pm, which is 5x lower than HashRand at $n = 136$ nodes. Even though Spurt and HashRand have the same asymptotic communication complexity, HashRand's computational efficiency allows it to produce beacons at a much higher rate. This difference between Hashes and DLog exponentiations is more visible at small n where HashRand uses $O(n^2 \log(n))$ computations per beacon as compared to Spurt's $O(n^2)$ DLog exponentiations. Overall, HashRand scales gracefully with n and is currently the best way to generate distributed randomness without a threshold setup and with Post-Quantum security.

7.2 SMR Evaluation

We also evaluate PQ-Tusk and compare it to classical Tusk without PQ-security. We test both protocols by offering increasing transaction loads and measuring the response rate and latency for that request rate. We then plot the response rate vs latency in Fig. 5 and check the value of response rate for which the latency has a disproportionate jump. Each transaction is of size 256 bytes.

PQ-secure signatures for SMR. We consider three PQ-secure signature schemes for use in PQ-Tusk: a) `DiLithium` based on Lattice cryptography [44], b) `SPHINCS-256` [14] based on the Hash-based Winternitz One-Time Signatures(WOTS), and c) `PICNIC` [25] based on the MPC-in-the-head approach with symmetric key primitives. We notice that the signing and verification times coupled with the signature size of a signature scheme have the highest impact on Tusk's latency and throughput. This is because Tusk has a $O(n)$ signing and $O(n^2)$ verification complexity per wave per node. Nodes in Tusk also broadcast $O(n^2)$ signatures per wave. `PICNIC` and `SPHINCS-256` have been optimized for the internet setting and therefore prioritize low public key and private key sizes. Hence, both have high signing and verification times in milliseconds. Both schemes also have high signature sizes(40 KB and 140 KB). `DiLithium` requires 220 and 70 microseconds to sign and verify a message respectively, and has a modest signature size of 4 KB for 128-bits of quantum security. `DiLithium` has larger public and secret keys compared to `PICNIC` and `SPHINCS-256`, which is not a problem in a permissioned SMR system.

In the WAN setting, PQ-Tusk has a response rate of 135k transactions per second at $n = 16$ nodes, with an optimal latency of 2.3 seconds per transaction confirmation. This response rate is only 10% lower than classical Tusk. PQ-Tusk is also very efficient compared to other prominent and implemented PQ-secure protocols like WaterBear [82] that uses local coins to generate distributed randomness. At $n = 16$ in a geo-distributed setting, WaterBear offers a peak response rate of 70k transactions per second with transaction size of 100 bytes in `m5.xlarge` machines (twice as powerful as `c5.large`) as compared to 135k transactions per second at 256 byte transaction size for PQ-Tusk.

At higher $n = 40$ and $n = 64$, PQ-Tusk offers a peak response rate of 122k transactions per second at a latency of 17.5 seconds per transaction, and 40k txns per second at a latency of 50 seconds. We do an ablation study of PQ-Tusk to identify the bottleneck at higher n. We implement and observe the response rate for Tusk with `DiLithium` signatures and BLS threshold signatures. This variant of Tusk underperforms compared to our PQ-Tusk at $n = 16$ with 20% lesser response rate at 108k transactions per second. We identify the higher signing and verification times of `DiLithium` signatures over `EdDSA` signatures as the bottleneck for PQ-Tusk. Therefore, with HashRand, we remove the Post-Quantum liveness bottleneck and establish that randomness from Hash functions is currently the most practical way to achieve Post-Quantum SMR.

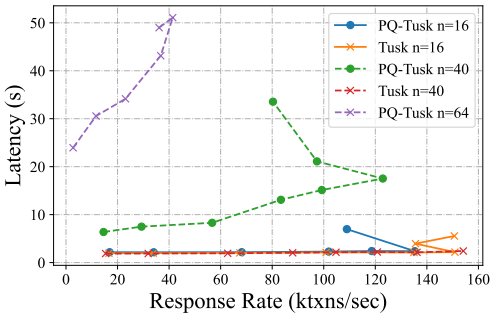

Figure 5: Post-Quantum SMR: The figure describes the latency-response rate curve for PQ-Tusk, and Tusk. PQ-Tusk has a response rate of 135k transactions per second at a latency of 2.3 seconds for $n = 16$, and 122k transactions per second at a latency of 17.5 seconds for $n = 40$ nodes.

8 DISCUSSION

8.1 VSS-SMR

We discuss an alternate approach for building an asynchronous random beacon protocol, titled VSS-SMR, which addresses the asynchronous SMR-Beacon circularity mentioned in Section 2.3 by using asynchronous SMR as a setup. This approach first generates a batch of $O(\lambda)$ beacons using our BAwVSS protocol and an SMR protocol that does not require a DKG setup. Abraham et al. [2] is one such Information-Theoretically (IT) secure asynchronous SMR protocol with a communication complexity of $O(n^4 \log(n))$ bits. Nodes use these $O(\lambda)$ beacons as common coins for an efficient asynchronous SMR protocol like FIN [43] with $O(\lambda n^3)$ communication. Then, nodes use our BAwVSS protocol with this efficient SMR instance to agree on a set of $n - t$ successful VSS instances and generate a fresh batch of $O(\beta + \lambda)$ beacons. Out of these beacons, nodes output $O(\beta)$ beacons and set aside $O(\lambda)$ beacons as common coins for the next SMR instance. Through this established pipeline, nodes generate beacons while ensuring sufficient beacons are available to ensure SMR terminates. For every coin used, FIN's termination probability reduces by a factor of $\frac{1}{3}$. This implies $O(\lambda)$ beacons are sufficient to ensure that FIN always terminates, except with probability negligible in λ. With a batch size of $\beta = O(n)$ beacons, this approach requires $O(\lambda n^2 \log(n))$ bits of communication per beacon per node, which can be reduced to $O(\lambda n \log(n))$ bits using AnyTrust sampling (Failure probability analysis required).

Comparison with HashRand. HashRand and VSS-SMR differ in their styles of termination with Monte-Carlo and Las-Vegas styles of termination, respectively. Comparing their efficiency, both protocols have the same VSS phase, whereas VSS-SMR's reconstruction phase is $O(n)$ factor more expensive than HashRand. Although we can use our AnyTrust sampling/shuffling for VSS-SMR to generate multiple beacons from the $n - t$ agreed VSSes through SMR, this technique requires further analysis in terms of failure probability.

HashRand's agreement phase uses Gather and c AA instances, with $O(n^2 \log(\frac{1}{1-\delta}))$ and $O(n^3 + cn^2 \log(\frac{1}{1-\delta}) \log\log(\frac{1}{1-\delta}))$ message and communication complexities. FIN requires $O(n^3)$ messages and $O(\lambda n^3)$ communication. Further, HashRand's agreement phase leads to a statistical security of $\log(\frac{1}{1-\delta})$ bits and FIN has a cryptographic security of $\lambda = \frac{\kappa}{3}$ bits against a quantum adversary, where κ is the output size of a collision-resistant Hash function. Asymptotically, HashRand scales better than FIN by an $O(n)$ factor in message complexity and by a $O(\kappa)$ factor in communication

complexity. Concretely, with 40 bits of statistical security (1 failure in a trillion beacons) and using a standard 256-bit Hash function, HashRand's $O(c \log(\frac{1}{1-\delta}) \log\log(\frac{1}{1-\delta}))$ factor scales better than VSS-SMR's $O(\kappa n)$ after $n = 45$. Further, HashRand's bootstrapping phase also has a $O(n)$ factor lower communication than Abraham et al. [2], enabling efficient reconfiguration.

8.2 AwVSS vs ACSS protocols

The Asynchronous Complete Secret Sharing (ACSS) is a stronger VSS primitive which ensures every honest node terminates with a secret share, even when the dealer is malicious. Owing to this stronger completeness property, ACSS is more expensive to achieve compared to AwVSS. However, secrets shared through an ACSS protocol can be reconstructed using Reed-Solomon Error Correction Codes (ECC), where nodes only send their secret shares to other nodes. This method of reconstruction has $O(n)$ communication per secret per node and is more efficient than AwVSS, where nodes must also attach a proof of share validity to secret shares (Merkle proofs for our AwVSS).

Given a Hash-based ACSS protocol with $O(n\beta)$ communication complexity, we can improve HashRand's communication to $O(n)$ per beacon per node using the batch reconstruction technique in Choudhury and Patra [26]. We can also combine the above VSS-SMR approach with the Vandermonde hyperinvertible matrix technique [29] to generate $O(n\beta)$ beacons from a single SMR instance. Overall, with a big enough β, this approach would also have a communication complexity of $O(n)$ bits per beacon per node. Therefore, improvements in Hash-based ACSS protocols can be utilized to improve PQ-secure asynchronous random beacons.

A recent work by Shoup and Smart [74] proposes a Hash-based ACSS protocol with $O(n\beta + n^3)$ communication in the good case with an honest dealer, and $O(n^2\beta + n^3)$ with a faulty dealer. However, this protocol uses a random beacon as a building block, and requires nodes to be online to listen to complaints. Further, it has a high $O(n^3)$ free term, so it is unclear how it would perform in practice.

8.3 Comparison with ADKG protocols

We set aside Post-Quantum security for this discussion and only compare performance of ADKG and HashRand. Asynchronous DKG protocols enable nodes to generate beacons efficiently using threshold unique signatures. Particularly, BLS threshold signatures [17] are highly efficient because of their non-interactive construction. However, the ADKG protocols required to establish a setup for BLS threshold signatures are very expensive. In summary, beacon generation using threshold signatures can be viewed as a highly expensive setup phase followed by efficient beacon generation. In contrast, at higher values of n, HashRand has a relatively cheaper, computationally efficient bootstrapping phase and relatively expensive beacon generation with $O(\log(n))$ factor more communication. Comparing performances, the current best implemented ADKG protocol by Das et al. [34] at $n = 136$ nodes takes 37 seconds to terminate versus 25 seconds for HashRand's bootstrapping phase in the same geo-distributed setup. This gap widens with increasing n with HashRand's computational efficiency helping it scale better than ADKG. Using ADKG is more favorable when its high setup cost can be *recovered* by efficiently generating sufficient number of beacons. In scenarios with a low reconfiguration period, the high cost of ADKG setup becomes a bottleneck, and HashRand's tradeoffs are more favorable. Similarly, with a high reconfiguration period, HashRand's relatively expensive beacon generation offers subpar performance compared to ADKG.

With Post-Quantum security, HashRand's beacon generation is vastly superior compared to other IT-secure beacons and Lattice-based threshold signature schemes [38]. We are also unaware of any practical asynchronous DKG protocols for these schemes. In our knowledge, HashRand is the only practical asynchronous beacon protocol with PQ security.

8.4 Setup costs and tradeoffs

HashRand requires a secure channel setup among participant nodes. In a Post-Quantum world, establishing secure channels needs PQ-secure TLS, which is a notable cost. However, this setup cost is only incurred once, after which nodes use efficient symmetric-key cryptography. In contrast, protocols relying on a PKI setup do not incur this cost, but instead use computationally expensive public key cryptography like encryption and digital signatures. Based on the discussions in Section 7.2 and the efficiency of symmetric-key cryptography, this tradeoff skews towards secure channels for all practical reconfiguration periods.

9 RELATED WORK

We use different trusted setup levels by Abraham and Yanai [8] to describe setup assumptions of related beacon protocols.

Synchronous and partially synchronous protocols. Beacon protocols in the synchronous world consists of protocols [15, 16, 23, 31, 72, 77] that use VSS driven by CRS or SRS setup(Level 4 in [8]). All these protocols require some variant of SMR to achieve the properties of a random beacon. We discuss OptRand [15], and Spurt [31] in this category. OptRand [15] is an optimistically responsive random beacon that uses OptSync's [75] commit rule to generate beacons at network speed. The protocol however depends on a synchronous timeout to implicate faulty leaders. Spurt is a partially synchronous protocol that uses n-parallel PVSS (requiring a Level 4 CRS setup) with leader-based secret aggregation to ensure unpredictability within $O(n^2)$ communication. The secrecy of the PVSS scheme depends on the DBDH assumption.Spurt internally uses the HotStuff [79] SMR protocol to enable honest nodes to terminate and output the beacon value.

Asynchronous protocols. Asynchronous random beacon protocols are bound by Freitas et al. 's [37] impossibility result. Protocols with a DKG setup(Level 5 in [8]) such as Cachin et al. [19] are not bound by this impossibility result because the output from threshold signatures is equivalent to distributed pseudorandomness and is a deterministic function of honest nodes' states.

Randomness Protocols with Las-Vegas style agreement have been traditionally employed in Asynchronous Distributed Key Generation (ADKG) [4, 5, 32, 34, 47, 65]. Kogias et al. [65] propose an Eventually Perfect Common Coin (EPCC) using a high-threshold AVSS scheme conducted over a secure channel setup(Level 2). Their protocol uses the DDH assumption with a RO. This protocol also

has a $O(n^3)$ computational complexity per node, which makes it very expensive in practice.

Other asynchronous protocols use a partially public setup like PKI with CRS or SRS(Level 4 setup). Abraham et al. [5] use PKI-based VRFs and Gather to agree on a random node's index. This protocol uses the SXDH assumption with a RO to achieve unpredictability. A recent work called Bingo [4] produces adaptively secure randomness using a Packed AVSS scheme, which requires a PKI and an SRS setup. Abraham et al., and Bingo have Ωn^2 computational complexity, which makes them computationally inefficient. Such protocols can be used for a one-shot ADKG and later use Cachin et al. 's [19] approach for generating pseudorandom beacons. This approach does not work with mobile participants and also suffers from key leakage attacks, which prompted a wave of research on mobile proactive [57, 81] and refreshable secret sharing [53]. Moreover, Kogias et al. 's approach is the only current way to achieve ADKG at a Level 2 setup (without PKI or CRS) in [8], whose expensiveness hinders its practical application.

Freitas et al. 's [37] protocol achieves deterministic termination with a statistical probability of disagreement with a Level 2 setup (pairwise secure channels). This protocol uses AVSS, Gather, and approximate agreement to output a random ϵ interval of numbers. The AVSS used in this protocol depends on Pedersen commitments and the DLog assumption for unconditional hiding and hence, does not require a RO for unpredictability. However, this protocol has a higher computation complexity of $O(n^3)$ per node and a high round complexity of $\log(\frac{n}{1-\delta})$ rounds.

Information-theoretic protocols. Statistically and information-theoretically secure schemes can be used to develop computationally efficient beacon protocols. For example, Patra et al. [69] propose a statistically secure AVSS scheme with $O(n^4 \log(n))$ communication complexity. This work combined with packed AVSS and Freitas et al.'s Monte-Carlo coin technique results in an overall communication complexity of $O(n^4 \log(n))$ per beacon.

10 CONCLUSION

We presented HashRand, a computationally efficient practical asynchronous random beacon protocol that requires a pairwise-channel setup. HashRand guarantees Monte-Carlo agreement, liveness, and unpredictability by only using the properties of a one-way Hash function. HashRand outputs each beacon with amortized $O(\lambda n \log(n))$ bits of communication per node and is computationally efficient with amortized $O(n \log(n))$ hash computations per beacon. We proved HashRand 's security using standard properties of one-way Hash functions against an adaptive and quantum adversary. HashRand provides post-quantum liveness for asynchronous SMR protocols with only $O(\log(n))$ more communication than threshold signatures. We also demonstrated HashRand 's scalability and utility in making post-quantum SMR practical.

We also find that this work, especially with its performance analysis, establishes an interesting research direction for secure distributed systems: when aiming for post-quantum security for distributed systems such as SMR or MPC, the protocol builders should consider hash-based approaches rather than directly resorting to the information-theoretic or statistically-secure protocols with significantly higher communication complexities.

ACKNOWLEDGEMENTS

We thank our anonymous shepherd and reviewers for helpful suggestions about the paper. We also thank Sourav Das for his insightful suggestions on the paper. This work was supported in part by NIFA award number 2021-67021-34252, the National Science Foundation under Grant Numbers CNS-2038986 and CNS-2038566, and the Army Research Office under Contract number W911NF-2020-221. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] Ittai Abraham, Yonatan Amit, and Danny Dolev. 2004. Optimal Resilience Asynchronous Approximate Agreement *(OPODIS'04)*. Springer-Verlag, Berlin, Heidelberg, 229–239.

[2] Ittai Abraham, Gilad Asharov, Arpita Patra, and Gilad Stern. 2023. Asynchronous Agreement on a Core Set in Constant Expected Time and More Efficient Asynchronous VSS and MPC. Cryptology ePrint Archive, Paper 2023/1130. https://eprint.iacr.org/2023/1130 https://eprint.iacr.org/2023/1130.

[3] Ittai Abraham, Naama Ben-David, and Sravya Yandamuri. 2022. Efficient and Adaptively Secure Asynchronous Binary Agreement via Binding Crusader Agreement. In *PODC '22*, Alessia Milani and Philipp Woelfel (Eds.). ACM, 381–391.

[4] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. 2023. Bingo: Adaptivity and Asynchrony in Verifiable Secret Sharing and Distributed Key Generation. In *CRYPTO 2023* (Santa Barbara, CA, USA). 39–70.

[5] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Reaching consensus for asynchronous distributed key generation. In *PODC'21*. 363–373.

[6] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically optimal validated asynchronous byzantine agreement. In *PODC 2019*. 337–346.

[7] Ittai Abraham and Gilad Stern. [n. d.]. Information Theoretic HotStuff. In *OPODIS 2020*, Vol. 184. 11:1–11:16.

[8] Ittai Abraham and Gilad Stern. 2019. *Trusted Setup assumptions,https://decentralizedthoughts.github.io/2019-07-19-setup-assumptions/.*

[9] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. 2021. High-threshold avss with optimal communication complexity. In *FC 2021*. Springer, 479–498.

[10] Michael Backes, Amit Datta, and Aniket Kate. 2013. Asynchronous computational VSS with reduced communication complexity. In *Cryptographers' Track at the RSA Conference*. Springer, 259–276.

[11] Michael Backes, Aniket Kate, and Arpita Patra. 2011. Computational verifiable secret sharing revisited. In *ASIACRYPT 2011*. Springer, 590–609.

[12] Akhil Bandarupalli, Adithya Bhat, Saurabh Bagchi, Aniket Kate, Chen-Da Liu-Zhang, and Michael K. Reiter. 2024. Delphi: Efficient Asynchronous Approximate Agreement for Distributed Oracles. In *IEEE/IFIP DSN 2024*. 14 pages.

[13] Akhil Bandarupalli, Adithya Bhat, Saurabh Bagchi, Aniket Kate, and Michael Reiter. 2023. HashRand: Efficient Asynchronous Random Beacon without Threshold Cryptographic Setup. Cryptology ePrint Archive, Paper 2023/1755. https://eprint.iacr.org/2023/1755

[14] Daniel J Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. 2015. SPHINCS: practical stateless hash-based signatures. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 368–397.

[15] Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. 2023. OptRand: Optimistically Responsive Reconfigurable Distributed Randomness. In *NDSS 2023*. The Internet Society.

[16] Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. 2021. RandPiper: Reconfiguration-Friendly Random Beacons with Quadratic Communication. In *CCS 2021*. 3502–3524.

[17] Alexandra Boldyreva. 2002. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Public Key Cryptography — PKC 2003*, Yvo G. Desmedt (Ed.). 31–46.

[18] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. 2010. Random Oracles in a Quantum World. Cryptology ePrint Archive, Paper 2010/428. https://eprint.iacr.org/2010/428.

[19] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2000. Random Oracles in Constantipole: Practical Asynchronous Byzantine Agreement Using Cryptography (Extended Abstract). In *PODC 2000*. 123–132.

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

[21] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. 2001. On adaptive vs. non-adaptive security of multiparty protocols. In *EUROCRYPT 2001*. Springer, 262–279.

[22] Ran Canetti and Tal Rabin. 1993. Fast asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. 42–51.

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

[24] Chainlink. 2023. *35+ Blockchain RNG Use Cases Enabled by Chainlink VRF*. https://chain.link/education-hub/rng-in-blockchain-use-cases

[25] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. 2017. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *CCS 2017*. 1825–1842.

[26] Ashish Choudhury and Arpita Patra. 2017. An Efficient Framework for Unconditionally Secure Multiparty Computation. *IEEE Transactions on Information Theory* 63, 1 (2017), 428–468.

[27] Ran Cohen, Abhi Shelat, and Daniel Wichs. 2019. Adaptively Secure MPC with Sublinear Communication Complexity. In *Advances in Cryptology – CRYPTO 2019*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer International Publishing, Cham, 30–60.

[28] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. 2009. Asynchronous multiparty computation: Theory and implementation. In *International workshop on public key cryptography*. Springer, 160–179.

[29] Ivan Damgård and Jesper Buus Nielsen. 2007. Scalable and Unconditionally Secure Multiparty Computation. In *Advances in Cryptology - CRYPTO 2007*, Alfred Menezes (Ed.). 572–590.

[30] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: A DAG-Based Mempool and Efficient BFT Consensus. In *Eurosys 2022*. 34–50.

[31] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. 2022. Spurt: Scalable Distributed Randomness Beacon with Transparent Setup. In *2022 IEEE Symposium on Security and Privacy (SP)*. 2502–2517.

[32] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. 2023. Practical Asynchronous High-threshold Distributed Key Generation and Distributed Polynomial Sampling. https://eprint.iacr.org/2022/1389 https://eprint.iacr.org/2022/1389.

[33] Sourav Das, Zhuolun Xiang, and Ling Ren. 2021. Asynchronous Data Dissemination and Its Applications. In *CCS 2021*. 2705–2721.

[34] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2518–2534.

[35] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *EUROCRYPT 2018*, Vol. 10821. 66–98.

[36] Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. 2022. GearBox: Optimal-Size Shard Committees by Leveraging the Safety-Liveness Dichotomy. In *CCS 2022*. 683–696.

[37] Luciano Freitas de Souza, Petr Kuznetsov, and Andrei Tonkikh. 2022. Distributed Randomness from Approximate Agreement. In *DISC 2022*, Vol. 246. 24:1–24:21.

[38] Rafael del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani Saarinen. 2024. Threshold Raccoon: Practical Threshold Signatures from Standard Lattice Assumptions. Cryptology ePrint Archive, Paper 2024/184. https://eprint.iacr.org/2024/184

[39] Danny Dolev, Nancy A Lynch, Shlomit S Pinter, Eugene W Stark, and William E Weihl. 1986. Reaching approximate agreement in the presence of faults. *Journal of the ACM (JACM)* 33, 3 (1986), 499–516.

[40] Danny Dolev and H. Raymond Strong. 1983. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput.* 12, 4 (1983), 656–666.

[41] Shlomi Dolev and Ziyu Wang. 2021. SodsBC/SodsBC++; SodsMPC: Post-Quantum Asynchronous Blockchain Suite for Consensus and Smart Contracts. In *Stabilization, Safety, and Security of Distributed Systems: 23rd International Symposium, SSS 2021*. 510–515.

[42] Sisi Duan, Michael K. Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT Made Practical. In *CCS 2018*. 2028–2041. https://doi.org/10.1145/3243734.3243812

[43] Sisi Duan, Xin Wang, and Haibin Zhang. 2023. Fin: Practical signature-free asynchronous common subset in constant time. In *CCS 2023*. 815–829.

[44] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), 238–268.

[45] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (April 1985), 374–382. https://doi.org/10.1145/3149.214121

[46] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Dumbo-NG: Fast Asynchronous BFT Consensus with Throughput-Oblivious Latency. In *CCS 2022*. ACM, 1187–1201.

[47] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Efficient Asynchronous Byzantine Agreement without Private Setups. In *ICDCS 2022*. IEEE, 246–257.

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

[49] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP 2017*. 51–68.

[50] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *STOC 1996*. 212–219.

[51] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice. In *NDSS 2022*.

[52] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. *Dumbo: Faster Asynchronous BFT Protocols*. 803–818.

[53] Christoph U. Günther, Sourav Das, and Lefteris Kokoris-Kogias. 2022. Practical Asynchronous Proactive Secret Sharing and Key Refresh. Cryptology ePrint Archive, Paper 2022/1586. https://eprint.iacr.org/2022/1586 https://eprint.iacr.org/2022/1586.

[54] Mads Haahr. 1999. random. org: Introduction to Randomness and Random Numbers. *Statistics,(June)* (1999), 1–4.

[55] Runchao Han, Haoyu Lin, and Jiangshan Yu. 2020. RandChain: A Scalable and Fair Decentralised Randomness Beacon. Cryptology ePrint Archive, Paper 2020/1033. https://eprint.iacr.org/2020/1033 https://eprint.iacr.org/2020/1033.

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

[57] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. 1995. Proactive secret sharing or: How to cope with perpetual leakage. In *annual international cryptology conference*. Springer, 339–352.

[58] Martin Hirt, Chen-Da Liu-Zhang, and Ueli Maurer. 2021. Adaptive Security of Multi-party Protocols, Revisited. In *TCC 2021 (Lecture Notes in Computer Science, Vol. 13042)*, Kobbi Nissim and Brent Waters (Eds.). Springer, 686–716.

[59] Shang-En Huang, Seth Pettie, and Leqi Zhu. 2023. Byzantine Agreement with Optimal Resilience via Statistical Fraud Detection. In *SODA 2023*, Nikhil Bansal and Viswanath Nagarajan (Eds.). SIAM, 4335–4353.

[60] Aniket Kate, Yizhou Huang, and Ian Goldberg. 2012. Distributed Key Generation in the Wild. *IACR Cryptol. ePrint Arch.* 2012 (2012), 377.

[61] Aniket Kate, Andrew Miller, and Tom Yurek. 2019. Brief Note: Asynchronous Verifiable Secret Sharing with Optimal Resilience and Linear Amortized Overhead. arXiv:1902.06095 [cs.CR]

[62] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *ASIACRYPT 2010*. Springer, 177–194.

[63] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All You Need is DAG. In *PODC 2021*. 165–175.

[64] John Kelsey, Luís TAN Brandão, Rene Peralta, and Harold Booth. 2019. *A reference for randomness beacons: Format and protocol version 2*. Technical Report. National Institute of Standards and Technology.

[65] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. 2020. Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures.. In *CCS 2020*. 1751–1767.

[66] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited. In *PODC 2020*, Yuval Emek and Christian Cachin (Eds.). 129–138.

[67] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *CCS 2016*. 31–42.

[68] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. 2015. Signature-free asynchronous binary Byzantine consensus with t< n/3, O (n2) messages, and O (1) expected time. *Journal of the ACM (JACM)* 62, 4 (2015), 1–21.

[69] Arpita Patra, Ashish Choudhary, and C Pandu Rangan. 2009. Efficient statistical asynchronous verifiable secret sharing with optimal resilience. In *International Conference on Information Theoretic Security*. Springer, 74–92.

[70] Phillip Rogaway and John Steinberger. 2008. Constructing cryptographic hash functions from fixed-key blockciphers. In *Advances in Cryptology–CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings 28*. Springer, 433–450.

[71] Sambhav Satija, Apurv Mehra, Sudheesh Singanamalla, Karan Grover, Muthian Sivathanu, Nishanth Chandran, Divya Gupta, and Satya Lokam. 2020. Blockene: A high-throughput blockchain over mobile devices. In *OSDI 2020*. 567–582.

[72] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. 2020. HydRand: Efficient Continuous Distributed Randomness. In *2020 IEEE Symposium on Security and Privacy (SP)*. 73–89.

[73] Victor Shoup. 2024. A Theoretical Take on a Practical Consensus Protocol. Cryptology ePrint Archive, Paper 2024/696. https://eprint.iacr.org/2024/696

[74] Victor Shoup and Nigel P. Smart. 2023. Lightweight Asynchronous Verifiable Secret Sharing with Optimal Resilience. Cryptology ePrint Archive, Paper 2023/536. https://eprint.iacr.org/2023/536

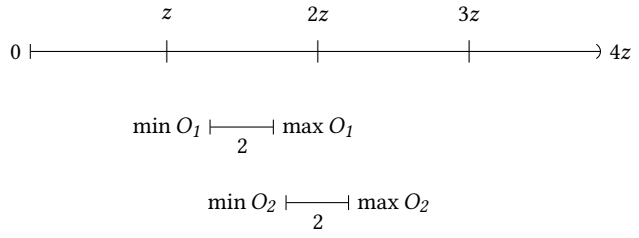

Figure 6: Monte-Carlo beacons: The picture describes a Monte-Carlo beacon in the context of leader election in an $n = 4$ system. In this context, the beacon must be a random number in the domain $\mathcal{D} = \{1, 2, 3, 4\}$. The nodes participate in the approximate common coin primitive by secret sharing a random secret from a domain $\mathcal{D}'$ of size $z\,|\mathcal{D}|$, where $z = \frac{2}{1-\delta}$ and $\epsilon = \frac{1}{z|\mathcal{D}|}$. Let O_i be set of outputs $o_{j,i}$ of honest nodes $j \in \mathcal{N}$ from the approximate common coin for beacon index i. O_i is an interval with $|O_i| \leq 2$ numbers in $\mathcal{D}' = [0, 4z)$. Every node j outputs $b_i = \left\lfloor \frac{o_{j,i}}{z} \right\rfloor + 1$ as the beacon output. If O_i is on different sides of a checkpoint, then nodes output different beacons resulting in a disagreement. The probability this event happening is $1 - \delta$ because the set O_i is distributed uniformly randomly in $\mathcal{D}'$. In this example, the second beacon b_2 corresponding to O_2 results in disagreement.

[75] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. 2020. On the optimality of optimistic responsiveness. In *CCS 2020*. 839–857.

[76] Open source contributors. 2019. *DRand - A Distributed Randomness Beacon Daemon - https://github.com/drand/drand.* https://github.com/drand/drand

[77] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. 2017. Scalable Bias-Resistant Distributed Randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*. 444–460.

[78] Xuechao Wang, Peiyao Sheng, Sreeram Kannan, Kartik Nayak, and Pramod Viswanath. 2023. TrustBoost: Boosting Trust among Interoperable Blockchains. In *CCS 2023*. 1–15. https://eprint.iacr.org/2022/1428.

[79] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus with linearity and responsiveness. In *PODC 2019*. 347–356.

[80] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. 2022. hbACSS: How to Robustly Share Many Secrets. In *NDSS 2022*.

[81] Thomas Yurek, Zhuolun Xiang, Yu Xia, and Andrew Miller. 2023. Long live the honey badger: Robust asynchronous dpss and its applications. (2023).

[82] Haibin Zhang, Sisi Duan, Boxin Zhao, and Liehuang Zhu. 2023. WaterBear: Practical Asynchronous BFT Matching Security Guarantees of Partially Synchronous BFT. In *USENIX Security Symposium'23*. 1–16.

A CALCULATING ANYTRUST SAMPLE SIZE

We use a hypergeometric probability distribution to model sampling from a group of nodes without repetition. The sampled set C must contain at least one honest node i whose secret will have a weight $w_{i,j} = 1 \forall j \in \mathcal{N}_{\text{hon}}$ in the aggregation. We know that the core set of Gather contains at least $t + 1$ honest nodes, out of which at least one honest node must be in the C. Therefore, the probability of at least one such honest node being on a committee of size c is given as follows.

$$p_c = 1 - \mathcal{H}.cdf(0, n, c, t + 1) \tag{1}$$

$\mathcal{H}$ is a hypergeometric distribution with total population n and $n - 2t \geq t + 1$ honest nodes part of the Gather core set G as the population with the desired feature. We plot the committee size $c = |\text{C}|$ for a given probability p_c with varying n in the Fig. 7.

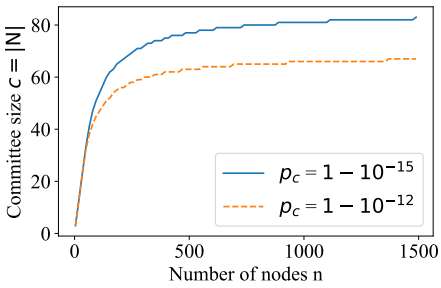

Figure 7: The plot describes the size of the committee $c = |\text{C}|$ with probability of at least one honest node in the core set G being part of the committee C, whose secrets eventually contribute to the beacon. For $p_c = 1 - 10^{-12}$, the committee size caps at $c = 60$ for higher values of n.

B EXTENDED RELATED WORK

AVSS **schemes.** There are many AVSS schemes with different hardness assumptions and corresponding computation and communication complexities. In the DLog world, Das et al.'s [33] scheme with Pedersen commitments has the lowest $O(\kappa n^2)$ communication complexity. It uses $O(n)$ exponentiations in the sharing phase and $O(n^2)$ exponentiations in the reconstruction phase. At the other end of the spectrum, information-theoretic (IT) VSS protocols such as statistical AVSS and perfect AVSS have communication complexities of $O(n^4 \log(n))$ and do not use any hardness assumption. Computational AVSS schemes such as Backes et al. [11] assume a one-way function for polynomial commitments. This variant of AVSS has a communication complexity of $O(\kappa n^3)$, less than IT protocols but higher than Das et al. Other AVSS schemes with differing commitment properties have also been proposed. One such property is the strong commitment [10] or completeness property used in ACSS protocols, necessary for AMPC and ADKG. ACSS protocols with $O(\kappa n^2)$ comm. complexity depend on a Level 4 setup with PKI and a CRS(for DLog NIZK proofs of valid encryption) [9, 33, 61] or SRS setup(powers of tau) [4, 10, 80] for achieving the completeness property. Kogias et al. 's [65] HAVSS achieves completeness with $O(\kappa n^3)$ complexity and a Level 2 setup. AVSS schemes with weaker commitment properties have also been proposed. Dolev et al. [41] proposed a weaker variant of AVSS that offers weak commitments, where a malicious dealer is only detected in the reconstruction phase, and honest nodes agree on the dealer being malicious. HashRand builds on top of this scheme to further improve the comm. complexity of this primitive. A weaker form of VSS called Asynchronous Weak Secret Sharing (AWSS) was proposed by Patra et al. [69] where a malicious dealer can force a subset of honest nodes to reconstruct $\perp$. The AwVSS primitive is stronger than AWSS because of its requirement of honest nodes agreeing on the maliciousness of the dealer.

Asynchronous SMR protocols. HoneyBadgerBFT [67] was the first practical asynchronous SMR protocol. It used n binary BA

instances to agree on an Asynchronous Common Subset (ACS) and requires expected $O(n\log(n))$ bits of randomness to terminate. The Dumbo family of protocols — Dumbo [52], Speeding Dumbo [51], and Dumbo-NG [46] — conduct SMR using Multi-Valued Validated BA (MVBA), where nodes elect a leader and its corresponding $n-t$ referenced RBC messages. These protocols use aggregatable threshold signatures to achieve a $O(n^2)$ communication complexity. Without aggregatable threshold signatures, the state-of-the-art MVBA [43] has $O(n^3)$ communication complexity. However, the MVBA family of protocols requires expected $O(\log(n))$ bits of randomness to elect the leader. Recent DAG-based SMR protocols like DAG-Rider [63] and Tusk [30] commit blocks in waves by electing wave leaders. Both MVBA and DAG-based protocols use coins for leader election where the protocol's safety depends on the coin's agreement property. However, protocols like HoneyBadgerBFT [67], Beat [42], and Dumbo 1 [52] depend on the coin only for achieving liveness in their Binary BA instances.

C BATCH AWVSS

We present the full BAwVSS protocol along with its security proofs in this section.

BAwVSS **security.** The Termination, Correctness, and Weak Commitment properties of our BAwVSS scheme follow Dolev et al.'s [41] scheme.

Lemma C.1. *Under collision resistance and preimage resistance of the hash function H, the* BAwVSS *protocol in Algorithm 2 satisfies Termination, Correctness, and Weak Commitment.*

Proof. If the dealer d is honest, every honest node will send out ECHOs by verifying their share tuples and their inclusion in the root vector. By the liveness and totality properties of reliable broadcast, every honest node will eventually terminate the sharing phase of the protocol, and at least $t+1$ honest nodes will have shares for all k secrets. If all nodes terminated BAwVSS.Sh(d) and start BAwVSS.Rec(.), then at least $t+1$ honest nodes must broadcast their shares to everyone. Since the degree of the sharing and nonce polynomial is t, these $t+1$ points are enough to reconstruct both share and nonce polynomials and verify the validity of the root. With the help of these $t+1$ honest nodes, every honest node terminates BAwVSS.Rec(.).

We prove correctness by contradiction. If the dealer D is honest, the honest nodes reach an agreement on the root vector, and at least $t+1$ honest nodes have shares for all k secrets. Assuming an honest node i reconstructs $s_i \neq s_i$ shared by the dealer, the node i must have reconstructed the polynomials $g_i'(x) \neq g_i(x)$ and $r_i'(x) \neq r_i(x)$. For this to happen, i must have used a share s_m not shared by the dealer D in Lagrange interpolation. However, since i validated the share s_m, the adversary must have generated a valid commitment $H(s_m, r_m)$ for a share not generated by the dealer. This implies that the adversary found a hash collision, which violates the collision resistance property of the function H. Hence, Algorithm 2 satisfies Correctness.

We prove weak commitment by contradiction under two cases: a) Honest nodes i, j reconstructed different secrets in the field $\mathcal{D}$ and b) Honest node i reconstructed a valid secret in $\mathcal{D}$ whereas j reconstructed $\perp$.

(1) Let i, j reconstruct $s_i, s_j \in field$ respectively. From the Termination property of BAwVSS.Sh(d), both i, j must terminate with the same root vector. Since $s_i \neq s_j$, two different degree t polynomials $g_i'(x)$ and $g_j'(x)$ construct two Merkle trees with the same root, which implies that there must be at least one hash collision while aiding the disagreement. This collision violates the collision-resistance property of the Hash function H.
(2) Let i, j reconstruct $s_i \in \mathcal{D}, \perp$ respectively. From the termination property of BAwVSS.Sh(d), i, j terminated with the same root vector. This disagreement implies that i and j used a different set of $t+1$ shares to construct their polynomials $g_i'(x)$ and $g_j'(x)$. i constructed a degree t polynomial and validated all n shares, which included the set of $t+1$ shares used by j to construct $g_j'(x)$. Since j reconstructed a different degree t polynomial with the shares validated by i, a malicious node must have sent a wrong share s_m to j, which had the same commitment as the corresponding share validated by i. This violates the collision-resistance property of the hash function H.

□

We prove the secrecy property of our BAwVSS scheme, where we establish that if no honest node invoked BAwVSS$_d$.Rec(i) for the ith secret in the batch, then the adversary has no information about it. We prove this property against an adaptive adversary using the definition of simulation-based security. We use a zero-knowledge simulator Sim to provide arguments of indistinguishability of the adversary's view when interacting with the simulator (who has no knowledge of the secrets) and a real dealer. We utilize the terminology and methods specified in Bingo [4] to build our proof. We define the adversary model, the secrecy game between the simulator Sim and the adversary $\mathcal{A}$, and the methodology followed by the simulator to prove indistinguishability.

Adversary model. We assume a probabilistic polynomial time (PPT) adaptive adversary $\mathcal{A}$ capable of corrupting nodes in the system at any time during the execution. Further, $\mathcal{A}$ controls the network and therefore decides the time and relative order of message delivery, with the restriction that messages must eventually be delivered. We model $\mathcal{A}$'s interactions with the system using a set of Oracles, using which $\mathcal{A}$ performs a specific action. We use Bingo's [4] definitions directly for this model. The functions provided by the oracles reflect $\mathcal{A}$'s power to schedule messages sent over the network and the power to corrupt nodes at any time during the execution.

(1) $\mathrm{O}_{\mathrm{BUF}}$ is a message buffer which allows the adversary to send a message to any honest node in the system. This oracle has three functions: (1) $\mathrm{O}_{\mathrm{BUF}}(\mathrm{add}, x)$ adds a message x to the message buffer, which allows $\mathcal{A}$ to send a message to an honest node from a faulty node, (2) $\mathrm{O}_{\mathrm{BUF}}(\mathrm{deliver}, j)$ which delivers the jth message in the buffer, and (3) $\mathrm{O}_{\mathrm{BUF}}(\mathrm{see})$, which allows $\mathcal{A}$ to see all the messages in the buffer.
(2) $\mathrm{O}_{\mathrm{CORR}}$ allows $\mathcal{A}$ to corrupt a node i in the system at an indexed step t by invoking $\mathrm{O}_{\mathrm{CORR}}(i, t)$. $\mathcal{A}$ gets access to the view of i at step t denoted by $\mathrm{view}_{i,t}$.

Algorithm 2 BAwVSS: Batch Asynchronous Weak Verifiable Secret Sharing (BAwVSS)

INPUT: $\mathcal{N}$,*RBC*,S_d // *RBC*: RBC protocol,S_d: secrets to share

Protocol BAwVSS$_d$.Sh(S_d):
upon receiving (ID.d,in,share,S_d): // As a dealer
// Sample $|S_d|$ nonces from a domain $\mathcal{D}_r$ at least twice the H's output range $R_d[i] \leftarrow Random(\mathcal{D}_r) \forall i \in \{1,\ldots,|S_d|\}$
// Use Shamir's SS scheme to create shares for secrets in S_d, $R_d[i]$
$\mathcal{S}_d[i] \leftarrow$Shamir.Split($S_d$ [i],n,t) $\forall i \in \{1,\ldots,|S_d|\}$
$\mathcal{R}_d[i] \leftarrow$Shamir.Split($R_d[i]$,n,$t$) $\forall i \in \{1,\ldots,|S_d|\}$
// Compute Commitments
$C_d[i] \leftarrow \langle H(\mathcal{R}_d[i][0], \mathcal{S}_d[i][0]), \ldots, H(\mathcal{R}_d[i][n], \mathcal{S}_d[i][n])\rangle \forall i \in \{1,\ldots,|S_d|\}$
// Compute Merkle Trees
$M_d[i]$ = Merkle($C_d[i]$) $\forall i \in \{1,\ldots,|S_d|\}$
$R_d \leftarrow \{M_d[0].\text{Root}, \ldots, M_d[k].\text{Root}\}$
$S_d[j][i] \leftarrow \langle \mathcal{S}_d[i][j], \mathcal{R}_d[i][j], M_d[i].\text{Proof}(j)\rangle \forall i \in \{1,\ldots,|S_d|\}, \forall j \in \{1,\ldots,n\}$
// Send shares to all nodes
for $j \in \{1,\ldots,n\}$ **do**
Send "ID.d, share, $S_d[j]$" to node j
end for
// Reliably Broadcast root vector
Invoke *RBC*.Broadcast(R_d)
// As a participant
upon receiving "ID.d,share,$S_d[j]$" and *RBC*.Init(R_d) from dealer d:
// Verify commitments and participate in *RBC* if all pass
if Merkle.Verify($R_d[i]$,$S_d[j][i]$.share,$S_d[j][i]$.proof) = 1$\forall i \in \{1,\ldots,|S_d|\}$ **then**
Send ECHO messages in d's RBC protocol *RBC*
end if
upon invoking *RBC*.Deliver(R_d) with share:
output (ID.d,out,share,R_d)
upon invoking *RBC*.Deliver(R_d) with no share:
output (ID.d,out,no-share,R_d)

Protocol BAwVSS$_d$.Rec(i):
$SS_d[i] \leftarrow \emptyset$
upon receiving (ID.d,in,recon,i):
if terminated with share **then**
for $j \in \{1,\ldots,n\}$ **do**
Send "ID.d, recon,$S_j[i]$" to node j
end for
end if
upon receiving (ID.d,recon,$S_d[m][i]$) from node m:
if Merkle.Verify($R_d[i]$,$S_d[m][i]$.share,$S_d[m][i]$.proof) = 1 **then**
$SS_d[i] \leftarrow SS_d[i] \cup \langle m, S_d[m][i]\rangle$
end if
upon $|SS_j[i]| \geq t+1$
// Combine shares and reverify root vector commitment
$\mathcal{S}_d[i], \mathcal{R}_d[i] \leftarrow$Shamir.Combine($SS_d[i]$,n,$t$)
$C_d[i] \leftarrow \langle H(\mathcal{R}_d[i][0], \mathcal{S}_d[i][0]), \ldots, H(\mathcal{R}_d[i][n], \mathcal{S}_d[i][n])\rangle \forall i \in \{1,\ldots,|S_d|\}$
if Merkle($C_d[i]$).Root = R_i **then**
output (ID.d,out,reconstructed,$\mathcal{S}_d[i][0]$,i)
else
output (ID.d,out,reconstructed,$\perp$,i)
end if

(3) O_S is a sharing oracle which allows $\mathcal{A}$ to control the time at which the sharing phase begins. $\mathcal{A}$ invokes $O_S(i)$ to let node i participate in the protocol.
(4) O_R is a reconstruction oracle which allows $\mathcal{A}$ to control the time at which reconstruction begins. $\mathcal{A}$ invokes $O_R(i,k)$ to enable node i to start reconstructing the kth secret in the batch.

Game definition. We define the secrecy game between the simulator Sim and the adversary $\mathcal{A}$ according to Bingo [4]. In this game, $\mathcal{A}$ chooses a batch of β secrets denoted by $S_{\mathcal{A}}$ and interacts either with the simulator Sim (denoted by $b = 0$), which has no knowledge of $S_{\mathcal{A}}$, or with a system of n nodes with a designated dealer (denoted by value $b = 1$), which knows $S_{\mathcal{A}}$. $\mathcal{A}$ does not know who they are interacting with and does not know the value of b. $\mathcal{A}$ invokes an initialization oracle $O_{\text{INIT}}(S_{\mathcal{A}})$ with the batch of secrets. The game chooses a bit b randomly and initializes the Sim if $b = 0$ or the designated dealer node with the secrets $S_{\mathcal{A}}$ if $b = 1$. $\mathcal{A}$ uses the described oracles to interact with the underlying player. At the end of the game, $\mathcal{A}$ must guess whether it played with the simulator or the real set of nodes.

The simulator Sim does not have any information about the secrets $S_{\mathcal{A}}$ in the sharing phase BAwVSS$_d$.$Sh(i) \forall i \in \{1,\ldots,|S_{\mathcal{A}}|\}$. However, if Sim does not know $S_{\mathcal{A}}$ in the reconstruction phase, then it would be trivial for $\mathcal{A}$ to recognize the difference between the two described scenarios. Therefore, the game provides the ith secret to Sim right before $\mathcal{A}$ instructs an honest node to start reconstruction of secret i. In other words, the game provides the secret to Sim at the latest possible step at which it would become trivial for $\mathcal{A}$ to determine if it were interacting with Sim or a real dealer [4].

We restate Bingo's definition of the secrecy game here.

Definition C.1. ***Secrecy*** *[4]: Consider a simulator* Sim *that interacts with adversary $\mathcal{A}$ on behalf of all honest nodes. We define $\mathcal{A}$'s advantage as* $\boldsymbol{Adv}^{secrecy}_{\mathcal{A}}(\lambda) = 2\Pr[G^{\mathcal{A}}_{secrecy}] - 1$, *where $G^{\mathcal{A}}_{secrecy}$ is defined as follows, given the dealer is party d who deals only after receiving a start message from the adversary:*

MAIN $G^{\mathcal{A}}_{secrecy}(\lambda)$:
$b \leftarrow^r \{0,1\}$;
$b' \leftarrow \mathcal{A}^{O_{\text{BUF}}(add,.),O_{\text{BUF}}(see),O^*,O_{\text{CORR}},O_{\text{INIT}},O_S,O_R}(1^\lambda)$
return $(b' = b)$
$O_{\text{INIT}}(s_0,\ldots,s_m)$
$S_i \leftarrow s_i \forall i \in [m]$
O_{CORR} *(i)*
if (i=d) // If the adversary corrupts the dealer
add $S_1,\ldots,S_m$ *to* $\text{State}_{\text{Sim}}$ *// Give secrets to* Sim
end if
remove i from set of honest nodes
O^* *(m)*

if (b=0)
run $\mathsf{Sim}(H, m)$
else
Initialize dealer d with $\vec{S}$ *and run* $\mathrm{O}_{\mathrm{BUF}}(\mathsf{deliver}, m)$
end if
$\underline{\mathrm{O}_{\mathrm{R}}(i,k)}$
// Instruct node i to reconstruct the kth secret
upon *finishing* $\mathrm{AwVSS}_d.\textsc{Sh}$*:*
add S_k *to* $\mathsf{State}_{\mathsf{Sim}}$ *// Provide secret to* Sim
start $\mathrm{AwVSS}_d.Rec(k)$

Secrecy holds if for all PPT adversaries $\mathcal{A}$*, there exists a negligible function* $\nu(\lambda)$ *such that* $\boldsymbol{Adv}^{secrecy}_{\mathcal{A}}(\lambda) < \nu(\lambda)$

Now, we formally define the secrecy property of our BAwVSS scheme. We prove secrecy of our scheme by following Bingo's proof approach of using a blinding polynomial to unconditionally hide the original polynomial. Bingo's PAVSS scheme uses KZG commitments [62] depending on a powers-of-tau ceremony and the q-Strong Diffie Hellman (q-SDH) assumption. Hence, Bingo instantiates Sim with a trapdoor τ_s of the powers-of-tau setup. This enables Sim to utilize any adversarial advantage $\mathbf{Adv}^{secrecy}_{\mathcal{A}}(\lambda)$ and break the q-SDH hardness assumption.

When $\mathcal{A}$ corrupts an honest node i at step t in the protocol, Sim must return a view $\mathsf{view}_{i,t}$ that is consistent with the state of the faulty nodes controlled by $\mathcal{A}$. Further, Sim must also ensure that $\mathsf{view}_{i,t}$ is consistent with the public information disseminated to all nodes in the sharing phase. Moreover, Sim only gets the information about the secret s_i right before reconstruction, which implies that Sim must construct the public commitment information without any knowledge of the secrets. This problem has been described as the *explainability* problem or commitment problem in prior literature on adaptive security [21, 58]. Bingo solves this problem by first creating a simulated commitment without any knowledge of the secrets. Later, when Sim learns the secret, it uses the blinding polynomial and the $Po\tau$ trapdoor to *fit* the secret to the commitment.

We use a similar approach to solve this commitment problem. We use unconditionally hiding commitments, which uses a blinding polynomial in a bigger domain $\mathcal{D}_r$, with each element at least twice the size of the output of H (for SHA256, the domain $\mathcal{D}_r$ is of size 512 bits). In We choose a large blinding polynomial size to support secret sharing across arbitrarily small or large domains $\mathcal{D}$. We assume the Hash function H is a Random Oracle (RO). In our approach, Sim has programmable access to H and can adaptively program the outputs of H. We utilize this access to solve the explainability problem described above by enabling Sim to fit the view $\mathsf{view}_{i,t}$ to the public commitment information.

Theorem C.2. *Under the assumption that the Hash function H is a programmable Random Oracle, the* BAwVSS *scheme in Algorithm 2 satisfies secrecy defined according to Definition C.1.*

Proof. We define the program followed by the simulator Sim on each oracle call made by the adversary $\mathcal{A}$. Then, we argue that $\mathcal{A}$ cannot distinguish between a real protocol execution and the simulated execution based on the Random Oracle assumption. We denote the set of honest nodes by $\mathcal{N}_{\mathsf{hon}}$ and the set of corrupted nodes by $\mathcal{N}_{\mathsf{corr}}$.

- When the oracle $\mathrm{O}_*(m)$ is called to start the game and if $b = 0$, Sim samples β random polynomials $f_i(x), r_i(x) \forall i \in \{1, \ldots, \beta\}$, with secrets $S_{\mathsf{Sim}} = \{f_1(0), \ldots, f_\beta(0)\}$. Sim then acts as the dealer running the Sharing protocol $\mathrm{BAwVSS}_{\mathsf{Sim}}.\textsc{Sh}(S_{\mathsf{Sim}})$ among the set of n nodes. The nodes in $\mathcal{N}_{\mathsf{corr}}$ would receive messages from honest nodes just like in a real execution. We denote the generated commitments in Line 6 using $\mathrm{C}_{\mathsf{Sim}}[i] \forall i \in \{1, \ldots, \beta\}$. The transcripts of messages between nodes in $\mathcal{N}_{\mathsf{hon}}$ are encrypted with an encryption scheme with forward secrecy (also called non-committal encryption [21]), which allows Sim to later change the content within these messages without changing the transcript.
- When $\mathcal{A}$ invokes $\mathrm{O}_{\mathrm{R}}(i,k)$, Sim receives the kth secret $S_{\mathcal{A}}[k]$. Consider the kth secret and nonce polynomial $f_k(x)$,$r_k(x)$. Sim creates a set of evaluations $\{S_{\mathcal{A}}[k], f_k(i_1), \ldots, f_k(i_j)\}$ at indices of points $0 \cup P_{\mathsf{corr}}$, where P_{corr} is the set of indices of nodes in set $\mathcal{N}_{\mathsf{corr}}$, and samples a random degree-t polynomial $f'_k(x)$ passing through these points. Similarly, Sim also samples a new nonce polynomial $r'_k(x)$ using evaluations $\{R(k)i_1, \ldots, R(k)i_j\}$ at points P_{corr}. Note that if $|\mathcal{N}_{\mathsf{corr}}| = t$, there is only one unique polynomial $f'_k(x)$ passing through the $t+1$ points.
  Next, Sim programs the random oracle to output values $\mathrm{C}_{\mathsf{Sim}}[k][i]$ for inputs $H(r'_k(i), f'_k(i)) \forall i \in \mathcal{N}$, with $H(r'_k(i), f'_k(i)) = \mathrm{C}_{\mathsf{Sim}}[k][i]$ $\forall i \in \mathcal{N}$. Note that this programming is consistent with all the public information in the protocol. The Merkle tree built on the commitments built from these new values will be equivalent to the Merkle tree built from old shares.
  Finally, for each node in set $\mathcal{N}_{\mathsf{hon}}$, Sim replaces the old Share messages (in Line 11) in the message buffer with new updated shares from polynomials $f'_k(x)$ and $r'_k(x)$. If specific honest nodes already processed the message, then Sim changes their internal state to update the kth share to the new value. The node i runs reconstruction by sending its updated secret share, which will be reconstructed to $\mathcal{A}$'s secret $S_{\mathcal{A}}[k]$.
- When $\mathcal{A}$ invokes $\mathrm{O}_{\mathrm{CORR}}(i)$ on node i, Sim checks if O_{R} has been invoked by $\mathcal{A}$ on any secret. If $\mathcal{A}$ invoked O_{R}, then Sim conducts the share replacement and oracle programming described in the previous bullet (Sim ignores this stage if it has already been performed), and then transfers the internal state of node i to $\mathcal{A}$. If $\mathcal{A}$ did not invoke O_{R} yet on any secret, then Sim directly transfers the internal state of i to $\mathcal{A}$ without making any changes.

We argue that the adversary $\mathcal{A}$ cannot distinguish a simulator Sim following the described program from a real protocol execution with probability $p > \nu(\lambda)$.

We first note that the generated public transcripts (Commitment Vectors and Merkle trees) come from probabilistically equivalent distributions of the RO. This is because Sim uses fully random values in O_* to generate the commitment vector $\mathrm{C}_{\mathsf{Sim}}[i] \forall i \in \{1, \ldots, \beta\}$. Similarly, the real dealer also chooses polynomial coefficients of both polynomials $f_{d,i}(x)$ and $r_{d,i}(x)$ $\forall i \in \{1, \ldots, \beta\}$ randomly. We also note that Sim has not programmed the RO yet, because of which both commitment vectors and Merkle trees come from probabilistically identical RO models. Therefore, $\mathcal{A}$ cannot decipher any information from public transcripts alone.

Next, we show that $\mathcal{A}$ also cannot decipher any distinguishing information from secret shares. From the properties of Shamir secret sharing, an adversary with at most t points on a degree-t

polynomial has no information about the encoded secret. In the sharing phase, $\mathcal{A}$ controls at most t nodes and knows at most t points on Sim's polynomials $f_{\mathsf{Sim},i}(x)$ and $r_{\mathsf{Sim},i}(x)\ \forall i \in \{1,\ldots,\beta\}$. Therefore, $\mathcal{A}$ cannot distinguish Sim's polynomials from d's polynomials in the sharing phase. In the reconstruction phase, Sim replaces the shares of all honest nodes to reflect $\mathcal{A}$'s provided secrets. Therefore, $\mathcal{A}$ cannot distinguish the simulated view and the real view from received secret shares alone.

Finally, we note that the only distinguishing factor between the real and simulated views is the state of the Random Oracle in the reconstruction phase, where Sim programs the RO in response to $\mathcal{A}$'s $\mathrm{O_R}$ invocations. In the simulated view, the Sim programs the RO at most $O(\beta n)$ times, $O(n)$ times per secret. This programming may result in $\mathcal{A}$ receiving conflicting responses from the RO. Therefore, we split this situation into the following two cases:

(1) Adversary $\mathcal{A}$ receives conflicting responses from the RO for the same input string. This only happens when the Sim overwrites the response encoded in the RO for an input string queried by $\mathcal{A}$. In this case, $\mathcal{A}$ recognizes that they are in a simulation. The probability of this event occurring is the probability that $\mathcal{A}$ queried the same string sampled by Sim in response to $\mathrm{O_R}$ invocation. As Sim samples these values randomly from domain $\mathcal{D}_r = 1^{2\lambda}$, the probability of a single query matching one of $O(\beta n)$ queries is $p = \frac{\beta n}{\exp(\lambda)}$. The probability of a PPT adversary $\mathcal{A}$ succeeding is $p_s \leq \frac{\mathrm{poly}(\lambda)\beta n}{\exp(\lambda)}$. As n and β are both polynomially bound, the probability of success is $p_s \leq \frac{\mathrm{poly}(\lambda)}{\exp(\lambda)}$, which is negligible in λ.

(2) $\mathcal{A}$ does not receive conflicting responses from the RO. In this case, $\mathcal{A}$ cannot gauge any distinguishing information between the real execution and the simulation. This is because from the Pigeon Hole principle, an expected 2^λ entries in the domain $\mathcal{D}_r$ map to each entry in RO's range. Therefore, $\mathcal{A}$ cannot infer any details from polynomial number of input-output queries from the RO.

Therefore, under the assumption that the Hash function H is a Random Oracle, the adversary $\mathcal{A}$'s advantage in the game $\mathrm{G}^{\mathcal{A}}_{secrecy}$ is bounded by the negligible function $\nu(\lambda) = \frac{\mathrm{poly}(\lambda)}{\exp(\lambda)}$, which gives $\mathbf{Adv}^{secrecy}_{\mathcal{A}}(\lambda) < \nu(\lambda)$. □