HashRand: Efficient Asynchronous Random Beacon *without* Threshold Cryptographic Setup

Akhil Bandarupalli*, Adithya Bhat†, Saurabh Bagchi*, Aniket Kate*‡, Michael Reiter§
*Purdue University
†Visa Research
‡Supra Research
§Duke University

***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. These protocols have found applications in blockchain technology, leading 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 assume partial or bounded synchronous networks. To overcome these limitations, we propose HashRand, a computation and communication-efficient asynchronous random beacon protocol that uses a secure Hash function to generate beacons and pairwise secure channels. HashRand has a per-node communication complexity of $\mathcal{O}(\lambda n \log(n))$ 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 $n = 160$ nodes, HashRand produces 1 beacon every second, which is at least 4x higher than Spurt. We also demonstrate the practical utility of HashRand by implementing a Post-Quantum secure Asynchronous SMR protocol, which has a response rate of over 122k txns per second over a WAN at $n = 40$ nodes.**

1. Introduction

Random beacons [51], [63] are a source of secure randomness that emit random numbers at regular intervals. They are often used in distributed system applications like committee election in sharded and Proof-of-Stake blockchains [29], [39], [42], [58]; common coins in asynchronous Byzantine Agreement (BA) [1], [4], [54], [56]; State Machine Replication (SMR) [24], [33], [36], [41], [42], [50], [55], [69]; asynchronous Multi-Party Computation (MPC) [23], [61]; and secure message mixing in anonymous communication [5], [53], [66].

Contemporary random beacon protocols are composed of a wide variety of trusted setup assumptions. Beacons from sources like Random.org [44] or NIST's random beacon project [51] require complete trust in the source. The source here becomes a single point of failure with respect to the system's liveness and safety as well as unpredictability and bias-resistance of beacon values. Approaches like DRand [63], Dfinity-DVRF [46], and Cachin et al. [19] mitigate these risks by distributing the trust across n nodes such that the system remains safe, live, and secure as long as a subset of $t + 1$ or more nodes are honest.

This setup (also called threshold setup) enables n nodes 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). Although the trusted setup phase for threshold signatures can be replaced with a Distributed Key Generation (DKG) protocol [38], [49], [52], DKG protocols are expensive in computation and communication and must be rerun each time participant nodes change. This constraint makes threshold signatures costly for applications with participant churn, such as Proof-of-Stake blockchains [39].

Many random beacon protocols that do not use a threshold setup have been proposed under a Public Key Infrastructure (PKI) setup or broadcast setup, where digital signatures can be used [12], [13], [25], [45], [59], [64]. These protocols use Verifiable Secret Sharing (VSS) and State Machine Replication (SMR) primitives as building blocks to output beacons. However, they depend on some form of network synchrony (bounded/partial) to achieve liveness and agreement amongst participant nodes. Hence, these protocols are not secure in an asynchronous network and cannot be used in asynchronous blockchains. Moreover, as demonstrated in HoneyBadgerBFT [55], synchronous protocols cannot take full advantage of the network speed and out-of-order message delivery, thereby offering subpar performance under active adversarial scheduling.

Many asynchronous randomness generation protocols that do not use threshold signatures function in a PKI/broadcast setup [2], [3], [26], [28], [37] or a pairwise-channels setup [30], [52] where the dealer sets up secure channels between participant nodes. However, these protocols have a high computational cost from n-parallel Asyn-

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)	DLog Expo Complexity (per node)	Setup	Cryptographic Assumption
Cachin et al. [19]	async.	✗	D	✓	$\mathcal{O}(\lambda n)$	$\mathcal{O}(n)$	DKG	pRO & CDH
RandShare [64]	async.	✗	LV	✗	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(n^3)$	DKG	DLog
Spurt [25]	partial sync.	✗	D	✗	$\mathcal{O}(\lambda n^2)$	$\mathcal{O}(n^2)^{\P}$	CRS & PKI	pRO & DBDH
Kogias et al. [52]	async.	✗	LV	✗	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(n^3)$	Secure channels	pRO & DDH
Das et al. [28]	async.	✗	LV	✗	$\mathcal{O}(\lambda n^2)$	$\mathcal{O}(n^3)$	PKI	pRO & DDH
Gao et al. [37]	async.	✗	LV	✗	$\mathcal{O}(\lambda n^2)$	$\mathcal{O}(n^3)$	PKI	pRO & SXDH
Abraham et al. [3]	async.	✗	LV	✗	$\mathcal{O}(\lambda n^2)$	$\mathcal{O}(n^3)$	PKI	pRO & SXDH
Bingo [2]	async.	✗	LV	✓	$\mathcal{O}(\lambda n^2)$	$\mathcal{O}(n^3)$	SRS & PKI	$q-$SDH
Freitas et al. [30]	async.	✓	MC	✗	$\mathcal{O}(\lambda n^2 \log(n))$	$\mathcal{O}(n^3)$	Secure channels	DLog
Patra et al. [57]	async.	✓	MC	✓	$\mathcal{O}(n^4 \log(n))$	0	Secure channels	None
Huang et al. [48]	async.	✓	LV	✓	$\Omega(n^6)$	0	Auth channels	None
HASHRAND	async.	✓	MC	✗	$\mathcal{O}(\lambda n \log(n))^{\dagger}$	0^{*}	Secure channels	CR & IH Hash[§]
HASHRAND	async.	✓	MC	✓	$\mathcal{O}(\lambda n \log(n))^{\dagger}$	0^{*}	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. ***DLog Expo complexity**: Concurrent beacon and coin protocols overwhelmingly depend on discrete log cryptography, which is the main reason for their high computational cost.HASHRAND instead uses $\mathcal{O}(cn^2 \log(n))$ Hash computations(100x cheaper than DLog expo) per beacon. †**Committee election**: HASHRAND elects an *AnyTrust* committee and reconstructs only those secrets. AnyTrust committees are much smaller in size than honest supermajority committees used in beacons like RandHerd [64] and Algorand [39].[§]**Hash**: HASHRAND uses a Collision Resistant(CR), and an Input Hiding(IH) Hash function to prove security against a classical adversary. The Input Hiding property [15, page 345], requires the Hash function to not reveal any information about the input. We also use a Quantum Random Oracle (RO) assumption against a Quantum adversary [16]. Other protocols use a Programmable RO (pRO) to link the properties of their beacon to the underlying hardness assumption. [¶]The leader performs $\mathcal{O}(n^2)$ operations every round, while other nodes perform $\mathcal{O}(n)$ operations each.

chronous VSS (AVSS). The current best AVSS protocol with $\mathcal{O}(\lambda n^2)$ communication complexity [27] uses compute-intensive discrete log cryptography. Each AVSS instance requires $\mathcal{O}(n^2)$ discrete log exponentiations per node, where each exponentiation takes approximately $100\times$ more time and energy to run than a one-way hash computation. This computation bottleneck induced by discrete log operations is visible in Spurt [25], a leader-based partially synchronous beacon protocol. Spurt produces only 15 beacons per minute at $n = 128$ nodes because of the $\mathcal{O}(n^2)$ discrete log operations per beacon performed by the leader.

Other asynchronous beacon protocols in the realm of statistical and information-theoretic security use only scalar arithmetic operations and therefore are computationally efficient. However, these protocols have very high communication complexities and are not efficient enough to be deployed at scale in practice. For example, the best protocol in this space is Patra et al.'s [57] statistical AVSS combined with Freitas et al.'s [30] common coin technique, with a total communication complexity of $\mathcal{O}(n^4 \log(n))$ bits per beacon per node. There also exist works like Backes et al. [9] that use collision-resistant Hash functions to build AVSS. This approach is computationally efficient because Hash computations are at least 100x cheaper than discrete log operations. However, this AVSS scheme also has a high communication complexity of $\mathcal{O}(n^2)$ bits per node, which translates to impractical $\mathcal{O}(n^3)$ bits per node for a beacon built with this AVSS scheme.

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

Our results. We propose HASHRAND, a hash-based asynchronous random beacon protocol that uses a collision-resistant Hash function to output random beacons. HASHRAND has an amortized communication complexity of $\mathcal{O}(\lambda n \log(n))$ bits per beacon per node, where λ is the range of the hash function in bits, and requires $\mathcal{O}(\lambda n \log(n))$ 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 message authentication codes (MACs). On top of computational efficiency and practical communication complexity, HASHRAND is also post-quantum secure, which is an added benefit over protocols that use discrete-log cryptography. This is because contemporary one-way hash

functions like SHA256 offer collision-resistance and pre-image resistance against a polynomial time quantum adversary. We offer a detailed comparison of HASHRAND with related works in Table 1.

Evaluation. We implement HASHRAND in Rust by using SHA256 hash function as the one-way function. We evaluate HASHRAND in a geo-distributed setting on AWS and measure the throughput in terms of the number of beacons output per minute. We also compare HASHRAND with Dfinity-DVRF [46] based on BLS threshold signatures. With $n = 40$ nodes, HASHRAND outputs 2819 beacons per minute, against 1505 beacons pm for Dfinity-DVRF. Even at higher values of $n = 160$, HASHRAND still outputs 60 beacons per minute or 1 every second, which is 4x higher than partially synchronous Spurt [25], which outputs 15 beacons per minute at $n = 128$ nodes.

An illustrative application. On top of implementing and evaluating HASHRAND as an asynchronous beacon, we demonstrate its practical utility by implementing a post-quantum secure asynchronous SMR protocol. We integrate HASHRAND with TUSK [24], 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 [35], a post-quantum secure public key signature scheme. PQ-TUSK has an optimal communication complexity of $\mathcal{O}(\lambda n)$ bits per transaction per node, which is much more practical than other post-quantum SMR protocols like WaterBear [69] with worst case $\mathcal{O}(\exp(n))$ 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 and 122k transactions per second for $n = 40$ nodes, with a latency of 2.3 seconds and 17.5 seconds, respectively.

2. Problem Setting and Solution Overview

2.1. System Model

We assume a set of n nodes connected by pairwise authenticated and secure channels and an asynchronous network with no time bound on message delivery. We consider a polynomial-time quantum adversary $\mathcal{A}$ defined according to Boneh et al. [16], who has access to a quantum computer and controls $t < \frac{n}{3}$ nodes. $\mathcal{A}$ also controls the network between honest nodes and can arbitrarily delay and reorder messages. 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.

2.2. Asynchronous Random Beacons

We define a random beacon protocol and its properties based on Spurt's [25] 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 incremental 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 $\langle x, b_x \rangle : x \in \mathbb{N}, b_x \in \mathcal{D}$. An honest node $n_i \in \mathcal{N}$ 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(.,.) for an index y, and outputting $\langle x, b_x \rangle$ $\forall x \in [1, y)$, an honest node n_i invokes $\mathcal{B}$.OPEN(y) to initiate generating $\langle y, b_y \rangle$. Such a protocol $\mathcal{B}$ is a random beacon protocol iff it satisfies the following properties.*

1) ***Agreement**: For a given index x, if honest nodes n_j and n_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.*

$$Pr\left\{\begin{array}{l} \langle x, b_{xj} \rangle \leftarrow n_j \\ \wedge \langle x, b_{xk} \rangle \leftarrow n_k \\ \wedge\, b_{xj} \neq b_{xk} \end{array}\right\} \leq \mathsf{negl}(\lambda)$$

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.*

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

Prior random beacon protocols [12], [13], [25] 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.

2.3. HASHRAND Key Insights

The conventional design of prior random beacon protocols using VSS and SMR [13], [25], [59], [64] cannot be applied to an asynchronous beacon because asynchronous SMR itself requires random beacons to terminate, which creates a circularity [25]. Moreover, Freitas et al.'s [30] impossibility result states that a deterministic asynchronous protocol without a private trusted setup cannot satisfy all three properties in Definition 2.1. Hence, any asynchronous random beacon protocol must choose between Monte-Carlo and Las-Vegas flavors of termination.[1]

HASHRAND is a Monte-Carlo style beacon protocol that builds on top of the asynchronous approximate common coin [30], which uses approximate agreement or ϵ-agreement primitive. HASHRAND guarantees Monte-Carlo agreement with a probability of δ, which is a statistical security parameter. HASHRAND uses n-parallel AVSS, Gather,

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.

and ϵ-agreement as building blocks and builds on top of two key observations that make it computationally efficient with a practical communication complexity.

1) Conventional AVSS schemes' commitment property, which requires honest nodes to always reconstruct a field element and is the major reason for their high computational complexity, is overkill for a random beacon protocol. As we will show, 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) protocol on top of Dolev et al.'s [32] AwVSS protocol. For a batch size of $\mathcal{O}(n)$, this BAwVSS scheme has an amortized sharing complexity of $\mathcal{O}(\lambda n \log(n))$ bits and a reconstruction complexity of $\mathcal{O}(\lambda n^2 \log(n))$ bits per secret.

2) Reconstructing $\mathcal{O}(n) \geq t+1$ secrets is not necessary to guarantee the unpredictability of a beacon. As long as we can ensure that at least one honest node's secret will always be reconstructed and be part of the beacon, we can ensure the beacon's unpredictability. Utilizing this observation, we propose a committee election procedure, which elects an AnyTrust committee of constant size guaranteed to have at least one honest node whose BAwVSS instance will be terminated and reconstructed by all honest nodes. All nodes later reconstruct the secrets of only those nodes part of the committee. We bootstrap the randomness required for committee election by generating more randomness in the startup rounds. This technique brings down the communication complexity of the reconstruction phase to $\mathcal{O}(\lambda n^2 \log(n))$ bits or $\mathcal{O}(\lambda n \log(n))$ bits per secret per node.

The first observation allows us to reduce computational complexity without increasing the communication complexity of the beacon. Additionally, each instance of Gather and ϵ-agreement can be shared across all n secrets in a BAwVSS instance, effectively amplifying the throughput by the batch size. However, even with this observation, the communication complexity per beacon is still $\mathcal{O}(\lambda n^3 \log(n))$ bits because every beacon output still requires reconstructing $\mathcal{O}(n)$ secrets.

The second observation enables HashRand to bring this communication complexity down by $\mathcal{O}(n)$ factor. Although conventional committee-based protocols only show performance effects at very large values of n, HashRand's AnyTrust committee achieves a low statistical failure probability for much smaller committee sizes. For example, in an $n = 160$ node system, a committee of size $c = 60$ is enough to guarantee AnyTrust assumption with a probability $p = 1 - 2^{-40}$. Both these additions make HashRand highly computationally efficient with a $\lambda n \log(n)$ per node communication complexity.

Security. As HashRand provides Monte-Carlo agreement, the honest nodes can output different values and violate properties in Definition 2.1 with probability $p \leq 1 - \delta + \mathsf{negl}(\lambda)$, which corresponds to λ bits of cryptographic security and $\log(\frac{1}{1-\delta})$ bits of statistical security against a quantum adversary. Statistical security is independent of $\mathcal{A}$'s computational power, as opposed to cryptographic security where the negl function changes with the adversary's computational power.

We prove HashRand 's agreement, liveness, and unpredictability properties under collision-resistance,pre-image resistance, and input hiding assumptions of a hash function against a classical adversary [15]. However, to prove security of HashRand against a quantum adversary, we assume the Hash function is a Quantum Random Oracle [16].

3. Building Blocks

In this section, we describe the key building blocks employed in HashRand.

3.1. Asynchronous weak Verifiable Secret Sharing (AwVSS)

We describe the Asynchronous weak VSS primitive as defined in Dolev et al. [27]. This primitive facilitates sharing a secret from a set $\mathcal{D}$, which can be a ring or a finite field. However, the polynomial must be evaluated at points from a set $\mathcal{E} \subseteq \mathcal{D}$, where polynomial interpolation is possible. If $\mathcal{D}$ is a finite field, $\mathcal{E} = \mathcal{D}$ whereas if $\mathcal{D}$ is a ring, $\mathcal{E}$ is the exceptional set of that ring [61].

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

- AwVSS_d.Sh*: A dealer n_d shares a secret $s_d \in \mathcal{D}$ among nodes in $\mathcal{N}$. At the end of* AwVSS.Sh*, at least $t+1$ honest nodes $n_i \in \mathcal{N}$ hold a secret share $s_{d,i}$.*
- AwVSS_i.Rec*: Every honest node $n_i \in \mathcal{N}$ reconstructs the secret s_d in a distributed fashion by broadcasting its secret share $s_{d,i}$ (if available). If the dealer n_d was honest, every honest n_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 n_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 n_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 n_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 n_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 the widely used Asynchronous Verifiable Secret Sharing (AVSS) primitive 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 the reconstruction phase in AwVSS.

We build on top of Dolev et al.'s [32] hash-based AwVSS scheme and give a brief description of it. In this protocol, a dealer n_d uses Shamir secret sharing to create secret shares and then creates a Merkle tree on top of it. n_d then sends individual shares along with Merkle proofs to nodes and reliably broadcasts the Merkle root r. An honest node participates in the RBC of 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. Towards reducing communication complexity in an amortized fashion, we also define a batched version of AwVSS scheme, where a protocol BAwVSS implementing Batched AwVSS has sub protocols BAwVSS$_d$.Sh(S) and BAwVSS$_j$.Rec(.). The dealer n_d shares a batch of multiple secrets S_d with BAwVSS$_d$.Sh(S_d) and honest nodes reconstruct each secret with BAwVSS$_.$.Rec(i): $\forall i \in \{1, \ldots, |S_d|\}$. In addition to the properties of AwVSS, BAwVSS also has the all-or-none property, where a node must either terminate BAwVSS$_d$.Sh(S_d) for all secrets in the batch S_d or should not terminate at all. This property is useful in HashRand to increase throughput.

3.2. Gather

We also employ the Gather primitive [3].

Definition 3.2. *Let $\mathcal{T}_i$ be any protocol amongst nodes $\mathcal{N}$ invoked by a node $n_i \in \mathcal{N}$. Let $\mathcal{T}_i$ implement the Totality property, which states that if an honest node n_j terminates $\mathcal{T}_i$, then every other honest node n_m terminates $\mathcal{T}_i$. A protocol* Gather *implementing the Gather primitive in conjunction with protocol $\mathcal{T}$ amongst $\mathcal{N}$ is defined by two sub primitives -* Gather.Start *and* Gather.Term(.)*. Every honest node n_i starts by invoking* Gather.Start *and terminates by invoking* Gather.Term(G_i)*. G_i is a set of node indices j such that n_i terminated $\mathcal{T}_j$ invoked by n_j. Given that $\mathcal{T}_.$ satisfies Totality, a protocol* Gather *implementing Gather has the following properties.*

- ***Binding Common Core****: Once the first honest node n_i outputs set G_i, out of all possible sets $G'_j \subseteq G_i$ with size $|G'| = n-t$, there exists a unique core set G such that every other honest node n_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 n_j to output a set $G_j \not\supseteq G$.*
- ***Termination****: If every honest node invokes* Gather.Start*, then every honest node n_i will eventually invoke* Gather.Term(G_i) *and output a set G_i: $G \subseteq G_i$.*

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

3.3. Approximate agreement

Finally, we use ϵ-agreement primitive [31].

Definition 3.3. *A protocol $\mathcal{E}$ implementing the ϵ-agreement primitive amongst nodes $\mathcal{N}$ is composed of $\mathcal{E}$.*Start*(.) and $\mathcal{E}$.*Term*(.), where every honest node n_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. A protocol implementing ϵ-agreement satisfies the following properties.*

- ***Termination****: If every honest node invokes $\mathcal{E}$.*Start*(.), then every honest node n_i must eventually invoke $\mathcal{E}$.*Term*(o_i).*
- ***ϵ-agreement****: For a given $\epsilon > 0$, the outputs of any pair of honest nodes n_i and n_j are within ϵ of each other. In other words, $|o_i - o_j| < \epsilon \quad \forall \{n_i, n_j\} \in \mathcal{N}$.*
- ***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 $\mathcal{M}$. In other words, $\min \mathcal{M} \leq o_i \leq \max \mathcal{M}$.*

We use the Binary Approximate Agreement protocol [10] as the ϵ-agreement protocol $\mathcal{E}$ in HashRand. This primitive proposes a more efficient ϵ-agreement protocol under a binary input assumption, where honest nodes' inputs to $\mathcal{E}$ are publicly known binary values. This protocol works using the crusader agreement primitive [1] and proceeds in rounds, where after each round, the range of values of honest nodes reduces by a factor of $\frac{1}{2}$. To reflect this behavior, we also define round-wise start and terminate primitives $\mathcal{E}$.StartRound($m_{r,i}$) and $\mathcal{E}$.EndRound($o_{r,i}$),specific to this protocol. It achieves ϵ-agreement with communication complexity $\mathcal{O}(n^2 \log^2(\frac{\Delta}{\epsilon}))$ bits and takes $\log(\frac{\Delta}{\epsilon})$ rounds to terminate, where $\Delta = \max(\mathcal{M}) - \min(\mathcal{M})$ is the initial range of inputs. The binary input assumption required by this protocol is satisfied in HashRand, where honest nodes input either 0 or 1 to each ϵ-agreement instance.

4. HashRand Design

4.1. Approximate Common Coin

Approximate common coin [30] is a deterministically terminating asynchronous protocol, which enables honest nodes to approximately agree on a random number. In this scheme, every node n_i shares a uniformly drawn secret s_i using an Asynchronous Verifiable Secret Sharing (AVSS) protocol. After instantiating AVSS, honest nodes

run GATHER and wait for its termination. A honest node n_j that terminates GATHER outputs a set of node indices G_j of size $\geq n-t$. This set G_j corresponds to all the AVSS $_i$ instances terminated by n_j. After terminating GATHER, nodes create an ϵ-agreement instance $\mathcal{E}_i$ for every node $i : n_i \in \mathcal{N}$. This step is necessary to enable nodes to approximately agree on a *representative weight* for each participant node. Node n_j inputs 1 to $\mathcal{E}_i$ if $i \in G_j$ and inputs 0 otherwise. After terminating all n $\mathcal{E}_i$: $\forall i \in \{1,\ldots,n\}$ instances with outputs $w_{i,j} \forall i \in \{1,\ldots,n\}$, node n_j initiates the reconstruction phase of AVSS. It waits until reconstructing AVSS $_i$ instances for which $w_{i,j} > 0$. n_j then aggregates the opened secrets s_i using a weighted average to generate o_j.

The described protocol enables nodes to approximately agree on a random number. ϵ-agreement guarantees approximate agreement on $o_.$ values, and liveness is guaranteed by the deterministic termination property of AVSS,Gather, and ϵ-agreement. The binding common core property of Gather guarantees unpredictability. After the first honest node terminates Gather, the binding core set G in Definition 3.2 ensures that every honest node terminates AVSS $_i \forall i \in G$. Therefore, all honest nodes input 1 to the ϵ-agreement instance $\mathcal{E}_i \forall i \in G$. Further, ϵ-agreement's Validity property in Definition 3.3 ensures that $w_{i,j} = 1 \forall i \in G, \forall j \in \{1,\ldots,n\}$. Therefore, G has at least $t+1$ honest nodes, whose secrets will always be reconstructed and added to the beacon.

This approximate common coin can be converted to a Monte-Carlo common coin [30] through rounding, where nodes round off their o_j values to the closest "checkpoint". The ratio of the distance between checkpoints and the degree of approximation in the approximate common coin influences the success probability of the Monte-Carlo coin.

Building a beacon. A strawman approach to building a beacon from this Monte-Carlo coin is to start a new coin instance after terminating the previous coin. However, this approach is impractical with a high latency, computation, and communication cost. Achieving a Monte-Carlo agreement probability of δ requires $\log(\frac{n}{1-\delta})$ rounds of ϵ-agreement. 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. This round complexity results in a high latency in a Wide-Area Network. Moreover, n-parallel AVSS phase in this coin costs $\mathcal{O}(n^2)$ bits of communication and $\mathcal{O}(n^2)$ discrete log exponentiations per node, which also creates a scalability bottleneck. We address these inefficiencies with two optimizations: a) Batching and pipelining to increase the throughput and reducing latency respectively, and b) Committee election for scalability.

4.2. Batching and Pipelining

We first observe that the approximate common coin's properties are unaffected if we replace the AVSS scheme in the coin with an AwVSS scheme with weak commitment. 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 efficient Hash-based AWVSS protocol proposed by Dolev et al. [32] and defined in Section 3.1.

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

1) We replace the conditionally hiding commitments in Dolev et al. with unconditionally hiding commitments based on Backes et al. [9]. 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 without a Random Oracle assumption.

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 Gather and the expensive ϵ-agreement 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 $\mathcal{O}(\beta)$ factor. Further, for a batch size $\beta = \mathcal{O}(n)$ secrets,n-parallel BAWVSS has an amortized sharing comm. complexity of $\mathcal{O}(\lambda n \log(n))$ bits per node, which is an $\mathcal{O}(\frac{\log(n)}{n})$ improvement over AVSS. We defer the full protocol and its security proofs to Appendix A.

Pipelining. We address the high latency of ϵ-agreement in HASHRAND using pipelining and piggybacking. Even though ϵ-agreement has a high round complexity, it still terminates deterministically in a fixed number of rounds. We leverage this deterministic termination property and 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 ϵ-agreement. 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 ϵ-agreement phases. HASHRAND moves to round $r+1$ from round r only after terminating the n-parallel BAWVSS $_{r,i}$ and its corresponding GATHER instance(if r is a multiple of ϕ), and round $r - r'$ of ϵ-agreement instances $\mathcal{E}_{r',j}$ corresponding to BAWVSS $_{r',j} \forall r' \in \{r, r-\phi, \ldots, r-r_t\}, j \in \{1,\ldots,n\}$. r_t is the number of rounds of ϵ-agreement required to achieve a success probability of δ.

This pipelining technique ensures that the entire pipeline is moving at the same pace, hence maintaining a consistent throughput of beacons. 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.

Piggybacking. In rounds $r : r \mod \phi = 0$ with BAwVSS, we piggyback the messages of ϵ-agreement instances on top of BAwVSS messages to maintain the same message complexity. This procedure does not affect the properties of BAwVSS or ϵ-agreement because both internally use all-to-all broadcasts. This piggybacking improves the practical efficiency of HASHRAND by having the same message complexity as the non-pipelined version in the optimistic case.

4.3. Committee election

The asymptotic comm. complexity per beacon for HASHRAND is dominated by the beacon opening phase, where secrets shared by nodes are reconstructed. Reconstructing a secret costs $\mathcal{O}(\lambda n \log(n))$ bits per secret per node. We know that at least $t+1$ honest nodes' secrets will always contribute to the beacon because of Gather's binding common core property. Therefore, the cost of the beacon is $\mathcal{O}(\lambda n^2 \log(n))$ bits per node and $\mathcal{O}(n^3 \log(n))$ overall, to reconstruct $\mathcal{O}(n)$ secrets.

Ensuring the unpredictability of the beacon requires the contribution of only one honest node. However, the Gather 's common core property and the corresponding ϵ-agreement phase 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 committee election procedure. After n-parallel BAwVSS and Gather, honest nodes agree on a constant-sized committee of nodes N and run an ϵ-agreement instance $\mathcal{E}_i \quad \forall i : n_i \in \mathsf{N}$ only for the elected node indices, as opposed to running an $\mathcal{E}_i \quad \forall i : n_i \in \mathcal{N}$.

Committee N requires the presence of only one honest node that was part of the core set G to guarantee unpredictability. Such a committee is called an AnyTrust committee and is much smaller in size than conventional supermajority or honest majority committees traditionally used in down sampled BFT systems [39] and sharding systems [29], respectively. Moreover, the use of committees in HASHRAND is fundamentally different than other committee-based protocols, where nodes in the committee run the protocol amongst themselves. In HASHRAND, all honest nodes participate in the approximate agreement phase, but elect a committee to choose which nodes' secrets to weigh and include in the beacon. Hence, HASHRAND runs only constant ϵ-agreement instances $\mathcal{E}_i$ for indices $i : n_j \in \mathsf{N}$ instead of running $\mathcal{O}(n)$ ϵ-agreement instances for all indices $i : n_i \in \mathcal{N}$.

However, this committee election procedure requires distributed randomness with agreement, where we run into the randomness-agreement circularity again. Moreover, the committee must be randomly elected with a secure beacon every round. Otherwise, the adversary can bias future beacons by preventing the committee members from being part of the Gather 's core set. To address this problem, we generate $\mathcal{O}(n)$ additional beacons for committee election once every ϕn rounds and use these beacons for electing committees for the next n beacon preparation phases. We present more details in Section 4.4.

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 n_i in round r of HASHRAND
$S_{r,i}$	The set of β secrets shared by n_i through round r's BAwVSS instance
N_r	The committee elected for BAwVSS $_{r,.}$
$G_{r,i}$	The output from GATHER.TERM(r,1) for a BAwVSS $_{r,.}$
$\mathcal{E}_{r,i}$	ϵ-agreement instance corresponding to BAwVSS $_{r,i}$ instantiated in HASHRAND round r
B	The list of all prepared beacons yet to be opened
B_c	The list of all prepared beacons for committee election

Calculating committee size. We calculate the size of the committee to be sampled using a hypergeometric probability distribution, which models sampling from a group of nodes without repetition. The committee N must contain at least one honest node whose secret will have a weight $w_{i,.} = 1$ 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 committee. Therefore, the probability of at least one such honest node being on a committee of size c is given as follows.

$$p_c = 1 - \mathcal{H}.cdf(0, n, c, t+1) \tag{2}$$

$\mathcal{H}$ 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 $c = |\mathsf{N}|$ for a given probability p_c with varying n in Fig. 2.

4.4. Protocol description

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

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 using Algorithm 2 as BAwVSS$_{r,i}$.SH(.) (Line 8). Once every ϕn rounds, nodes increase the batch size $\beta' = \beta + n$ to output enough beacons for committee election for the next n BAwVSS instances(Line 8), where each BAwVSS and Gather instance requires an unpredictable random beacon to elect c nodes from $\mathcal{N}$. After instantiating BAwVSS, nodes also call GATHER.START and instantiate round r for each ϵ-agreement instance $\mathcal{E}_{r',*}$ started in the last $r' \in [r - r_t, r]$ rounds(Line 8).

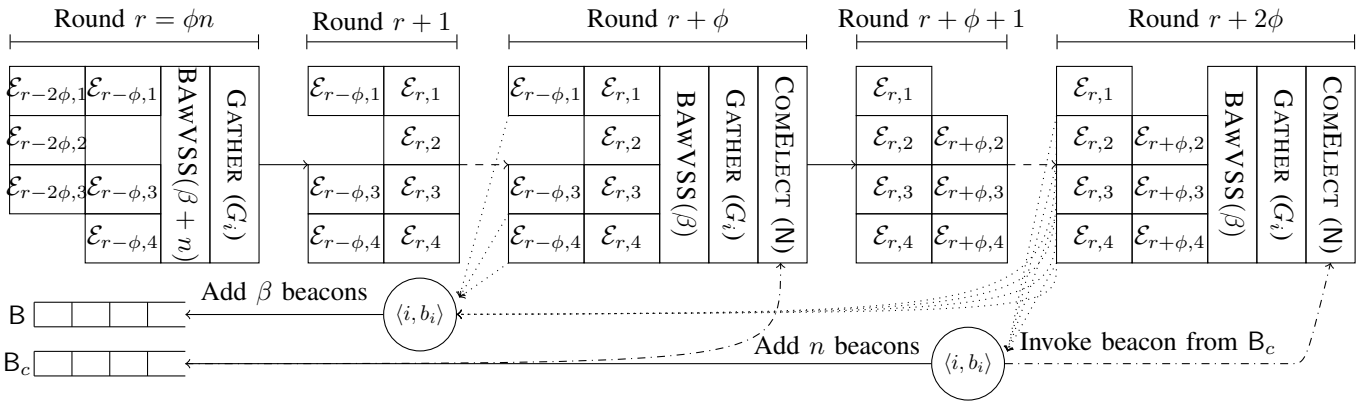

Figure 1: HASHRAND **pipeline**: We describe the pipeline of HASHRAND with batch size β period $\phi = \frac{r_t}{2}$. HASHRAND initiates a new BAWVSS instance once every ϕ rounds and pipelines the corresponding ϵ-agreement instances $\mathcal{E}_{r,i}$. HASHRAND also initiates preparation of n more beacons once every ϕn rounds. These beacons are prepared without committee election, and therefore run $\mathcal{E}_{r,j}$ for all n nodes. 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 electing committees for future beacons in rounds $r : r \mod \phi = 0, r \mod n \neq 0$.

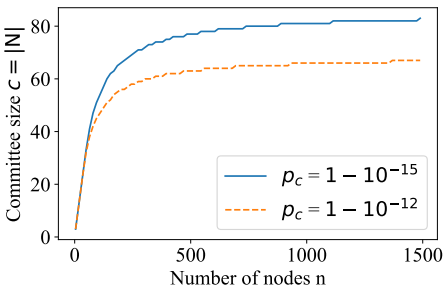

Figure 2: The plot describes the size of the committee $c = |\mathsf{N}|$ with probability of at least one honest node in the core set G being part of the committee N, whose secrets eventually contribute to the beacon. For $p_c = 1 - 10^{-12}$, the committee size caps at $c = 60$ for higher values of n.

After terminating Gather, node i calls GATHER.TERM(I) and outputs set G_i. Next, nodes elect a committee using COMELECT procedure, where they open the beacons prepared for committee election(Line 19). We note that nodes do not elect a committee in rounds where randomness for future committee election is being proposed. Once COMELECT terminates, the nodes instantiate ϵ-agreement $\mathcal{E}_{r,i} \forall i \in \mathsf{N}_r$ only for indices i in the committee N_r, by calling $\mathcal{E}$.START(r,i) (1) if $n_i \in G_i$ and calling $\mathcal{E}$.START(r,i) (0) if $n_i \notin G_i \forall i \in \mathsf{N}_r$(Line 19). Nodes terminate round r only after all ϵ-agreement instances $\mathcal{E}_{r',*} : r' \geq r - r_t$ terminate round r (Line 21). 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 23). These beacons are ready to be opened.

Open phase. Once an honest node n_i gets a request to open a beacon for index k, it identifies the n-parallel BAWVSS instance BAWVSS$_{r',\cdot}$ and the index of the secret in the batch b corresponding to the given index k(Line 36). n_i broadcasts its shares by calling BAWVSS$_{r',j}$.REC(b) $\forall j \in \mathsf{N}_{r'}$. It then waits until it reconstructs all secrets for which the ϵ-agreement instance $\mathcal{E}_{r',j}$ terminated with weight $w_{r',j} > 0$(Line 43). Then, n_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 46).

HASHRAND's preparation phase adds β beacons to the beacon queue B. These β beacons are completely independent and uniformly randomly chosen from the domain $\mathcal{D}$, composing $\log(|\mathcal{D}|)$ bits of entropy. They can be revealed according to the target application.

Verification. Any external client consuming beacons from HASHRAND 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. HASHRAND Analysis

5.1. Security Analysis

We defer the security analysis of BAWVSS scheme to Appendix A. We prove the liveness, correctness, and weak commitment properties using a collision-resistance and preimage resistance properties. We prove the secrecy of BAWVSS using the input hiding assumption of the Hash function H, first mentioned in [15, page 345]. As opposed to Dolev et al.'s [32] AWVSS scheme which has perfect correctness and computational secrecy guarantee in

Algorithm 1 HASHRAND protocol

INPUT: $\mathcal{D}$,δ,β,ϕ ▷ β: Number of secrets in each BAwVSS instance, ϕ: Period of BAwVSS instances

$\mathcal{B}$.PREP(1,.) beacon prepare phase:
$\mathcal{D}'$: Domain of size at least $\lfloor \frac{4}{1-\delta} \rfloor |\mathcal{D}|$
$r \leftarrow 0$; $\epsilon \leftarrow \frac{1}{n\mathcal{D}'}$; $r_t \leftarrow \log(\frac{4n\mathcal{D}}{1-\delta})$
$c : p_c \geq \frac{2+\delta}{3}$ ▷ Calculate committee size from Eq. (2) and Fig. 2
$\mathsf{B} \leftarrow \{\}$; $\mathsf{B}_c \leftarrow \{\}$ ▷ List of prepared beacons; List of prepared beacons for committee election

upon receiving (ID.i, START,r)
▷ Start new BAwVSS instance every ϕ rounds
  if $r \mod \phi = 0$ **then**
  ▷ More beacons for COMELECT every ϕn rounds
    if $r \mod n = 0$, **then** $\beta' \leftarrow \beta + n$
    else $\beta' \leftarrow \beta$
    ▷ Form a batch of β' random numbers
    $S_{r,i}[j] \leftarrow$ Random($\mathcal{D}'$) $\forall j \in \{1, \ldots, \beta'\}$
    ▷ Share secrets using BAwVSS
    Invoke BAwVSS$_{r,i}$.SH($S_{r,i}$)
    Invoke GATHER.START ()
  end if
  ▷ Start next round of ϵ-agreement
  for all $\mathcal{E}_{r',j} \forall r' \in [r - r_t, r] \wedge r \mod \phi = 0, \forall j \in \mathsf{N}_{r'}$ **do**
    $\mathcal{E}_{r',j}$.STARTROUND(r)
  end for
upon invoking GATHER.TERM(G_i) for round r:
  ▷ Upon terminating Gather instantiated in round r, elect a committee and begin ϵ-agreement
  $\mathsf{N} \leftarrow$COMELECT(r)
  $\forall j \in \mathsf{N} : \begin{cases} \mathcal{E}_{r,j}.\text{START}(1), \textbf{if } j \in G_i \\ \mathcal{E}_{r,j}.\text{START}(0), \textbf{otherwise} \end{cases}$
  ▷ Begin ϵ-agreement for BAwVSS in round r
  ▷ Wait until terminating round r for all ϵ-agreement instances in the past r_t rounds
upon invoking $\mathcal{E}_{r',j}$.ENDROUND(r) $\forall r' \in [r - r_t, r] \wedge r \mod \phi = 0, \forall j \in \mathsf{N}_{r'}$:
  ▷ Terminate for ϵ-agreement instances that completed r_t rounds
  if $\exists \mathcal{E}_{r',.} : r' = r - r_t$ **then**
  ▷ Add beacons generated to prepared beacons list
    If $r \mod n \neq 0$ **then** $\mathsf{B} \leftarrow \mathsf{B} \cup \langle r', \langle 0, \beta \rangle \rangle$
    Else $\mathsf{B} \leftarrow \langle r', \langle n, \beta + n \rangle \rangle$; $\mathsf{B}_c \leftarrow \mathsf{B}_c \cup \langle r', \langle 0, n \rangle \rangle$ ▷ Beacons in B_c are for committee election
  end if
  $r \leftarrow r + 1$ and **goto** Line 7 ▷ Increment round

procedure COMELECT(r)
▷ Find closest committee election beacon
  if $r < r_t$ **then**
    return $\{1, \ldots, n - t\}$
  end if
  $m \leftarrow \frac{(r - r_t) \mod \phi n}{\phi}$; $q \leftarrow r - r_t - \phi m$
  $\mathsf{N} \leftarrow c$ indices sampled from $\mathcal{N}$ using randomness generated from $\mathcal{B}$.OPEN(q, $\langle m \rangle$) ▷ Using a Pseudorandom function on the beacon's output
  return N
end procedure

$\mathcal{B}$.OPEN(k) beacon open phase:
upon receiving (ID.i, RECON,k) ▷ Beacon Open phase
$r' \leftarrow \lfloor \frac{k}{\beta} \rfloor \phi$; $b \leftarrow k \mod \beta$
  if $r' \mod n = 0$ **then**
    $b \leftarrow b + n$ ▷ First n are for committee election
  end if
  for $j \in \mathsf{N}_{r'}$ **do**
    $w_{j,i} \leftarrow \mathcal{E}_{r',j}$.TERM() ▷ Output of $\mathcal{E}_{r',j}$
  end for
  $\forall j \in \mathsf{N}_{r'}$: invoke BAwVSS$_{r',j}$.REC(b)
  ▷ Wait until reconstructing only those secrets for which $w_{j,i} > 0$
  $\forall j \in \mathsf{N}_{r'} : x_j \leftarrow \begin{cases} \text{BAwVSS}_{r',j}.\text{REC(b)}.\text{TERM}(), \text{if } w'_j \neq 0 \\ 0, \text{otherwise} \end{cases}$
  ▷ Replace $\perp$ reconstructions with 0
  $\forall j \in \mathsf{N}_{r'} : y_j \leftarrow \begin{cases} 0, \text{if } x_j = \perp \\ x_j, \text{otherwise} \end{cases}$
  $o \leftarrow \left(\sum_{j \in \mathsf{N}} y_j.w_{j,i} \right)$
  invoke $\mathcal{B}$.OPEN(k+1)
  output $\left\langle k, \left\lfloor \frac{o}{\lfloor \frac{4}{1-\delta} \rfloor} \right\rfloor \right\rangle$ and **return**

the RO model, our BAwVSS scheme offers computational correctness with $p = 1 - \mathsf{negl}(\lambda)$ and perfect secrecy under the Input Hiding assumption of Hash functions.

HASHRAND **security.** We start by proving the agreement, liveness, and unpredictability properties for rounds without committee election, initiated every ϕn rounds. Using the properties of beacons generated during these rounds, we prove the properties of beacons initiated in all other rounds.

Theorem 5.1. *Assuming correctness and termination properties of* BAwVSS*, and secure Gather, and ϵ-agreement protocols,* HASHRAND *satisfies liveness for beacon indices $i \in \{0, \ldots, \beta + n\}$.*

Proof. From the termination property of BAwVSS in Definition 3.1, we know that BAwVSS scheme satisfies totality, which ensures that every honest node n_i terminates Gather. Every honest node therefore starts ϵ-agreement instance $\mathcal{E}_{0,i} \forall n_i \in \mathcal{N}$. From the properties of ϵ-agreement in Definition 3.3, if all nodes invoke $\mathcal{E}$.START(i), then every honest node terminates $\mathcal{E}$.START(i). Hence, every honest node terminates $\mathcal{B}$.PREP(0,$\beta + n$). In the opening phase $\mathcal{B}$.OPEN(l) $\forall l \in \{0, \beta + n\}$, every honest node n_j only waits to reconstruct secrets initiated by nodes n_i for which the output of $\mathcal{E}_{0,i}$, $w_{j,i} > 0$. An output $w_{j,i} > 0$ implies that at least one honest node n_k input 1 to $\mathcal{E}_{0,i}$, meaning which n_k terminated n_i's BAwVSS instance. By the termination property of BAwVSS, every honest node must eventually terminate and thereby reconstruct the secret shared by n_i. Therefore, n_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 ϵ-agreement protocols,* HASHRAND *satisfies unpredictability property in Definition 2.1 for beacon indices $i \in \{0, \ldots, \beta + n\}$.*

Proof. Every honest node n_j calculates a weighted average of secrets. Gather's binding core primitive guarantees that at least $t+1$ honest nodes' secrets will always have $w_{i,.} = 1$. This set of nodes is fixed before the first honest node terminating Gather begins ϵ-agreement. On top, no honest node starts $\mathcal{B}$.OPEN(.) until all $\mathcal{E}_{0,i} \forall i \in \{1, \ldots, n\}$ terminate, which forces $\mathcal{A}$ to commit to the secrets that contribute to the beacon before any honest node starts $\mathcal{B}$.OPEN(.). The correctness property of BAwVSS ensures at least $t+1$ honest secrets are always reconstructed and become part of the beacon. From secrecy of BAwVSS, we guarantee unpredictability and bias-resistance for all $\beta + n$ beacons. We note that the unconditional hiding property of BAwVSS ensures perfect unpredictability of the beacon. □

Theorem 5.3. *Assuming secure* BAwVSS*,* GATHER*, and ϵ-agreement,* HASHRAND *described in Algorithm 1 satisfies the agreement property in Definition 2.1 with probability $p \geq \frac{2+\delta}{3} - \mathsf{negl}(\lambda)$ 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 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, and Unpredictability unconditionally and satisfy 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 $r > r_t; r \mod n \neq 0$ requires a committee to be elected from previously agreed upon beacon. From the security properties of first $\beta + n$ beacons in Theorems 5.1 to 5.3, this committee election beacon terminates at all honest nodes, has an agreement probability of $p_c = \frac{2+\delta}{3} - \mathsf{negl}(\lambda)$, and is unpredictable. Therefore, every honest node starts ϵ-agreement instance $\mathcal{E}_{r,i} \forall i \in \mathsf{N}$ with probability p_c, and eventually terminates $\mathcal{E}_{r,i} \forall i \in \mathsf{N}$, guaranteeing liveness with probability p_c.

The elected committee N has at least one honest node n_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 committee election until it terminates Gather, which implies n_j's secret will have weight $w_{j,.} = 1$. Therefore, the output beacon will always the contribution of at least one honest node with probability $p_c p_h = (\frac{2+\delta}{3})^2 - \mathsf{negl}(\lambda) > \delta - \mathsf{negl}(\lambda)$. This guarantees unpredictability.

After electing a committee with probability p_c and having at least one honest node $n_j \in G$ part of the agreed-upon committee N, ϵ-agreement 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. □

Post Quantum security. We translate the proofs listed so far to be secure against a polynomial time quantum adversary. For this reduction, we require the Quantum Random Oracle Model(QROM) [16] assumption to deduce that Grover's algorithm [40], the current best quantum algorithm to find collisions, needs at least $2^{\lambda/3}$ operations to find a collision with probability at least $\frac{1}{2}$. Using this hypothesis, we assume collision and preimage resistance are hard for a polynomial time quantum adversary. These properties of H are enough to prove correctness, and weak commitment of our BAwVSS scheme. Additionally, the unconditional hiding property of the commitment scheme used in BAwVSS directly results from the definition of a Random Oracle, which outputs a randomly chosen string of size $\mathcal{O}(\lambda)$ bits for every new input. As the size of the input is much larger than H's output size, the QROM assumption directly gives unconditional polynomial hiding.

Adaptive security proof sketch. We provide a brief explanation of how HASHRAND is adaptively secure. We first consider the version of HASHRAND without committee election, where every node participates in the ϵ-agreement instance of every other node in the system. Agreement and liveness properties of HashRand under an adaptive adversary directly follow from Theorems 5.1 and 5.3. However, proving unpredictability requires us to establish the existence of a zero-knowledge simulator $\mathcal{S}$ that can simulate the system's functionality without any knowledge of honest nodes' states. The adversary $\mathcal{A}$ interacting with $\mathcal{S}$ must not be able to recognize the difference between the simulated view and the real view with probability $p > 0$.

With an adaptive adversary, the main challenge of simulating such a view has been the explainability problem (also called the commitment problem) [8], [20], where the simulator $\mathcal{S}$ must explain the state of an honest node corrupted by the adversary during the protocol. Suppose the adversary corrupts an honest node i after i broadcasted its commitment vector in BAwVSS. $\mathcal{S}$ must explain the state of i that resulted in the commitment vector with zero knowledge about i's state. In prior protocols where unconditionally binding commitment schemes have been used, this task is hard because the simulator must invert the commitment to produce a consistent state [8].

We avoid this problem in HASHRAND by using an unconditionally hiding commitment scheme, where multiple possible inputs can generate each commitment string. We assume the Hash function H is a programmable Random Oracle (RO) whose outputs for any given input can be programmed by the simulator. When the adversary corrupts

node i, the simulator generates a new set of shares and programs H to return the same commitments generated by i's old shares. Then, the simulator rewinds the tapes of the adversary and other honest nodes to replace i's old shares with the newer shares. A similar technique based on unconditionally hiding commitments was used in Bingo [2] to prove the adaptive security of their AVSS scheme. Additionally, our RO assumption does not involve recording the adversary's queries. Hence, we also avoid the problem of recording a quantum adversary's queries and achieve post-quantum security in the Quantum RO model [16].

The adaptive security of HashRand with committee election also follows from the unpredictability, agreement, and liveness of the beacons used for electing committees. This is because all parties participate in the ϵ-agreement instances of committee members, which ensures deterministic termination and agreement amongst honest nodes.

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 $\mathcal{O}(\beta n \log(n) + \lambda n^2)$ bits of communication per node, with each node sending β merkle proofs and RBCing a vector of β commitments. With Cachin-Tessaro's RBC [17], this complexity becomes $\mathcal{O}(\beta\lambda n \log(n) + \lambda n^2 \log(n))$ per node for each n-parallel BAwVSS. The Gather primitive requires $\mathcal{O}(n^2)$ comm. bits per node. The ϵ-agreement phase for a round that does not use committee election ($r \mod n = 0$) requires n BINAA instances, with each round of one instance costing $\mathcal{O}(n \log(\frac{\mathcal{D}}{1-\delta}))$ bits per node. Overall, for $\mathcal{O}(n)$ instances running for $\log(\frac{n\mathcal{D}}{1-\delta})$ rounds costs $\mathcal{O}(n^2 \log(n) \log^2(\frac{\mathcal{D}}{1-\delta}))$ bits per node. For every beacon, the opening phase requires every node to broadcast secret shares of $\mathcal{O}(n)$ secrets with evaluation proofs to all other nodes, which gives this phase a communication complexity of $\mathcal{O}(\lambda n^2 \log(n))$ bits per node per beacon. Therefore, for β beacons, the overall comm. complexity per node without committee election is $\mathcal{O}(\beta\lambda n \log(n) + \lambda n^2 \log(n) + n^2 \log(n) \log^2(\frac{\mathcal{D}}{1-\delta}) + \beta\lambda n^2 \log(n)) = \mathcal{O}(\beta\lambda n^2 \log(n))$ per node for $\beta = \mathcal{O}(n)$. Hence, the asymptotic complexity per beacon is dominated by the opening phase.

With committee election, each batch of β beacons requires honest nodes to open a previous beacon with $\mathcal{O}(\lambda n^2 \log(n))$ per node. The nodes run only c ϵ-agreement instances,and require opening only c secrets. This results in a communication complexity of $\mathcal{O}(\beta\lambda n \log(n) + \lambda n^2 \log(n) + cn \log(n) \log^2(\frac{\mathcal{D}}{1-\delta}) + \beta\lambda cn \log(n))$ per a β beacons. For $\beta = \mathcal{O}(n)$, the communication complexity per node comes to $\mathcal{O}(\lambda n \log(n))$ bits. As beacon generation for committee election happens once every ϕn rounds, the average comm. complexity per beacon totals $\frac{\lambda n^2 \log(n)}{n} + (1 - \frac{1}{n})\lambda n \log(n) = \mathcal{O}(\lambda n \log(n))$ 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 $\mathcal{O}(n^2 \log(n))$ hash computations to verify $\mathcal{O}(n)$ secret shares each for $\mathcal{O}(n)$ secrets part of the beacon. With committee election, each node only computes $\mathcal{O}(cn \log(n))$ hashes for at most c secrets part of the beacon. Since beacons for committee election are only generated once every ϕn rounds, the average is $\mathcal{O}(cn \log(n))$ hash computations per beacon.

6. Applications

We present two asynchronous SMR protocol instantiations with HASHRAND providing common coins for liveness in asynchrony. We present a) PQ-TUSK, a high-throughput PQ-secure SMR protocol that uses digital signatures and a PKI setup, and b) DAG-RIDER with HASHRAND, a PKI-free SMR protocol for asynchronous cross-chain consensus.

6.1. 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 [50] and FIN [34] 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 [14] for liveness. Prior to HASHRAND, the PQ-secure random beacon protocol with the best communication complexity is Freitas et al. [30] with Dolev et al.'s [32] AWVSS. With this beacon, the communication complexity of PQ-secure asynchronous SMR is $\mathcal{O}(n^2 \log(n))$ bits per block per node.

We present PQ-TUSK, an efficient post-quantum secure asynchronous SMR protocol built on top of TUSK [24]. Our choice of TUSK is motivated by its exceptional performance in a Wide Area Network. 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. Overall, PQ-TUSK has a communication complexity of $\mathcal{O}(\lambda n \log(n))$ bits per block per node, which is an $\mathcal{O}(n)$ factor better than the current best PQ-secure asynchronous SMR protocol, and only an $\mathcal{O}(\log(n))$ factor higher than TUSK.

6.2. Cross-chain Consensus

We also use HASHRAND to create the first asynchronous cross-chain consensus protocol, which does not require a PKI setup. A cross-chain consensus protocol is a key building block in creating a combined ledger with boosted

trust from individual blockchains. Such a combined ledger is secure even when the adversary controls 1/3rd fraction of blockchains. Recent work TrustBoost [65] 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 and Tendermint. 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.

Using HASHRAND, we propose a signature and PKI-free asynchronous cross-chain consensus protocol. We use DAG-RIDER [50], a DAG-based atomic broadcast protocol in combination with HASHRAND, to create a PKI-free asynchronous SMR protocol. DAG-RIDER uses only Bracha's RBC and a common coin as building blocks, which, with HASHRAND, enables the protocol to commit transactions without using any signatures. This SMR protocol has a communication complexity of $\mathcal{O}(\lambda n^2 \log(n))$ bits per transaction, which is $\mathcal{O}(\log(n)/n)$ factor more efficient than IT-HotStuff in the worst case, and only $\mathcal{O}(\log(n))$ 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 [46] based on BLS threshold signatures [14]. We also evaluate PQ-TUSK with HASHRAND providing common coins.

Implementation. We implement HASHRAND in Rust with the `tokio` library as our asynchronous runtime[2]. We use `SHA256` as our hash function H. We use Cachin-Tessaro's [17] computationally efficient Reliable Broadcast protocol in HASHRAND and use the Erasure Codes library[3]. We then integrate HASHRAND with TUSK 's [24] codebase and use HASHRAND 's beacons to elect wave leaders in TUSK[4]. Further, we implement PQ-TUSK by replacing the quantum-insecure `EdDSA` signatures in TUSK with PQ-secure `DiLithium` Signatures from the pqcrypto[5] 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 on Amazon Web Services (AWS) with varying nodes $n = 16, 40, 64, 112$, and 160. We run our protocol on `c5.large` nodes, each with 2 cores and 4GB RAM. As opposed to `t3a.medium` machines (also with 2 cores and 4GB RAM) used in many prior works, `c5.large` machines do not have burstable CPU credits, which ensures predictable and reproducible performance. We also evaluate PQ-TUSK on the same testbed with $n = 16, 40, 64$ nodes.

Network. 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 [46], which uses BLS threshold signatures [14]. Prior works such as Spurt [25] and OptRand [12] directly used DRand [63] as a benchmark for a beacon with threshold setup. However, we note that DRand's codebase does not produce beacons at full throttle 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`[6] on the `bls12-381` curve, which internally uses the `blst`[7] 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 [25] numbers. We ensure a fair comparison by matching Spurt's geo distribution of nodes and benchmarking HASHRAND in machines with the same number of CPUs and memory as Spurt. We also benchmark the performance of PQ-TUSK with PQ-security against a non PQ-secure TUSK [24].

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 full throttle.

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.

2. https://github.com/akhilsb/hashrand-rs
3. https://github.com/rust-rse/reed-solomon-erasure
4. https://github.com/akhilsb/pqsmr-rs
5. https://github.com/rustpq/pqcrypto
6. https://github.com/filecoin-project/blstrs
7. https://github.com/supranational/blst

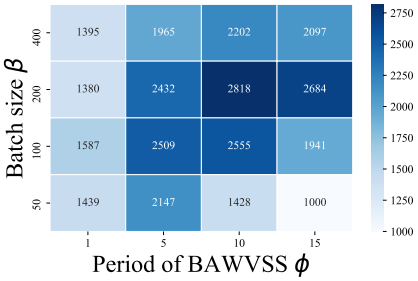

Figure 3: β **and** ϕ **vs Beacons/min** The figure shows the number of beacons emitted by HASHRAND per minute for $n = 40$ nodes with configuration parameters batch size β in each BAWVSS instance and the number of BAWVSS instances. For every value of n, there exists a sweet spot of β and ϕ that gives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

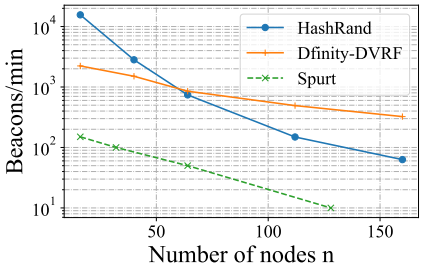

Figure 4: **Scalability results**: The figure shows the number of beacons produced per minute by HASHRAND, Dfinity-DVRF [46], and the numbers reported by Spurt [25] in the same geo-distributed setting. HASHRAND's $\mathcal{O}(n^2 \log(n))$ Hash comp. cost results in a steeper slope than Dfinity's $\mathcal{O}(n)$ comp. 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 $n = 160$ without a threshold setup, which is 4x higher than Spurt at $n = 128$.

Similarly, at a lower period, the nodes saturate their network bandwidth by consecutively engaging in 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. The same reasoning also applies for column-based analysis, where for a given ϕ, there is an optimal β that gives maximum throughput of beacons. For $n = 40$ nodes, HASHRAND produces a maximum throughput of 2818 beacons pm at $(\beta, \phi) = (200, 10)$.

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 81% more beacons than Dfinity. However, HASHRAND scales as $\mathcal{O}(n^2 \log(n))$ 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. We also observe the constant committee size in HASHRAND and $\mathcal{O}(cn \log(n))$ complexity taking effect for higher n, where the slope of HASHRAND decreases with increasing n. Due to this improvement, HASHRAND outputs 63 beacons pm or 1 beacon every second at $n = 160$ nodes without a threshold setup.

HASHRAND vastly outperforms Spurt at all values of n. Particularly, at $n = 128$, Spurt outputs only 15 beacons pm, which is 4x lower than HASHRAND at $n = 160$ 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 $\mathcal{O}(n^2 \log(n))$ computations per beacon as compared to Spurt's $\mathcal{O}(n^2)$ discrete log 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. We consider three PQ-secure signature schemes for use in PQ-TUSK: a) `DiLithium` based on Lattice cryptography [35], b) `SPHINCS-256` [11] based on the Hash-based Winternitz One-Time Signatures(WOTS), and c) `PICNIC` [22] 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 $\mathcal{O}(n)$ signing and $\mathcal{O}(n^2)$ verification complexity per wave per node. Nodes in TUSK also broadcast $\mathcal{O}(n^2)$ signatures per wave. `PICNIC` and `SPHINCS-256` 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. Hence, we choose the `DiLithium` scheme to implement certificates in PQ-TUSK.

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 [69]

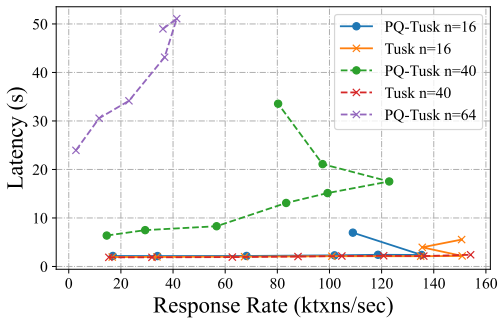

Figure 5: **Post-Quantum SMR**: The figure describes the latency-response rate curve for PQ-TUSK, and TUSK. PQ-TUSK has a response rate of 135k txns per second at a latency of 2.3 seconds for $n = 16$, and 122k txns per second at a latency of 17.5 seconds for $n = 40$ nodes.

that uses local coins to generate distributed randomness. At $n = 16$ in a geo-distributed setting, WaterBear offers a peak response rate of only 23k txns per second in `t3a.medium` machines(with same specs as our machines, but with burstable CPU) as compared to 135k for PQ-TUSK.

At higher $n = 40$ and $n = 64$, PQ-TUSK offers a peak response rate of 122k txns 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 instead of `EdDSA`. This variant of TUSK underperforms compared to our PQ-TUSK at $n = 16$ with 20% lesser response rate at 108k txns per second. At $n = 40$, it performs very similar to PQ-TUSK, with a peak response rate of 122k txns per second and a latency of 13 seconds. Based on this information, 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 that has plagued prior PQ-SMR protocols like WaterBear, and establish that randomness from Hash functions is the most practical way to achieve Post-Quantum security.

8. Related work

We segregate various randomness protocols based on their network and setup assumptions. We use Abraham and Yanai's [6] description of different levels of trusted setups to describe setup assumptions of related beacon protocols. We also describe the cryptographic hardness assumptions used in these protocols.

Synchronous and partially synchronous protocols. Beacon protocols in the synchronous world consists of protocols [12], [13], [21], [25], [59], [64] that use VSS driven by CRS or SRS setup(Level 4 in [6]). All these protocols require some variant of Byzantine Agreement to achieve the properties of a random beacon. We discuss RandPiper [13], OptRand [12], and Spurt [25] in this category, as they are the most relevant.

BRandPiper [13] is an adaptively secure, leader-based synchronous beacon protocol that has a communication complexity of $\mathcal{O}(\kappa n^3)$, which uses a VSS scheme with unconditionally hiding homomorphic commitments. OptRand [12] is an optimistically responsive random beacon that uses OptSync's [62] commit rule to generate beacons at network speed. The protocol however depends on a synchronous timeout to implicate faulty leaders in the pessimistic case. 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 $\mathcal{O}(n^2)$ communication. The secrecy of the PVSS scheme depends on the DBDH assumption, which along with an RO used for NIZK proofs in the protocol, provides unpredictability of the beacon's output. Spurt internally uses the HotStuff [67] 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 [30] impossibility result, which states that any protocol that converts local randomness from honest nodes into global randomness with agreement amongst honest nodes must have infinitely many executions. Protocols with a DKG setup(Level 5 in [6]) 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. Another protocol requiring a DKG setup is RandShare [64], which uses AVSS and Asynchronous Byzantine Agreement (ABA) to achieve agreement and unpredictability. The ABA instances consume pseudorandomness from the DKG setup to overcome the bound.

Randomness Protocols with Las-Vegas style agreement have been traditionally employed in Asynchronous Distributed Key Generation (ADKG) [2], [3], [26], [28], [37], [52]. Kogias et al. [52] 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 to prove the unpredictability of their coin. This protocol also has a $\mathcal{O}(n^3)$ comp. 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). Gao et al. [37] use a combination of PAVSS, and the binding core abstraction in GATHER to produce beacons. This work proves unpredictability using the DDH and the RO assumption. Abraham et al. [3] also 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 [2] produces adaptively secure randomness using a Packed AVSS scheme, which requires a PKI and an SRS setup. Gao et al., Abraham et al., and Bingo have $\mathcal{O}(n^3)$ DLog expo. 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 [18], [47], [60], [68] and refreshable secret sharing [43], respectively. Moreover, Kogias et al. 's approach is the only current way to achieve ADKG at a Level 2 setup(without PKI or CRS) in [6], whose expensiveness hinders its practical application.

Freitas et al. 's [30] protocol achieves deterministic termination with a statistical probability of disagreement with a Level 2 setup (pairwise secure channels). This protocol uses AVSS, GATHER, and ϵ-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 $\mathcal{O}(n^3)$ per node and a high round complexity of $\log(\frac{n}{1-\delta})$ rounds.

Information-theoretic protocols. Protocols in the realm of IT and statistical security use different techniques to generate distributed randomness. Huang et al. [48] use statistical fraud detection by Byzantine nodes to improve the complexity of Ben-Or's local coin-based coin flipping to $\Omega(n^6)$ from $\mathcal{O}(\exp(n))$, at the expense of tolerating a lesser fraction of faults. This form of work uses only authenticated channels. Another style of work by Patra et al. [57] propose a statistically secure AVSS scheme with $\mathcal{O}(n^4 \log(n))$ comm. complexity. This work combined with packed AVSS and Freitas et al.'s Monte-Carlo coin technique results in an overall communication complexity of $\mathcal{O}(n^4 \log(n))$ per beacon.

9. 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 $\mathcal{O}(\lambda n \log(n))$ bits of communication per node and is computationally efficient with amortized $\mathcal{O}(n \log(n))$ hash computations per beacon. We proved HASHRAND 's security using standard properties of one-way Hash functions against a classical adversary and using the Quantum ROM against a quantum adversary. HASHRAND provides post-quantum liveness for asynchronous SMR protocols with only $\mathcal{O}(\log(n))$ more communication than threshold signatures. We also demonstrated HASHRAND 's scalability and utility in making post-quantum SMR practical.

References

[1] Ittai Abraham, Naama Ben-David, and Sravya Yandamuri. Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 381–391. ACM, 2022.

[2] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation. Cryptology ePrint Archive, Paper 2022/1759, 2022. https://eprint.iacr.org/2022/1759.

[3] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 363–373, 2021.

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

[5] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder – scalable, robust anonymous committed broadcast. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1233–1252, New York, NY, USA, 2020. Association for Computing Machinery.

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

[7] Ittai Abraham and Gilad Stern. Information theoretic hotstuff. In Quentin Bramas, Rotem Oshman, and Paolo Romano, editors, *24th International Conference on Principles of Distributed Systems, OPODIS 2020, December 14-16, 2020, Strasbourg, France (Virtual Conference)*, volume 184 of *LIPIcs*, pages 11:1–11:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[8] Renas Bacho and Julian Loss. On the adaptive security of the threshold bls signature scheme. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 193–207, New York, NY, USA, 2022. Association for Computing Machinery.

[9] Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In *Advances in Cryptology–ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings 17*, pages 590–609. Springer, 2011.

[10] Akhil Sai Bandarupalli. Efficient $o(n^2)$ byzantine fault-tolerant asynchronous approximate agreement, 03-30-2023, 2023. https://akhilsb.github.io/posts/2023/3/bp3/.

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

[12] Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. Optrand: Optimistically responsive reconfigurable distributed randomness. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023.

[13] Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. Randpiper: Reconfiguration-friendly. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 3502–3524, New York, NY, USA, 2021. Association for Computing Machinery.

[14] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In Yvo G. Desmedt, editor, *Public Key Cryptography — PKC 2003*, pages 31–46, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[15] Dan Boneh and Victor Shoup. A graduate course in applied cryptography v0.6: https://crypto.stanford.edu/ dabo/pubs/abstracts/bookshoup.html, https://toc.cryptobook.us/, 2023.

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

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

[18] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 88–97, 2002.

[19] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: Practical asynchronous byzantine agreement using cryptography (extended abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, page 123–132, New York, NY, USA, 2000. Association for Computing Machinery.

[20] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. On adaptive vs. non-adaptive security of multiparty protocols. In *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20*, pages 262–279. Springer, 2001.

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

[22] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 1825–1842, 2017.

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

[24] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: A dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 34–50, New York, NY, USA, 2022. Association for Computing Machinery.

[25] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. Spurt: Scalable distributed randomness beacon with transparent setup. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2502–2517, 2022.

[26] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling, 2023. https://eprint.iacr.org/2022/1389.

[27] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 2705–2721, New York, NY, USA, 2021. Association for Computing Machinery.

[28] Sourav Das, Tom Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. Cryptology ePrint Archive, Report 2021/1591, 2021. https://ia.cr/2021/1591.

[29] Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 683–696, 2022.

[30] Luciano Freitas de Souza, Petr Kuznetsov, and Andrei Tonkikh. Distributed randomness from approximate agreement. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*, volume 246 of *LIPIcs*, pages 24:1–24:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

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

[32] Shlomi Dolev and Ziyu Wang. 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, Virtual Event, November 17–20, 2021, Proceedings*, page 510–515, Berlin, Heidelberg, 2021. Springer-Verlag.

[33] Sisi Duan, Michael K. Reiter, and Haibin Zhang. Beat: Asynchronous bft made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 2028–2041, New York, NY, USA, 2018. Association for Computing Machinery.

[34] Sisi Duan, Xin Wang, and Haibin Zhang. Practical signature-free asynchronous common subset in constant time. Cryptology ePrint Archive, Paper 2023/154, 2023. https://eprint.iacr.org/2023/154.

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

[36] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo-ng: Fast asynchronous BFT consensus with throughput-oblivious latency. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1187–1201. ACM, 2022.

[37] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Efficient asynchronous byzantine agreement without private setups. In *42nd IEEE International Conference on Distributed Computing Systems, ICDCS 2022, Bologna, Italy, July 10-13, 2022*, pages 246–257. IEEE, 2022.

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

[39] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

[40] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[41] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Speeding dumbo: Pushing asynchronous BFT closer to practice. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022.

[42] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. *Dumbo: Faster Asynchronous BFT Protocols*, page 803–818. Association for Computing Machinery, New York, NY, USA, 2020.

[43] Christoph U. Günther, Sourav Das, and Lefteris Kokoris-Kogias. Practical asynchronous proactive secret sharing and key refresh. Cryptology ePrint Archive, Paper 2022/1586, 2022. https://eprint.iacr.org/2022/1586.

[44] Mads Haahr. random. org: Introduction to randomness and random numbers. *Statistics,(June)*, pages 1–4, 1999.

[45] Runchao Han, Haoyu Lin, and Jiangshan Yu. Randchain: A scalable and fair decentralised randomness beacon. Cryptology ePrint Archive, Paper 2020/1033, 2020. https://eprint.iacr.org/2020/1033.

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

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

[48] Shang-En Huang, Seth Pettie, and Leqi Zhu. Byzantine agreement with optimal resilience via statistical fraud detection. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 4335–4353. SIAM, 2023.

[49] Aniket Kate, Yizhou Huang, and Ian Goldberg. Distributed key generation in the wild. *IACR Cryptol. ePrint Arch.*, 2012:377, 2012.

[50] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 165–175, New York, NY, USA, 2021. Association for Computing Machinery.

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

[52] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. Cryptology ePrint Archive, Report 2019/1015, 2019. https://ia.cr/2019/1015.

[53] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 887–903, New York, NY, USA, 2019. Association for Computing Machinery.

[54] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. Cryptology ePrint Archive, Report 2020/842, 2020. https://ia.cr/2020/842.

[55] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42, 2016.

[56] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. 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):1–21, 2015.

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

[58] Sambhav Satija, Apurv Mehra, Sudheesh Singanamalla, Karan Grover, Muthian Sivathanu, Nishanth Chandran, Divya Gupta, and Satya Lokam. Blockene: A high-throughput blockchain over mobile devices. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 567–582, 2020.

[59] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. Hydrand: Efficient continuous distributed randomness. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 73–89, 2020.

[60] David A Schultz, Barbara Liskov, and Moses Liskov. Mobile proactive secret sharing. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 458–458, 2008.

[61] Victor Shoup and Nigel P. Smart. Lightweight asynchronous verifiable secret sharing with optimal resilience. Cryptology ePrint Archive, Paper 2023/536, 2023. https://eprint.iacr.org/2023/536.

[62] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the optimality of optimistic responsiveness. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 839–857, 2020.

[63] Open source contributors. Drand - a distributed randomness beacon daemon - https://github.com/drand/drand, 2019.

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

[65] Xuechao Wang, Peiyao Sheng, Sreeram Kannan, Kartik Nayak, and Pramod Viswanath. Trustboost: Boosting trust among interoperable blockchains. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, CCS '23, page 1–15, 2023. https://eprint.iacr.org/2022/1428.

[66] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, page 179–192, USA, 2012. USENIX Association.

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

[68] Thomas Yurek, Zhuolun Xiang, Yu Xia, and Andrew Miller. Long live the honey badger: Robust asynchronous dpss and its applications. 2023. https://eprint.iacr.org/2022/971.

[69] Haibin Zhang, Sisi Duan, Boxin Zhao, and Liehuang Zhu. Waterbear: Practical asynchronous bft matching security guarantees of partially synchronous bft. In *USENIX Security Symposium'23*, pages 1–16, 2023. https://eprint.iacr.org/2022/021.

Appendix

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 [32] scheme.

Lemma A.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 n_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(n_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(.).

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

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

2: Protocol BAwVSS$_d$.SH(S_d):
3: **upon** receiving (ID.d,in,SHARE,S_d): ▷ As a dealer
▷ Sample $|S_d|$ nonces from a domain $\mathcal{D}'$ at least twice the H's output range $R_d[i] \leftarrow Random(\mathcal{D}')\forall i \in \{1,\ldots,|S_d|\}$
▷ Use Shamir's SS scheme to create shares for secrets in S_d, $R_d[i]$
4: $\mathcal{S}_d[i] \leftarrow$SHAMIR.SPLIT($S_d$ [i],n,t) $\forall i \in \{1,\ldots,|S_d|\}$
5: $\mathcal{R}_d[i] \leftarrow$SHAMIR.SPLIT($R_d[i]$,n,$t$) $\forall i \in \{1,\ldots,|S_d|\}$
▷ Compute Commitments
6: $\mathsf{C}_d[i] \leftarrow \langle H(\mathcal{R}_d[i][0], \mathcal{S}_d[i][0]),\ldots,H(\mathcal{R}_d[n][0], \mathcal{S}_d[i][n])\rangle \forall i \in \{1,\ldots,|S_d|\}$
▷ Compute Merkle Trees
7: $M_d[i] =$ MERKLE($\mathsf{C}_d[i]$) $\forall i \in \{1,\ldots,|S_d|\}$
8: $\mathsf{R}_d \leftarrow \{M_d[0].\text{ROOT},\ldots,M_d[k].\text{ROOT}\}$
9: $\mathsf{S}_d[j][i] \leftarrow \langle \mathcal{S}_d[i][j], \mathcal{R}_d[i][j], M_d[i].\text{PROOF}(j)\rangle \forall i \in \{1,\ldots,|S_d|\}, \forall j \in \{1,\ldots,n\}$
▷ Send shares to all nodes
10: **for** $j \in \{1,\ldots,n\}$ **do**
11: Send "ID.d, SHARE, $\mathsf{S}_d[j]$" to node n_j
12: **end for**
▷ Reliably Broadcast root vector
13: Invoke RBC.BROADCAST(R_d)
▷ As a participant
14: **upon** receiving "ID.d,SHARE,$\mathsf{S}_d[j]$" and RBC.INIT(R_d) from dealer n_d:
▷ Verify commitments and participate in RBC if all pass
15: **if** MERKLE.VERIFY($\mathsf{R}_d[i]$,$\mathsf{S}_d[j][i]$.SHARE,$\mathsf{S}_d[j][i]$.PROOF) $= 1 \forall i \in \{1,\ldots,|S_d|\}$ **then**
16: Send ECHO messages in n_d's RBC protocol RBC
17: **end if**
18: **upon** invoking RBC.DELIVER(R_d) with share:
19: **output** (ID.d,out,SHARE,R_d)
20: **upon** invoking RBC.DELIVER(R_d) with no share:
21: **output** (ID.d,out,NO-SHARE,R_d)

1: Protocol BAwVSS$_d$.REC(i):
2: $SS_d[i] \leftarrow \emptyset$
3: **upon** receiving (ID.d,in,RECON,i):
4: **if** terminated with SHARE **then**
5: **for** $j \in \{1,\ldots,n\}$ **do**
6: Send "ID.d, RECON,$\mathsf{S}_j[i]$" to node n_j
7: **end for**
8: **end if**
9: **upon** receiving (ID.d,RECON,$\mathsf{S}_d[m][i]$) from node n_m:
10: **if** MERKLE.VERIFY($\mathsf{R}_d[i]$,$\mathsf{S}_d[m][i]$.SHARE,$\mathsf{S}_d[m][i]$.PROOF) $= 1$ **then**
11: $SS_d[i] \leftarrow SS_d[i] \cup \langle m, \mathsf{S}_d[m][i]\rangle$
12: **end if**
upon$|SS_j[i]| \geq t+1$
▷ Combine shares and reverify root vector commitment
13: $\mathcal{S}_d[i], \mathcal{R}_d[i] \leftarrow$ SHAMIR.COMBINE($SS_d[i]$,n,t)
14: $\mathsf{C}_d[i] \leftarrow \langle H(\mathcal{R}_d[i][0], \mathcal{S}_d[i][0]),\ldots,H(\mathcal{R}_d[i][n], \mathcal{S}_d[i][n])\rangle \forall i \in \{1,\ldots,|S_d|\}$
15: **if** MERKLE($\mathsf{C}_d[i]$).ROOT = R_i **then**
16: **output** (ID.d,out,RECONSTRUCTED,$\mathcal{S}_d[i][0]$,i)
17: **else**
18: **output** (ID.d,out,RECONSTRUCTED,$\perp$,i)
19: **end if**

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(n_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(n_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.

□

Next, we prove the unconditional hiding property of our BAwVSS scheme using the Input Hiding assumption of a Hash function. The same assumption was also used in Backes et al. 's [9] computational AVSS scheme.

Lemma A.2. *Under the input hiding assumption of the hash function H as stated in Boneh and Shoup [15, page 345],*

the BAwVSS *protocol* BAwVSS *in Algorithm 2 satisfies unconditional hiding.*

Proof. We use Boneh and Shoup's [15, page 345] Input Hiding assumption of collision-resistant Hash functions to prove this lemma. This assumption states that a commitment computed as $H(R, x)$, where R is a random variable at least twice the size of H's output, statistically hides x. This implies a distribution $H(R, x_1)$ is statistically indistinguishable from $H(R, x_2)$ for all $x_1, x_2 \in \mathcal{D}$, thereby providing unconditional hiding. □