

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

Akhil Bandarupalli
Purdue University
West Lafayette, IN, USA
abandaru@purdue.edu

Adithya Bhat
Visa Research
Foster City, CA, USA
haxolotl.research@gmail.com

Saurabh Bagchi
Purdue University
West Lafayette, IN, USA
sbgachi@purdue.edu

Aniket Kate
Purdue University / Supra Research
West Lafayette, IN, USA
aniket@purdue.edu

Michael K. Reiter
Duke University / Chainlink Labs
Durham, NC, USA
michael.reiter@duke.edu

Abstract

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

CCS Concepts

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

Keywords

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

ACM Reference Format:

Akhil Bandarupalli, Adithya Bhat, Saurabh Bagchi, Aniket Kate, and Michael K. Reiter. 2024. Random Beacons in Monte Carlo: Efficient Asynchronous Random Beacon *without* Threshold Cryptography. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*, October 14–18, 2024, Salt Lake City, UT, USA. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3658644.3670326>

1 Introduction

Random beacon schemes [56, 68] emit random numbers at regular intervals. They are often used as a source of secure randomness in applications such as committee election in sharded and Proof-of-Stake blockchains [33, 44, 47, 63]; common coins in asynchronous Byzantine Agreement (BA) [3, 6, 58, 60]; State Machine Replication (SMR) [27, 41, 46, 47, 55, 59, 73]; asynchronous Multi-Party Computation (MPC) [25, 66]; and also in layer-2 blockchain applications in the DeFi and GameFi space [21]. The demand for secure randomness from these applications has recently surged and continues to grow further [21].

Contemporary random beacon protocols make a wide variety of trusted setup assumptions. Beacons from sources such as Random.org [49] or NIST's random beacon project [56] place complete trust in the source. The sources here become single points of failure for the system's liveness and safety as well as unpredictability and bias-resistance of beacon values. Approaches like DRand [68], Dfinity-DVRF [51], and Cachin et al. [17] mitigate these risks by distributing the trust across some n nodes such that the system remains safe, live, and secure as long as a subset of $t + 1$ or more nodes are honest.

Under the hood, these approaches employ a threshold cryptographic setup among n nodes that enables them to generate (n, t) -threshold unique signatures, which offer unpredictability and bias-resistance against an adversary corrupting $t < \frac{n}{3}$ nodes ($t < \frac{n}{2}$ in a synchronous network). Here, the trusted setup phase for threshold signatures can be replaced with a Distributed Key Generation (DKG) protocol [43, 54, 57]. However, DKG protocols are expensive in computation and communication and must be re-executed every time nodes' participation changes. This constraint makes



This work is licensed under a Creative Commons Attribution International 4.0 License.

threshold signatures costly for applications with churn, such as Proof-of-Stake blockchains [44].

As a result, a few recent random beacon proposals [13, 14, 28, 50, 64, 69] avoid the threshold setup and the threshold unique signature approach. Instead, they employ a Public Key Infrastructure (PKI) setup or broadcast setup, where digital signatures and secure channels can be used to realize Verifiable Secret Sharing (VSS) and State Machine Replication (SMR) primitives and build a beacon service on top of those. However, they depend on some form of network synchrony (bounded/partial) to achieve liveness and agreement amongst participant nodes, and cannot be used in the asynchronous communication setting. Indeed, as demonstrated by Miller et al. [59], synchronous protocols cannot take full advantage of the network speed and out-of-order message delivery, thereby offering subpar performance under active adversarial scheduling.

Asynchronous randomness generation protocols that do not use threshold signatures are available: Some of those use a PKI setup [4, 5, 29, 31, 42], while other need a pairwise-channels setup [34, 57]. However, all of these protocols have a high computational cost due to discrete-log-based commitments used in their n -parallel Asynchronous VSS (AVSS) phases. For example, in protocols with PKI setup like Abraham et al. [5] and Bingo [4], the computational cost of n -parallel AVSS amounts to $\Omega(n^2)$ discrete log exponentiations and $O(n^2)$ bilinear pairing computations per node per beacon. As an exponentiation and a pairing respectively consume $100\times$ and $1000\times$ more compute time and energy than a cryptographic hash, there is significant scope for improvement. In protocols with only secure channels such as Kogias et al. [57] and Freitas et al. [34], the cost of n -parallel AVSS increases to $O(n^3)$ discrete log exponentiations per beacon per node. Overall, high computational cost becomes a scalability bottleneck for the existing asynchronous randomness generation protocols as n grows.

Asynchronous beacon protocols in the realm of Information-Theoretic and Statistical security utilize computation-efficient techniques that do not use any expensive cryptographic operations. For example, Huang et al. [53] use a technique called statistical fraud detection to detect misbehavior by Byzantine-faulty nodes. This approach has a communication complexity of $\Omega(n^6)$ bits per beacon. Although such approaches are computationally efficient, their high communication complexity hinders their practical use.

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

Our solution. We present HASHRAND, a computation-efficient asynchronous random beacon protocol that uses hash functions and a secure channel setup to output random beacons. HASHRAND has an amortized $O(\lambda n \log(n))$ communication complexity per beacon per node, where λ is the range of the hash function in bits, and requires amortized $O(\lambda n \log(n))$ Hash computations per beacon per node. HASHRAND requires only a pairwise secure channel setup, where nodes can send private, authenticated messages to each other. HASHRAND is computationally efficient because it only uses lightweight cryptographic primitives like hash functions and authenticated symmetric encryption. On top of computational efficiency and practical communication complexity, HASHRAND is also

post-quantum secure assuming that cryptographic hash functions like SHA256 behave like a Random Oracle against a polynomial-time quantum adversary [16]. We offer a detailed comparison of HASHRAND with related works in Table 1.

Evaluation. We implement HASHRAND in Rust and evaluate it in a geo-distributed setting on AWS by measuring the throughput in terms of the number of beacons output per minute. We also compare HASHRAND with Dfinity-DVRF [51] based on BLS threshold signatures and Spurt [28], which require a trusted DKG setup and a PKI setup, respectively. Dfinity-DVRF's computation and communication complexities are equivalent to Cachin et al. [17]. At $n = 40$ nodes, HASHRAND outputs 1837 beacons per minute, against 1109 beacons pm for Dfinity-DVRF and 100 beacons pm for Spurt. The computational efficiency of Hash functions allows HASHRAND to produce beacons at a higher throughput at lower n .

With increasing n , HASHRAND's throughput drops faster than Dfinity-DVRF, with HASHRAND doing worse than Dfinity-DVRF at $n = 64$. This is because Dfinity-DVRF requires $O(n)$ discrete-log exponentiations per beacon whereas HASHRAND scales as $n^2 \log(n)$ hash computations per beacon at $n < 64$ (Refer to Table 1 for details). However, recall that Dfinity-DVRF is not post-quantum secure and needs DKG for a threshold signature setup, which is expensive for systems with churn. On the other hand, HASHRAND continues to scale much better than threshold-setup-free Spurt [28], outputting $5\times$ the throughput of Spurt at $n = 136$ nodes. Hence, HASHRAND is the optimal choice in situations where threshold setup is infeasible.

An illustrative application. On top of implementing and evaluating HASHRAND as an asynchronous beacon, we demonstrate its practical utility by realizing a post-quantum secure asynchronous SMR protocol. We integrate HASHRAND with Tusk [27], 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 [39], a post-quantum secure public key signature scheme. PQ-Tusk has an optimal communication complexity of $O(\lambda n)$ bits per transaction per node, which is much more practical than other post-quantum SMR protocols like WaterBear [73] with worst case $O(\exp(n))$ communication complexity. We also demonstrate the practicality of PQ-SMR by evaluating our SMR protocol PQ-Tusk in a geo-distributed setting. PQ-Tusk achieves a throughput of 135k transactions per second for $n = 16$ nodes with a latency of 2.3 seconds, compared to 150k transactions per second at a latency of 2.1 seconds for Tusk. We also propose a signature-free asynchronous SMR protocol based on DAG-Rider [55] for cross-chain consensus [70], an application where signatures are expensive and infeasible.

2 Solution Overview

2.1 System Model

We assume a set of n nodes \mathcal{N} connected by pairwise authenticated and secure channels. We consider a polynomial-time quantum adaptive adversary \mathcal{A} defined according to Boneh et al. [16], who has access to a quantum computer and controls $t < \frac{n}{3}$ nodes. The channels connecting honest nodes are private and \mathcal{A} cannot distinguish the contents of each message from a uniformly random bitstring within polynomial time. An efficient post-quantum symmetric encryption scheme like AES can satisfy this assumption.

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

| Protocol | Network | Post Quantum Security | Termination Flavor | Adaptive Security | Communication Complexity (per node) | Computational Complexity (per node) | Setup | Cryptographic Assumption |
|---------------------|---------------|-----------------------|--------------------|-------------------|-------------------------------------|-------------------------------------|-----------------|--------------------------|
| Cachin et al. [17] | async. | ✗ | D | ✓ | $O(\lambda n)$ | $O(n)$ DLE | DKG | pRO & CDH |
| RandShare [69] | async. | ✗ | LV | ✗ | $O(\lambda n^3)$ | $O(n^3)$ DLE | DKG | DLog |
| Spurt [28] | partial sync. | ✗ | D | ✗ | $O(\lambda n^2)$ | $O(n^2)$ DLE [¶] | CRS & PKI | pRO & DBDH |
| Kogias et al. [57] | async. | ✗ | LV | ✗ | $O(\lambda n^3)$ | $O(n^3)$ DLE | Secure channels | pRO & DDH |
| Das et al. [31] | async. | ✗ | LV | ✗ | $O(\lambda n^2)$ | $O(n^3)$ DLE | PKI | pRO & DDH |
| Gao et al. [42] | async. | ✗ | LV | ✗ | $O(\lambda n^2)$ | $\Omega(n^2)$ DLE | PKI | pRO & SXDH |
| Abraham et al. [5] | async. | ✗ | LV | ✗ | $O(\lambda n^2)$ | $\Omega(n^2)$ DLE | PKI | pRO & SXDH |
| Bingo [4] | async. | ✗ | LV | ✓ | $O(\lambda n^2)$ | $\Omega(n^2)$ DLE | SRS & PKI | q -SDH |
| Freitas et al. [34] | async. | ✓ | MC | ✗ | $O(\lambda n^2 \log(n))$ | $O(n^3)$ DLE | Secure channels | DLog |
| Huang et al. [53] | async. | ✓ | LV | ✓ | $\Omega(n^6)$ | 0 | Auth channels | None |
| HASHRAND | async. | ✓ | MC | ✓ | $O(\lambda n \log(n))^\dagger$ | $O(n \log(n))$ H [*] | Secure channels | pRO [§] |

Termination Flavor: Monte-Carlo (MC) style protocols terminate in a deterministic number of rounds. But, even after termination, honest nodes can disagree on the beacon with a practically negligible probability $p = 1 - \delta$. A lower p implies a higher round complexity. Las Vegas (LV) style protocols terminate only upon agreement amongst honest nodes on the beacon value. These protocols can have infinitely many executions and we report the expected runtime complexities of these protocols. Deterministic (D) protocols offer agreement and a deterministic runtime. **Computational complexity:** Concurrent beacon and coin protocols use discrete log exponentiations (denoted by DLE), which is the main reason for their high computational cost. HASHRAND instead uses $n^2 \log(n)$ Hash computations ($100\times$ cheaper than DLog expo and $1000\times$ cheaper than a pairing computation) at low n and $O(n \log(n))$ Hashes at high n per beacon per node (denoted by H). [†]**AnyTrust Sampling:** HASHRAND has a per-node communication of $O(\lambda c n \log(n))$ bits per beacon. HASHRAND randomly samples a set of nodes C of size c and reconstructs secrets shared only by those nodes. We set the size of $c = |C|$ to ensure at least one honest node is present in C with high probability. However, we note c is a small constant ($c < 70$ for $n = 1024$) at higher values of n , because of which we do not include it in our communication cost. [§]**Hash function:** HASHRAND assumes a Hash function to be a programmable Random Oracle (pRO) assumption to prove security against an adaptive and quantum adversary. The techniques in recent works [65, 66] can be used to replace the pRO in HASHRAND with a collision-resistant and linear hiding Hash function. [¶]The leader performs $O(n^2)$ operations every round, while other nodes perform $O(n)$ operations each.

The network is asynchronous, where \mathcal{A} can arbitrarily delay and reorder messages. However, it cannot drop messages between two honest nodes.

2.2 Asynchronous Random Beacons

We define a random beacon protocol and its properties based on Spurt's [28] definition and adapt it to asynchrony. Honest nodes participating in a beacon protocol \mathcal{B} output tuples of the form $\langle i, b_i \rangle$, where $i \in \mathbb{N}$ is a unique beacon index and $b_i \in \mathcal{D}$ is an element from a predefined domain of numbers \mathcal{D} .

Definition 2.1. A protocol \mathcal{B} amongst n nodes \mathcal{N} consists of two subprotocols $\mathcal{B}.\text{PREP}(\cdot, \cdot)$ and $\mathcal{B}.\text{OPEN}(\cdot)$, with outputs of the form $\langle x, b_x \rangle : x \in \mathbb{N}, b_x \in \mathcal{D}$. An honest node $i \in \mathcal{N}$ invokes $\mathcal{B}.\text{PREP}(x, y) : x, y \in \mathbb{N}, x \leq y$ to prepare beacons from index x to y . Upon terminating the preparation phase $\mathcal{B}.\text{PREP}(x, y)$ until index y , and outputting $\langle x, b_x \rangle \forall x \in [1, y]$, an honest node i invokes $\mathcal{B}.\text{OPEN}(y)$ to initiate generating $\langle y, b_y \rangle$. Such a protocol \mathcal{B} is a random beacon protocol if and only if it satisfies the following properties.

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

protocol \mathcal{B} outputs a value.

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

Prior random beacon protocols [13, 14, 28] 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 [34].

2.3 HASHRAND Key Insights

Conventional random beacon protocols (without the threshold setup) consist of two building blocks: VSS and consensus [14, 28, 64, 69]. In these protocols, each node uses VSS to share a uniformly random element from a given domain. Next, the nodes use some version of consensus to agree on a set of $t + 1$ nodes whose VSS instances have been successfully terminated by every node in the system. Finally, the nodes reconstruct the secrets shared by these $t + 1$ nodes and aggregate them to output the beacon. This template guarantees the contribution of at least one honest node to the beacon, which ensures its unpredictability. The liveness and agreement guarantees of such beacon protocols come from SMR's liveness and safety properties, respectively.

This design paradigm cannot be applied to an asynchronous beacon. This is because asynchronous SMR requires random beacons to terminate because of the FLP impossibility result [40]. This creates a circular dependency between the beacon and SMR in asynchrony [28]. A recent impossibility result by Freitas et al. [34] throws light on methods to resolve this circularity. It states that it

is impossible for a deterministic asynchronous protocol to achieve agreement, deterministic liveness, and unpredictability. Hence, an asynchronous random beacon protocol must choose between the Monte-Carlo and Las Vegas flavors of termination.¹

We propose **HASHRAND**, a Monte-Carlo style beacon protocol with deterministic termination and a fixed probability of successful execution denoted by δ . Mainly, **HASHRAND** breaks the beacon-SMR circularity in the conventional design paradigm by replacing the use of asynchronous SMR with the approximate agreement (AA) primitive [35]. An AA protocol allows nodes to agree on values within a configured ϵ distance. Further, AA protocols offer deterministic termination and do not require access to random beacons. However, employing AA in **HASHRAND** requires us to trade off a small probability of agreement failure, where nodes output different beacons. This is because the approximation in the agreement property of AA induces a small, yet finite probability that nodes output different beacons.

HASHRAND overview. **HASHRAND** uses n -parallel AVSS [19], and approximate agreement [35] as building blocks. Prior AVSS protocols using discrete-log cryptography are communication efficient but have a high computational complexity, which is a performance bottleneck [30]. Although other AVSS protocols using Hash functions are computationally efficient [9], they have an $O(n)$ order increase in communication compared to discrete-log based AVSS schemes, which makes them practically inefficient. In **HASHRAND**, we overcome two challenges related to Hash-based AVSS using two key observations that enable **HASHRAND** to be computationally efficient with a practical communication complexity.

(1) We address the high communication cost of hash-based AVSS by making the following observation. AVSS's commitment property requires honest nodes to always reconstruct a valid element in the domain, even when the dealer is faulty. This commitment property is the major reason for prior AVSS protocols being communication intensive [9]. However, we note that this strong property is overkill for a random beacon protocol. As we show (Section 5.1), a weaker commitment property, where nodes can reconstruct a value \perp designating an invalid sharing conducted by a malicious dealer, is enough to guarantee all the desired properties of a beacon. Using this weakened commitment property, we build a Batched Asynchronous weak VSS (BAwVSS) on top of Dolev et al.'s [37] AwVSS. For a batch size of $O(n)$, this BAwVSS scheme has an amortized sharing complexity of $O(\lambda \log(n))$ bits per node.

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

¹Monte-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.

Table 2: Symbols and notations used in **HASHRAND**

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

that are a part of this set, which brings down the communication complexity of the reconstruction phase to $O(\lambda n^2 \log(n))$ bits or $O(\lambda n \log(n))$ bits per beacon per node.

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

Security. **HASHRAND** is a Monte-Carlo beacon protocol with success probability $p \leq 1 - \delta + \text{negl}(\lambda)$. This corresponds to λ bits of cryptographic security and $\log(\frac{1}{1-\delta})$ bits of statistical security against a quantum adversary. We prove **HASHRAND**'s properties against a polynomial-time quantum adaptive adversary \mathcal{A} under the assumption that the hash function H is a Random Oracle. We first prove **HASHRAND** is adaptively secure against a classical adversary and extend our proof to a quantum adversary using Boneh et al.'s [16] template for proving post-quantum security.

3 Building Blocks

3.1 Asynchronous weak Verifiable Secret Sharing (AwVSS)

We describe the Asynchronous weak VSS primitive as defined by Dolev et al. [30].

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

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

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

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

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

We build on top of Dolev et al.'s [37] hash-based AwVSS protocol. In this protocol, a dealer d uses Shamir secret sharing to create secret shares for a secret s_d in domain \mathcal{D} . \mathcal{D} can either be a ring or a finite field. The dealer d then creates a Merkle tree on top of the n secret shares. d then sends individual shares along with Merkle proofs to nodes and reliably broadcasts the Merkle root r . An honest node participates in the reliable broadcast of the root only upon verifying its Merkle proof. During the reconstruction phase, nodes that terminated with a share, broadcast their shares and Merkle proofs. Upon receiving $t + 1$ shares, every node reconstructs the entire share vector and the Merkle tree and verifies if the root broadcasted in the sharing phase r matches the reconstructed root r' . It outputs the shared secret if $r = r'$ and outputs \perp otherwise.

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

3.2 Gather

We also employ the Gather primitive [5].

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

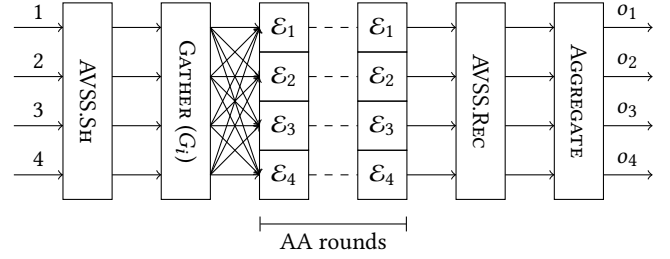


Figure 1: Approximate Common Coin: The structure of Freitas et al.'s [34] protocol with $n = 4$ nodes. Each node shares a random secret using AVSS, invokes Gather, and participates in n AA instances to agree on a weight for each node. Then nodes reconstruct and aggregate secrets to output o_i .

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

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

3.3 Approximate agreement

We also use Approximate Agreement [36] (AA) primitive.

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

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

We use the Binary Approximate Agreement protocol [10] as the Approximate Agreement (AA) protocol \mathcal{E} in HASHRAND . This protocol works under a binary input assumption, where honest nodes' inputs to \mathcal{E} are publicly known binary values, for example, 0 and 1. This protocol proceeds in rounds, where after each round, the range of honest nodes reduces by a factor of $\frac{1}{2}$. A prior version of the protocol required $O(n^2 \log^2(\frac{1}{\epsilon}))$ bits of communication and $\log(\frac{1}{\epsilon})$ rounds for achieving approximate agreement. However, this communication can be improved to $O(n^2 \log(\frac{1}{\epsilon}) \log \log(\frac{1}{\epsilon}))$ bits using FIFO broadcast [1]. In this protocol [10], instead of broadcasting the entire value, a node broadcasts a constant-sized update

of how its value changed in the previous round. Therefore, a node can construct another node's value in a round by applying updates over the value broadcast in the first round.

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

3.4 Approximate Common Coin

We utilize the approximate common coin [34] primitive.

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

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

We describe an overview of Freitas et al. [34] in Fig. 1.

4 Design

We build HASHRAND from the Approximate Common Coin primitive. A strawman approach for building a beacon from this primitive is to start a brand new coin instance after terminating and reconstructing the previous coin. This approach has high computation, communication, and latency costs because of two main reasons. First, the n -parallel $\text{AVSS}.\text{SH}$ and $\text{AVSS}.\text{REC}$ phases cost $\mathcal{O}(n^3)$ discrete log exponentiations per node if using discrete-log-based AVSS [30]. If we instead use Backes et al.'s [9] Hash-based AVSS, each beacon incurs $\mathcal{O}(n^3)$ communication per node. Second, the AA phase has a high round complexity. Achieving a Monte-Carlo agreement probability of δ requires $\log(\frac{n}{1-\delta})$ rounds of AA. For example, generating a single beacon with $\delta = 1 - 2^{-40}$, equivalent to one expected failure in a trillion beacons, requires 200 round trips of communication at $n = 40$. We address these problems with two optimizations: (a) Batching and pipelining with Hash-based AwVSS for increasing throughput and reducing latency, and (b) AnyTrust sampling for reducing communication complexity.

4.1 Batching and Pipelining

AwVSS's weak commitment property enables every honest node to agree on the maliciousness of the dealer during the reconstruction phase and discard the contributions of these nodes. This observation enables us to use the AwVSS protocol proposed by Dolev et al. [37] and defined in Section 3.1. Directly using this AwVSS protocol in the strawman approach reduces the complexity of n -parallel AwVSS by a $\mathcal{O}(n)$ factor to $\mathcal{O}(\lambda n^2 \log(n))$ bits per node.

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

(1) We replace the computationally hiding commitments in Dolev et al. with unconditionally hiding commitments based on Backes et al. [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 i th secret share, and builds a Merkle tree on this commitment vector. This change allows us to prove the security of HASHRAND against an adaptive adversary.

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

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

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

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

4.2 AnyTrust Sampling

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

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

The sampled set \mathcal{C} requires the presence of only one honest node that was part of the core set G to guarantee unpredictability. This condition is called the AnyTrust condition, which is satisfied at

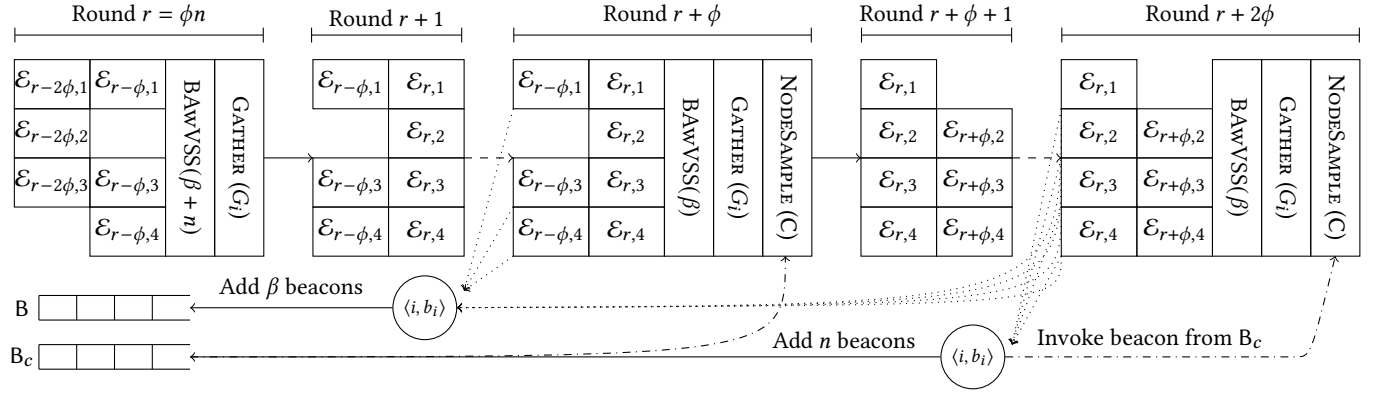


Figure 2: HASHRAND pipeline: We describe the pipeline of HASHRAND for $n = 4$ with batch size β period $\phi = \frac{r_t}{2}$. HASHRAND initiates a new BAWVSS instance once every ϕ rounds and pipelines the corresponding AA instances $\mathcal{E}_{r,i}$. HASHRAND also initiates preparation of n more beacons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

much smaller sizes of sampled sets than conventional supermajority or honest majority committees traditionally used in down sampled BFT systems [44, 48], and sharding systems [33], respectively. Hence, with AnyTrust sampling, HASHRAND needs to reconstruct secrets from at most $c = |C|$ nodes.

However, this sampling procedure requires distributed randomness with agreement, where we run into the randomness-agreement circularity again. Moreover, the set must be randomly sampled with a secure beacon for every new n -parallel BAWVSS instantiation. Otherwise, the adversary can bias future beacons by preventing the sampled members from being part of the Gather's core set. To address this problem, we generate cn additional beacons once every ϕn rounds and use these beacons for sampling for the next n beacon preparation phases. We calculate the size of the sample set C based on the hypergeometric distribution.

4.3 Protocol description

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

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

After terminating Gather, node i calls $\text{GATHER.TERM}(t)$ and outputs set G_i . Next, nodes sample the set C using NODESAMPLE

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

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

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

5 Analysis

We analyze HASHRAND in this section. We start by proving HASHRAND's security against an adversary corrupting at most $\frac{1}{3}$ faults. Then, we present the communication and computation complexity of HASHRAND. We give a rough proof sketch for brevity and refer the reader to the full version [11] for exhaustive proofs.

Algorithm 1 HASHRAND protocol

```

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

```

```

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

```

```

► GATHER phase
13: upon invoking GATHER.TERM( $G_i$ ) for round  $r$ :
14:   $C \leftarrow \text{NODESAMPLE}(r)$ 
15:   $\forall j \in C : \begin{cases} \mathcal{E}_{r,j}.\text{START}(1), & \text{if } j \in G_i \\ \mathcal{E}_{r,j}.\text{START}(0), & \text{otherwise} \end{cases}$  // Start AAA

```

```

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

```

```

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

```

5.1 Security Analysis

We assume that the Hash function H is a Random Oracle (or ROM) for our proofs. We prove the liveness, correctness, and weak commitment properties using collision-resistance and preimage-resistance properties offered by H under the ROM. We prove the secrecy of BAwVSS against a PPT adaptive adversary \mathcal{A} . Our BAwVSS scheme offers computational correctness with $p = 1 - \text{negl}(\lambda)$ and perfect secrecy in the ROM. We defer the adaptive security analysis of our BAwVSS scheme to the full version of the paper [11].

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

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

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

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

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

Further, we also show that \mathcal{A} cannot bias a beacon $\langle i, b_i \rangle$ after the first honest node invokes $\mathcal{B}.\text{OPEN}(i)$. An honest node j invokes $\mathcal{B}.\text{OPEN}(k) : k \in \{1, \dots, \beta + n\}$ until all $\mathcal{E}_{0,i} \forall i \in \{1, \dots, n\}$ terminate at j . From the ϵ -agreement property of AA, we know that $|w_{j,i} - w'_{j,i}| \leq \epsilon \forall j, j' \in \mathcal{N}_{\text{hon}}$ and $\forall i \in \mathcal{N}$. Therefore, the maximum difference between two honest nodes' weighted sums $|o_j - o_i| \leq n\epsilon D' \leq 1, \forall i, j \in \mathcal{N}_{\text{hon}}$. Except for the cases where an

honest weighted sum o_i is on the decision boundary of the rounding operation (which occurs with probability δ and agreement is already violated), \mathcal{A} cannot bias the beacon's value after an honest node j starts reconstruction. \square

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

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

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 $i \in \{0, \dots, \beta + n\}$ satisfy Liveness, Unpredictability, and Agreement with probability $p \geq \frac{2+\delta}{3} - \text{negl}(\lambda)$, HASHRAND described in Algorithm 1 will output beacons $(i, b_i) \forall i > 0$ that satisfy the properties in Definition 2.1 with probability $p \geq \delta - \text{negl}(\lambda)$.*

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

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

Sampling with an adaptive adversary. We show that HASHRAND with AnyTrust sampling is adaptively secure under the secure erasure model [24, 32], where nodes can securely erase specified portions of their random execution tapes. In the BAWVSS phase, every honest node erases the sampled secrets and corresponding secret shares of nodes from its tape simultaneously while sending out the SHARE messages.

We utilize secure erasures to prove unpredictability. Consider BAWVSS and GATHER instantiated in round r for which honest nodes use AnyTrust sampling. An honest node i invokes $\mathcal{B}.\text{OPEN}(j)$ for sampling for round r only after terminating GATHER of round r (Line 14). From the unpredictability property of the beacons used for sampling, we know that \mathcal{A} cannot distinguish the sample members from uniformly random until i terminates GATHER. Further, from the properties of GATHER, the sampled node set C_r must contain at least one honest node j part of the core set G_r with probability $p_h = \frac{2+\delta}{3}$. Moreover, as node i already terminated j 's BAWVSS instance, j would have already erased all data about its shared secret. Therefore, if \mathcal{A} corrupts j after knowing its membership

in the committee, it would only get j 's share of the secret. \mathcal{A} can corrupt at most t nodes and access t shares, which provides no information about j 's secret and thereby ensures unpredictability.

In the no-erasures model of adaptive security [24], where the adversary gets access to the entire tape of a corrupted node, the above argument for adaptive security will not hold. The adversary can adaptively corrupt all committee members, gain access to the secrets shared by them, and know the beacons beforehand, which breaks unpredictability. HASHRAND needs to be run without AnyTrust sampling in this model. The per-node communication complexity in this model increases to $O(\lambda n^2 \log(n))$ bits per beacon.

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

Post Quantum security. We translate the proofs of BAWVSS and HASHRAND to be secure against a polynomial time quantum adversary. For this reduction, we require the Quantum Random Oracle Model (QROM) [16] to deduce that Grover's algorithm [45], the current best quantum algorithm to find collisions, needs at least $2^{\lambda/3}$ operations to find a collision with probability at least $\frac{1}{2}$. We also employ the history-free reduction technique proposed by Boneh et al. [16] to prove that HASHRAND is secure against a polynomial time quantum adversary. This technique states that a proof in the classical RO model can be translated to a proof in the quantum RO model, provided the proof simulates a RO in a *history-free* fashion. This implies that the RO's responses to a given query must not depend on its previous responses and query history. Our proofs (both static and adaptive) satisfy this condition because we do not program the RO based on prior query history. Hence, we also avoid the problem of recording a quantum adversary's queries and achieve post-quantum security in the Quantum RO model [16].

5.2 Complexity Analysis

We analyze the communication and computation complexity of HASHRAND. The n -parallel BAWVSS phase in round $r : r \bmod \phi = 0$ with β secrets costs $O(\beta n \log(n) + \lambda n^2)$ bits of communication per node, with each node sending β Merkle proofs and reliably broadcasting a vector of β commitments. With Cachin-Tessaro's RBC [18], this complexity becomes $O(\beta \lambda n \log(n) + \lambda n^2 \log(n))$ per node for each n -parallel BAWVSS. The Gather primitive requires $O(n^2)$ bits per node. The AA phase for a round that does not use AnyTrust sampling ($r \bmod n = 0$) requires n BINAA instances, with each round of one instance costing $O(n \log \log(\frac{D}{1-\delta}))$ bits per node. Overall, $O(n)$ BINAA instances running for $\log(\frac{nD}{1-\delta})$ rounds costs $O(n^2 \log(n) \log(\frac{D}{1-\delta}) \log \log(\frac{D}{1-\delta}))$ bits per node. For every beacon, the opening phase requires every node to broadcast secret shares of $O(n)$ secrets with evaluation proofs to all other nodes, which gives this phase a communication complexity of $O(\lambda n^2 \log(n))$ bits per node per beacon. Therefore, for β beacons, the overall communication per node without sampling is $O(\beta \lambda n \log(n) + \lambda n^2 \log(n) + n^2 \log(n) \log(\frac{D}{1-\delta}) \log \log(\frac{D}{1-\delta}) +$

$\beta\lambda n^2 \log(n) = O(\beta\lambda n^2 \log(n))$ bits per node for $\beta = O(n)$. The asymptotic communication complexity per beacon is dominated by the opening phase.

With AnyTrust sampling, each batch of β beacons requires honest nodes to open a beacon in B_C with $O(\lambda n^2 \log(n))$ per node. Then, the nodes run only $c = |C_r|$ AA instances and require opening only c secrets. This results in a communication complexity of $O(\beta\lambda n \log(n) + \lambda n^2 \log(n) + cn \log(\frac{D}{1-\delta}) \log \log(\frac{D}{1-\delta}) + \beta\lambda cn \log(n))$ per β beacons. For $\beta = O(n)$, the communication complexity per node comes to $O(\lambda n \log(n))$ bits. As beacon generation for sampling happens once every ϕn rounds, the average communication complexity per beacon totals $\frac{\lambda n^2 \log(n)}{n} + (1 - \frac{1}{n})\lambda n \log(n) = O(\lambda n \log(n))$ bits per beacon per node. The computation complexity can also be calculated in a similar fashion. For rounds $r : r \bmod n = 0$, each node performs $O(n^2 \log(n))$ hash computations to verify $O(n)$ secret shares each for $O(n)$ secrets part of the beacon. With sampling, each node computes $O(cn \log(n))$ hashes for at most c secrets part of the beacon, and the average is $O(n \log(n))$ hashes per beacon.

6 Applications

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

Post Quantum SMR. We demonstrate the utility of `HASHRAND` by implementing a Post-Quantum asynchronous SMR protocol using `HASHRAND` for Post-Quantum Liveness. There exist asynchronous SMR protocols like `DAG-RIDER` [55] and `FIN` [38] 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 [15] for liveness. Prior to `HASHRAND`, the PQ-secure random beacon protocol with the best communication complexity is Freitas et al. [34] with Dolev et al.'s [37] `AwVSS`. With this beacon, the communication complexity of PQ-secure asynchronous SMR is $O(n^2 \log(n))$ bits per block per node.

We present `PQ-Tusk`, an efficient post-quantum secure asynchronous SMR protocol built on top of `Tusk` [27]. Our choice of `Tusk` is motivated by its exceptional performance in a WAN. We create `PQ-Tusk` by using `HASHRAND` to provide the distributed randomness necessary for electing wave leaders. We tune `HASHRAND`'s configuration parameters β and ϕ to ensure that `HASHRAND` prepares exactly one beacon per wave. Since revealing a prepared beacon in `HASHRAND` takes only one round trip in `HASHRAND`, `PQ-Tusk` can commit blocks at network speed. However, `Tusk` uses quantum-insecure EdDSA signatures for certificate generation. We replace this scheme with an appropriate PQ-secure signature scheme that offers maximum performance benefits in this context. Using PQ-secure signatures in `Tusk` does not induce any *asymptotic* overheads. However, PQ-secure signatures do induce a computation and communication overhead over traditional EdDSA signatures. We refer to Section 7.2 for a detailed discussion on PQ-secure signatures for SMR. Overall, `PQ-Tusk` has a communication complexity of $O(\lambda n \log(n))$ bits per block per node, which is an $O(n)$ factor better than the current best PQ-secure asynchronous SMR protocol, and only an $O(\log(n))$ factor higher than PQ-insecure `Tusk`.

Cross chain consensus. We also use `HASHRAND` to create the first asynchronous cross-chain consensus protocol without a PKI setup. A cross-chain consensus protocol is a key building block in creating a combined ledger with boosted trust from individual blockchains. Recent work `TrustBoost` [70] 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` [71]. Further, `TrustBoost` also notes a limitation in the underlying IBC protocol, which prevents the use of public-key signatures in cross-chain consensus. These issues motivated `TrustBoost` to use Information-theoretic `HotStuff` (IT-HS) [7], which does not use any signatures. Moreover, these issues are also the main roadblock in running asynchronous cross-chain consensus. We propose a signature and PKI-free asynchronous cross-chain consensus protocol. We use `DAG-RIDER` [55], a DAG-based atomic broadcast protocol with `HASHRAND`, to create a PKI-free asynchronous SMR protocol. This SMR protocol has a communication complexity of $O(\lambda n^2 \log(n))$ bits per transaction, which is $O(\log(n)/n)$ factor more efficient than IT-`HotStuff` in the worst case, and only $O(\log(n))$ factor higher than IT-HS under a stable leader.

7 Evaluation

We describe the evaluation of `HASHRAND` in this section. We evaluate `HASHRAND` in a geo-distributed setting and compare it with `Dfinity-DVRF` [51] based on BLS threshold signatures [15]. We also evaluate `PQ-Tusk` with `HASHRAND` providing common coins.

Implementation. We implement `HASHRAND` in Rust with the `tokio` library as our asynchronous runtime. We use SHA256 as our hash function H . We use Cachin-Tessaro's [18] computationally efficient Reliable Broadcast protocol in `HASHRAND` and use the Erasure Codes library. We then integrate `HASHRAND` with `Tusk`'s [27] codebase and use `HASHRAND`'s beacons to elect wave leaders in `Tusk`. Further, we implement `PQ-Tusk` by replacing the quantum-insecure EdDSA signatures in `Tusk` with PQ-secure `DiLiThium` Signatures from the `pqcrypto` library.

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

Baselines. We compare `HASHRAND` with an asynchronous beacon protocol based on `Dfinity-DVRF` [51], which uses BLS threshold signatures [15]. Prior works such as `Spurt` [28] and `OptRand` [13] directly used `DRand` [68] as a benchmark for a beacon with threshold setup. However, we note that `DRand`'s codebase does not produce beacons at network speed and thereby misrepresents the true pace

of beacon generation with a threshold setup. To ensure a fair comparison, we implement the Dfinity-DVRF protocol using the Rust-based BLS signature library `blstron` on the `bls12-381` curve, which internally uses the `blstpairing` 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 [28] stated numbers. We ensure a fair comparison by running `HASHRAND` using the same geo distribution of nodes and configuration of machines as in Spurt. We compare the throughput of beacons produced by each protocol using the metric beacons per minute. We also benchmark the performance of `PQ-Tusk` with PQ-security against a non PQ-secure `Tusk` [27].

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 `B.OPEN(i+1)` immediately after they terminate `B.OPEN(i)` to generate beacons at network speed.

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

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

We observe the AnyTrust sampling procedure taking effect with increasing n . At small n , the set size c grows with n , resulting in a computation and communication complexity of $O(n^2 \log(n))$. However, with increasing n , c becomes constant and the complexities

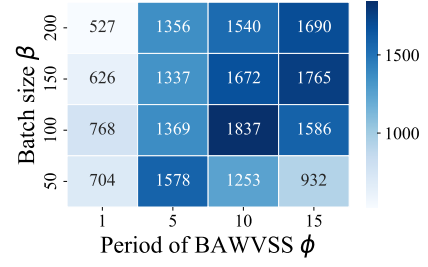


Figure 3: The rate of beacon generation (/min) vs β and ϕ . The figure shows the number of beacons emitted by `HASHRAND` per minute for $n = 40$ nodes with configuration parameters batch size β in each BAWVSS instance and the number of BAWVSS instances. For every value of n , there exists a sweet spot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

grow as $O(n \log(n))$. In Fig. 4, the rate of decrease of throughput of `HASHRAND` decreases with increasing n , where at higher n , `HASHRAND` scales more gracefully. Due to this improvement, `HASHRAND` outputs 78 beacons pm at $n = 136$ nodes without a threshold setup.

Hardware-accelerated Hash functions. The performance can further be improved using efficient hash functions accelerated by hardware. For example, cryptographic hash functions built from fixed-key block ciphers like AES [62] can be accelerated using the AES-NI hardware instruction set provided by many concurrent processors. These hash computations offer 20 \times speed up over conventional SHA256 computations. With such specialized hash functions, we expect `HASHRAND` to outperform Dfinity at higher values of n .

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

7.2 SMR Evaluation

We also evaluate `PQ-Tusk` and compare it to classical `Tusk` without PQ-security. We test both protocols by offering increasing transaction loads and measuring the response rate and latency for that request rate. We then plot the response rate vs latency in Fig. 5 and check the value of response rate for which the latency has a disproportionate jump. Each transaction is of size 256 bytes. **PQ-secure signatures for SMR.** We consider three PQ-secure signature schemes for use in `PQ-Tusk`: a) DiLiThium based on Lattice cryptography [39], b) SPHINCS-256 [12] based on the Hash-based Winternitz One-Time Signatures(WOTS), and c) PICNIC [22]

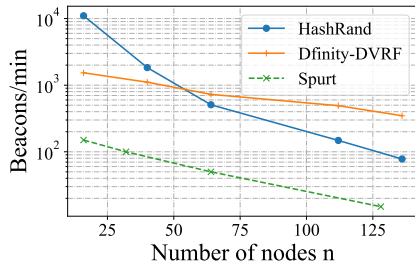


Figure 4: Scalability results: The figure shows the number of beacons produced per minute by `HASHRAND`, `Dfinity-DVRF` [51], and the numbers reported by `Spurt` [28] in the same geo-distributed setting. `HASHRAND`'s $O(n^2 \log(n))$ Hash computational cost results in a steeper slope than `Dfinity`'s $O(n)$ computational cost. `Dfinity`'s cost doesn't include the cost for ADKG. `HASHRAND`'s computational efficiency allows it to produce to produce 1 beacon per second at $n = 136$ without a threshold setup, which is 5x higher than `Spurt` at $n = 128$.

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

In the WAN setting, `PQ-Tusk` has a response rate of 135k transactions per second at $n = 16$ nodes, with an optimal latency of 2.3 seconds per transaction confirmation. This response rate is only 10% lower than classical `Tusk`. `PQ-Tusk` is also very efficient compared to other prominent and implemented PQ-secure protocols like `WaterBear` [73] that uses local coins to generate distributed randomness. At $n = 16$ in a geo-distributed setting, `WaterBear` offers a peak response rate of 70k transactions per second with transaction size of 100 bytes in m5.xlarge machines (twice as powerful as c5.large) as compared to 135k transactions per second at 256 byte transaction size for `PQ-Tusk`.

At higher $n = 40$ and $n = 64$, `PQ-Tusk` offers a peak response rate of 122k transactions per second at a latency of 17.5 seconds per transaction, and 40k txns per second at a latency of 50 seconds. We do an ablation study of `PQ-Tusk` to identify the bottleneck at higher n . We implement and observe the response rate for `Tusk` with `DiLithium` signatures and BLS threshold signatures. This variant of `Tusk` underperforms compared to our `PQ-Tusk` at $n = 16$ with 20% lesser response rate at 108k transactions per second. We identify the higher signing and verification times of `DiLithium` signatures

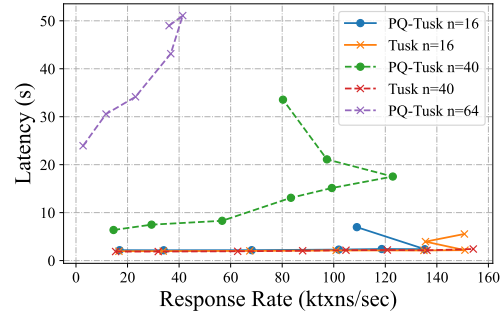


Figure 5: Post-Quantum SMR: The figure describes the latency-response rate curve for `PQ-Tusk`, and `Tusk`. `PQ-Tusk` has a response rate of 135k transactions per second at a latency of 2.3 seconds for $n = 16$, and 122k transactions per second at a latency of 17.5 seconds for $n = 40$ nodes.

over EdDSA signatures as the bottleneck for `PQ-Tusk`. Therefore, with `HASHRAND`, we remove the Post-Quantum liveness bottleneck and establish that randomness from Hash functions is currently the most practical way to achieve Post-Quantum SMR.

8 Discussion

8.1 VSS-SMR

We discuss an alternate approach for building an asynchronous random beacon protocol, titled `VSS-SMR`, which addresses the asynchronous SMR-Beacon circularity mentioned in Section 2.3 by using asynchronous SMR as a setup. This approach first generates a batch of $O(\lambda)$ beacons using our `BAWVSS` protocol and an SMR protocol that does not require a DKG setup. Abraham et al. [2] is one such Information-Theoretically (IT) secure asynchronous SMR protocol with a communication complexity of $O(n^4 \log(n))$ bits. Nodes use these $O(\lambda)$ beacons as common coins for an efficient asynchronous SMR protocol like `FIN` [38] with $O(\lambda n^3)$ communication. Then, nodes use our `BAWVSS` protocol with this efficient SMR instance to agree on a set of $n - t$ successful VSS instances and generate a fresh batch of $O(\beta + \lambda)$ beacons. Out of these beacons, nodes output $O(\beta)$ beacons and set aside $O(\lambda)$ beacons as common coins for the next SMR instance. Through this established pipeline, nodes generate beacons while ensuring sufficient beacons are available to ensure SMR terminates. For every coin used, `FIN`'s termination probability reduces by a factor of $\frac{1}{3}$. This implies $O(\lambda)$ beacons are sufficient to ensure that `FIN` always terminates, except with probability negligible in λ . With a batch size of $\beta = O(n)$ beacons, this approach requires $O(\lambda n^2 \log(n))$ bits of communication per beacon per node, which can be reduced to $O(\lambda n \log(n))$ bits using `AnyTrust` sampling (Failure probability analysis required).

Comparison with `HASHRAND`. `HASHRAND` and `VSS-SMR` differ in their styles of termination with Monte-Carlo and Las-Vegas styles of termination, respectively. Comparing their efficiency, both protocols have the same VSS phase, whereas `VSS-SMR`'s reconstruction phase is $O(n)$ factor more expensive than `HASHRAND`. Although we can use our `AnyTrust` sampling/shuffling for `VSS-SMR` to generate

multiple beacons from the $n - t$ agreed VSSes through SMR, this technique requires further analysis in terms of failure probability.

HASHRAND's agreement phase uses GATHER and c AA instances, with $O(n^2 \log(\frac{1}{1-\delta}))$ and $O(n^3 + cn^2 \log(\frac{1}{1-\delta}) \log \log(\frac{1}{1-\delta}))$ message and communication complexities. FIN requires $O(n^3)$ messages and $O(\lambda n^3)$ communication. Further, HASHRAND's agreement phase leads to a statistical security of $\log(\frac{1}{1-\delta})$ bits and FIN has a cryptographic security of $\lambda = \frac{\kappa}{3}$ bits against a quantum adversary, where κ is the output size of a collision-resistant Hash function. Asymptotically, HASHRAND scales better than FIN by an $O(n)$ factor in message complexity and by a $O(\kappa)$ factor in communication complexity. Concretely, with 40 bits of statistical security (1 failure in a trillion beacons) and using a standard 256-bit Hash function, HASHRAND's $O(c \log(\frac{1}{1-\delta}) \log \log(\frac{1}{1-\delta}))$ factor scales better than VSS-SMR's $O(\kappa n)$ after $n = 45$. Further, HASHRAND's bootstrapping phase also has a $O(n)$ factor lower communication than Abraham et al. [2], enabling efficient reconfiguration.

8.2 AwVSS vs ACSS protocols

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

Given a Hash-based ACSS protocol with $O(n\beta)$ communication complexity, we can improve HASHRAND's communication to $O(n)$ per beacon per node using the batch reconstruction technique in Choudhury and Patra [23]. We can also combine the above VSS-SMR approach with the Vandermonde hyperinvertible matrix technique [26] to generate $O(n\beta)$ beacons from a single SMR instance. Overall, with a big enough β , this approach would also have a communication complexity of $O(n)$ bits per beacon per node. Therefore, improvements in Hash-based ACSS protocols can be utilized to improve PQ-secure asynchronous random beacons.

A recent work by Shoup and Smart [66] proposes a Hash-based ACSS protocol with $O(n\beta + n^3)$ communication in the good case with an honest dealer, and $O(n^2\beta + n^3)$ with a faulty dealer. However, this protocol uses a random beacon as a building block, and requires nodes to be online to listen to complaints. Further, it has a high $O(n^3)$ free term, so it is unclear how it would perform in practice.

8.3 Comparison with ADKG protocols

We set aside Post-Quantum security for this discussion and only compare performance of ADKG and HASHRAND. Asynchronous DKG protocols enable nodes to generate beacons efficiently using threshold unique signatures. Particularly, BLS threshold signatures [15] are highly efficient because of their non-interactive construction. However, the ADKG protocols required to establish a setup for BLS threshold signatures are very expensive. In summary, beacon generation using threshold signatures can be viewed as a

highly expensive setup phase followed by efficient beacon generation. In contrast, at higher values of n , HASHRAND has a relatively cheaper, computationally efficient bootstrapping phase and relatively expensive beacon generation with $O(\log(n))$ factor more communication. Comparing performances, the current best implemented ADKG protocol by Das et al. [31] at $n = 136$ nodes takes 37 seconds to terminate versus 25 seconds for HASHRAND's bootstrapping phase in the same geo-distributed setup. This gap widens with increasing n with HASHRAND's computational efficiency helping it scale better than ADKG. Using ADKG is more favorable when its high setup cost can be *recovered* by efficiently generating sufficient number of beacons. In scenarios with a low reconfiguration period, the high cost of ADKG setup becomes a bottleneck, and HASHRAND's tradeoffs are more favorable. Similarly, with a high reconfiguration period, HASHRAND's relatively expensive beacon generation offers subpar performance compared to ADKG.

8.4 Setup costs and tradeoffs

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

9 Related work

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

Synchronous and partially synchronous protocols. Beacon protocols in the synchronous world consists of protocols [13, 14, 20, 28, 64, 69] that use VSS driven by CRS or SRS setup (Level 4 in [8]). All these protocols require some variant of SMR to achieve the properties of a random beacon. We discuss OptRand [13], and Spurt [28] in this category. OptRand [13] is an optimistically responsive random beacon that uses OptSync's [67] commit rule to generate beacons at network speed. The protocol however depends on a synchronous timeout to implicate faulty leaders. Spurt is a partially synchronous protocol that uses n-parallel PVSS (requiring a Level 4 CRS setup) with leader-based secret aggregation to ensure unpredictability within $O(n^2)$ communication. The secrecy of the PVSS scheme depends on the DBDH assumption. Spurt internally uses the HotStuff [71] SMR protocol to enable honest nodes to terminate and output the beacon value.

Asynchronous protocols. Asynchronous random beacon protocols are bound by Freitas et al.'s [34] impossibility result. Protocols with a DKG setup (Level 5 in [8]) such as Cachin et al. [17] are not bound by this impossibility result because the output from threshold signatures is equivalent to distributed pseudorandomness and is a deterministic function of honest nodes' states.

Randomness Protocols with Las-Vegas style agreement have been traditionally employed in Asynchronous Distributed Key Generation (ADKG) [4, 5, 29, 31, 42, 57]. Kogias et al. [57] propose an

Eventually Perfect Common Coin (EPCC) using a high-threshold AVSS scheme conducted over a secure channel setup (Level 2). Their protocol uses the DDH assumption with a RO. This protocol also has a $O(n^3)$ computational complexity per node, which makes it very expensive in practice.

Other asynchronous protocols use a partially public setup like PKI with CRS or SRS (Level 4 setup). Abraham et al. [5] use PKI-based VRFs and Gather to agree on a random node's index. This protocol uses the SXDH assumption with a RO to achieve unpredictability. A recent work called Bingo [4] produces adaptively secure randomness using a Packed AVSS scheme, which requires a PKI and an SRS setup. Abraham et al., and Bingo have $\Omega(n^2)$ computational complexity, which makes them computationally inefficient. Such protocols can be used for a one-shot ADKG and later use Cachin et al.'s [17] 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 [52, 72] and refreshable secret sharing [48]. Moreover, Kogias et al.'s approach is the only current way to achieve ADKG at a Level 2 setup (without PKI or CRS) in [8], whose expensiveness hinders its practical application.

Freitas et al.'s [34] protocol achieves deterministic termination with a statistical probability of disagreement with a Level 2 setup (pairwise secure channels). This protocol uses AVSS, GATHER, and approximate agreement to output a random ϵ interval of numbers. The AVSS used in this protocol depends on Pedersen commitments and the DLog assumption for unconditional hiding and hence, does not require a RO for unpredictability. However, this protocol has a higher computation complexity of $O(n^3)$ per node and a high round complexity of $\log(\frac{n}{1-\delta})$ rounds.

Information-theoretic protocols. Statistically and information-theoretically secure schemes can be used to develop computationally efficient beacon protocols. For example, Patra et al. [61] propose a statistically secure AVSS scheme with $O(n^4 \log(n))$ communication complexity. This work combined with packed AVSS and Freitas et al.'s Monte-Carlo coin technique results in an overall communication complexity of $O(n^4 \log(n))$ per beacon.

10 Conclusion

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

We also find that this work, especially with its performance analysis, establishes an interesting research direction for secure distributed systems: when aiming for post-quantum security for distributed systems such as SMR or MPC, the protocol builders

should consider hash-based approaches rather than directly resorting to the information-theoretic or statistically-secure protocols with significantly higher communication complexities.

Acknowledgements

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

References

- [1] Ittai Abraham, Yonatan Amit, and Danny Dolev. 2004. Optimal Resilience Asynchronous Approximate Agreement (*OPODIS'04*). Springer-Verlag, Berlin, Heidelberg, 229–239.
- [2] Ittai Abraham, Gilad Asharov, Arpita Patra, and Gilad Stern. 2023. Asynchronous Agreement on a Core Set in Constant Expected Time and More Efficient Asynchronous VSS and MPC. *Cryptology ePrint Archive*, Paper 2023/1130. <https://eprint.iacr.org/2023/1130> <https://eprint.iacr.org/2023/1130>.
- [3] Ittai Abraham, Naama Ben-David, and Sravya Yandamuri. 2022. Efficient and Adaptively Secure Asynchronous Binary Agreement via Binding Crusader Agreement. In *PODC '22*, Alessia Milani and Philipp Woelfel (Eds.). ACM, 381–391.
- [4] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. 2023. Bingo: Adaptivity and Asynchrony in Verifiable Secret Sharing and Distributed Key Generation. In *CRYPTO 2023* (Santa Barbara, CA, USA), 39–70.
- [5] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Reaching consensus for asynchronous distributed key generation. In *PODC'21*, 363–373.
- [6] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically optimal validated asynchronous byzantine agreement. In *PODC 2019*, 337–346.
- [7] Ittai Abraham and Gilad Stern. [n. d.]. Information Theoretic HotStuff. In *OPODIS 2020*, Vol. 184, 11:1–11:16.
- [8] Ittai Abraham and Gilad Stern. 2019. *Trusted Setup assumptions*, <https://decentralizedthoughts.github.io/2019-07-19-setup-assumptions/>.
- [9] Michael Backes, Aniket Kate, and Arpita Patra. 2011. Computational verifiable secret sharing revisited. In *ASIACRYPT 2011*. Springer, 590–609.
- [10] Akhil Bandarupalli, Adithya Bhat, Saurabh Bagchi, Aniket Kate, Chen-Da Liu-Zhang, and Michael K. Reiter. 2024. Delphi: Efficient Asynchronous Approximate Agreement for Distributed Oracles. In *IEEE/IFIP DSN 2024*, 14 pages.
- [11] Akhil Bandarupalli, Adithya Bhat, Saurabh Bagchi, Aniket Kate, and Michael Reiter. 2023. HashRand: Efficient Asynchronous Random Beacon without Threshold Cryptographic Setup. *Cryptology ePrint Archive*, Paper 2023/1755. <https://eprint.iacr.org/2023/1755>
- [12] Daniel J Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. 2015. SPHINCS: practical stateless hash-based signatures. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 368–397.
- [13] Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. 2023. OptRand: Optimistically Responsive Reconfigurable Distributed Randomness. In *NDSS 2023*. The Internet Society.
- [14] Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. 2021. RandPiper: Reconfiguration-Friendly Random Beacons with Quadratic Communication. In *CCS 2021*, 3502–3524.
- [15] Alexandra Boldyreva. 2002. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Public Key Cryptography – PKC 2003*, Yvo G. Desmedt (Ed.), 31–46.
- [16] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. 2010. Random Oracles in a Quantum World. *Cryptology ePrint Archive*, Paper 2010/428. <https://eprint.iacr.org/2010/428>.
- [17] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2000. Random Oracles in Constantipole: Practical Asynchronous Byzantine Agreement Using Cryptography (Extended Abstract). In *PODC 2000*, 123–132.
- [18] C. Cachin and S. Tessaro. 2005. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, 191–201.

- [19] Ran Canetti and Tal Rabin. 1993. Fast asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. 42–51.
- [20] Ignacio Cascudo and Bernardo David. 2017. SCRAPE: Scalable randomness attested by public entities. In *International Conference on Applied Cryptography and Network Security*. Springer, 537–556.
- [21] Chainlink. 2023. *35+ Blockchain RNG Use Cases Enabled by Chainlink VRF*. <https://chain.link/education-hub/rng-in-blockchain-use-cases>
- [22] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Rasmacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. 2017. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *CCS 2017*. 1825–1842.
- [23] Ashish Choudhury and Arpita Patra. 2017. An Efficient Framework for Unconditionally Secure Multiparty Computation. *IEEE Transactions on Information Theory* 63, 1 (2017), 428–468.
- [24] Ran Cohen, Abhi Shelat, and Daniel Wichs. 2019. Adaptively Secure MPC with Sublinear Communication Complexity. In *Advances in Cryptology – CRYPTO 2019*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer International Publishing, Cham, 30–60.
- [25] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. 2009. Asynchronous multiparty computation: Theory and implementation. In *International workshop on public key cryptography*. Springer, 160–179.
- [26] Ivan Damgård and Jesper Buus Nielsen. 2007. Scalable and Unconditionally Secure Multiparty Computation. In *Advances in Cryptology – CRYPTO 2007*, Alfred Menezes (Ed.). 572–590.
- [27] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: A DAG-Based Mempool and Efficient BFT Consensus. In *Eurosys 2022*. 34–50.
- [28] Sourav Das, Vinit Krishnan, Irene Miriam Isaac, and Ling Ren. 2022. Spurt: Scalable Distributed Randomness Beacon with Transparent Setup. In *2022 IEEE Symposium on Security and Privacy (SP)*. 2502–2517.
- [29] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. 2023. Practical Asynchronous High-threshold Distributed Key Generation and Distributed Polynomial Sampling. <https://eprint.iacr.org/2022/1389> <https://eprint.iacr.org/2022/1389>
- [30] Sourav Das, Zhuolun Xiang, and Ling Ren. 2021. Asynchronous Data Dissemination and Its Applications. In *CCS 2021*. 2705–2721.
- [31] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2518–2534.
- [32] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *EUROCRYPT 2018*, Vol. 10821. 66–98.
- [33] Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. 2022. GearBox: Optimal-Size Shard Committees by Leveraging the Safety-Liveness Dichotomy. In *CCS 2022*. 683–696.
- [34] Luciano Freitas de Souza, Petr Kuznetsov, and Andrei Tonkikh. 2022. Distributed Randomness from Approximate Agreement. In *DISC 2022*, Vol. 246. 24:1–24:21.
- [35] Danny Dolev, Nancy A Lynch, Shlomit S Pinter, Eugene W Stark, and William E Weihl. 1986. Reaching approximate agreement in the presence of faults. *Journal of the ACM (JACM)* 33, 3 (1986), 499–516.
- [36] Danny Dolev and H. Raymond Strong. 1983. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput.* 12, 4 (1983), 656–666.
- [37] Shlomi Dolev and Ziyu Wang. 2021. SodsBC/SodsBC++: Post-Quantum Asynchronous Blockchain Suite for Consensus and Smart Contracts. In *Stabilization, Safety, and Security of Distributed Systems: 23rd International Symposium, SSS 2021*. 510–515.
- [38] Sisi Duan, Xin Wang, and Haibin Zhang. 2023. Fin: Practical signature-free asynchronous common subset in constant time. In *CCS 2023*. 815–829.
- [39] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), 238–268.
- [40] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (April 1985), 374–382. <https://doi.org/10.1145/3149.214121>
- [41] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Dumbo-NG: Fast Asynchronous BFT Consensus with Throughput-Oblivious Latency. In *CCS 2022*. ACM, 1187–1201.
- [42] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Efficient Asynchronous Byzantine Agreement without Private Setups. In *ICDCS 2022*. IEEE, 246–257.
- [43] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology* 20 (2007), 51–83.
- [44] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP 2017*. 51–68.
- [45] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *STOC 1996*. 212–219.
- [46] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice. In *NDSS 2022*.
- [47] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster Asynchronous BFT Protocols. 803–818.
- [48] Christoph U. Günther, Sourav Das, and Lefteris Kokoris-Kogias. 2022. Practical Asynchronous Proactive Secret Sharing and Key Refresh. *Cryptology ePrint Archive*, Paper 2022/1586. <https://eprint.iacr.org/2022/1586>.
- [49] Mads Haahr. 1999. random.org: Introduction to Randomness and Random Numbers. *Statistics, (June)* (1999), 1–4.
- [50] Runchao Han, Haoyu Lin, and Jiangshan Yu. 2020. RandChain: A Scalable and Fair Decentralized Randomness Beacon. *Cryptology ePrint Archive*, Paper 2020/1033. <https://eprint.iacr.org/2020/1033> <https://eprint.iacr.org/2020/1033>
- [51] Timo Hanke, Mahnush Movahedi, and Dominic Williams. 2018. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548* (2018).
- [52] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. 1995. Proactive secret sharing or: How to cope with perpetual leakage. In *annual international cryptography conference*. Springer, 339–352.
- [53] Shang-En Huang, Seth Pettie, and Leqi Zhu. 2023. Byzantine Agreement with Optimal Resilience via Statistical Fraud Detection. In *SODA 2023*, Nikhil Bansal and Viswanath Nagarajan (Eds.). SIAM, 4335–4353.
- [54] Aniket Kate, Yizhou Huang, and Ian Goldberg. 2012. Distributed Key Generation in the Wild. *IACR Cryptol. ePrint Arch.* 2012 (2012), 377.
- [55] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All You Need is DAG. In *PODC 2021*. 165–175.
- [56] John Kelsey, Luis TAN Brandão, Rene Peralta, and Harold Booth. 2019. *A reference for randomness beacons: Format and protocol version 2*. Technical Report. National Institute of Standards and Technology.
- [57] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. 2020. Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures. In *CCS 2020*. 1751–1767.
- [58] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited. In *PODC 2020*, Yuval Emek and Christian Cachin (Eds.). 129–138.
- [59] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *CCS 2016*. 31–42.
- [60] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. 2015. Signature-free asynchronous binary Byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time. *Journal of the ACM (JACM)* 62, 4 (2015), 1–21.
- [61] Arpita Patra, Ashish Choudhary, and C Pandu Rangan. 2009. Efficient statistical asynchronous verifiable secret sharing with optimal resilience. In *International Conference on Information Theoretic Security*. Springer, 74–92.
- [62] Philipp Rogaway and John Steinberger. 2008. Constructing cryptographic hash functions from fixed-key blockciphers. In *Advances in Cryptology—CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17–21, 2008. Proceedings 28*. Springer, 433–450.
- [63] Sambhav Satija, Apurv Mehra, Sudheesh Singanamalla, Karan Grover, Muthian Sivathanu, Nishanth Chandran, Divya Gupta, and Satya Lokam. 2020. Blockene: A high-throughput blockchain over mobile devices. In *OSDI 2020*. 567–582.
- [64] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. 2020. HydRand: Efficient Continuous Distributed Randomness. In *2020 IEEE Symposium on Security and Privacy (SP)*. 73–89.
- [65] Victor Shoup. 2024. A Theoretical Take on a Practical Consensus Protocol. *Cryptology ePrint Archive*, Paper 2024/696. <https://eprint.iacr.org/2024/696>
- [66] Victor Shoup and Nigel P. Smart. 2023. Lightweight Asynchronous Verifiable Secret Sharing with Optimal Resilience. *Cryptology ePrint Archive*, Paper 2023/536. <https://eprint.iacr.org/2023/536>
- [67] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. 2020. The optimality of optimistic responsiveness. In *CCS 2020*. 839–857.
- [68] Open source contributors. 2019. *DRand - A Distributed Randomness Beacon Daemon* - <https://github.com/drand/drand>. <https://github.com/drand/drand>
- [69] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. 2017. Scalable Bias-Resistant Distributed Randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*. 444–460.
- [70] Xuechao Wang, Peiyao Sheng, Sreeram Kannan, Kartik Nayak, and Pramod Viswanath. 2023. TrustBoost: Boosting Trust among Interoperable Blockchains. In *CCS 2023*. 1–15. <https://eprint.iacr.org/2022/1428>.
- [71] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus with linearity and responsiveness. In *PODC 2019*. 347–356.
- [72] Thomas Yurek, Zhuolun Xiang, Yu Xia, and Andrew Miller. 2023. Long live the honey badger: Robust asynchronous dpss and its applications. (2023).
- [73] Haibin Zhang, Sisi Duan, Boxin Zhao, and Liehuang Zhu. 2023. WaterBear: Practical Asynchronous BFT Matching Security Guarantees of Partially Synchronous BFT. In *USENIX Security Symposium '23*. 1–16.