HashRand: Efficient Asynchronous Random Beacon without Threshold Cryptographic Setup

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

Abstract—Regular access to unpredictable and bias-resistant randomness is important for applications such as blockchains, voting, and secure distributed computing. Distributed random beacon protocols address this need by distributing trust across multiple nodes, with the majority of them assumed to be honest. These protocols have found applications in blockchain technology, leading to the proposal of several distributed random beacon protocols, with some already implemented. However, many current random beacon systems rely on threshold cryptographic setups or exhibit high computational costs, while others assume partial or bounded synchronous networks. To overcome these limitations, we propose HashRand, a computation and communication-efficient asynchronous random beacon protocol that uses a secure Hash function to generate beacons and pairwise secure channels. HashRand has a per-node communication complexity of $O(n \log(n))$ bits per beacon. The computational efficiency of HashRand is attributed to the two orders of magnitude lower time of a one-way Hash computation compared to discrete log exponentiation. Interestingly, besides reduced overhead, HashRand achieves Post-Quantum security by leveraging the secure Hash function against quantum adversaries, setting it apart from other random beacon protocols that use discrete log cryptography. In a geo-distributed testbed of $n = 160$ nodes, HashRand produces 1 beacon every second, which is at least 4x higher than Spurt. We also demonstrate the practical utility of HashRand by implementing a Post-Quantum secure Asynchronous SMR protocol, which has a response rate of over 122k txns per second over a WAN at $n = 40$ nodes.

1. Introduction

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

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

This setup (also called threshold setup) enables $n$ nodes to generate $(n, t)$-threshold unique signatures, which offer unpredictability and bias-resistance against an adversary corrupting $t < \frac{n}{3}$ nodes ($t < \frac{n}{4}$ in a synchronous network). Although the trusted setup phase for threshold signatures can be replaced with a Distributed Key Generation (DKG) protocol [38], [49], [52], DKG protocols are expensive in computation and communication and must be rerun each time participant nodes change. This constraint makes threshold signatures costly for applications with participant churn, such as Proof-of-Stake blockchains [39].

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

Many asynchronous randomness generation protocols that do not use threshold signatures function in a PKI/broadcast setup [2], [3], [26], [28], [37] or a pairwise-channels setup [30], [52] where the dealer sets up secure channels between participant nodes. However, these protocols have a high computational cost from $n$-parallel Asyn-
chronous VSS (AVSS). The current best AVSS protocol with $O(\lambda n^2)$ communication complexity [27] uses computation-intensive discrete log cryptography. Each AVSS instance requires $O(n^2)$ discrete log exponentiations per node, where each exponentiation takes approximately $100 \times$ more time and energy to run than a one-way hash computation. This computation bottleneck induced by discrete log operations is visible in Spurt [25], a leader-based partially synchronous beacon protocol. Spurt performs only 15 beacons per minute at $n = 128$ nodes because of the $O(n^2)$ discrete log operations per beacon performed by the leader.

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

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

**Our results.** We propose HASHRand, a hash-based asynchronous random beacon protocol that uses a collision-resistant Hash function to output random beacons. HASHRand has an amortized communication complexity of $O(\lambda n \log(n))$ bits per beacon per node, where $\lambda$ is the range of the hash function in bits, and requires $O(\lambda n \log(n))$ Hash computations per beacon per node. HASHRand requires only a pairwise secure channel setup, where nodes can send private, authenticated messages to each other. HASHRand is computationally efficient because it only uses lightweight cryptographic primitives like hash functions and message authentication codes (MACs). On top of computational efficiency and practical communication complexity, HASHRand is also post-quantum secure, which is an added benefit over protocols that use discrete-log cryptography. This is because contemporary one-way hash

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Network</th>
<th>Post Quantum Security</th>
<th>Termination Flavor</th>
<th>Adaptive Security</th>
<th>Communication Complexity (per node)</th>
<th>DLog Expo Complexity (per node)</th>
<th>Setup</th>
<th>Cryptographic Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cachin et al. [19]</td>
<td>async.</td>
<td>✓</td>
<td>D</td>
<td>✓</td>
<td>$O(\lambda n)$</td>
<td>$O(n^2)$</td>
<td>DKG</td>
<td>pRO &amp; CDH</td>
</tr>
<tr>
<td>RandShare [64]</td>
<td>async.</td>
<td>✓</td>
<td>LV</td>
<td>✓</td>
<td>$O(\lambda n^2)$</td>
<td>$O(n^3)$</td>
<td>Secure channels</td>
<td>pRO &amp; DDH</td>
</tr>
<tr>
<td>Spurt [25]</td>
<td>partial sync.</td>
<td>✓</td>
<td>D</td>
<td>✓</td>
<td>$O(\lambda n^2)$</td>
<td>$O(n^2)$</td>
<td>CRS &amp; PKI</td>
<td>pRO &amp; DDBH</td>
</tr>
<tr>
<td>Kogias et al. [52]</td>
<td>async.</td>
<td>✓</td>
<td>LV</td>
<td>✓</td>
<td>$O(\lambda n^3)$</td>
<td>$O(n^3)$</td>
<td>Secure channels</td>
<td>DLog</td>
</tr>
<tr>
<td>Das et al. [28]</td>
<td>async.</td>
<td>✓</td>
<td>LV</td>
<td>✓</td>
<td>$O(\lambda n^2)$</td>
<td>$O(n^3)$</td>
<td>PKI</td>
<td>pRO &amp; DDH</td>
</tr>
<tr>
<td>Gao et al. [37]</td>
<td>async.</td>
<td>✓</td>
<td>LV</td>
<td>✓</td>
<td>$O(\lambda n^2)$</td>
<td>$O(n^3)$</td>
<td>PKI</td>
<td>pRO &amp; SXDH</td>
</tr>
<tr>
<td>Abraham et al. [3]</td>
<td>async.</td>
<td>✓</td>
<td>LV</td>
<td>✓</td>
<td>$O(\lambda n^2)$</td>
<td>$O(n^3)$</td>
<td>PKI</td>
<td>pRO &amp; SXDH</td>
</tr>
<tr>
<td>Bingo [2]</td>
<td>async.</td>
<td>✓</td>
<td>LV</td>
<td>✓</td>
<td>$O(\lambda n^2)$</td>
<td>$O(n^3)$</td>
<td>DLog</td>
<td></td>
</tr>
<tr>
<td>Freitas et al. [30]</td>
<td>async.</td>
<td>✓</td>
<td>MC</td>
<td>✓</td>
<td>$O(n^4 \log(n))$</td>
<td>$O(n^3)$</td>
<td>Secure channels</td>
<td></td>
</tr>
<tr>
<td>Patra et al. [57]</td>
<td>async.</td>
<td>✓</td>
<td>LV</td>
<td>✓</td>
<td>$\Omega(n^5)$</td>
<td>$0^*$</td>
<td>Auth channels</td>
<td></td>
</tr>
<tr>
<td>Huang et al. [48]</td>
<td>async.</td>
<td>✓</td>
<td>MC</td>
<td>✓</td>
<td>$O(\lambda n \log(n))$</td>
<td>$0^*$</td>
<td>Secure channels</td>
<td></td>
</tr>
<tr>
<td>HASHRand</td>
<td>async.</td>
<td>✓</td>
<td>MC</td>
<td>✓</td>
<td>$O(\lambda n \log(n))$</td>
<td>$0^*$</td>
<td>CR &amp; IH Hash</td>
<td>pRO</td>
</tr>
</tbody>
</table>

**Termination Flavor:** Monte-Carlo(MC) style protocols terminate in a deterministic number of rounds. But, even after termination, honest nodes can disagree on the beacon with a practically negligible probability $p = 1 - \delta$. A lower $p$ implies a higher round complexity. Las-Vegas(LV) style protocols terminate only upon agreement amongst honest nodes on the beacon value. These protocols can have infinitely many executions and we report the expected runtime complexities of these protocols. Deterministic(D) protocols offer agreement and a deterministic runtime. **DLog Expo complexity:** Concurrent beacon and coin protocols overwhelmingly depend on discrete log cryptography, which is the main reason for their high computational cost. HASHRand instead uses $O(cn^3 \log(n))$ Hash computations(100x cheaper than DLog expo) per beacon. **Committee election:** HASHRand elects an AnyTrust committee and reconstructs only those secrets. AnyTrust committees are much smaller in size than honest supermajority committees used in beacons like RandHerd [64] and Algornad [39]. **Hash:** HASHRand uses a Collision Resistant(CR), and an Input Hiding(IH) Hash function to prove security against a classical adversary. The Input Hiding property [15, page 345], requires the Hash function to not reveal any information about the input. We also use a Quantum Random Oracle (RO) assumption against a Quantum adversary [16]. Other protocols use a Programmable RO (pRO) to link the properties of their beacon to the underlying hardness assumption. The leader performs $O(n^2)$ operations every round, while other nodes perform $O(n)$ operations each.
functions like SHA256 offer collision-resistance and pre-image resistance against a polynomial time quantum adversary. We offer a detailed comparison of HASHRAND with related works in Table 1.

**Evaluation.** We implement HASHRAND in Rust by using SHA256 hash function as the one-way function. We evaluate HASHRAND in a geo-distributed setting on AWS and measure the throughput in terms of the number of beacons per minute. We also compare HASHRAND with Dfinity-DVRF [46] based on BLS threshold signatures. With \( n = 40 \) nodes, HASHRAND outputs 2819 beacons per minute, against 1505 beacons pm for Dfinity-DVRF. Even with \( n = 160 \) nodes, HASHRAND still outputs 60 beacons per minute, with a latency of 2.3 seconds and 17.5 seconds, respectively.

**An illustrative application.** On top of implementing and evaluating HASHRAND as an asynchronous beacon, we demonstrate its practical utility by implementing a post-quantum secure asynchronous SMR protocol. We integrate HASHRAND with TUSK [24], the State-of-the-art asynchronous SMR protocol to form PQ-TUSK, in which HASHRAND provides the randomness needed for wave leader election. We replace TUSK’s digital signatures with DiLithium [35], a post-quantum secure public key signature scheme. PQ-TUSK has an optimal communication complexity of \( O(\lambda n) \) bits per transaction per node, which is much more practical than other post-quantum SMR protocols like WaterBear [69] with worst case \( O(exp(n)) \) communication complexity. We also demonstrate the practicality of PQ-SMR by evaluating our SMR protocol PQ-TUSK in a geo-distributed setting. PQ-TUSK achieves a throughput of 135k transactions per second for \( n = 16 \) nodes and 122k transactions per second for \( n = 40 \) nodes, with a latency of 2.3 seconds and 17.5 seconds, respectively.

### 2. Problem Setting and Solution Overview

#### 2.1. System Model

We assume a set of \( n \) nodes connected by pairwise authenticated and secure channels and an asynchronous network with no time bound on message delivery. We consider a polynomial-time quantum adversary \( A \) defined according to Boneh et al. [16], who has access to a quantum computer and controls \( t < \frac{n}{3} \) nodes. \( A \) also controls the network between honest nodes and can arbitrarily delay and reorder messages. The channels connecting honest nodes are private and \( A \) cannot distinguish the contents of each message from a uniformly random bitstring within polynomial time. An efficient post-quantum symmetric encryption scheme like AES can satisfy this assumption.

#### 2.2. Asynchronous Random Beacons

We define a random beacon protocol and its properties based on Spurt’s [25] definition and adapt it to asynchrony. Honest nodes participating in a beacon protocol \( B \) output tuples of the form \((i, b_i)\), where \( i \in \mathbb{N} \) is a unique incremental beacon index and \( b_i \in D \) is an element from a predefined domain of numbers \( D \).

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

1) **Agreement:** For a given index \( x \), if honest nodes \( n_j \) and \( n_k \) output \((x, b_{xj})\) and \((x, b_{xk})\) respectively, then \( b_{xj} = b_{xk} \), except with a probability negligible in \( \lambda \).

\[
Pr\left\{ (x, b_{xj}) \leftarrow n_j \\
\land (x, b_{xk}) \leftarrow n_k \\
\land b_{xj} \neq b_{xk} \right\} \leq \text{neg}(\lambda)
\]

2) **Liveness:** Every honest node \( j \) must eventually output a beacon \((x, b_x)\) for all \( x \geq 1 \).

3) **Bias-resistance and Unpredictability:** Given that no honest node invoked \( B.OPEN(x) \) for an index \( x \geq 1 \), a polynomial time quantum adversary \( A \) should not have any advantage in predicting the beacon output \( b_x \).

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

Prior random beacon protocols [12], [13], [25] also guaranteed agreement with probability 1 and unpredictability with \( A \)'s advantage negligible in \( \lambda \). However, both properties cannot be achieved with probability 1.

#### 2.3. HASHRAND Key Insights

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

HASHRAND is a Monte-Carlo style beacon protocol that builds on top of the asynchronous approximate common coin [30], which uses approximate agreement or \( \epsilon \)-agreement primitive. HASHRAND guarantees Monte-Carlo agreement with a probability of \( \delta \), which is a statistical security parameter. HASHRAND uses \( n \)-parallel AVSS, Gather, Gather, Gather.
and $\epsilon$-agreement as building blocks and builds on top of two key observations that make it computationally efficient with a practical communication complexity.

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

2) Reconstructing $O(n) \geq t + 1$ secrets is not necessary to guarantee the unpredictability of a beacon. As long as we can ensure that at least one honest node’s secret will always be reconstructed and be part of the beacon, we can ensure the beacon’s unpredictability. Utilizing this observation, we propose a committee election procedure, which elects an AnyTrust committee of constant size guaranteed to have at least one honest node whose BAVSS instance will be terminated and reconstructed by all honest nodes. All nodes later reconstruct the secrets of only those nodes part of the committee. We bootstrap the randomness required for committee election by generating more randomness in the startup rounds. This technique brings down the communication complexity of the reconstruction phase to $O(\lambda n^2 \log(n))$ bits or $O(\lambda n \log(n))$ bits per secret per node. The first observation allows us to reduce computational complexity without increasing the communication complexity of the beacon. Additionally, each instance of Gather and $\epsilon$-agreement can be shared across all $n$ secrets in a BAVSS instance, effectively amplifying the throughput by the batch size. However, even with this observation, the communication complexity per beacon is still $O(\lambda n^3 \log(n))$ bits because every beacon output still requires reconstructing $O(n)$ secrets.

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

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

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

3. Building Blocks

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

3.1. Asynchronous weak Verifiable Secret Sharing (AwVSS)

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

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

- AwVSS.SH: A dealer $n_d$ shares a secret $s_d \in \mathcal{D}$ among nodes in $\mathcal{N}$. At the end of AwVSS.SH, at least $t+1$ honest nodes $n_i \in \mathcal{N}$ hold a secret share $s_{d,i}$.
- AwVSS.REC: Every honest node $n_i \in \mathcal{N}$ reconstructs the secret $s_{d,i}$ in a distributed fashion by broadcasting its secret share $s_{d,i}$ (if available). If the dealer $n_d$ was honest, every honest $n_i$ outputs $s_{d}$ and $\bot$ otherwise. A protocol AwVSS implements $(n, t)$-Asynchronous weak VSS if it satisfies the following properties.

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

- Correctness: If $n_d$ is honest, then each honest node upon terminating AwVSS.REC, outputs the shared secret $s_d \in \mathcal{D}$ with probability $p \geq 1 - \text{negl}(\lambda)$.

- Secrecy: If $n_d$ is honest and no honest node has started AwVSS.REC, then an adversary that corrupts up to $t$ nodes has no information about $s_d$.

- Weak Commitment: Even if $n_d$ is malicious, with probability at least $p \geq 1 - \text{negl}(\lambda)$, there exists a value
s^* \in D \cup \{\perp\} at the end of AWVSS.R, such that all honest nodes output s^* at the end of AWVSS.REC phase.

This primitive is strictly weaker than the widely used Asynchronous Verifiable Secret Sharing (AVSS) primitive because AVSS requires the honest nodes to hold shares of a secret s \in D. In AWVSS, nodes can hold shares of a secret s \in \{D, \perp\}. A malicious dealer who conducts an invalid sharing is detected in the sharing phase in AVSS and the reconstruction phase in AwVSS.

We build on top of Dolev et al.'s [32] hash-based AwVSS scheme and give a brief description of it. In this protocol, a dealer n_d uses Shamir secret sharing to create secret shares and then creates a Merkle tree on top of it. n_d then sends individual shares along with Merkle proofs to nodes and reliably broadcasts the Merkle root r. An honest node participates in the RBC of root only upon verifying its Merkle proof. During the reconstruction phase, nodes that terminated with a share broadcast their shares and Merkle proofs. Upon receiving t+1 shares, every node reconstructs the entire share vector and the Merkle tree and verifies if the root broadcasted in the sharing phase r matches the reconstructed root r'. It outputs the shared secret if r = r' and outputs \perp otherwise.

**Batched AwVSS.** Towards reducing communication complexity in an amortized fashion, we also define a batched version of AwVSS scheme, where a protocol BAwVSS implementing Batched AwVSS has sub protocols BAWVSS.R,SH(S) and BAWVSS.REC(.). The dealer n_d shares a batch of multiple secrets S_d with BAWVSS.R,SH(S_d) and honest nodes reconstruct each secret with BAWVSS.REC(i); \forall i \in \{1, \ldots, |S_d|\}. In addition to the properties of AwVSS, BAWVSS also has the all-or-none property, where a node must either terminate BAWVSS.R,SH(S_d) for all secrets in the batch S_d or should not terminate at all. This property is useful in HASHRAND to increase throughput.

### 3.2. Gather

We also employ the Gather primitive [3].

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

- **Binding Common Core:** Once the first honest node n_i outputs set G_i, out of all possible sets G_j \subseteq G_i with size |G_j| = n - t, there exists a unique core set G such that every other honest node n_j's output G_j \cap G_i \supseteq G. Moreover, the core set G is binding, meaning the adversary A cannot force any other honest node n_j to output a set G_j \not\supseteq G.

- **Termination:** If every honest node invokes GATHER.START, then every honest node n_i will eventually invoke GATHER.TERM(G_i) and output a set G_i: G \subseteq G_i.

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

### 3.3. Approximate agreement

Finally, we use \epsilon-agreement primitive [31].

**Definition 3.3.** A protocol E implementing the \epsilon-agreement primitive amongst nodes N is composed of E.START(.) and E.TERM(.), where every honest node n_i starts by invoking E.START(n_i) with a value n_i \in \mathbb{R} and terminates by invoking E.TERM(n_i) with output n_i. A protocol implementing \epsilon-agreement satisfies the following properties.

- **Termination:** If every honest node invokes E.START(.), then every honest node n_i must eventually invoke E.TERM(n_i).

- **\epsilon-agreement:** For a given \epsilon > 0, the outputs of any pair of honest nodes n_i and n_j are within \epsilon of each other. In other words, |n_i - n_j| < \epsilon \quad \forall \{n_i, n_j\} \in N.

- **Validity:** Let M be the set of honest nodes' initial values n_i. The decision value of every honest node must be within the range of initial inputs of honest nodes M. In other words, min M \leq n_i \leq max M.

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

### 4. HASHRAND Design

#### 4.1. Approximate Common Coin

Approximate common coin [30] is a deterministically terminating asynchronous protocol, which enables honest nodes to approximately agree on a random number. In this scheme, every node n_i shares a uniformly drawn secret s_i using an Asynchronous Verifiable Secret Sharing (AVSS) protocol. After instantiating AVSS, honest nodes
run GATHER and wait for its termination. A honest node $n_j$ that terminates GATHER outputs a set of node indices $G_j$ of size $\geq n - t$. This set $G_j$ corresponds to all the AVSS instances terminated by $n_j$. After terminating GATHER, nodes create an $\epsilon$-agreement instance $E_i$ for every node $i : n_i \in \mathcal{N}$. This step is necessary to enable nodes to approximately agree on a representative weight for each participant node. Node $n_j$ inputs 1 to $E_i$ if $i \in G_j$ and inputs 0 otherwise. After terminating all $n \in \mathcal{E}_i : \forall i \in \{1, \ldots, n\}$ instances with outputs $w_{i,j} \forall i \in \{1, \ldots, n\}$, node $n_j$ initiates the reconstruction phase of AVSS. It waits until reconstructing AVSS instances for which $w_{i,j} > 0$. $n_j$ then aggregates the opened secrets $s_i$ using a weighted average to generate $o_j$.

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

This approximate common coin can be converted to a Monte-Carlo common coin [30] through rounding, where nodes round off their $o_j$ values to the closest "checkpoint". The ratio of the distance between checkpoints and the degree of approximation in the approximate common coin influences the success probability of the Monte-Carlo coin. **Building a beacon.** A strawman approach to building a beacon from this Monte-Carlo coin is to start a new coin instance after terminating the previous coin. However, this approach is impractical with a high latency, computation, and communication cost. Achieving a Monte-Carlo agreement probability of $\delta$ requires $\log(\frac{1}{\delta^2})$ rounds of $\epsilon$-agreement. For example, generating a single beacon with $\delta = 1 - 2^{-40}$, equivalent to one expected failure in a trillion beacons, requires 200 round trips of communication. This round complexity results in a high latency in a Wide-Area Network. Moreover, $n$-parallel AVSS phase in this coin costs $O(n^2)$ bits of communication and $O(n^2)$ discrete log exponentiations per node, which also creates a scalability bottleneck. We address these inefficiencies with two optimizations: a) Batching and pipelining to increase the throughput and reducing latency respectively, and b) Committee election for scalability.

### 4.2. Batching and Pipelining

We first observe that the approximate common coin's properties are unaffected if we replace the AVSS scheme in the coin with an AwVSS scheme with weak commitment. AwVSS’s weak commitment property enables every honest node to agree on the maliciousness of the dealer during the reconstruction phase and discard the contributions of these nodes. This observation enables us to use the efficient Hash-based AwVSS protocol proposed by Dolev et al. [32] and defined in Section 3.1.

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

1) We replace the conditionally hiding commitments in Dolev et al. with unconditionally hiding commitments based on Backes et al. [9]. Along with the share polynomial $f(x)$ of degree $t$, the dealer also computes a nonce polynomial $R(x)$ of degree $t$ with a secret $R$ of size twice the range of hash function $H$ (implies a 512-bit nonce for SHA256). The dealer then computes $H(R(i), f(i))$ as a commitment for the $i$th secret share, and builds a Merkle tree on this commitment vector. This change allows us to prove the security of HASHRAND without a Random Oracle assumption.

2) We batch $\beta$ secrets in each BAwVSS instance and reliably broadcast a vector of $\beta$ Merkle roots. This amortizes the cost of RBC over $\beta$ secrets. This change allows us to amplify the throughput of HASHRAND by sharing Gather and the expensive $\epsilon$-agreement phase over all the secrets in the batch to produce $\beta$ beacons at the end. Replacing AVSS in the coin with BAwVSS drastically reduces the computation cost of each beacon by replacing DLog exponentiations with Hashes. It also amplifies the throughput by a $O(\beta)$ factor. Further, for a batch size $\beta = O(n)$ secrets, $n$-parallel BAwVSS has an amortized sharing comm. complexity of $O(\lambda n \log(n))$ bits per node, which is an $O\left(\frac{\log(n)}{n}\right)$ improvement over AVSS. We defer the full protocol and its security proofs to Appendix A. **Pipelining.** We address the high latency of $\epsilon$-agreement in HASHRAND using pipelining and piggybacking. Even though $\epsilon$-agreement has a high round complexity, it still terminates deterministically in a fixed number of rounds. We leverage this deterministic termination property and create a pipeline of beacons with a pipelining period $\phi$. For example, $\phi = 10$ implies one $n$-parallel BAwVSS instantiation for every 10 rounds of $\epsilon$-agreement. Using pipelining, HASHRAND maintains a steady throughput of beacons with a constant latency by instantiating $n$-parallel BAwVSS every $\phi$ rounds and pipelining the corresponding $\epsilon$-agreement phases. HASHRAND moves to round $r + 1$ from round $r$ only after terminating the $n$-parallel BAwVSS $r,i$ and its corresponding GATHER instance (if $r$ is a multiple of $\phi$), and round $r - r'$ of $\epsilon$-agreement instances $E_{r',j}$ corresponding to BAwVSS $r',j \forall r' \in \{r, r - \phi, \ldots, r - r_1\}, j \in \{1, \ldots, n\}$. $r_i$ is the number of rounds of $\epsilon$-agreement required to achieve a success probability of $\delta$.

This pipelining technique ensures that the entire pipeline is moving at the same pace, hence maintaining a consistent throughput of beacons. The length of the pipeline is determined by the probability of agreement $\delta$ and $n$. For constant $n$, a higher $\delta$ implies a longer pipeline to fill, which increases the startup latency. However, once full, HASHRAND keeps up a continuous and constant throughput of $\frac{\beta}{\phi}$ Monte-Carlo beacons per round.
Piggybacking. In rounds \( r : r \mod \phi = 0 \) with BA\(W\)\(VSS\), we piggyback the messages of \( \epsilon \)-agreement instances on top of BA\(W\)\(VSS\) messages to maintain the same message complexity. This procedure does not affect the properties of BA\(W\)\(VSS\) or \( \epsilon \)-agreement because both internally use all-to-all broadcasts. This piggybacking improves the practical efficiency ofHASHRAND by having the same message complexity as the non-pipelined version in the optimistic case.

4.3. Committee election

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

Ensuring the unpredictability of the beacon requires the contribution of only one honest node. However, the Gather’s common core property and the corresponding \( \epsilon \)-agreement phase guarantees the contribution of at least \( t + 1 \) honest nodes, which is overkill and results in higher communication complexity. We address this issue by using a committee election procedure. After \( n \)-parallel BA\(W\)\(VSS\) and Gather, honest nodes agree on a constant-sized committee of nodes \( N \) and run an \( \epsilon \)-agreement instance \( \mathcal{E}_i \) \( \forall i : n_i \in N \) only for the elected node indices, as opposed to running an \( \mathcal{E}_i \) \( \forall i : n_i \in \mathcal{N} \).

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

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

Calculating committee size. We calculate the size of the committee to be sampled using a hypergeometric probability distribution, which models sampling from a group of nodes without repetition. The committee \( N \) must contain at least one honest node whose secret will have a weight \( w_{i_1} = 1 \) in the aggregation. We know that the core set of Gather contains at least \( t + 1 \) honest nodes, out of which at least one honest node must be in the committee. Therefore, the probability of at least one such honest node being on a committee of size \( c \) is given as follows.

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

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

4.4. Protocol description

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

**Beacon preparation phase.** HASHRAND takes batch size \( \beta \) and instantiation period of BA\(W\)\(VSS\) \( \phi \), where \( \phi \leq \log(\frac{n}{2\varepsilon}) \), as configurable inputs. Once every \( \phi n \) rounds, nodes initiate BA\(W\)\(VSS\) by selecting \( \beta \) random secrets and secret sharing them using Algorithm 2 as BA\(W\)\(VSS\)\(_{r,i}\).SH(.) (Line 8). Once every \( \phi n \) rounds, nodes increase the batch size \( \beta' = \beta + n \) to output enough beacons for committee election for the next \( n \) BA\(W\)\(VSS\) instances (Line 8), where each BA\(W\)\(VSS\) and Gather instance requires an unpredictable random beacon to elect \( c \) nodes from \( \mathcal{N} \). After instantiating BA\(W\)\(VSS\), nodes also call GATHER.START and instantiate round \( r \) for each \( \epsilon \)-agreement instance \( \mathcal{E}_{r,i} \) started in the last \( r' \in [r - r_t, r] \) rounds (Line 8).

Table 2: Symbols and notations used in HASHRAND

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \delta )</td>
<td>Statistical security parameter</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>Computational security parameter</td>
</tr>
<tr>
<td>( \beta )</td>
<td>Batch size in BA(W)(VSS)</td>
</tr>
<tr>
<td>( \phi )</td>
<td>Period of BA(W)(VSS)</td>
</tr>
<tr>
<td>( \phi_n )</td>
<td>Current round of HASHRAND</td>
</tr>
<tr>
<td>( r )</td>
<td>Batch size in BA(W)(VSS)</td>
</tr>
<tr>
<td>( \mathcal{E}_{r,i} )</td>
<td>( \epsilon )-agreement instance corresponding to BA(W)(VSS) (_{r,i}) instantiated in HASHRAND round ( r )</td>
</tr>
<tr>
<td>( B )</td>
<td>The list of all prepared beacons yet to be opened</td>
</tr>
<tr>
<td>( B_i )</td>
<td>The list of all prepared beacons for committee election</td>
</tr>
</tbody>
</table>

\[
\sum_{c=1}^{n} \binom{n}{c} \cdot \binom{n-c}{t-c} = \binom{2n}{t}
\]

where \( \binom{n}{c} \) is the binomial coefficient.
Committee size $c = |N|$ eventually contribute to the beacon. For $G = c$ Figure 2: The plot describes the size of the committee for future committee election is being proposed. Once nodes do not elect a committee in rounds where random-

Figure 1: HASH function for future committee election is being proposed. Once protocol reaches round $r$, the beacon initiated through BAWVSS in round $r - r_t$ are terminated and added to the beacon queue B (Line 23). These beacons are ready to be opened.

**Open phase.** Once an honest node $n_i$ gets a request to open a beacon for index $k$, it identifies the $n$-parallel BAWVSS instance $\text{BA VWSS}_{r,i}$, and the index of the secret in the batch $b$ corresponding to the given index $k$ (Line 36). $n_i$ broadcasts its shares by calling $\text{BA VWSS}_{r,i}.\text{REC}(b) \forall j \in N_{r,i}$. It then waits until it reconstructs all secrets for which the $\epsilon$-agreement instance $E_{r,j}$ terminated with weight $w_{r,j} > 0$ (Line 43). Then, $n_i$ outputs a weighted average of reconstructed secrets and rounds it off to the nearest checkpoint, which is a multiple of the domain size $|D|$ (Line 46).

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

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

5. HASHRAND Analysis

5.1. Security Analysis

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

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

After terminating GATHER, node $i$ calls GATHER.TERM$(1)$ and outputs set $G_i$. Next, nodes elect a committee using COMSELECT procedure, where they open the beacons prepared for committee election (Line 19). We note that nodes do not elect a committee in rounds where randomness for future committee election is being proposed. Once COMSELECT terminates, the nodes instantiate $\epsilon$-agreement $E_{r,j} \forall i \in N_r$ only for indices $i$ in the committee $N_{r,i}$, by calling $E.\text{START} (r, i)$ (1) if $n_i \in G_i$ and calling $E.\text{START} (r, i)$ (0) if $n_i \notin G_i \forall i \in N_r$ (Line 19). Nodes terminate round $r$ only after all $\epsilon$-agreement instances $E_{r' j} : r' \geq r - r_t$ terminate round $r$ (Line 21). Once protocol reaches round $r$, the beacon $\text{BA VWSS}_{r,i}$ round $r - r_t$ are terminated and added to the beacon queue B (Line 23). These beacons are ready to be opened.
the RO model, our BAWVSS scheme offers computational correctness with $p = 1 - \negl(\lambda)$ and perfect secrecy under the Input Hiding assumption of Hash functions.

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

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

**Proof.** From the termination property of BAWVSS in Definition 3.1, we know that BAWVSS scheme satisfies totality, which ensures that every honest node $n_i$ terminates Gather. Every honest node therefore starts $\epsilon$-agreement instance $E_{0,i}, \forall n_i \in \mathcal{N}$. From the properties of $\epsilon$-agreement in Definition 3.3, if all nodes invoke $E$.Start(i), then every honest node terminates $E$.Start(i). Hence, every honest node terminates $B$.Prep(0, $\beta + n$). In the opening phase $B$.Open(0) $\forall l \in \{0, \beta + n\}$, every honest node $n_j$ only waits to reconstruct secrets initiated by nodes $n_l$ for which the output of $E_{0,i}, w_{ij} > 0$. An output $w_{ij} > 0$ implies that at least one honest node $n_j$ input 1 to $E_{0,i}$, meaning which $n_{ik}$ terminated $n_i$'s BAWVSS instance. By the termination property of BAWVSS, every honest node must eventually terminate and thereby reconstruct the secret shared by $n_i$. Therefore, $n_j$ and all honest nodes terminate $B$.Open(l) $\forall l \in \{0, \ldots, \beta + n\}$. \qed
Theorem 5.2. Assuming secure BAWVSS, Gather and ε-agreement protocols, HashRAND satisfies unpredictability property in Definition 2.1 for beacon indices \( i \in \{0, \ldots, \beta + n\} \).

Proof. Every honest node \( n_j \) calculates a weighted average of secrets. Gather’s binding core primitive guarantees that at least \( t + 1 \) honest nodes’ secrets will always have \( w_i = 1 \). This set of nodes is fixed before the first honest node terminating Gather begins ε-agreement. On top, no honest node starts \( B.\text{OPEN}() \) until all \( E_0, i \in \{1, \ldots, n\} \) terminate, which forces \( \mathcal{A} \) to commit to the secrets that contribute to the beacon before any honest node starts \( B.\text{OPEN}() \). The correctness property of BAWVSS ensures at least \( t + 1 \) honest secrets are always reconstructed and become part of the beacon. From secrecy of BAWVSS, we guarantee unpredictability and bias-resistance for all \( \beta + n \) beacon. We note that the unconditional hiding property of BAWVSS ensures perfect unpredictability of the beacon.

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

Proof. From the weak commitment property of BAWVSS, at the end of \( B.\text{OPEN}(i) \) for an index \( i \), honest nodes each have a weighted sum of secrets which correspond to a random range of \( \epsilon = 2 \) numbers in the domain \( [0, \frac{1}{\epsilon - 2}] \) with probability \( p \geq 1 - \text{negl}(\lambda) \). When divided by \( \frac{1}{2^2} \) and rounded off to the closest number, the probability that all honest nodes agree is \( p = \frac{2^4}{3} - \text{negl}(\lambda) \).

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

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

Proof. Every batch of beacons instantiated in rounds \( r > r_i; r \mod n \neq 0 \) requires a committee to be elected from previously agreed upon beacon. From the security properties of first \( \beta + n \) beacons in Theorems 5.1 to 5.3, this committee election beacon terminates at all honest nodes, has an agreement probability of \( p_e = \frac{2^3}{3} - \text{negl}(\lambda) \), and is unpredictable. Therefore, every honest node starts ε-agreement instance \( E_r, i \in N \) with probability \( p_e \), and eventually terminates \( E_r, i \in N \), guaranteeing liveness with probability \( p_e \).

The elected committee \( N \) has at least one honest node \( n_j \) that is part of the core set \( G \) with probability \( p_h = \frac{2^4}{3} \). Additionally, no honest node opens the beacon for committee election until it terminates Gather, which implies \( n_j \)’s secret will have weight \( w_{j_i} = 1 \). Therefore, the output beacon will always the contribution of at least one honest node with probability \( p_e p_h = \left(\frac{2^4}{3}\right)^2 - \text{negl}(\lambda) > \delta - \text{negl}(\lambda) \). This guarantees unpredictability.

After electing a committee with probability \( p_e \) and having at least one honest node \( n_j \in G \) part of the agreed-upon committee \( N \), ε-agreement and weighted average of random secrets enables honest nodes to agree on the beacon value with probability \( p = p_e p_h p_n = \left(\frac{2^4}{3}\right)^3 - \text{negl}(\lambda) > \delta - \text{negl}(\lambda) \), where \( p_n \) is the probability that the weighted average results in the same beacon at all nodes.

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

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

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

We avoid this problem in HashRAND by using an unconditionally hiding commitment scheme, where multiple possible inputs can generate each commitment string. We assume the Hash function \( H \) is a programmable Random Oracle (RO) whose outputs for any given input can be programmed by the simulator. When the adversary corrupts
node $i$, the simulator generates a new set of shares and programs $H$ to return the same commitments generated by $i$’s old shares. Then, the simulator rewinds the tapes of the adversary and other honest nodes to replace $i$’s old shares with the newer shares. A similar technique based on unconditionally hiding commitments was used in Bingo [2] to prove the adaptive security of their AVSS scheme. Additionally, our RO assumption does not involve recording the adversary’s queries. Hence, we also avoid the problem of recording a quantum adversary’s queries and achieve post-quantum security in the Quantum RO model [16].

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

5.2. Complexity Analysis

We analyze the communication and computation complexity of HashRand. The $n$-parallel BAWVSS phase in round $r: r \mod \phi = 0$ with $\beta$ secrets costs $O(\beta n \log(n) + \lambda n^2)$ bits of communication per node, with each node sending $\beta$ merkle proofs and RBCombing a vector of $\beta$ commitments. With Cachin-Tessaro’s RBC [17], this complexity becomes $O(\beta n \log(n) + \lambda n^2 \log(n))$ per node for each $n$-parallel BAWVSS. The Gather primitive requires $O(n^2)$ comm. bits per node. The $c$-agreement phase for a round that does not use committee election ($r \mod n = 0$) requires $n$ BINAA instances, with each round of one instance costing $O(n \log(\frac{n}{\rho^2}))$ bits per node. Overall, for $O(n)$ instances running for $\log(\frac{n^2}{\rho^2})$ rounds costs $O(n^2 \log(n) \log(\frac{\rho^2}{n}))$ 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 $\beta$ beacons, the overall comm. complexity per node without committee election is $O(\beta n \log(n) + \lambda n^2 \log(n) + n^2 \log(n) \log(\frac{n}{\rho^2}) + \beta n^2 \log(n)) = O(\beta n^2 \log(n))$ per node for $\beta = O(n)$. Hence, the asymptotic complexity per beacon is dominated by the opening phase.

With committee election, each batch of $\beta$ beacons requires honest nodes to open a previous beacon with $O(\lambda n^2 \log(n))$ per node. The nodes run only $c$ $c$-agreement instances, and opening only $c$ secrets. This results in a communication complexity of $O(\beta n \log(n) + \lambda n^2 \log(n) + cn \log(n) \log(\frac{n}{\rho^2}) + \beta cn \log(n))$ per $\beta$ beacons. For $\beta = O(n)$, the communication complexity per node comes to $O(\lambda n \log(n))$ bits. As beacon generation for committee election happens once every $\phi n$ rounds, the average comm. complexity per beacon totals $\frac{\lambda n^2 \log(n)}{n} + (1 - \frac{1}{n}) \lambda n \log(n) = O(\lambda n \log(n))$ bits per beacon per node.

The computation complexity can also be calculated in a similar fashion. For rounds $r: r \mod n = 0$, each node performs $O(n^2 \log(n))$ hash computations to verify $O(n)$ secret shares each for $O(n)$ secrets part of the beacon. With committee election, each node only computes $O(cn \log(n))$ hashes for at most $c$ secrets part of the beacon. Since beacons for committee election are only generated once every $\phi n$ rounds, the average is $O(cn \log(n))$ hash computations per beacon.

6. Applications

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

6.1. Post Quantum SMR

We demonstrate the utility of HashRand by implementing a Post-Quantum asynchronous SMR protocol using HashRand for Post-Quantum Liveness. There exist asynchronous SMR protocols like DAG-RIDER [50] and FIN [34] that achieve post-quantum safety by using only symmetric key primitives like Message Authentication Codes(MACs), which are PQ-secure. However, these protocols depend on quantum-insecure random beacons like BLS threshold signatures [14] for liveness. Prior to HashRand, the PQ-secure random beacon protocol with the best communication complexity is Freitas et al. [30] with Doele et al.’s [32] 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 [24]. Our choice of TUSK is motivated by its exceptional performance in a Wide Area Network. We create PQ-TUSK by using HashRand to provide the distributed randomness necessary for electing wave leaders. We tune HashRand’s configuration parameters $\beta$ and $\phi$ to ensure that HashRand prepares exactly one beacon per wave. Since revealing a prepared beacon in HashRand takes only one round trip in HashRand, PQ-TUSK can commit blocks at network speed. However, TUSK uses quantum-insecure EdDSA signatures for certificate generation. We replace this scheme with an appropriate PQ-secure signature scheme that offers maximum performance benefits in this context. Overall, PQ-TUSK has a communication complexity of $O(\lambda n \log(n))$ 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 TUSK.

6.2. Cross-chain Consensus

We also use HashRand to create the first asynchronous cross-chain consensus protocol, which does not require a PKI setup. A cross-chain consensus protocol is a key building block in creating a combined ledger with boosted
trust from individual blockchains. Such a combined ledger is secure even when the adversary controls 1/3rd fraction of blockchains. Recent work TrustBoost [65] implements cross-chain consensus using a smart contract that runs on each participating blockchain. These contracts communicate using an inter-blockchain communication (IBC) protocol. A major challenge noted by TrustBoost is the computational expense of signature verification in prominent consensus protocols like HotStuff and Tendermint. Further, TrustBoost also notes a limitation in the underlying IBC protocol, which prevents the use of public-key signatures in cross-chain consensus. These issues motivated TrustBoost to use Information-theoretic HotStuff (IT-HS) [7], which does not use any signatures. Moreover, these issues are also the main roadblock in running asynchronous cross-chain consensus.

Using HashRand, we propose a signature and PKI-free asynchronous cross-chain consensus protocol. We use DAG-Rider [50], a DAG-based atomic broadcast protocol in combination with HashRand, to create a PKI-free asynchronous SMR protocol. DAG-Rider uses only Brach’s RBC and a common coin as building blocks, which, with HashRand, enables the protocol to commit transactions without using any signatures. This SMR protocol has a communication complexity of $O(\lambda n^2 \log(n))$ 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 [46] based on BLS threshold signatures [14]. Prior works such as Spurt [25] and OptRand [12] directly used DRand [63] as a benchmark for a beacon with threshold setup. However, we note that DRand’s codebase does not produce beacons at full throttle and thereby misrepresents the true pace of beacon generation with a threshold setup. To ensure a fair comparison, we implement the Dfinity-DVRF protocol using the Rust-based BLS signature library blst

Using HashRand, we propose a signature and PKI-free asynchronous cross-chain consensus protocol. We use DAG-Rider [50], a DAG-based atomic broadcast protocol in combination with HashRand, to create a PKI-free asynchronous SMR protocol. DAG-Rider uses only Brach’s RBC and a common coin as building blocks, which, with HashRand, enables the protocol to commit transactions without using any signatures. This SMR protocol has a communication complexity of $O(\lambda n^2 \log(n))$ 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.

7.1. HashRand evaluation

We configure HashRand to emit beacons for committee election in asynchronous SMR or blockchain sharding. For this application, the domain of beacons $D$ 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 $10^{15}$ beacons. Even with an optimistic $10^6$ beacons being invoked everyday, the average expected time for a failure is 2739 years. Assuming a domain $D$ of size 2048, we choose the domain $D'$ for BAWVSS to be a finite field of size $|D'| > 2^{50}$. Nodes invoke $B$.Open(i+1) immediately after they terminate $B$.Open(i) to generate beacons at full throttle.

**Tuning $\beta$ and $\phi$.** HashRand’s runtime configuration parameters comprise the BAWVSS batch size $\beta$, and the period of BAWVSS instantiation $\phi$. 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 $\beta$, there is an optimal $\phi$ 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.
Batch size
10
100
400
without a threshold setup and with Post-Quantum security. Currently, the best way to generate distributed randomness is through Hash-based Winternitz One-Time Signatures (WOTS), SPHINCS-256, and PICNIC based on Lattice cryptography [35], b) SPHINCS-256 [11] based on the Hash-based Winternitz One-Time Signatures (WOTS), and c) PICNIC [22] based on the MPC-in-the-head approach with symmetric key primitives. We notice that the signing and verification times coupled with the signature size of a signature scheme have the highest impact on TUSK’s latency and throughput. This is because TUSK has a $O(n)$ 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 both have high signing and verification times in milliseconds. Both schemes also have high signature sizes (40 KB and 140 KB). DiLithium requires 220 and 70 microseconds to sign and verify a message respectively, and has a modest signature size of 4 KB for 128-bits of quantum security. DiLithium has larger public and secret keys compared to PICNIC and SPHINCS-256, which is not a problem in a permissioned SMR system. Hence, we choose the DiLithium scheme to implement certificates in PQ-TUSK.

In the WAN setting, PQ-TUSK has a response rate of 135k transactions per second at $n = 16$ nodes, with an optimal latency of 2.3 seconds per transaction confirmation. This response rate is only 10% lower than classical TUSK. PQ-TUSK is also very efficient compared to other prominent and implemented PQ-secure protocols like WaterBear [69].
that uses local coins to generate distributed randomness. At \(n = 16\) in a geo-distributed setting, WaterBear offers a peak response rate of only 23k txns per second in \(t=3a\_medium\) machines (with same specs as our machines, but with burstable CPU) as compared to 135k for PQ-Tusk.

At higher \(n = 40\) and \(n = 64\), PQ-Tusk offers a peak response rate of 122k txns per second at a latency of 17.5 seconds per transaction, and 40k txns per second at a latency of 50 seconds. We do an ablation study of PQ-Tusk to identify the bottleneck at higher \(n\). We implement and observe the response rate for Tusk with DiLithium signatures instead of EdDSA. This variant of Tusk underperforms compared to our PQ-Tusk at \(n = 16\) with 20% lesser response rate at 108k txns per second. At \(n = 40\), it performs very similar to PQ-Tusk, with a peak response rate of 122k txns per second and a latency of 13 seconds. Based on this information, we identify the higher signing and verification times of DiLithium signatures over EdDSA signatures as the bottleneck for PQ-Tusk. Therefore, with HashRand, we remove the Post-Quantum liveness bottleneck that has plagued prior PQ-SMR protocols like WaterBear, and establish that randomness from Hash functions is the most practical way to achieve Post-Quantum security.

8. Related work

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

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

BRandPiper [13] is an adaptively secure, leader-based synchronous beacon protocol that has a communication complexity of \(O(n^3)\), which uses a VSS scheme with unconditionally hiding homomorphic commitments. OptRand [12] is an optimistically responsive random beacon that uses OptSync’s [62] commit rule to generate beacons at network speed. The protocol however depends on a synchronous timeout to implicate faulty leaders in the pessimistic case. Spurt is a partially synchronous protocol that uses n-parallel PVSS (requiring a Level 4 CRS setup) with leader-based secret aggregation to ensure unpredictability within \(O(n^2)\) communication. The secrecy of the PVSS scheme depends on the DBDH assumption, which along with an RO used for NIZK proofs in the protocol, provides unpredictability of the beacon’s output. Spurt internally uses the HotStuff [67] protocol to enable honest nodes to terminate and output the beacon value.

Asynchronous protocols. Asynchronous random beacon protocols are bound by Freitas et al.’s [30] impossibility result, which states that any protocol that converts local randomness from honest nodes into global randomness with agreement amongst honest nodes must have infinitely many executions. Protocols with a DKG setup (Level 5 in [6]) such as Cachin et al. [19] are not bound by this impossibility result because the output from threshold signatures is equivalent to distributed pseudorandomness and is a deterministic function of honest nodes’ states. Another protocol requiring a DKG setup is RandShare [64], which uses AVSS and Asynchronous Byzantine Agreement (ABA) to achieve agreement and unpredictability. The ABA instances consume pseudorandomness from the DKG setup to overcome the bound.

Randomness Protocols with Las-Vegas style agreement have been traditionally employed in Asynchronous Distributed Key Generation (ADKG) [2], [3], [26], [28], [37], [52]. Kogias et al. [52] propose an Eventually Perfect Common Coin (EPCC) using a high-threshold AVSS scheme conducted over a secure channel setup (Level 2). Their protocol uses the DDH assumption with a RO to prove the unpredictability of their coin. This protocol also has a \(O(n^3)\) comp. complexity per node, which makes it very expensive in practice.

Other asynchronous protocols use a partially public setup like PKI with CRS or SRS (Level 4 setup). Gao et al. [37] use a combination of PAVSS, and the binding core abstraction in GATHER to produce beacons. This work proves unpredictability using the DDH and the RO assumption. Abraham et al. [3] also use PKI-based VRFs and Gather to agree on a random node’s index. This protocol uses the SXDH assumption with a RO to achieve unpredictability. A recent work called Bingo [2] produces adaptively secure randomness using a Packed AVSS scheme, which requires a PKI and an SRS setup. Gao et al., Abraham et al., and Bingo have \(O(n^3)\) DLog expo. complexity, which makes them computationally inefficient. Such protocols can be used for a one-shot ADKG and later use Cachin et
al.’s [19] approach for generating pseudorandom beacons. This approach does not work with mobile participants and also suffers from key leakage attacks, which prompted a wave of research on mobile proactive [18], [47], [60], [68] and refreshable secret sharing [43], respectively. Moreover, Kogias et al.’s approach is the only current way to achieve ADKG at a Level 2 setup (without PKI or CRS) in [6], whose expensiveness hinders its practical application.

Freitas et al.’s [30] protocol achieves deterministic termination with a statistical probability of disagreement with a Level 2 setup (pairwise secure channels). This protocol uses AVSS, GATHER, and ε-agreement to output a random ε interval of numbers. The AVSS used in this protocol depends on Pedersen commitments and the DL 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 $\log\left(\frac{n^3}{\epsilon}\right)$ rounds.

**Information-theoretic protocols.** Protocols in the realm of IT and statistical security use different techniques to generate distributed randomness. Huang et al. [48] use statistical fraud detection by Byzantine nodes to improve the complexity of Ben-Or’s local coin-based coin flipping to $\Omega\left(n^3\right)$ from $O\left(\exp(n)\right)$, at the expense of tolerating a lesser fraction of faults. This form of work uses only authenticated channels. Another style of work by Patra et al. [57] propose a statistically secure AVSS scheme with $O\left(n^3 \log(n)\right)$ 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.

### 9. Conclusion

We presented HASHRAND, a computationally efficient practical asynchronous random beacon protocol that requires a pairwise-channel setup. HASHRAND guarantees Monte-Carlo agreement, liveness, and unpredictability by only using the properties of a one-way Hash function. HASHRAND outputs each beacon with amortized $O(\lambda n \log(n))$ 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 a classical adversary and using the Quantum ROM against a 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.

### References


Appendix

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

BAwVSS security. The Termination, Correctness, and Weak Commitment properties of our BAwVSS scheme follow Dolev et al.'s [32] scheme.

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

Proof. If the dealer $n_d$ is honest, every honest node will send out ECHOs by verifying their share tuples and their inclusion in the root vector. By the liveness and totality properties of reliable broadcast, every honest node will eventually terminate the sharing phase of the protocol, and at least $t + 1$ honest nodes will have shares for all $k$ secrets. If all nodes terminated BAwVSS.SH$(n_d)$ and start BAwVSS.REC($\cdot$), then at least $t + 1$ honest nodes must broadcast their shares to everyone. Since the degree of the sharing and nonce polynomial is $t$, these $t + 1$ points are enough to reconstruct both share and nonce polynomials and verify the validity of the root. With the help of these $t + 1$ honest nodes, every honest node terminates BAwVSS.REC($\cdot$).
We prove correctness by contradiction. If the dealer $D$ is honest, the honest nodes reach an agreement on the root vector, and at least $t + 1$ honest nodes have shares for all $k$ secrets. Assuming an honest node $i$ reconstructs $s_i \neq s_i$ shared by the dealer, the node $i$ must have reconstructed the polynomials $g_j(x) \neq g_i(x)$ and $r_j(x) \neq r_i(x)$. For this to happen, $i$ must have used a share $s_m$ not shared by the dealer $D$ in Lagrange interpolation. However, since $i$ validated the share $s_m$, the adversary must have generated a valid commitment $H(s_m, r_m)$ for a share not generated by the dealer. This implies that the adversary found a hash collision, which violates the collision resistance property of the function $H$. Hence, Algorithm 2 satisfies Correctness.

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

1) Let $i, j$ reconstruct $s_i, s_j \in \text{field}$ respectively. From the Termination property of BAWVSS, $\text{Sh}(n_d)$, both $i, j$ terminate with the same root vector. Since $s_i \neq s_j$, two different degree $t$ polynomials $g_j(x)$ and $g_j'(x)$ construct two Merkle trees with the same root, which implies that there must be at least one hash collision while aiding the disagreement. This collision violates the collision-resistance property of the Hash function $H$.

2) Let $i, j$ reconstruct $s_i \in D, \perp$ respectively. From the termination property of BAWVSS, $\text{Sh}(n_d)$, $i, j$ terminated with the same root vector. This disagreement implies that $i$ and $j$ used a different set of $t + 1$ shares to construct their polynomials $q_i(x)$ and $q_j(x)$. $i$ constructed a degree $t$ polynomial and validated all $n$ shares, which included the set of $t + 1$ shares used by $j$ to construct $q_j(x)$. Since $j$ reconstructed a different degree $t$ polynomial with the shares validated by $i$, a malicious node must have sent a wrong share $s_m$ to $j$, which had the same commitment as the corresponding share validated by $i$. This violates the collision-resistance property of the hash function $H$. $

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

Lemma A.2. Under the input hiding assumption of the hash function $H$ as stated in Boneh and Shoup [15, page 345],
the BAVSS protocol BAVSS in Algorithm 2 satisfies unconditional hiding.

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