Article | Received 26 December 2024; Accepted 12 March 2025; Published 26 March 2025
https://doi.org/10.55092/blockchain20250006

Faster dynamic asynchronous distributed random beacon

Hongjian Yang[1], Yuan Lu[1,*], Wu Chen[2], Yang Zhang[2], Rong Wei[2] and Jieyu Li[2]

[1] Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] Beijing Institute of Satellite Information Engineering, Beijing, China

* Correspondence author; E-mail: luyuan@iscas.ac.cn

Highlights:

- Efficient asynchronous distributed key reconfiguration.
- Faster dynamic asynchronous distributed random beacon.

Abstract: random beacons are crucial components in blockchain consensus, secure multiparty computation, and decentralized applications, providing high-quality randomness for these applications. However, random beacon services operated by a single organization face centralization issues and cannot be fully trusted by mission-critical applications due to possible breaches and collusion. Asynchronous distributed random beacon protocols are proposed as a promising alternative to such centralized services, since they can generate high-quality randomnesses that are unbiased and unpredictable for critical applications in the adversarial asynchronous Internet. However, they either suffer from expensive communication overhead or lack accommodation for efficient dynamic participation. To address these issues, we propose a practical asynchronous random beacon protocol that can be efficiently reconfigured to support rotations of participating nodes, reducing the reconfiguration's communication complexity from $O(\lambda n^3)$ to $O(\lambda \kappa n^2)$, where λ is the cryptography security parameter, n is the size of nodes in the network, and κ is the small size of a any-trust sub-committee (which approximates a constant number about several dozens). We also demonstrate the performance and security of our scheme through thorough analysis and extensive experiments.

Keywords: distributed random beacon; asynchronous multi-party protocol; fault-tolerance system; threshold cryptosystem reconfiguration

1. Introduction

Random beacon is a fundamental cryptographic primitive to provide high-quality unpredictable randomness, which is essential for many blockchain applications, e.g., it is a key building block for decentralized gambling and lotteries [1, 2], asynchronously secure consensus protocols [3, 4, 5], secure multiparty computation protocols for privacy-preserving smart contracts [6, 7], and more. Most random beacons in the wild are implemented through some centralized web services, allowing users to obtain fresh random numbers via HTTP requests (e.g., NIST [8], Random.org [9]). However, such random beacon services place too much trust in a single organization. That said, in the worst case if the centralized service providers are compromised, those random beacons would become fully predictable by the adversary, thus completely violating the desired security and causing severe vulnerabilities in mission-critical applications.

To address the overtrust issue of centralized random beacon services, researchers have

proposed distributed random beacon protocols [10, 11, 12, 13, 14, 15, 16, 17], making a set of distributed nodes to collectively generate high-quality randomness in a distributed manner, such that the returned randomness would be unpredictable and unbiased, as long as the number of corrupted nodes does not exceed a certain threshold. For mission-critical blockchain applications, asynchronous distributed random beacons (ADRBs) are particularly enticing, as they can thrive in the adversarial asynchronous Internet environment, despite the network messages could be arbitrarily delayed by the adversary. However, as Table 1 illustrates, the existing asynchronous distributed random beacon protocols still suffer from the following problems:

Practical ADRBs from threshold cryptosystems cannot accommodate dynamic changes of participating node. It is fairly known that ADRBs can be efficiently constructed from (non-interactive) threshold cryptosystems like BLS threshold signature [14, 18, 19]. For example, the seminal study of Cachin et al. [18] presents the first practical asynchronous random beacon protocol from Shamir secret share based non-interactive threshold cryptosystem, conditioned on computational hardness assumptions like the intractability of the Diffie-Hellman problem or the RSA problem, with a moderate communication complexity of $O(\lambda n^2)$ per generated randomness. Nevertheless, threshold cryptosystems require an initialization setup phase via so-called Asynchronous Distributed Key Generation (ADKG) [20, 21, 22, 23, 24, 25], which initially disperses the threshold private keys to a fixed set of participating nodes in a distributed manner. Such setup assumption phase fixes the identities of participating nodes, from which, each node has to obtain an exclusive threshold private key as a Shamir secret share of the master private key, and subsequently use the key share during the whole execution of ADRB protocols. Consequently, any change of participating nodes after the initialization phase might require a re-configuration to refresh the threshold private keys to accommodate the leaving nodes and the joining nodes, by executing another ADKG protocol. Unfortunately, the ADKG based re-configuration could be very complicated and extremely computationally costly (e.g. at least incurs cubic communication complexity), thus introducing cumbersome overhead when participating nodes are dynamically changing.

Other dynamic-friendly ADRBs have hindersome complexities and questionable practicability. Besides those efficient ADRBs built from pre-established threshold cryptosystems, it is also known that one can construct ADRBs on the fly without the setup assumption of threshold cryptosystems, relying on Asynchronous Verifiable Secret Sharing (AVSS) [20, 26, 27, 28, 29, 21, 22] in combination of asynchronous Byzantine agreement (ABA) modules. Here AVSS is a two-phase protocol that allows a dealer to confidentially disperse a secret across n participating parties during a sharing phase, after which a reconstructing phase can be invoke to let the honest parties collectively recover the earlier committed secret. Then, the core idea becomes that each node executes an AVSS protocol to verifiably share a randomly sampled secret, and then all nodes collectively execute some ABA protocol(s) to decide a set of completed AVSS protocols, thus inducing an unbiased randomness from those solicited AVSS protocols. However, the communication overhead of such AVSS-based approach is typically very high, often reaching $O(\lambda n^3)$. This is because the communication lower bound of any AVSS protocol is quadratic due to the seminal Dolev-Reischuk bound [30][1], so the overall communication complexity of n distinct AVSS protocols naturally causes a cubic communication overhead, which is certainly asymptotically worse than the common $O(\lambda n^2)$ complexity of ADRBs built from threshold cryptosystems, and significantly hurts the performance of such AVSS-based ADRBs. Moreover, most of these protocols [20, 26, 27, 21, 22] only realize a weak variant of distributed random beacon, such that with a non-negligible probability, their each output might be predictable by the adversary in advance, or the honest participants might even return different output values.

[1]Note that Dolev-Reischuk bound [30] is originally about Byzantine broadcast instead of AVSS, but AVSS implies an asynchronous reliable broadcast in a complexity-preserving manner, so the Dolev-Reischuk lower bound is also applied to AVSS.

Given the limitations of existing ADRB protocols, we ask the following question:

Can we design a practical asynchronous distributed random beacon protocol that simultaneously supports efficient dynamic participation, e.g., both its randomness generation and participant reconfiguration attain only (quasi-)quadratic communication overhead?

Table 1. Comparison with existing ADRB protocols and their reconfiguration approaches, where λ is the cryptographic security parameter (e.g. several hundred bits) and κ is a statistic security parameter with constant size.

Design Paradigm	**Randomness Generation Protocols**	**Per-Randomness Communication**	**Reconfiguration Protocols**	**Reconfiguration Communication**
Threshold crytosystem + ADKG	Cachin *et al.* [18]	$O(\lambda n^2)$	ADKG w/ private key over field [23]	$O(\lambda n^3)$
	[18] + Gurkan *et al.* [31]		ADKG w/ private key over group [20]	
Direclty from AVSS without pre-configured threshold crytosystem	Dolev *et al.* [29]	$O(\lambda n^3)$ or more	n.a.	n.a.
	Abraham *et al.* [21, 20]			
	Bandarupalli *et al.* [28]			
	Gao *et al.* [22]			
	Das *et al.* [32]			
Threshold crytosystem + more efficient reconfiguration	**This paper §4.2.** adapted from [18, 31, 33]	$O(\lambda n^2)$	**This paper §4.3.** our new design	$O(\lambda \kappa n^2)$

1.1. Our contribution

This paper answers the above question in the affirmative, by constructing a novel practical reconfiguration-friendly asynchronous distributed random beacon protocol from Public Verifiable Secret Sharing (PVSS) and Multi-valued Validated Byzantine Agreement (MVBA), in order to support efficient change of participating nodes and simultaneously attain low communication overhead. In greater details, our contributions can be summarized as follows:

- We design a novel and efficient asynchronous distributed random beacon protocol based on a recent threshold non-interactive signature scheme (that is efficiently realized through the variant of Shamir secret sharing over elements in the cyclic groups instead of fields). The feature further empowers efficient and aggregatable PVSS construction on top of such special kind of secret sharing, which later becomes a key factor of reducing the reconfiguration overhead by taking advantage of PVSS aggregation.
- Moreover, we design an efficient reconfiguration protocol for the above asynchronous distributed random beacon protocol, by utilizing (i) an aggregatable publicly verifiable secret sharing mechanism specially tailored for group elements and (ii) an communication-optimized asynchronous multi-valued Byzantine agreement protocol. The resulting reconfiguration phase requires only $O(\lambda \kappa n^2)$, in contrast to the $O(\lambda n^3)$ communication cost of trivial reconfiguration from ADKG. Furthermore, as the parameter κ is asymptotically constant (cf. §5.1. for the rigorous analysis of the parameter choice), our result attains asymptotic improvement over the state-of-the-art reconfiguration protocols.
- Furthermore, we prove the security properties and complexity metrics of our proposed asynchronous distributed random beacon protocol and its corresponding reconfiguration protocol. Thereby, we demonstrate the security assurance of our protocol against any probabilistic polynomial-time bounded adversary that might corrupt up to $t < n/3$ nodes

in the system,[2] and reveal the rationale of designs behind performance improvement.
- Finally, we implement the reconfiguration protocol for the proposed ADRB scheme, with another dedicated optimization of reducing the number of needed PVSS for aggregation (i.e. reducing n PVSS transcripts used during the reconfiguration phase towards a constant number independent of n), and we conduct extensive evaluations to experimentally compare with a baseline reconfiguration protocol optimized from one of the state-of-the-art ADKG protocol from Das et al. [23]. [3] As by Table 2 highlights, when $n \geq 16$ nodes, our design achieves 36%-41% reduction in the critical reconfiguration latency, demonstrating its superior practicality in the dynamic setting with rotating participants.

Table 2. Experimental comparison between our reconfiguration protocol and a naive reconfiguration protocol using an ADKG optimized from Das *et al.* [23] (cf. §6. for implementation details and concrete experimental setup).

Protocols	**Latency for varying scales**			
	$n = 4$	$n = 16$	$n = 31$	$n = 64$
Das *et al.* [23]	0.69 sec	7.20 sec	26.47 sec	111.18 sec
This work	0.50 sec (↓26.7%)	4.58 sec (↓36.4%)	15.64 sec (↓40.9%)	69.63 sec (↓37.4%)

1.2. Structure of remaining sections

The rest of the paper is organized as follows: §2 introduces the cryptographic and distributed computing primitives that are used by our designs; §3 formulates the problem of asynchronous distributed random beacon with dynamic participation; §4 explains the details of our proposed protocols, including the initialization protocol, the random beacon generation protocol, and the reconfiguration protocol; §5 formally analyzes the security and performance of our protocol design; §6 presents the experimental evaluations of our design to demonstrate its feasibility; §7 takes a careful review of the pertinent relevant studies in the synchronous setting, and discusses their limitations in contrast to this work; finally, §8 summarizes this work.

2. Preliminary

For completeness of our presentation, this section would provide a brief introduction to the cryptographic primitives and distributed protocols that would be used in our protocol design.

2.1. Multi-valued validated byzantine agreement (MVBA)

The concept of Byzantine Agreement (BA) [34] was introduced by Lamport et al. in 1982. BA ensures that all honest nodes achieve a common output in a distributed network, even in the presence of Byzantine nodes, which may arbitrarily misbehave like sending equivocating messages to different nodes or simply stay silent.

To introduce the advanced notion of multi-valued validated Byzantine agreement (MVBA) [5, 35, 36, 4], we first briefly recall the classic definition of Asynchronous Binary Agreement (ABA), which is a distributed multiparty protocol enabling all participating honest nodes to reach a binary consensus (i.e., output 0 or 1) in an asynchronous network environment, despite the presence of Byzantine nodes. MVBA, on the other hand, is a move advanced consensus notion that allows all honest nodes to agree on one of the input values from all nodes in an asynchronous network environment against $t < n/3$ colluding malicious nodes.

[2] Note that in a dynamic system, both the system scale n and the number of faulty nodes t can change dynamically. Nevertheless, we generally use n and t to denote these varying parameters unless otherwise specified or when context eliminates ambiguity.

[3] For fair comparison, both implementations adopt the same underlying cryptographic primitives and use the same cryptographic libraries. Hence, our experiments effectively demonstrate the efficiency gain from our protocol design (instead of else engineering optimizations).

Throughout this paper, we would adopt the state-of-the-art MVBA protocol Dumbo-MVBA as the instantiation for asynchronous consensus component, which is known for its asymptotically optimal communication complexity of only $O(\ell n + \lambda n^2)$ (where ℓ represents the bit-length of consensus input and λ is the bit-length of a cryptographic security parameter). In greater details, any MVBA protocol, including Dumbo-MVBA, shall satisfy the following properties except with negligible probability:

- **Termination.** If all honest parties receive value inputs to activate MVBA, each honest party outputs a value.
- **Agreement.** If any two honest parties output v and v', then $v = v'$.
- **Valiaity.** If any honest parties output a value v, then the value must be able to pass a predefined assertion.

2.2. Publicly verifiable secret sharing (PVSS)

Secret Sharing is a widely-adopted technique that can divide a secret value into n fragments called secret shares, requiring at least a threshold of shares to reconstruct the secret value. For a distributed system with up to t malicious parties, the threshold is typically $t+1$. Shamir Secret Sharing [37] is the most classical construction of secret sharing, and it begins with constructing a polynomial $f(x)$ whose zero-point $f(0)$ encodes the secret value s. As such, the polynomial can be further evaluated at n distinct (and non-zero) points x_i to obtain a set of n different evaluations y_i. These points (x_i, y_i) are then assigned to each node P_i, and y_i is the so-called Shamir's secret share, and x_i is simply a pre-specified index of P_i. Later, the secret value s can be recovered using the Lagrange interpolation approach, once any $t+1$ evaluations from distinct nodes have been collected.

Verifiable Secret Sharing (VSS) [38, 39, 40] adds a verifiability property to the Secret Sharing scheme. In VSS, the dealer distributes the secret shares along with commitments to the polynomial coefficients of the secret values. This allows participants to verify the validity of the secret shares during the secret recovery phase using these commitments.

(Non-interactive) Public Verifiable Secret Sharing (PVSS) [41, 42, 43] further extends each participant's local verifiability in VSS to public verifiability, allowing anyone to verify that the PVSS script indeed carries all participants' valid secret shares (in some encrypted form). In this paper, we can use a special *aggregatable* variant of Scrape PVSS [44] proposed by Gurkan et al. [31], we refer the unfamiliar readers to literature [31, 22] for detailed description of this aggregatable PVSS scheme. Informally speaking, a (non-interactive aggregatable) PVSS scheme consists of four algorithms/protocols: **PVSS.Deal**, **PVSS.Verify**, **PVSS.Aggregate**, and **PVSS.Reconstruct**, which executes as follows:

- **The PVSS.Deal** algorithm is used to generate the PVSS transcript, which consists of the commitments and encryptions of all participants' secret shares.
- **The PVSS.Aggregate** algorithm can be used to aggregate several different PVSS transcripts into a single PVSS transcript, such that, the resulting secret shares (carried by the aggregated PVSS) is also combined as the summation of previous shares.
- **The PVSS.Verify** algorithm allows each participant to verify the validity of the PVSS transcript, such that a valid PVSS script would carry the ciphertext encrypting the valid secret shares of all participants. Moreover, the verification function can also verify an aggregated PVSS script, such that telling the identities of nodes contributing in this aggregated PVSS.
- **The PVSS.Reconstruct** protocol is an interactive procedure executed by all (honest) participants to collectively reconstruct the shared secret carried by a certain PVSS transcript, despite the arbitrary influence of the malicious ones. Here the PVSS transcript can also be an aggregated one.

In addition to the aggregatable PVSS scheme proposed by [31], we may consider employing a PVSS scheme based on Paillier encryption combined with Pedersen/Feldman

polynomial commitments. This approach enhances efficiency and supports secret sharing over fields, instead of being restricted to groups. However, the Paillier encryption based PVSS lacks aggregatability. To address this, our dedicated implementation (cf. §6.1.) incorporates a specialized aggregation technique that selects random PVSS transcripts from a sub-committee with an honest majority. This ensures that even a straightforward concatenation of PVSS transcripts, taken from half the participants in the honest-majority subcommittee, maintains an asymptotically linear size in n.

2.3. Non-interactive unique threshold signature

(Non-interactive unique) threshold signature (TSG) [33, 45] is a representative scheme of distributed cryptosystem, enabling a sufficiently large subset of multiple users to collaborate on signing any given message (through a single step of communication). In a distributed network with n users denoted as $\{P_i\}_{i\in[n]}$, the scheme ensures that a subset of $n-t$ (honest) users can cooperate to robustly generate the signature for any message m, but it is computationally infeasible for t colluding (malicious) users to produce a valid signature for all messages without the cooperation of other $n-2t$ honest users, which is known as unforgeability of threshold signature. More detailedly, a threshold signature scheme consists of the following algorithms:

- **TSG.KeyGen**: It generates the public key pk, a vector of individual public keys $\{pk_i\}_{i\in[n]}$, and a vector of secret key shares $\{sk_i\}_{i\in[n]}$ of the private key sk. The public keys pk and $\{pk_i\}_{i\in[n]}$ are published to all participants, and each secret key share sk_i is sent exclusively to the participant P_i. Though this step can be performed by a centralized algorithm through some trusted third-party (TTP) service, we do not assume such TTP and would invoke a distributed protocol executed by all participants, called ADKG [25, 20, 21, 22], to conduct the setup in a decentralized manner.
- **TSG.PartialSign**: Each participant P_i partially signs the given message using their secret share of the private key sk_i, thus obtaining a share of signature σ_i (which is also called the i-th "partial" signature).
- **TSG.PartialVerify**: Every participant P_i can verify that σ_j is a valid partial signature computed by P_j regarding the given message, using P_j's individual public key pk_j;
- **TSG.Combine**: Given a set of $n-t$ valid partial signatures from distinct participants regarding the given message, anyone can aggregate these $n-t$ signature shares into a valid "full" signature σ of the message, which is known as robustness of threshold signature; Note that as we require the threshold signature scheme unique, namely, all $n-t$ subsets of valid signature shares would be combined into the unique σ.
- **TSG.Verify**: everyone can verify the validity of the aggregated signature σ using the public key pk.

In the paper, we can adopt the non-interactive unique threshold signature scheme due to Gurkan *et al.* [31] if we adopt the same authors' proposal of aggregatable PVSS. However, since we can also adopt Paillier encryption based PVSS to extend the secret key sharing over standard prime fields, BLS threshold signature scheme can also be used (thanks to that our implementation §6.1. adopts an aggregating technique to support Paillier encryption based PVSS without hurting communication efficiency). According to data points reported in [31], BLS threshold signature would outperform in both performance and size, and we stick with this more efficient approach in our implementation.

2.4. Cryptographic hash function as random oracle

A hash function $H : \{0,1\}^* \leftarrow \{0,1\}^\lambda$ can map arbitrary input strings to an output with fixed length. Usually, a cryptographic hash function H is considered to be collision-resistance, i.e., it is computationally infeasible for any P.P.T. adversary to find a pair of $x \neq x'$ s.t. $H(x) = H(x')$

with non-negligible probability. In this paper, we consider an ideal model of cryptographic hash functions—random oracle [46]. Essentially, during our security analysis, we replace H with an ideal black-box functionality (called random oracle). For each input that was not queried before, the random oracle samples an output uniformly at random, and it also internally maintains a table to record all queried inputs and their corresponding outputs, such that for some already queried input, it can return what it earlier has outputted.

2.5. *Erasure coding*

A (k,n)-erasure code scheme [47] consists of two deterministic algorithms **Enc** and **Dec**. The **Enc** algorithm maps any vector $\mathbf{v} = (v_1, \cdots, v_k)$ of k data fragments (called message word) into an vector $\mathbf{m} = (m_1, \cdots, m_n)$ of n coded fragments (called code word), such that any k elements in the code word $\mathbf{m}$ would be sufficient to reconstruct the message word $\mathbf{v}$ due to the **Dec** algorithm. More formally, a (k,n)-erasure code scheme has the following syntax:

- $\mathbf{Enc}(\mathbf{v}) \to \mathbf{m}$. On input a vector $\mathbf{v} \in \mathscr{B}^k$ (message word), this *deterministic* encode algorithm outputs a vector $\mathbf{m} \in \mathscr{B}^n$ (code word). Note that $\mathbf{v}$ contains k data fragments and $\mathbf{m}$ contains n coded fragments, and $\mathscr{B}$ denotes the field of each fragment.
- $\mathbf{Dec}(\{(i,m_i)\}_{i\in S}) \to \mathbf{v}$. On input a set $\{(i,m_i)\}_{i\in S}$ where $m_i \in \mathscr{B}$, and $S \subset [n]$ and $|S| = k$, this *deterministic* decode algorithm outputs the message word $\mathbf{v} \in \mathscr{B}^k$.

We require (k,n)-erasure code scheme is *maximum distance separable*, namely, the original data fragments $\mathbf{v}$ can be recovered from any k-size subset of the coded fragments $\mathbf{m}$, which can be formally defined as:

- **Correctness of erasure code**. For any $\mathbf{v} \in \mathscr{B}^k$ and any $S \subset [n]$ that $|S| = k$, $\Pr[\mathbf{Dec}(\{(i,m_i)\}_{i\in S}) = \mathbf{v} \mid \mathbf{m} := (m_1, \cdots, m_n) \leftarrow \mathbf{Enc}(\mathbf{v})] = 1$. If a vector $\mathbf{m} \in \mathscr{B}^n$ is indeed the code word of some message word $\mathbf{v} \in \mathscr{B}^k$, we say the $\mathbf{m}$ is well-formed; otherwise, we say the $\mathbf{m}$ is ill-formed.

Through the paper, we consider a $(t+1,n)$-erasure code scheme where $3t+1 = n$. Besides, we emphasize the erasure code scheme would implicitly choose a proper field $\mathscr{B}$ according to the actual length of each element in $\mathbf{v}$, such that the encoding causes only constant blow-up in size, namely, the bits of $\mathbf{m}$ are larger than the bits of $\mathbf{v}$ by at most a constant factor. There are a few well-known instantiations of such primitive like Rabin's [48], Reed-Solomon [49] and numerous their variants.

2.6. *Position-binding vector commitment*

For an established position-binding n-vector commitment (**VC**), there is a tuple of algorithms $(\mathbf{VCom}, \mathbf{Open}, \mathbf{VerifyOpen})$. On input a vector $\mathbf{m}$ of any n elements, the algorithm **VCom** produces a commitment vc for the vector $\mathbf{m}$. On input $\mathbf{m}$ and vc, the **Open** algorithm can reveal the element m_i committed in vc at the i-th position while producing a short proof π_i, which later can be verified by **VerifyOpen**.

Formally, a position-binding **VC** scheme (without hiding) is abstracted as:

- $\mathbf{VC.Setup}(\lambda, n, \mathscr{M}) \to pp$. Given security parameter λ, the size n of the input vector, and the message space $\mathscr{M}$ of each vector element, it outputs public parameters pp, which are implicit inputs to all the following algorithms. We explicitly require $\mathscr{M} = \{0,1\}^*$, such that one **VC** scheme can commit any n-sized vectors.
- $\mathbf{VCom}(\mathbf{m}) \to (vc; aux)$. On input a vector $\mathbf{m} = (m_1, ..., m_n)$, it outputs a commitment string vc and an auxiliary advice string aux. We might omit aux for presentation simplicity. Note we do not require the hiding property, and then let **VCom** to be a deterministic algorithm.
- $\mathbf{Open}(vc, m_i, i; aux) \to \pi_i$. On input $m_i \in \mathscr{M}$, $i \in [n]$, the commitment vc and advice aux, it produces an opening string π to prove that m_i is the i-th committed element. We might omit aux for presentation simplicity.

- **VerifyOpen**$(vc, m_i, i, \pi_i) \rightarrow 0/1$. On input $m_i \in \mathscr{M}$ and $i \in [n]$, the commitment vc, and an opening proof π, the algorithm outputs 0 (accept) or 1 (reject).

An already established **VC** scheme shall satisfy **correctness** and **position binding**:

- **Correctness of VC**. An established **VC** scheme with public parameter pp is correct, if for all $\mathbf{m} \in \mathscr{M}^n$ and $i \in [n]$, $\Pr[\mathbf{VC.VerifyOpen}(vc, m_i, i, \mathbf{VC.Open}(vc, m_i, i, aux)) = 1 \mid (vc, aux) \leftarrow \mathbf{VC.VCom}(\mathbf{m})] = 1$.
- **Position binding**. An established **VC** scheme with public parameter pp is said position binding, if for any P.P.T. adversary A, $\Pr[\mathbf{VC.VerifyOpen}(vc, m, i, \pi) = \mathbf{VC.VerifyOpen}(vc, m', i, \pi') = 1 \wedge m \neq m' \mid (vc, i, m, m', \pi, \pi') \leftarrow A(pp)] < negl(\lambda)$, where $negl(\lambda)$ is a negligible function in λ.

There are a few simple solutions [50, 51, 52] to achieve the above position-binding **VC** notion without hiding. A simplistic example is hash Merkle tree [50], where vc is a $O(\lambda)$-bit hash value committing a vector of messages organized as a binary hash tree, and the openness π is $O(\lambda \log n)$-bit as it carries .

3. Problem formulation

For rigorousness, this section would formulate the problem solved in the paper by introducing our security modeling, including the considered system, threats, and security goals. Namely, we will precisely define the admissible abilities of the adversary to reflect the realistic threats, and provide the desired security goals that we aim at realizing against such the adversary. Along the way, we will also present the complexity metrics that would be used to measure the efficiency of asynchronous random beacon protocols.

3.1. System and threat modeling

We extend the standard model of reliable asynchronous fully-meshed message-passing network with authenticated peer-to-peer connections [53, 4] into the setting of dynamic participation. For the purpose, we consider the set of participating nodes is changing by consecutive configurations $C^{(0)}, C^{(1)}, C^{(2)}, \cdots$, where each configuration $C^{(i)}$ has $n^{(i)}$ participating nodes with up to $t^{(i)} < n^{(i)}/3$ malicious corruptions, in the spirit of the standard practice of dynamic fault-tolerance systems [54, 55, 56].

In greater details, our system and threat modeling can be formalized as follows:

- **Public key infrastructure.** We let the universe U denote the set of all potential participating nodes in our dynamic ADRB system. Every node in U has its public keys (which can be used for digital signature or public key encryption) known by the whole universe. In practice, this can be facilitated through some bulletin board based public key infrastructure (e.g., either some centralized certificate authority or some decentralized alternative approach such as proof of stake [55, 56]).
- **Initial configuration.** Initially, there is a committee of n participating nodes $\{P_1, P_2, ..., P_n\}$, denoted as $C^{(0)}$, which is a subset of the nodes in universe U. This initial committee might execute some initialization protocol, and then continuously generate a sequence of random number, distributedly, by executing the ADRB protocol, until the committee of participating nodes are reconfigured.
- **Dynamic participation by reconfiguration.** To accommodate the desired join of some newly spawning nodes and the planned departure of some current participating nodes, we let the system "periodically" conduct reconfiguration to rotate the committee of participating nodes. That said, once if the current committee $C^{(i)}$ has generated a specific number of random beacons (e.g., K), the committee $C^{(i)}$ would stop generating randomness, and a new committee $C^{(i+1)}$ (which is still a subset of nodes in universe U but might probably consist of a list of participants very different from the previous configuration $C^{(i)}$'s participants) would be launched to replace the role of the committee

$C^{(i)}$ to distributedly generate the coming K random numbers, until the next committee $C^{(i+2)}$ will be launched.

- **Reliable authenticated asynchronous network.** We consider that the communication network is fully asynchronous without any form of network timing assumption, indicating that all messages might suffer from an unpredictable delay of transmission, although they would eventually deliver their destinations. Additionally, for any two nodes P_i and P_j in the universe U, a secure authenticated channel can be established, such that all messages sent between any two honest nodes couldn't be tampered, although they can be arbitrarily delayed or reordered.
- **Computationally bounded adversary controlling up to $n/3$ Byzantine corruptions.** We assume there is a probabilistic polynomial-time (P.P.T.) bounded static adversary A in the network capable of corrupting up to $t^{(i)} < n^{(i)}/3$ nodes for each configuration $C^{(i)}$. Here "static" means that the adversary would choose the set of corrupted parties before the protocol starts. The adversary can coordinate the attacks of all corrupted nodes and also schedule the delay of all network messages, as long as its attacking strategies are P.P.T. computable.

REMARK ON RECONFIGURATION. During the reconfiguration phase, when the new committee $C^{(i+1)}$ just launches, the old committee $C^{(i)}$ might still stay online to execute some auxiliary protocol to help the new committee efficiently complete some necessary setup of threshold cryptosystem, thus empowering the new committee to efficiently generate random beacons after the execution of such reconfiguration phase.

3.2. Security goals

Informally speaking, a (dynamic) asynchronous distributed random beacon protocol is said secure, if it realizes the following properties with all but negligible probability w.r.t. the security model described in Section 3.1.:

- **Liveness.** The protocol would continuously generate a sequence of random values (with bounded per-randomness communication complexity), despite the influence of reconfiguration, malicious corruptions, and purely asynchronous network.
- **Consistency.** For any two honest nodes, their generated random value (at any index of their output sequence) is same.
- **Unbiasability and unpredictability.** The adversary A cannot manipulate the distribution of the output random values, i.e., it is infeasible for A to make the distribution (computationally) distinguishable from the uniform distribution. Moreover, we require an even stronger security assurance that A also cannot predict any generated random value better than guessing, before the first honest node releases its exclusive secret share of this generated randomness.

3.3. Quantitative efficiency metrics

Besides security, we are also interested in constructing efficient ADBR protocols. For the purpose, we consider the following standard quantitative metrics as performance indicators of asynchronous fault-tolerant protocols:

- **Communication complexity**. We consider the standard notion of bit communication complexity, which characterizes the expected number of bits sent by the honest parties during the protocol.
- **Round complexity**. We follow the standard approach due to Canetti and Rabin [53] to measure the running time of protocols by asynchronous rounds. Essentially, this measurement counts the number of communication "steps", when the protocol is embedded into a lock-step timing model.

4. Protocols for dynamic asynchronous random beacon

Now we are ready to elaborate on the detailed construction of our dynamic ADRB protocol, which can achieve all desired security properties including liveness, unbiasability and unpredictability, in a dynamic fully asynchronous network with resilience against arbitrary P.P.T. attacks, as long as the computationally bounded adversary controlling less than $n^{(i)}/3$ malicious nodes for each configuration $C^{(i)}$.

Overview. As briefly shown in Figure 1, our dynamic ADRB construction consists of three asynchronous fault-tolerant protocols, which are responsible for initialization, reconfiguration, and random beacon generation, respectively. The initialization protocol is essentially an asynchronous distributed key generation (ADKG) protocol, such that the participants of the initial configuration can distributedly set up a threshold cryptosystem proposed by Gurkan *et al.* [31] for threshold signature. Furthermore, the beacon protocol can leverage the resulting threshold signature to efficiently generate a beacon of randomness in a distributed manner, assuming the random oracle model. Moreover, when the configuration is rotating, i.e., the members of $C^{(i)}$ stop generating beacon and the participants of $C^{(i+1)}$ start, the reconfiguration protocol is invoked to help the new configuration efficiently set up a new threshold cryptosystem for threshold signature (in the assistance of the old configuration), which would be asymptotically more efficient than merely executing another ADKG.

The detailed execution flows of our initialization, reconfiguration and beacon protocols can be respectively described as follows.

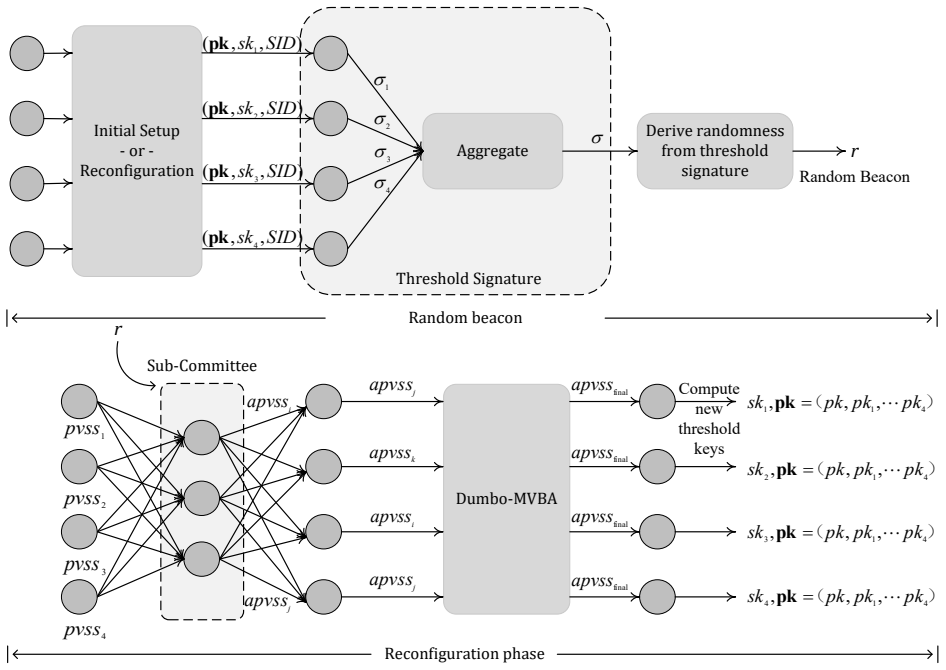

Figure 1. The execution flow of our re-configurable ADRB protocol (exemplified by the case of no joining or leaving node).

4.1. Initialization

Before the initial configuration's participants can collectively generate the random values, we first let them set up the threshold cryptosystem due to Gurkan *.et al.* [31], which later can be used for threshold signature. For the purpose, we can leverage the state-of-the-art ADKG protocols [21, 22] for Gurkan *.et al.*'s threshold cryptosystem. For the completeness of presentation, we would re-introduce the high-level rationales behind these designs hereunder,

but refrain from repeating their tedious security analysis, and recommend readers interested in more details refer to the original publications [21, 22].

4.1.1. Private-setup free MVBA in the absence of established threshold cryptosystem

Recall two facts that (i) the celebrated FLP "impossibility" [57] states that no deterministic protocols can ever exist to solve the problem of asynchronous agreement against even a single crash corruption and (ii) ADKG implies asynchronous Byzantine agreement and therefore is also subject to the FLP "impossibility". Therefore, we necessarily rely on a randomized asynchronous Byzantine agreement protocol as a core building block to facilitate the procedure of ADKG, which is quite non-trivial, since we face a circular problem: (i) to efficiently implement ADKG for threshold cryptosystem, we might require an efficient asynchronous Byzantine agreement protocol, and (ii) to realize some asymptotically efficient asynchronous Byzantine agreement, an established threshold cryptosystem (for coin flipping [4, 18]) is usually desired in practice.[4]

That said, we have to overcome the challenge to realize some efficient asynchronous Byzantine agreement in the absence of any threshold cryptosystem. For the purpose, we adopt the state-of-the-art private-setup free coin flipping protocol in asynchrony [22], which can be plugged into any existing MVBA protocols [5, 3, 35, 36, 4] to replace their coin flipping component requiring the private setup of established threshold cryptosystem. The resulting private-setup free MVBA protocols attain expected $O(\ell n^2 + \lambda n^3)$ communication complexity and expected $O(1)$ round complexity, where ℓ is the bit-length of MVBA input and λ represents the length of cryptographic security parameter. Here we can also adopt other common-coin based private-setup free asynchronous consensus protocols [32, 21].

4.1.2. Expected constant-round ADKG for Gurkan *.et al.'s* threshold cryptosystem

Given the efficient private-setup free MVBA, we are readily to implement an efficient ADKG protocol for Gurkan et al.'s threshold cryptosystem that can terminate in expected $O(1)$ round with $O(\lambda n^3)$ communication complexity. Such ADKG simulates a trusted functionality that (i) generates a public key pk, a vector of individual public keys $\{pk_i\}_{i\in[n]}$, and a vector of secret key shares $\{sk_i\}_{i\in[n]}$ of some private key sk corresponding to the public key pk, and also (ii) publishes pk and $\{pk_i\}_{i\in[n]}$ to the whole network, and privately distributes each secret key share sk_i to the corresponding node P_i.

Specially, the ADKG based initialization protocol is executed by all initial participants in $C^{(0)}$, and can be straightforwardly constructed as follows:

- Each participating node P_i samples a private secret s_i, computes an aggregatable non-interactive PVSS script pvss_i sharing the secret, and then multicasts pvss_i to all participating nodes. Note that the implementation of aggregatable non-interactive PVSS is also from Gurkan et al. as an extension of SCRAP PVSS.
- Then, everyone node P_i waits for $n-f$ PVSS scripts sent from distinct nodes, combines them to obtain an aggregated PVSS script apvss_i, and then takes apvss_i as input to one MVBA protocol (with external validity specified to check the input is indeed an PVSS script aggregating $n-f$ distinct nodes' PVSS).
- Finally, each node can get the common aggregated PVSS script returned by the MVBA protocol, which carries the desired public keys pk and $\{pk_i\}_{i\in[n]}$, and more importantly, every node P_i can decrypt the returned PVSS script to get its own secret key share sk_i.

[4]We remark that there is another approach using local coins (instead of common coins from threshold cryptography) to realize randomized asynchronous Byzantine agreement protocols [58]. However, such local-coin protocols suffer from a prohibitively large round complexity that is exponentially large in the number of participating nodes. Moreover, the exponential round complexity is not just an efficiency issue, it is also illed in our computationally-bounded setting (though these protocols are fine in the information-theoretic setting), because the computing ability of the adversary is at least linear in round complexity. That means, an exponential round complexity enables an adversary to have an exponentially-bounded time to break any computationally-secure cryptographic primitives (e.g., the standard signatures and polynomial commitments). This corresponds to another reason that we focus on common-coin based protocols.

4.2. Random beacon generation

Once Gurkan *et al.*'s threshold cryptosystem has been established among the participating nodes (i.e. every node receives all the public keys and its own exclusive private key share), these distributed nodes can collectively compute randomness values, in an efficient manner with a single round of communication and quadratic communicated bits per randomness.

In particular, the random beacon generation protocol during each configuration $C^{(i)}$ is executed by all its participating nodes as described in Algorithm 1:

- Every node P_i invokes the **TSG.PartialSign** function using its own secret key share sk_i to compute a partial signature σ_i, on input a common message denoted as $C^{(i)}||k$, where $C^{(i)}$ represents the current number of configuration and $k = \{1, 2, \cdots, K\}$ are consecutive numbers repressing the k-th random value generated during the course of the current configuration $C^{(i)}$.
- Then every P_i multicasts σ_i, and waits for sufficient valid partial signatures sent from a set S of $t+1$ distinct participating nodes, so it can invoke the **TSG.Combine** function to aggregate $\{\sigma_j\}_{j\in S}$ as the final full signature σ. Finally, everyone can obtain a public common random value $H(\sigma)$, where H is a cryptographic hash function modeled as random oracle.

The above process of generating randomness would be sequentially iterated with the increment of variable k, until $k > K$. After that, the current configuration $C^{(i)}$ would hand over the duty of beacon generation to the new configuration $C^{(i+1)}$, which would involve our reconfiguration protocol in order to efficiently set up the threshold cryptosystem among the new configuration, as described in the next subsection.

REMARK ON SUPPORTING CONCURRENT APPLICATIONS USING DIFFERENT LABEL STRINGS. As illustrated in Algorithm 1, we can easily leverage the same setup among the participants to launch multiple random beacons for various applications, by specifying a unique string of identifier prefix (denoted as *LABEL*) to each application. The security intuition behind the approach is straightforward: due to the threshold version EUF-CMA unforgeability of threshold signature, even if the adversary already queries many valid signatures on messages with a valid prefix *LABEL*, it still cannot forge a valid threshold signature for another unqueried message with prefix *LABEL*$'$. That said, the other concurrently executing beacon protocols with different application prefixes would not mutually influence their unpredictability.

Algorithm 1 Random Beacon Generation with an application identifier *LABEL*

// Code for each node P_i in the configuration $C^{(i)}$
// Note that every P_i has obtained $(pk, pk_1, \cdots, pk_{n^{(i)}}, sk_i)$ from ADKG (4.1) or reconfiguration (4.3)
1: **for** $k \in \{1, 2, \cdots, K\}$ **do**
2: $\quad\sigma_i \leftarrow$ **TSG.PartialSign**$(sk_i, LABEL||C^{(i)}||k)$
3: $\quad$**multicast** the message $(C^{(i)}, k, \sigma_i)$ to all nodes in the universe U
4: $\quad$**enter** the reconfiguration protocol
// Code for every node in the universe
5: $\quad$**wait** for valid $\{\sigma_j\}_{j\in S}$ sent from a set S of $t+1$ distinct nodes s.t.
$\qquad$**TSG.PartialVerify**$(pk_j, LABEL||C^{(i)}||k, \sigma_i) = 1$
6: $\quad\sigma \leftarrow$ **TSG.Combine**$(\{\sigma_j\}_{j\in S})$
7: $\quad$**return** $r = H(\sigma)$ as the k-th random value during configuration $C^{(i)}$
Note: we can use multiple different *LABEL*s to instantiate several independent random beacon protocols from the same setup, such that these independent instantiations can be dedicatedly used for varying applications.

4.3. Reductive reconfiguration

Careful readers might note that the protocol for beacon generation (4.2) necessarily requires a pre-established setup of Gurkan et al.'s threshold cryptosystem, but we only earlier

Algorithm 2 Reconfiguration (code for each node P_i in the configuration $C^{(i)}$ or $C^{(i+1)}$ where $i \geq 0$)

// Code for each node P_i in the new configuration $C^{(i+1)}$

send message $(\textbf{Election}, C^{(i+1)})$ to all nodes in the old configuration $C^{(i)}$
wait for receiving valid $\{\sigma_j\}_{j\in S}$ sent from $t^{(i)}+1$ distinct nodes in the old configuration $C^{(i)}$, s.t. $\textbf{TSG.PartialVerify}(pk_j, C^{(i)}||\text{“}Elect'', \sigma_i) = 1$
$\sigma \leftarrow \textbf{TSG.Combine}(\{\sigma_j\}_{j\in S})$
compute $r_{elect} = H(\sigma)$
use r_{elect} to randomly permute the identities of nodes in the new configuration $C^{(i+1)}$, and select the first κ nodes to form a sub-committee $SC^{(i+1)}$
each P_i samples a private randomness s_i, invokes **PVSS.Deal** to compute a PVSS transcript pvss_i sharing s_i, and multicast pvss_i to all nodes in the recently selected sub-committee $SC^{(i+1)}$
if P_i is in the sub-committee $SC^{(i+1)}$ **then**
    wait for $t^{(i+1)}+1$ valid $\{\text{pvss}_\text{j}\}_{\text{j}\in\text{S}}$ sent from distinct nodes in $C^{(i+1)}$
    combine $\{\text{pvss}_\text{j}\}_{\text{j}\in\text{S}}$ to form an aggregated PVSS script apvss_i
    send the aggregated PVSS script apvss_i to all nodes in the old configuration $C^{(i)}$
wait for receiving the same m_j codeword from at least $t^{(i)}+1$ distinct nodes in the old configuration $C^{(i)}$
    send m_j to all nodes in the new configuration $C^{(i)}$
wait for receiving codewords $\{m_j\}$ from at least $2t^{(i+1)}+1$ distinct nodes in the new configuration $C^{(i+1)}$
    try to decode $\text{apvss}_\text{final}$ from the received $2t^{(i+1)}+1$ codewords
    if decoding fails:
        while for receiving more codeword from a different node in the new configuration
            try to decode until $\text{apvss}_\text{final}$ can be successfully decoded
decrypt the exclusive secret key share sk_i from the ciphertext carried by $\text{apvss}_\text{final}$
compute the public keys $pk, pk_1, \cdots, pk_n$ from the commitment carried by $\text{apvss}_\text{final}$
finish reconfiguration and **start** beacon generation (4.2) using key materials of the established threshold cryptosystem (i.e., sk_i, $pk, pk_1, \cdots, pk_n$)

// Code for each node P_i in the old configuration $C^{(i)}$

wait for $n^{(i+1)} - t^{(i+1)}$ $(\textbf{Election}, C^{(i+1)})$ messages sent from distinct nodes in the new configuration $C^{(i+1)}$
$\sigma_i \leftarrow \textbf{TSG.PartialSign}(sk_i, C^{(i)}||\text{“}Elect'')$
send σ_i to all nodes in the old configuration $C^{(i)}$ and the new configuration $C^{(i+1)}$
wait for receiving valid $\{\sigma_j\}_{j\in S}$ sent from $t^{(i)}+1$ distinct nodes S which is a subset of the old configuration $C^{(i)}$, s.t. $\textbf{TSG.PartialVerify}(pk_j, C^{(i)}||\text{“}Elect'', \sigma_i) = 1$
$\sigma \leftarrow \textbf{TSG.Combine}(\{\sigma_j\}_{j\in S})$
compute $r_{elect} = H(\sigma)$
use r_{elect} to randomly permute the identities of nodes in the new configuration $C^{(i+1)}$, and select the first κ nodes to form a sub-committee $SC^{(i+1)}$
wait for the first valid apvss_j (which aggregates $t^{(i+1)}+1$ pvss scripts from different nodes in the new configuration $C^{(i+1)}$) from any node sub-committee $SC^{(i+1)}$
start an MVBA protocol running among the old configuration $C^{(i)}$ using apvss_j as input (where the external validity of MVBA is specified to validate the input is indeed a PVSS script aggregating $t+1$ pvss scripts from different nodes in the new configuration $C^{(i+1)}$, and all common randomnesses required by MVBA are generated using Algorithm 1 from the existing random beacon setup with letting the application identifier $LABEL$ to be “$MVBA$”)
wait for MVBA outputs an aggregated PVSS script $\text{apvss}_\text{final}$
chunk $\text{apvss}_\text{final}$ into $t^{(i+1)}+1$ fragments represented as $(m_1, \cdots, m_{t^{(i+1)}})$
run Cauchy Reed-Solomon algorithm [59] to encode $(m_1, \cdots, m_{t^{(i+1)}})$ into $n^{(i+1)}$ Reed-Solomon codewords represented as $(m_1, \cdots, m_{n^{(i+1)}})$
for each P_j in the new committee $C^{(i+1)}$: send Reed-Solomon codeword m_j to it
terminate

describe how to implement such setup for the initial configuration $C^{(0)}$ through ADKG (4.1) and haven't yet presented how to realize such setups for all the following configurations including $C^{(1)}$, $C^{(2)}$, $\cdots$ etc. In this subsection, we will complete this remaining crucial part by presenting our reconfiguration protocol, which would be reductively executed by two consecutive configurations $C^{(i)}$ and $C^{(i+1)}$, such that the previous configuration $C^{(i)}$ with

already-established threshold cryptosystem can help the new configuration $C^{(i+1)}$ set up Gurkan et al.'s threshold cryptosystem as well.

As Algorithm 2 describes, our reconfiguration phase executes as follows:

- **Electing a sub-committee** (lines 1-5, lines 21-27). The new configuration participants would send the old configuration members a message of $(\textbf{Election}, C^{(i+1)})$, such that if a participant P_i of the old configuration receives $n^{(i+1)} - t^{(i+1)}$ such messages, P_i would compute a partial threshold signature σ_i and multicast it to all nodes of the two consecutive configurations, which means all nodes can expect to receive $t^{(i)}+1$ such partial signatures and derive a common randomness $r_{elect} = H(\sigma)$ from the aggregate signature σ. So a κ-sized sub-committee $SC^{(i+1)}$ can be elected out of the new configuration, as all nodes can locally compute the identities of $SC^{(i+1)}$ using the common randomness r_{elect} as seed. Note that κ is a statistical security parameter (for electing a small κ-sized any-trust group from a large set with 2/3-super honest majority), which typically is merely a couple of dozens due to earlier cryptographic practice that now has been quit standard.
- **Sub-committee members aggregate PVSS** (lines 6-10). Once the sub-committee is publicly elected according to the common randomness generated by the old configuration, all new configuration participants would compute a PVSS script sharing a uniformly sampled random value and then send the PVSS script to all nodes in the sub-committee. As such, each honest node in the sub-committee would eventually receive at least $t^{(i+1)}+1$ valid PVSS scripts sent from distinct nodes, thus aggregating them. If the aggregation of PVSS scripts is completed, the sub-committee members would multicast the aggregated PVSS to all nodes in the old configuration.
- **Old committee decides a unique aggregate PVSS** (lines 29-30). With all but negligible probability, the nodes in the old configuration can eventually receive a valid aggregated PVSS script from some node in the elected sub-committee (which also combines at least $t^{(i+1)}+1$ different nodes' PVSS scripts). As such, the old configuration participants would start an MVBA protocol using the received aggregated PVSS as input, and then, the MVBA protocol would enable the old configuration participants to decide a common and valid aggregated PVSS script.

  We remark that the MVBA protocol requires unpredictable common randomnesses (a.k.a. common coins) to realize expected constant-round termination against a fully asynchronous adversary. It is fortunate that the old committee members have a sufficient setup to generate such common coins. More specifically, for the k-th invocation to common coins within MVBA, the old committee members just follow Algorithm 1 to exchange their partial signatures on "$MVBA$"$||C^{(i)}||k$, then form the unique threshold signature on "$MVBA$"$||C^{(i)}||k$, and finally hash the threshold signature to obtain the $k-th$ common coin used within this MVBA.
- **Old committee transfers the finalized PVSS to new committee** (lines 31-34, lines 11-17). Note that the old configuration members do not necessarily need the aggregated PVSS decided by MVBA, and they have to transfer the PVSS script to the new committee members. The trivial idea of letting the old configuration members simply broadcast the PVSS script would incur cubic communication, thus failing to attain our desired quadratic complexity. To this end, we use the technique of online Reed-Solomon decoding [49, 59] used in [60], which enables the old committee reliably transfer the PVSS script to the new committee, yet still preserves the desired quadratic communication complexity.

  In greater detail, every old committee member encodes the PVSS script according to the size of the new committee (line 31-32), such that each new committee member is corresponding to a particular code fragement of the PVSS script. Then, each new participant can receive its corresponding code fragement from at least $t^{(i)}+1$ old members, where $t^{(i)}+1$ is greater than 1/3 of the old committee size (line 11). Note that this code fragement sent from $t^{(i)}+1$ old members must be the correctly computed

code fragement of the true PVSS script agreed by MVBA (because there are only $t^{(i)}$ malicious members in the old committee), so the new committee members can further exchange their correct code fragements, thus finally decoding the correct PVSS script.

- **Key derivation from decided PVSS** (lines 18-20). Finally, all nodes in the new configuration can receive a sufficient number of honest codewords encoding the final PVSS script and thus decode it, enabling them obtain all necessary key materials (including all public keys and the private key share) necessary for the new epoch of ADRB execution.

REMARK ON COMMUNICATION-EFFICIENT IMPLEMENTATION. In the above protocol, MVBA protocol must be extension protocols that can accommodate large input. The suggested choice is Dumbo-MVBA [3] that uses vector commitment (VC) and erasure code to realize optimal communication, otherwise, careless choice of instantiation (e.g., using the designs from Cachin et al.'s [4], Abraham et al.'s [5] or Guo et al.'s [36] MVBA protocols) might cause cubic communication cost. Noticeably, Dumbo-MVBA requires its participants to have an already-established threshold signature scheme for functionality and efficiency, which is why we choose the old committee instead of the new committee to execute MVBA (because the new committee does not yet establish threshold signature).

Our reconfiguration phase may be of independent interest, as it efficiently realizes an asynchronous distributed key reconfiguration (ADKR) protocol, which enables an asynchronous fault-tolerant system with an existing threshold cryptosystem to efficiently generate a new threshold cryptosystem for a reconfigured set of participants. Informally, ADKR can be viewed as a simplified version of ADKG, where an already-established threshold cryptosystem among a set of current participants is leveraged to distributedly generate a new threshold cryptosystem for a reconfigured participant set. Compared to the previous ADKR protocol [61], which relies on an honest-majority sub-committee of at least several hundred nodes to achieve quadratic communication complexity, our design significantly improves efficiency. As we will show in the following analysis, our approach requires only an any-trust sub-committee consisting of a few dozen nodes while maintaining the desired security and performance guarantees.

5. Analysis

This section will analyze the security of our proposed asynchronous distributed random beacon protocol by proving that it satisfies unbiasability, unpredictability, and liveness. Then, we analyze the performance of the protocol to give the communication complexity of the protocol. Note that in this section, without loss of generality, we let each epoch's committee has the same number n of participants for presentation simplicity.

5.1. Analysis for sub-committee size

In this section, we will analyze in a quantitative way that the number of committee nodes tends to a constant value as the number of nodes increases.

Lemma 1 Assuming the security of non-interactive unique threshold signature scheme and cryptographic hash function as random oracle, there is at least one honest node in the randomly elected sub-committee, with all but negligible probability in the sub-committee size κ.

Proof 1 Recall that there are $3t+1$ nodes in the network, of which $2t+1$ are honest and t are malicious. Let κ be the number of sub-committee nodes and K represent the number of honest nodes in the sub-committee. Then the probability that $K \geq 1$, i.e. the chance of selecting at least one honest party in the sub-committee, is thereby:

$$\Pr(K \geq 1) = 1 - (\frac{t}{3t+1})^{\kappa} \geq 1 - (\frac{1}{3})^{\kappa} \tag{1}$$

if cryptographic hash function is random oracle and non-interactive unique threshold signature scheme cannot be predicted by the adversary. This is because if an adversary can efficiently predict the threshold signature (which is the only computationally feasible way to guess the output of random oracle), then we can construct an efficient adversary to forge threshold signature, which violates the security properties of the threshold signature scheme.

This completes the proof as $(\frac{1}{3})^{\kappa}$ is a negligible function in κ.

Given the above statement, we can further have the following simple corollary to decide the concrete size of sub-committee $SC^{(i+1)}$ as a constant number independent to the system scale n, for any desired statistic security level.

Corollary 1 To ensure that the selected sub-committee $SC^{(i+1)}$ contains at least one honest node (i.e. indeed an any-trust sub-committee) with probability of at least $1-p$, the (minimal) size κ of sub-committee $SC^{(i+1)}$ is asymptotically constant in n.

Proof 2 First, a trivial lower bound of κ is:

$$\kappa = t + 1 \tag{2}$$

which is because $t+1$ nodes contain at least one honest node (since there are at most t malicious nodes).

Second, another constraint of sub-committee size κ is $(\frac{1}{3})^{\kappa} \leq p$, which renders another non-trivial lower bound:

$$\kappa = \lceil \log_3(\frac{1}{p}) \rceil \tag{3}$$

Therefore, the tight lower bound of κ is $\min(t+1, \lceil \log_3(\frac{1}{p}) \rceil) = \min(\lfloor \frac{n-1}{3} \rfloor + 1, \lceil \log_3(\frac{1}{p}) \rceil)$. For a given p, the term $\lceil \log_3(\frac{1}{p}) \rceil$ is independent to n and thus asymptotically smaller than the term $\lfloor \frac{n-1}{3} \rfloor$, resulting in an asymptotic (tight) lower bound $\kappa \geq \lceil \log_3(\frac{1}{p}) \rceil$.

This completes the proof of the asymptotic behavior of sub-committee size κ.

As an exemplary example to justify the size of any-trust sub-committee is indeed very small, we can parameterize $p = 10^{-10}$, which intuitively means a failure of any-trust sub-committee selection happens after expected 10^{10} reconfigurations. If ADRB is reconfigured after every hour, then this translates into a sub-committee failure after more than 1,140,795 years in expectation, which clearly is a sufficiently conservative choice of statistic security level. Given such an overly conservative failure probability $p = 10^{-10}$, we still can use a very small any-trust sub-committee size $\kappa = \lceil \log_3(\frac{1}{p}) \rceil = 21$. In practice, if we use a less conservative failure probability such as $p = 10^{-8}$ (which is a widely adopted statistic secure parameter in many sharding blockchains [62, 63], and still corresponds to expected one million years to encounter a single failure of any-trust sub-committee selection, if ADRB is reconfigured for about every 5 days), then sub-committee size κ can be even smaller $\lceil \log_3(10^8) \rceil = 17$. Clearly, any-trust sub-committee is very small in practice.

5.2. Security proof for ADRB

Proof for liveness. We first prove the liveness of our dynamic ADRB protocol, by showing its randomness generation phase (Algorithm 1) and reconfiguration phase (Algorithm 2) can terminate, according to Lemma 2 and Lemma 3, respectively.

Lemma 2 If all honest nodes enter the random beacon generation protocol, then all honest nodes terminate from the protocol, with all but negligible probability.

Proof 3 Recall that we consider an optimal asynchronous adversary that can corrupt up to t nodes in the network where $t < \frac{n}{3}$. During the random beacon generation phase, the adversary can prevent at most t malicious nodes from sending any messages, while all other $n-t$ honest nodes would sign and broadcast the $SID||epoch$ using their private keys sk_i. That means, every

honest node can eventually collect a sufficient number (at least $n-t$) valid threshold signature shares $\{\sigma_j\}_{j\in S}$. This enables them to successfully aggregate the set of signature shares with all but negligible probability, and subsequently derive the random beacon r, otherwise, either the correctness or robustness of the threshold signature scheme is violated.

While if the correctness or robustness of the threshold signature scheme can be breached with non-negligible probability, a contradiction arises, as we adopt secure threshold signature scheme. This completes the proof.

Lemma 3 If all honest nodes enter the reconfiguration protocol, then all honest nodes terminate from the termination protocol, with all but negligible probability.

Proof 4 In the reconfiguration phase, the adversary controls t nodes to either refrain from sending messages or delay messages sent by honest nodes. Since the number of committee nodes is less than $t+1$, there are still more than $t+1$ nodes in the network capable of executing PVSS and MVBA. Additionally, according to Lemma 1, our sub-committee is an any-trust group with at least one honest party with all but negligible probability. This ensures that non-sub-committee nodes eventually receive at least one correctly aggregated PVSS transcript, allowing all honest nodes to enter MVBA protocol and thus terminate the reconfiguration phase (otherwise MVBA's termination is violated).

However, since we adopt secure MVBA and PVSS, the lemma must hold, otherwise, there is a contradiction that the computationally-bounded adversary breaks the security properties of secure MVBA or PVSS, or it can break Lemma 1 (indicating that it can forge threshold signature). This completes the proof.

Theorem 1 The dynamic ADRB protocol proposed in this paper ensures liveness, in the presence of a computationally bounded fully asynchronous adversary corrupting up to $t^{(i)} < |C^{(i)}|/3$ nodes for each committee $C^{(i)}$.

Proof 5 Following Lemmas 2 and 3, our proposed protocol guarantees termination in all phases, even when the adversary statistically corrupts up to t nodes (in each committee) and can arbitrarily delay message transmission in the fully asynchronous network. Therefore, our proposed asynchronous distributed random beacon protocol ensures liveness.

Proof for consistency. Then we demonstrate that the honest nodes generate the common random values. The crux of proving the statement is showing that the reconfiguration protocol can set up a threshold cryptosystem that is consistent among all honest nodes. Namely, they return the same public keys $pk, pk_1, \cdots, pk_n$ to all honest nodes in $C^{(i+1)}$ and return a secret key share sk_i to each node $P_i \in C^{(i+1)}$ s.t. $pk_i = g^{sk_i}$, and more importantly, $\{sk_i\}$ are evaluations at the same polynomial.

Lemma 4 If all honest nodes enter the reconfiguration protocol, then with all but negligible probability, each honest node $P_i \in C^{(i+1)}$ terminates from the protocol with obtaining its exclusive private key $\{sk_i\}$ and the common public keys $pk, pk_1, \cdots, pk_n$, s.t. (i) $pk_i = g^{sk_i}$ and (ii) any $n-t$ group elements from $pk_1, \cdots, pk_n$ can interpolate pk.

Proof 6 The lemma is reducible to the agreement and external validity properties of MVBA: (I) the agreement property ensures that any two honest nodes obtain the same aggregated PVSS script from MVBA; and (ii) the external validity property ensures the returned aggregated PVSS script must be valid, s.t. it contains a polynomial commitment that is consistent with the common $pk, pk_1, \cdots, pk_n$ and it carries a ciphertext that can be decrypted by each P_i to obtain the correct secret key sk_i.

Lemma 5 If Lemma 4 holds, then the honest nodes in $C^{(i+1)}$ can collectively generate common random values.

Proof 7 This is reducible the uniqueness of threshold signature scheme: if a consistent threshold cryptosystem is established according to Lemma 4, then the honest nodes can collectively produce the unique threshold signature, from which, the common random value can be derived (as hash function is deterministic).

Theorem 2 The dynamic ADRB protocol proposed in this paper ensures consistency, in the presence of a computationally bounded fully asynchronous adversary corrupting up to $t^{(i)} < |C^{(i)}|/3$ nodes for each committee $C^{(i)}$.

Proof 8 The statement is true by following Lemma 4 and Lemma 5. Particularly, Lemma 4 ensures that the reconfiguration protocol can setup a consistent threshold cryptosystem, and Lemma 5 further guarantees that, from such consistently generated threshold cryptosystem, consistent random values can be continuously produced.

Proof for unbiasability and unpredictability. Then we are ready to prove the unbiasability and unpredictability of our proposed ADRB protocol. To do so, we first show the reconfiguration scheme does not leak the honest nodes' private keys to the adversary, and then argue the unforgeability of threshold signature, which implies the unbiasability and unpredictability of generated randomness in the random oracle model.

Lemma 6 It is computationally infeasible for the adversary to guess the private key share of any honest node that is obtained from the reconfiguration protocol.

Proof 9 This is reducible to the secrecy of PVSS scheme. Given that the private key share of each honest node is an aggregation of $t+1$ secret shares containing at least one share from some honest node, each node's private key share is uniformly sampled. Therefore, if the adversary predicts the secret key share, it literally breaks the hardness problem of discrete logarithm, i.e. it can efficiently computes r from public key g^r where r is randomly sampled.

Theorem 3 The asynchronous distributed random beacon protocol proposed in this paper satisfies unbiasability and unpredictability.

Proof 10 Conditioned on the correctness of Lemma 6, this theorem is reducible to the unforgeability of threshold signature scheme in the random oracle model. Considering that the random value $r = H(\sigma)$ where H is a cryptographic hash function model as random oracle and σ is a threshold signature for a fresh new message $C^{(i)}||k$, if the adversary can predict or bias the distribution of r with a non-negligible probability, then the adversary can query either bias the distribution of random oracle's output (which arises a contradiction to the definition of random oracle), or it can have non-negligible probability to predict σ, which can be translated into a non-negligible to forge a valid threshold signature.

5.3. Complexity analysis of ADRB

First, we analyze the complexity of the reconfiguration phase. Non-committee nodes must send PVSS transcripts to committee nodes. After aggregation by the committee nodes, the PVSS transcript is broadcast to all non-committee nodes. The communication complexity of this stage is $O(\lambda\kappa n^2)$, where κ is the number of nodes in any-trust sub-committee. As already analyzed in §5.1., κ is asymptotically a small constant about 20 or even less. Then, each node takes the aggregated PVSS transcript as input to a Dumbo-MVBA protocol. The communication complexity in this phase is $O(\lambda n^2)$. Finally, the old committee members encode the Dumbo-MVBA and send the codeword to new committe members, such that the new committee can reconstruct the same Dumbo-MVBA output and thus finish the reconfiguration. This step also costs $O(\lambda n^2)$. By summing the communication complexity of the above three major phases, the total communication complexity of our reconfiguration protocol is $O(\kappa\lambda n^2)$.

Then we analyze the communication complexity of randomness generation. There is only one step of communication: each node multicast a threshold signature share, resulting in an overall communication complexity of $O(\lambda n^2)$. The remaining steps of random beacon generation are executed locally without incurring additional communication overhead. Therefore, the collective communication cost of each generate random value is $O(\lambda n^2)$.

6. Performance optimization, implementation, and experiments

6.1. *Reducing PVSS needed for aggregation towards better performance*

Every participant computes a PVSS transcript and broadcasts it, which unfortunately is very expensive as PVSS is computationally intensive. Later, these PVSS transcripts seem to be necessary as they are needed to aggregate, and we require there are at least one PVSS sent from honest node used for the aggregation, which is critical to ensure the secrecy of private key shares generated from the reconfiguration protocol.

Careful readers might quickly identify a straightforward optimization: it is unnecessary for all parties to broadcast a PVSS. Instead, it suffices for only $2t+1$ nodes to do so, ensuring that everyone eventually receives $t+1$ PVSS transcripts (which will include at least one honest PVSS) for aggregation. To implement this improvement, we simply modify line 6 in Algorithm 2 as follows: each P_i, *if it is one of the* $2t+1$ *participants with smallest indices*, samples a private randomness s_i, invokes **PVSS.Deal** to compute a PVSS transcript pvss_i sharing s_i, and multicast pvss_i to all nodes in the recently selected sub-committee $SC^{(i+1)}$.

Nevertheless, despite the above straight optimization, the number of PVSS is still $2t+1 = O(n)$, which is not satisfied, as when instantiating our ADRB design, we attempt to further reduce the number of needed PVSS towards constant instead of $2t+1$ which is still linear in n. Our key observation towards realizing a constant number of PVSS needed for aggregation is that we only need an honest majority sub-group $SC'^{(i+1)}$ of participants to compute and multicast PVSS transcripts, and we actually can use the established randomness beacon to sample such honest-majority sub-group (which has $n/2$ nodes to be honest) from the super honest-majority set of all participants (which has $n/3$ nodes to be honest).

To realize the above idea, we can update line 6 in Algorithm alg:reconfig as follows: all participants invoke the random beacon established among the old committee to get a randomness r, use r to randomly permute the set of $C^{(i+1)}$ and select the first η nodes (after random permutation) as another sub-committee $SC'^{(i+1)}$ (which is different from $SC^{(i+1)}$); then for each $P_i \in SC'^{(i+1)}$, it samples a private randomness s_i, invokes **PVSS.Deal** to compute a PVSS transcript pvss_i sharing s_i, and multicast pvss_i to all nodes in $SC^{(i+1)}$. Subsequently, we also need to replace line 8 in Algorithm 2 with the following execution, to reflect that there are only a sub-committee $SC'^{(i+1)}$ that are still sending PVSS scripts to nodes in the sub-committee $SC^{(i+1)}$: wait for $\lceil \frac{SC'^{(i+1)}+1}{2} \rceil$ valid $\{\text{pvss}_\text{j}\}_{\text{j}\in S}$ sent from distinct nodes in $SC'^{(i+1)} \subset C^{(i+1)}$.

The security of the above optimization is still guaranteed by the following lemma, which states that the sub-committee $SC^{(i+1)}$ has an honest majority.

Lemma 7 Given a super honest-majority committee C consisting n nodes with $2n/3$ honest nodes, then its random η-size sub-committee SC' is honest-majority, with all but negligible probability in η; and asymptotically, the minimal choice of η is constant and independent to n when n becomes large, for any fixed failure probability p of selecting malicious-majority SC'.

Proof 11 Recall that there are $3t+1$ nodes in the network, of which $2t+1$ are honest and t are malicious. Let η be the number of sub-committee nodes and K represent the statistic representing the number of honest nodes in the sub-committee. Clearly, statistic K is a binomial random variable following distribution $K \sim Bin(\eta, p)$, where $p = 2/3$ when n approaches infinite (as p has an asymptotic behavior approximating $\lim_{t\to\infty} \frac{2t+1}{3t+1}$).

According to the lower-tail Chernoff bound, we have the following inequality:

$$\Pr(K \leq (1-\delta)\mathrm{E}(K)) \leq \exp(-\frac{\delta^2 \mathrm{E}(X)}{2}) \tag{4}$$

where $\mathrm{E}(K) = \frac{2\eta}{3}$ is the expectation of binomial statistic K, and $\delta \in (0,1)$ to ensure that the above inequality holds.

Then, if we parameterize $\delta = \frac{1}{4}$, the probability of failed sub-committee selection SC'

with $K \leq \frac{\eta}{2}$ (i.e. the chance of selecting a malicious-majority sub-committee) can thereby be expressed as follows, using the inequality from the lower tail of Chernoff bound:

$$\Pr(K \leq \eta/2) = \Pr(K \leq (1-(1/4))\mathrm{E}(K)) \leq \exp(-\frac{(1/4)^2\mathrm{E}(X)}{2}) = \exp(-\frac{\eta}{48}) \quad (5)$$

which is clearly a negligible probability decreasing exponentially in η. This completes the proof of the first part of the lemma.

Then we prove the second part of the lemma as a simple corollary of the above proof. For any desired failure probability $\Pr(K \leq \eta/2) = p$, we can have the minimal size of η:

$$\eta \geq \lceil -48\ln(p) \rceil \quad (6)$$

whose lower bound clearly is a constant number asymptotically irrelevant to n. This complete the proof.

Therefore, we can leverage the above fact to randomly select another honest-majority sub-committee SC' to compute and multicast PVSS transcripts, which in combination with another any-trust committee SC that receive and aggregate PVSS transcripts, significantly reduce the overall overhead caused by the expensive PVSS operations.

6.2. Implementation details and setup of experiments

Our protocol is implemented in the programming language of Python 3. We use existing and widely-adopted libraries for cryptographic operations, such as pypairing, betterpairing, hashlib, phe, etc. For handling concurrent asynchronous tasks, we use gevent library to start multiple micro-threads. Communication between each pair of nodes P_i and P_j is performed over two TCP sockets—one from P_i to P_j and the other from P_j to P_i. Each node consists of there separated processes: one process for handling our ADRB protocol, one process to handle message sending, and another process to handle message receiving. For fair comparison, we use the same programming language, same cryptographic libraries, and same communication layer to implement a baseline benchmark, which is an optimized variant of Das et al.'s ADKG protocol [23] by replacing its overly complex MVBA component with the state-of-the-art Dumbo-MVBA protocol. For PVSS scheme used in both implementations, we adopt the Paillier encryption based approach. We also use BLS12-381 (Barreto-Lynn-Scott family with an embedding degree of 12) as the underlying elliptic curve instantiation, which is a pairing friendly elliptic curve implementation widely adopted in numerous real-world applications like the implementations of KZG polynomial commitments and Groth16 zk-SNARK. For VC scheme, we adopt the paradigm of hash tree from Merkle [64].

Finally, we evaluate the baseline protocol and our ADRB protocol by deploying their implementations in a high-performance server that installs Ubuntu 20.04 TLS and is equipped with a 2.7 GHz Intel Xeon Platinum 8280 CPU having 224 cores and 1 TB main memory. We conducted experiments of both implementations under varying number of nodes for $n =$ 4, 16, 31, and 64.[5] We measure two empirical metrics for reconfiguration: latency and communication overhead. Here latency represents the time spent by the honest nodes to obtain for the public keys and exclusive private key of the threshold cryptosystem for the new epoch, and communication overhead (per node) represents how many bytes sent by each node during the reconfiguration phase. For beacon generation, we measure the (average) latency spent by the honest nodes to collectively generate each random value. We execute the experiment under each scale for at least twice, in order to obtain each data point averaged over different executions and all participating nodes.

[5]For our choices of system scales that are relatively small (e.g. not greater than 64), the sizes of sub-committees SC and SC' are $t+1$ and $2t+1$ respectively (instead of asymptotic constant for larger system scales). As aforementioned, such choices of committee sizes are necessary in small systems to ensure SC and SC' to be any-trust group and honest-majority group, respectively.

6.3. *Evaluation results: efficiency of beacon generation and reconfiguration*

Performance of the reconfiguration protocol. In Figures 2 and 3, we compare our reconfiguration protocol with a baseline benchmark optimized from the state-of-the-art ADKG protocol of Das *et al.* [23] from the aspects of execution latency and communication cost, respectively.

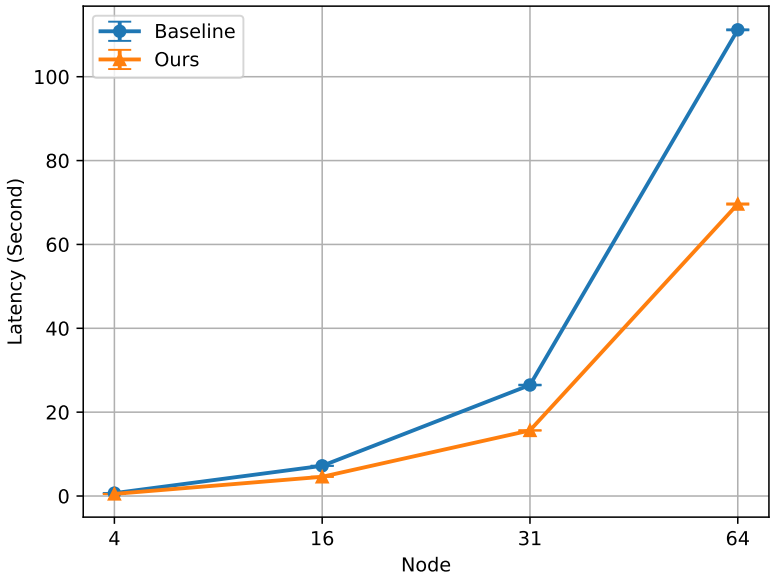

Figure 2. Our reconfiguration vs. the baseline based on ADKG: execution latency.

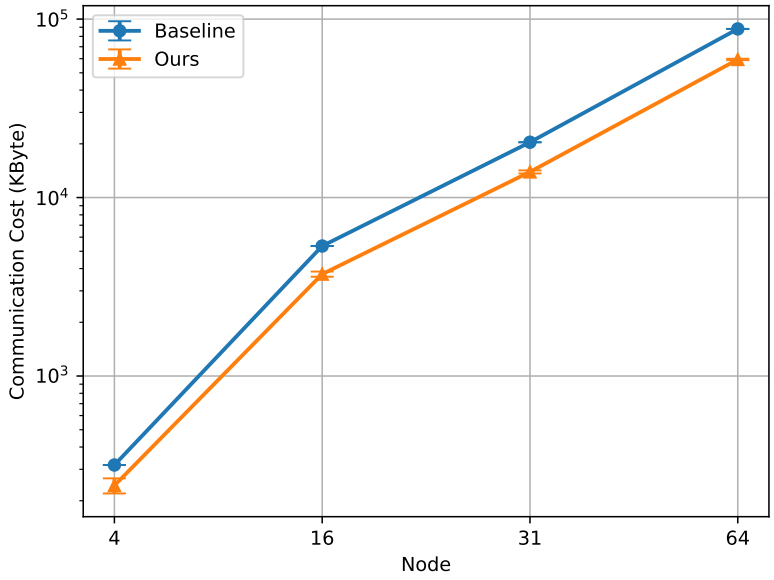

Figure 3. Our reconfiguration vs. the baseline based on ADKG: per-node communication.

Regarding the execution latency, as the number of nodes increases, the performance gain of our scheme becomes increasingly evident compared to the baseline scheme. This is because, in our scheme, only a small honest-majority subset of nodes is required to compute and broadcast PVSS to another even smaller any-trust sub-committee, whereas in the baseline

scheme, all honest nodes must broadcast PVSS to all other nodes. Consequently, as the number of nodes grows, the efficiency gain of our scheme becomes more pronounced, as it saves an increasing number of PVSS operations, which are computationally expensive.

Regarding communication overhead, our scheme demonstrates significantly better performance compared to the baseline scheme. This improvement arises because, in our scheme, only a small honest-majority subset of nodes needs to compute and broadcast PVSS transcripts to an even smaller any-trust sub-committee, greatly reducing the number of transferred PVSS transcripts. Furthermore, a deeper analysis of the data points plotted in Figure 3 reveals that our proposed scheme exhibits a higher standard deviation compared to the baseline. This is due to our usage of random sub-committees for sending and receiving PVSS transcripts, causing nodes within the sub-committees to send more bytes than nodes outside the sub-committees. Nevertheless, even the most communication-intensive node in our scheme is significantly more efficient than in the baseline scheme.

Performance of distributed generation of random value. To demonstrate the efficiency of asynchronous distributed randomness generation, we implemented two different existing approaches for distributedly generating randomness beacon: one from threshold BLS signature [19, 33] (denoted by BLS-Beacon in Figure 4) and the other one from the standard Σ-protocol for zero-knowledge (zk) proof-of-knowledge (PoK) of Diffie-Hellman (DH) tuple [18, 65] (denoted by DH-Beacon in Figure 4). Noticeably, we refrain from using the VUF scheme from Gurkan [31] et al. for efficiency consideration. From Figure 4, we can tell that DH-Beacon is much faster than BLS-Beacon, which is very understandable since DH-Beacon avoid the computationally heavy operations of cryptographic pairing. Nevertheless, BLS-Beacon has an additional property of public verifiability, which means every party across the globe can verify the random value that is indeed generated from the current epoch's committee. Therefore, there could be a meaningful property-performance trade-off between the two approaches of BLS-Beacon and DH-Beacon, and we suggest practitioners to choose one out of them according to the demands of their application setting.

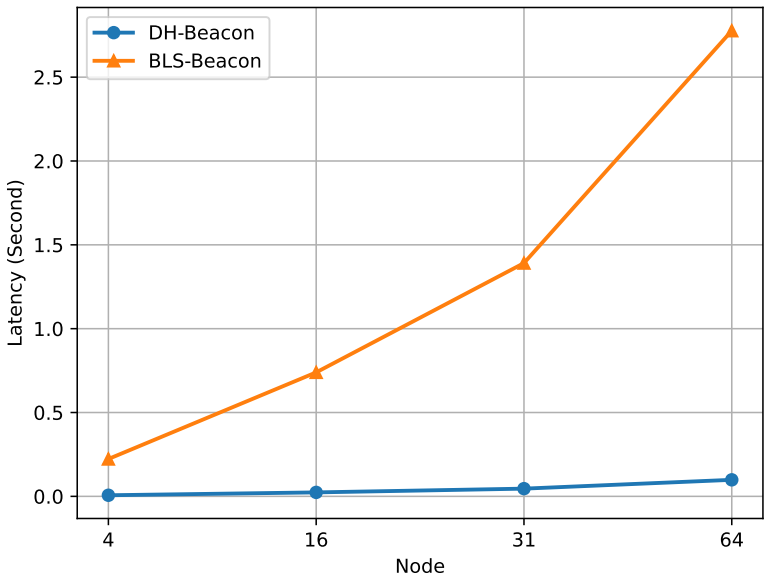

Figure 4. Latency of per-randomness generation: the approach based on BLS signature (BLS-Beacon) vs. the approach based on zk-PoK for DH tuple (DH-Beacon).

7. Additional related work

Besides the closely related studies about asynchronous distributed random beacon that we have already discussed in the introduction, there are also a line of studies on the topic of distributed random beacon in the fundamentally different synchronous setting. This section will briefly introduce their techniques and limitations.

Choi *et al.* [66] provide a comprehensive framework for constructing distributed random beacons, introducing commonly used construction blocks. They analyze and compare the security aspects (unbiasedness and unpredictability) of previous research works. Additionally, they identify corresponding attack vectors and propose countermeasures. A careful analysis of the performance and applicability of these previous works is also conducted.

RandShare [11], an asynchronous random beacon protocol designed by Syta et al., uses VSS with BA, resulting in a communication complexity of $O(\lambda n^3)$, which is not suitable for large-scale node deployment. To address this, they proposed the RandHound protocol, which can be deployed on a large scale. RandHound reduces communication complexity from $O(\lambda n^3)$ to $O(\lambda nc^2)$ by grouping servers and performing secret sharing within each group, where c is the group size. Furthermore, they introduced the RandHerd protocol, which further reduces communication overhead to $O(\lambda c^2 \log n)$. RandHerd divides the participating nodes into subgroups through a one-time setup phase and utilizes aggregation with a communication tree to generate random values, achieving significant communication overhead reduction.

Schindler *et al.* constructed Hydrand [12], a distributed random beacon protocol that does not use Distributed Key Generation (DKG) and does not require a trusted setup. Using PVSS with a variant of repeated BA, and assuming synchronous networks, Hydrand achieves a communication complexity of $O(\lambda n^2)$.

Bhat *et al.* constructed the random beacon protocols of GRandPiper (Good Pipelined Random Beacon) and BRandPiper (Better RandPiper) [67] under the synchronous network model, using a state machine replication protocol with (publicly) verifiable secret sharing. GRandPiper utilizes PVSS with a communication complexity of $O(\lambda n^2)$. BRandPiper, on the other hand, attains adaptive security by using VSS with a communication complexity of $O(\lambda f n^2)$, where f is the actual number of faulty nodes and $f \leq t$ if t is the upper bound of fault tolerance. Additionally, they designed a synchronous reconfiguration mechanism with a communication overhead of $O(\lambda n^2)$. Bhat *et al.* [16] also constructed a linear-sized Aggregatable PVSS with an optimistically responsive synchronous SMR protocol. Based on this, they developed a random beacon protocol, OptRand, which has a communication complexity of $O(\lambda n^2)$ and is $O(t)$ rounds faster than RandPiper regarding reconfiguration.

Das *et al.* used PVSS with a state machine replication protocol under a semi-synchronous network model to construct a distributed stochastic beacon protocol called SPURT [10], with a communication complexity of $O(\lambda n^2)$. SPURT optimizes the aggregation phase of PVSS by allowing each node to calculate the future leader node in advance and send the share for subsequent rounds. The future leader node starts aggregation upon receiving the share, decentralizing communication and computation across various phases. This approach reduces the communication overhead of the leader nodes in the aggregation phase. In terms of computation, the scheme enhances computational efficiency by using a multi-exponential operation technique to further improve performance.

However, all above results rely on some form of network synchrony for ensuring security, and unavoidably, they fail to guarantee the basic securities in a fully asynchronous network considered by us. While in the asynchronous setting, as discussed in the introduction, most distributed key reconfiguration protocols adapted from ADKG exhibit cubic communication complexity. While an earlier distributed key reconfiguration protocol [61] achieves quadratic communication complexity, it necessitates a committee consisting of several hundred nodes, making its effectiveness questionable in typical system scales. Notably, a very recent follow-up

work by Feng et al. [68], as inspired by this study, further reduces the required committee size to several dozen nodes while simultaneously ensuring the critical adaptive security.

8. Discussions and conclusion

Limitations and applications. We note that our reconfiguration protocol achieves only static security. However, this limitation is mitigated by the fact that the system undergoes periodic reconfiguration. In particularly, in many sharding blockchain systems, periodic reconfiguration is also considered sufficient to defend against mildly adaptive adversaries—those that can adaptively corrupt selected committees but only after a delay longer than the reconfiguration period. Another limitation of our approach is its reliance on non-interactive PVSS with aggregability, which currently supports only group-element secret shares and is therefore incompatible with popular discrete logarithm (DLog)-based cryptosystems that require secret shares over prime fields. Developing an adaptively secure and DLog-compatible distributed reconfiguration protocol while maintaining low complexity remains an interesting problem, which is only answered very recently in a follow-up work [68] inspired by this study.

Despite these limitations, our design in this paper still enables broader applications in the asynchronous distributed computing systems. In particular, it facilitates dynamic participation in proof-of-stake blockchains that rely on asynchronous fault-tolerant consensus, as the key challenge in such settings is efficiently re-configuring the asynchronous random beacon when the set of participants are periodically rotating due to the change of stakes.

Concluding remark. We introduce a dynamic asynchronous practical random beacon protocol that leverages Dumbo-MVBA to achieve more efficient consensus on aggregated PVSS, significantly reducing communication overhead during the reconfiguration phase. The protocol guarantees agreement, liveness, and unpredictability and unbiasness. Our theoretical analysis shows that the overall communication complexity for each reconfiguration is $O(\kappa\lambda n^2)$, where λ is the cryptographic security parameter (e.g., the bit length of an element in the group over an elliptic curve) and κ is a small statistical security parameter (typically 20 or less). Compared to the existing reconfiguration approach optimized from ADKG, our reconfiguration protocol is significantly more efficient, achieving a greatly reduced latency—approximately only 60% of the previous result—when the system scale $n \geq 16$ nodes.

Acknowledgments

This work is partially supported by National Key R&D Project of China (No. 2022YFB2701600), NSFC (No. 62102404), CAS Project for Young Scientists in Basic Research (No. YSBR-035), and the Youth Innovation Promotion Association CAS.

Conflicts of Interests

The authors declared that they have no conflicts of interests.

Authors' contribution

Conceptualization, Y.H.J., L.Y.; Investigation, Y.H.J., L.Y.; Writing - original draft, Y.H.J., L.Y., C.W., Z.Y., W.R., L.J.Y; Writing - review & editing, Y.H.J., L.Y.; Project administration, L.Y., Y.H.J.; Supervision, L.Y.; All authors have read and agreed to the published version of the manuscript.

References

[1] Bentov I, Kumaresan R, Miller A. Instantaneous Decentralized Poker. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security*, Hong Kong, China, December 3–7, 2017, pp. 410–440.

[2] Kumaresan R, Moran T, Bentov I. How to Use Bitcoin to Play Decentralized Poker. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Denver, CO, USA, October 12–16, 2015, pp. 195–206.

[3] Lu Y, Lu Z, Tang Q, Wang G. Dumbo-MVBA: Optimal Multi-valued Validated Asynchronous Byzantine Agreement, Revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, Virtual Event, Italy, August 3–7, 2020, pp. 129–138.

[4] Cachin C, Kursawe K, Petzold F, Shoup V. Secure and Efficient Asynchronous Broadcast Protocols. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference*, Santa Barbara, California, USA, August 19–23, 2001, pp. 524–541.

[5] Abraham I, Malkhi D, Spiegelman A. Validated Asynchronous Byzantine Agreement with Optimal Resilience and Asymptotically Optimal Time and Word Communication. *arXiv* 2018, arXiv:1811.01332.

[6] Lu D, Yurek T, Kulshreshtha S, Govind R, Kate A, *et al.* HoneyBadgerMPC and AsynchroMix: Practical Asynchronous MPC and its Application to Anonymous Communication. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, London, UK, November 11–15, 2019, pp. 887–903.

[7] Zyskind G, Nathan O, Pentland A. Enigma: Decentralized Computation Platform with Guaranteed Privacy. *arXiv* 2015, arXiv:1506.03471.

[8] Information Technology Laboratory Computer Security Division. Interoperable Randomness Beacons: CSRC, 2021. Available: https://csrc.nist.gov/projects/interoperable-randomness-beacons (Accessed on 16 Sep. 2024).

[9] Mads Haahr. True Random Number Service, 2021. Available: https://www.random.org/ (Accessed on 16 Sep. 2024).

[10] Das S, Krishnan V, Isaac IM, Ren L. Spurt: Scalable Distributed Randomness Beacon with Transparent Setup. In *2022 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, May 22–26, 2022, pp. 2502–2517.

[11] Syta E, Jovanovic P, Kogias EK, Gailly N, Gasser L, *et al.* Scalable Bias-Resistant Distributed Randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, May 22–24, 2017, pp. 444–460.

[12] Schindler P, Judmayer A, Stifter N, Weippl E. Hydrand: Efficient Continuous Distributed Randomness. In *2020 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, May 18–20, 2020, pp. 73–89.

[13] Schindler P, Judmayer A, Hittmeir M, Stifter N, Weippl E. RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness. In *28th Annual Network and Distributed System Security Symposium (NDSS)*, virtually, February 21–25, 2021 .

[14] Hanke T, Movahedi M, Williams D. Dfinity Technology Overview Series, Consensus System. *arXiv preprint arXiv:1805.04548* 2018 .

[15] Han R, Yu J, Lin H. RandChain: Decentralised Randomness Beacon from Sequential Proof-of-Work. *IACR Cryptol. ePrint Arch.* 2020 2020:1033.

[16] Bhat A, Shrestha N, Kate A, Nayak K. OptRand: Optimistically Responsive Reconfigurable Distributed Randomness. In *30th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, February 27–March 3, 2023 .

[17] Cherniaeva A, Shirobokov I, Shlomovits O. Homomorphic Encryption Random Beacon. Cryptology ePrint Archive, Paper 2019/1320, 2019.

[18] Cachin C, Kursawe K, Shoup V. Random Oracles in Constantipole: Practical Asynchronous Byzantine Agreement Using Cryptography (extended abstract). In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, Portland, Oregon, USA, July 16–19, 2000, pp. 123–132.

[19] Boneh D, Lynn B, Shacham H. Short Signatures from the Weil Pairing. In *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security*, Gold Coast, Australia, December 9–13, 2001, pp. 514–532.

[20] Abraham I, Jovanovic P, Maller M, Meiklejohn S, Stern G. Bingo: Adaptivity and Asynchrony in Verifiable Secret Sharing and Distributed Key Generation. In *Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 20–24, 2023, pp. 39–70.

[21] Abraham I, Jovanovic P, Maller M, Meiklejohn S, Stern G, *et al.* Reaching Consensus for Asynchronous Distributed Key Generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, Virtual Event, Italy, July 26–30, 2021, pp. 363–373.

[22] Gao Y, Lu Y, Lu Z, Tang Q, Xu J, *et al.* Efficient Asynchronous Byzantine Agreement without Private Setups. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, Bologna, Italy, July 10–13, 2022, pp. 246–257.

[23] Das S, Yurek T, Xiang Z, Miller A, Kokoris-Kogias L, *et al.* Practical Asynchronous Distributed Key Generation. In *2022 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, May 23–25, 2022, pp. 2518–2534.

[24] Kokoris Kogias E, Malkhi D, Spiegelman A. Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, Virtual Event, USA, November 9–13, 2020 pp. 1751–1767.

[25] Das S, Xiang Z, Kokoris-Kogias L, Ren L. Practical Asynchronous High-threshold Distributed Key Generation and Distributed Polynomial Sampling. In *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, USA, August 9–11, 2023, pp. 5359–5376.

[26] Cachin C, Kursawe K, Lysyanskaya A, Strobl R. Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, Washington, DC, USA, November 18–22, 2002, pp. 88–97.

[27] Choudhury A, Patra A. Brief Announcement: Efficient Optimally Resilient Statistical AVSS and its Applications. In *Proceedings of the 2012 ACM symposium on Principles of distributed computing*, Madeira, Portugal, July 16–18, 2012, pp. 103–104.

[28] Bandarupalli A, Bhat A, Bagchi S, Kate A, Reiter MK. Random Beacons in Monte Carlo: Efficient Asynchronous Random Beacon without Threshold Cryptography. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, Salt Lake, Utah, USA, October 14–18, 2024, pp. 2621–2635.

[29] Dolev S, Wang Z. SodsBC/SodsBC++ & SodsMPC: Post-quantum Asynchronous Blockchain Suite for Consensus and Smart Contracts. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, Virtual Event, November 17–20, 2021, pp. 510–515.

[30] Dolev D, Reischuk R. Bounds on Information Exchange for Byzantine Agreement. *J. ACM (JACM)* 1985 32(1):191–204.

[31] Gurkan K, Jovanovic P, Maller M, Meiklejohn S, Stern G, *et al.* Aggregatable Distributed Key Generation. In *Annual International Conference on the Theory and Applications of*

Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, pp. 147–176.
[32] Das S, Duan S, Liu S, Momose A, Ren L, *et al.* Asynchronous Consensus without Trusted Setup or Public-Key Cryptography. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, Salt Lake, UT, USA, October 14–18, 2024, pp. 3242–3256.
[33] Boldyreva A. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *International Workshop on Public Key Cryptography*, Miami, FL, USA, January 6–8, 2003, pp. 31–46.
[34] Lamport L, Shostak R, Pease M. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 1982 4(3):382–401.
[35] Duan S, Wang X, Zhang H. FIN: Practical Signature-Free Asynchronous Common Subset in Constant Time. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, Copenhagen, Denmark, November 26–30, 2023, pp. 815–829.
[36] Guo B, Lu Y, Lu Z, Tang Q, Xu J, *et al.* Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice. In *29th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, April 24–28, 2022 .
[37] Shamir A. How to Share a Secret. *Commun. ACM* 1979 22(11):612–613.
[38] Choc B, Goldwasser S, Micali S, Awerbuch B. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *26th Annual Symposium on Foundations of Computer Science*, Portland, Oregon, USA, October 21–23, 1985, pp. 383–395.
[39] Feldman P. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *28th Annual Symposium on Foundations of Computer Science*, Los Angeles, California, USA, October 27–29, 1987, pp. 427–438.
[40] Pedersen TP. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference*, Santa Barbara, California, USA, August 11–15, 1991, pp. 129–140.
[41] Stadler M. Publicly Verifiable Secret Sharing. In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques*, Saragossa, Spain, May 12–16, 1996, pp. 190–199.
[42] Fujisaki E, Okamoto T. A Practical and Provably Secure Scheme for Publicly Verifiable Secret Sharing and Its Applications. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques*, Espoo, Finland, May 31–June 4, 1998, pp. 32–46.
[43] Schoenmakers B. A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference*, Santa Barbara, California, USA, August 15–19, 1999, pp. 148–164.
[44] Cascudo I, David B. SCRAPE: Scalable Randomness Attested by Public Entities. In *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017*, Kanazawa, Japan, July 10–12, 2017, pp. 537–556.
[45] Shoup V. Practical Threshold Signatures. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques*, Bruges, Belgium, May 14–18, 2000, pp. 207–220.
[46] Bellare M, Rogaway P. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, Fairfax, Virginia, USA, November 3–5, 1993, pp. 62–73.
[47] Blahut RE. *Theory and Practice of Error Control Codes*, Addison-Wesley1983.
[48] Rabin MO. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. *JACM* 1989 36(2):335–348.
[49] Reed IS, Solomon G. Polynomial Codes Over Certain Finite Fields. *J. Soc. Ind. Appl.*

Math. 1960 8(2):300–304.

[50] Merkle RC. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques*, Santa Barbara, California, USA, August 16–20, 1987, pp. 369–378.

[51] Catalano D, Fiore D. Vector Commitments and Their Applications. In *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography*, Nara, Japan, February 26–March 1, 2013, pp. 55–72.

[52] Libert B, Yung M. Concise Mercurial Vector Commitments and Independent Zero-Knowledge Sets with Short Proofs. In *7th Theory of Cryptography Conference*, Zurich, Switzerland, February 9–11, 2010, pp. 499–517.

[53] Canetti R, Rabin T. Fast Asynchronous Byzantine Agreement with Optimal Resilience. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, San Diego, CA, USA, May 16–18, 1993, pp. 42–51.

[54] Yurek T, Xiang Z, Xia Y, Miller A. Long Live The Honey Badger: Robust Asynchronous DPSS and its Applications. In *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, USA, August 9–11, 2023, pp. 5413–5430.

[55] Kiayias A, Russell A, David B, Oliynykov R. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 20–24, 2017, pp. 357–388.

[56] Daian P, Pass R, Shi E. Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake. In *Financial Cryptography and Data Security: 23rd International Conference*, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, pp. 23–41.

[57] Fischer MJ, Lynch NA, Paterson MS. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 1985 32(2):374–382.

[58] Ben-Or M. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols (Extended Abstract). In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, Montreal Quebec, Canada, August 17–19, 1983, pp. 27–30.

[59] Plank JS, Xu L. Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Network Storage Applications. In *Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)*, Cambridge, Massachusetts, USA, July 24–26, 2006, pp. 173–180.

[60] Das S, Xiang Z, Ren L. Asynchronous Data Dissemination and its Applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, Virtual Event, Republic of Korea, November 15–19, 2021, pp. 2705–2721.

[61] Günther CU, Das S, Kokoris-Kogias L. Practical Asynchronous Proactive Secret Sharing and Key Refresh. Cryptology ePrint Archive, Paper 2022/1586, 2022.

[62] Kokoris-Kogias E, Jovanovic P, Gasser L, Gailly N, Syta E, *et al.* OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *2018 IEEE symposium on security and privacy (SP)*, San Francisco, California, USA, May 21–23, 2018, pp. 583–598.

[63] Zamani M, Movahedi M, Raykova M. RapidChain: Scaling Blockchain via Full Sharding. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, Toronto, ON, Canada, October 15–19, 2018, pp. 931–948.

[64] Merkle RC. Method of Providing Digital Signatures, 1982. US Patent 4,309,569.

[65] Hazay C, Lindell Y, Hazay C, Lindell Y. Sigma protocols and efficient zero-knowledge. *Efficient Secure Two-Party Protocols: Techniques and Constructions* 2010 pp. 147–175.

[66] Choi K, Manoj A, Bonneau J. SoK: Distributed Randomness Beacons. In *2023 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, May 21–25, 2023,

pp. 75–92.

[67] Bhat A, Shrestha N, Luo Z, Kate A, Nayak K. RandPiper - Reconfiguration-Friendly Random Beacons with Quadratic Communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. November 15–19, 2021, pp. 3502–3524.

[68] Feng H, Gao Y, Lu Y, Tang Q, Xu J. Practical Asynchronous Distributed Key Reconfiguration and Its Applications. Cryptology ePrint Archive, Paper 2025/149, 2025.