Rondo: Scalable and Reconfiguration-Friendly Randomness Beacon

Xuanji Meng*, Xiao Sui*✉, Zhaoxin Yang*✉, Kang Rong†, Wenbo Xu†✉,
Shenglong Chen†, Ying Yan†, Sisi Duan*✉
*Tsinghua University, †Blockchain Platform Division, Ant Group,
mxj21@mails.tsinghua.edu.cn,{suixiao, zhaoxin_yang, duansisi}@tsinghua.edu.cn
{rongkang.rong, xuwenbo.xwb, shenglong.chensl, fuying.yy}@antgroup.com
✉ Corresponding authors

***Abstract*—We present Rondo, a scalable and reconfiguration-friendly distributed randomness beacon (DRB) protocol in the partially synchronous model. Rondo is the first DRB protocol that is built from batched asynchronous verifiable secret sharing (bAVSS) and meanwhile avoids the high $O(n^3)$ message cost, where n is the number of nodes. Our key contribution lies in the introduction of a new variant of bAVSS called batched asynchronous verifiable secret sharing with partial output (bAVSS-PO). bAVSS-PO is a weaker primitive than bAVSS but allows us to build a secure and more scalable DRB protocol. We propose a bAVSS-PO protocol Breeze. Breeze achieves the optimal $O(n)$ messages for the sharing stage and allows Rondo to offer better scalability than prior DRB protocols. Additionally, to support the reconfiguration, we introduce Rondo-BFT, a dynamic and partially synchronous Byzantine fault-tolerant protocol inspired by Dyno (S&P 2022). Unlike Dyno, Rondo-BFT provides a communication pattern that generates randomness beacon output periodically, making it well-suited for DRB applications.**

We implement our protocols and evaluate the performance on Amazon EC2 using up to 91 instances. Our evaluation results show that Rondo achieves higher throughput than existing works and meanwhile offers better scalability, where the performance does not degrade as significantly as n grows.

I. Introduction

Randomness beacon, a notion formalized by Rabin [1], refers to a service that provides a continuous stream of shared randomness. Randomness beacon is useful in numerous applications such as Proof-of-Stake based blockchains [2, 3], blockchain sharding [4], electronic voting [5], asynchronous Byzantine fault-tolerant (BFT) consensus protocols [6]–[9], anonymous communication [10]–[13], smart contract based decentralized finance (DeFi) applications and non-fungible tokens (NFTs) [14, 15].

While there exists some trusted centralized randomness sources [16, 17], many recent efforts have been made to distributed randomness beacons (DRB) that remove the need for a trusted third party [18]–[27]. DRB can typically be built from three different cryptographic primitives: verifiable delay functions (VDF) [22], threshold cryptosystems that require distributed key generation (DKG) [24]–[27], and verifiable secret sharing (VSS) [18, 19, 21, 23]. VDF-based approaches enable efficient DRB with unlimited and open participation. However, the fixed delay introduced by VDF makes DRB only work in synchronous networks, where a known upper bound exists for message transmission and processing. Meanwhile, threshold cryptosystems make it possible to build DRB in partially synchronous networks (where the upper bound for message transmission and processing is unknown [28]) or fully asynchronous networks (there does not exist an upper bound). The DKG setup, however, especially during system reconfiguration (allowing the joining and leaving of nodes), can be costly. VSS-based DRB protocols thus become a perfect choice to avoid the need for DKG.

Existing VSS-based DRB protocols all employ a paradigm that combines VSS and Byzantine fault-tolerant state machine replication (BFT-SMR), assuming a system with n nodes and at most t of them are corrupted. A typical VSS-based protocol proceeds as follows. First, multiple VSS instances are queried and each node shares its secret. Then, the BFT-SMR commits the secrets of $t + 1$ nodes. Finally, nodes reconstruct and aggregate the secrets, ensuring that the aggregated beacon value is unbiased and unpredictable. The paradigm works both as a standalone randomness beacon protocol and an *on-chain* randomness beacon provider. In the latter case, the BFT-SMR can be replaced with a blockchain, and the randomness values generated can be used on-chain, e.g., for smart contract-based DeFi applications.

Such a paradigm suffers from some limitations we seek to address. First, as parallel VSS instances are needed for every round (by default, one beacon value is expected to be generated in each round), the performance of the protocols usually degrades significantly as n grows. Second, most VSS-based protocols [19, 21, 23, 29] still build DRB under the synchronous timing assumption and are not that robust in wide area networks. Third, although VSS-based protocols are reconfiguration-friendly, it is still unclear how to build reconfigurable randomness beacon in systems that do not make a synchronous assumption. Indeed, the only partially synchronous DRB, Spurt [18], does not support reconfiguration, mainly because the underlying BFT-SMR does not support reconfiguration. Meanwhile, the only reconfiguration-friendly DRB solutions, RandPiper [19] and Optrand [29], assume a synchronous network.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.230549
www.ndss-symposium.org

Thus, an interesting research question to answer is:

Can we build a distributed randomness beacon protocol that is both scalable and reconfiguration-friendly in the partially synchronous network?

Our approach and our results. We propose Rondo, a scalable and the first reconfiguration-friendly randomness beacon protocol in the partially synchronous model. Our starting point is using batched asynchronous verifiable secret sharing (bAVSS) [36]–[38] to build DRB. Specifically, bAVSS allows the *dealer* to share a batch of B secrets at once, and the secret can be reconstructed independently. As we can share B secrets at once, the VSS phase only needs to be executed once every B rounds and the amortized latency of beacon outputs can be significantly reduced compared to previous works.

Unfortunately, all known bAVSS protocols have $O(n^2)$ messages. As n parallel instances are required to build DRB, the DRB protocol has $O(n^3)$ messages and the bAVSS phase might be the bottleneck of the system. Additionally, just as mentioned above, it is still unclear how to support reconfiguration in a partially synchronous network.

Compared to prior VSS-based DRB protocols, Rondo introduces two crucial building blocks to enhance the scalability and support reconfiguration: a new primitive called *batched asynchronous verifiable secret sharing with partial output (bAVSS-PO)*, and a new dynamic Byzantine fault-tolerant state machine replication protocol. A comparison between Rondo and existing works is summarized in Table I.

▷ *Batched asynchronous verifiable secret sharing with partial output (bAVSS-PO) and the Breeze protocol.* We propose a new bAVSS-PO primitive and a protocol called Breeze. bAVSS-PO has two desirable properties essential for building a DRB. First, it supports batching, which significantly reduces the amortized latency of beacon output. Second, bAVSS-PO is weaker than AVSS in the sense that each node requires validation data to validate its secret share. Despite this weaker requirement, the primitive retains the other properties of AVSS and is sufficient to build a DRB protocol.

Our Breeze protocol has $O(n)$ messages for the sharing stage, which is optimal for VSS. Indeed, $O(n)$ messages are required even for a dealer to share the secret shares. The communication paradigm is inspired by that for consistent broadcast (CBC) [39]. By setting the batch size B as $O(\log n)$, Breeze achieves $O(\lambda n)$ communication and is the best result known so far. Indeed, existing state-of-the-art protocols achieve $O(\lambda n)$ amortized communication at the cost of $B = O(n)$ [38,40]. Due to the reliance on digital signatures, Breeze is provably secure only in the computational model. Meanwhile, to support batching, we use Bulletproofs [41] for batch verification inspired by hbACSS [37]. Compared to known works that also adopt Bulletproofs for building batched VSS [37], we optimize the computational cost by a factor of B for batch verification.

▷ *Dynamic Byzantine fault-tolerant state machine replication (BFT-SMR).* Although VSS supports reconfiguration of the nodes relatively easily (e.g., via proactive secret sharing [42, 43] or asking the nodes to start fresh VSS instances), BFT-SMR cannot easily do so in the partially synchronous network. Inspired by a recent work on dynamic BFT called Dyno [44], we propose Rondo-BFT, a new partially synchronous and reconfiguration-friendly BFT protocol, which might be of independent interest. Different from Dyno that can be viewed as a reconfiguration-friendly version of PBFT [45], Rondo-BFT can be viewed as a reconfiguration-friendly version of HotStuff [46]. The HotStuff-style pipelining communication pattern is a better *fit* for randomness beacon, mainly because the beacons can be output periodically.

Our contributions. We make the following contributions:

- We provide a new DRB protocol Rondo. Rondo consists of a new bAVSS-PO primitive and dynamic BFT-SMR. The batching feature of bAVSS-PO and the dynamic feature of BFT-SMR allow us to achieve scalability and reconfiguration simultaneously. To our knowledge, Rondo is the first reconfiguration-friendly DRB in the partially synchronous model.
- We propose Breeze, a bAVSS-PO protocol. Breeze is both communication-efficient and computation-efficient. Breeze achieves $O(n)$ messages for the sharing stage, in contrast to $O(n^2)$ by prior bAVSS protocols. Meanwhile, the computational cost for batch verification is reduced by a factor of B compared to existing state-of-the-art batched VSS. Although the use of Breeze makes Rondo achieve slightly higher communication complexity than existing state-of-the-art DRB protocol, we show that Rondo achieves better scalability thanks to the optimization in message complexity and computational complexity.
- We propose Rondo-BFT, a dynamic BFT-SMR protocol in the partially synchronous model. Rondo-BFT can be viewed as a dynamic version of HotStuff that supports the joining and leaving of nodes, which is of independent interests.
- Our evaluation results on Amazon EC2 using up to 91 instances across four different regions show that Rondo achieves high throughput (beacon/min) and scalability. Unlike prior works, VSS does not become the bottleneck of the system anymore.

II. Related Work

AVSS, ACSS, and HACSS. In the unconditional security setting (also known as information-theoretical model where the only assumption is authenticated channels), AVSS usually incurs high communication [47]–[49]. In this setting, there are two types: perfectly secure AVSS [50, 51] and statistically secure AVSS [52, 53]. In the computational setting, Cachin et al. [54] (assuming the DL assumption) proposed the first practical AVSS with optimal $O(n^2)$ messages assuming $n > 3t$. Many follow-up works optimize the communication complexity [55, 56]. If AVSS achieves the *completeness* property, it is sometimes called asynchronous complete secret sharing (ACSS). Additionally, if ACSS supports a high threshold (e.g., $2t + 1$ out of $3t + 1$ for the $n > 3t$ setting), it is also called high-threshold ACSS (HACSS) [55, 57, 58].

In this work, we propose a new bAVSS-PO primitive that has a weaker *commitment* property than ACSS. Additionally, we introduce an efficient bAVSS-PO protocol named Breeze. Apart from Breeze's message-efficient design, our technique of supporting batching is generic and can be integrated with any ACSS protocol. To demonstrate this, we integrate our approach with HAVEN [55] and build a batched HACSS protocol called

protocol	timing	fault tolerance	liveness	unpredictability	bias-resistance	re-configurable	communication	message	computation	crypto. primitives	crypto. assumptions*	setup
Cachin et al. [27]	async.	1/3	✓	✓	✓	✗	$O(\lambda n^2)$	$O(n^2)$	$O(n)$	th-sig	CDH	DKG
RandHerd [26]	async.	1/3	✓	✓	✓	✗	$O(\lambda c^2 \log n)^\clubsuit$	$O(c^2)$	$O(c^2 \log n)^\clubsuit$	PVSS +CoSi	DLog	DKG
Hashrand [30]	async.	1/3	✓	✓	✓	✗	$O(\lambda n^2 \log n)$	$O(n^2)$	$O(\lambda n \log n)^\dagger$	AwVSS¶ +Hash	RO	secure channels
BRandPiper [19]	sync.	1/2	✓	✓	✓	✓	$O(\lambda n^3)$	$O(n^3)$	$O(n^2)$	VSS	$q-$SDH	SRS
GRandPiper [19]	sync.	1/2	✓	$t+1$	✓	✓	$O(\lambda n^2)$	$O(n^2)$	$O(n^2)$	PVSS	$q-$SDH	SRS
GRandline [31]	sync.	1/2	✓	✓	✓	✗	$O(\lambda n^2)$	$O(n^2)$	$O(n)$	PVSS +pairing +th-sig	AGM +Co-OMDL††	DKG
Optrand [29]	sync.	1/2	✓	✓	✓	✓	$O(\lambda n^2)$	$O(n^2)$	$O(n^2)^\ddagger$	PVSS +Pairing	$q-$SDH +SXDH	SRS +PKI
Drand [32]	sync.	1/2	✓	✓	✓	✗	$O(\lambda n^2)$	$O(n^2)$	$O(n)$	th-sig	Gap-CDH	DKG
Scrape [33]	sync.	1/2	✓	✓	✓	✗	$O(\lambda n^4)$	$O(n^3)$	$O(n^2)$	PVSS +Pairing	DDH**	CRS +PKI
Spurt [18]	partial	1/3	✓	✓	✓	✗	$O(\lambda n^2)$	$O(n^2)$	$O(n^2)^\ddagger$	PVSS +Pairing	DBDH	CRS +PKI
Dfinity [34]	partial	1/3	✓	✓	✓	✗	$O(\lambda n^3)^\S$	$O(n^3)$	$O(n)$	th-sig	CDH	DKG
Rondo (this work)	partial	1/3	✓	✓	✓	✓	$O(\lambda n^2 \log n)$	$O(n^2)$	$O(n)$	bAVSS-PO	DLog	CRS +PKI

TABLE I: Comparison of existing randomness beacon protocols. *All protocols in the table assume random oracles.†All the protocols count the complexity of exponentiation, except Hashrand [30], which counts hash functions. ‡The computational cost is $O(n^2)$ for the dealer and $O(n)$ for other nodes. **This scheme can be instantiated both under the Decisional Diffie Hellman (DDH) assumption in the random oracle model and in the plain model under the Decisional Bilinear Square (DBS) assumption. §The complexity is $O(\lambda n^2)$ in the best case. ¶AwVSS stands for asynchronous weak verifiable secret sharing. The main difference between this scheme and ordinary VSS is that it allows nodes to reconstruct $\perp$ in the reconstruction phase. ††Co-OMDL stands for co-one-more discrete logarithm [35], which is stronger than ordinary OMDL assumption. $^\clubsuit$ Randherd employs a committee-sampling approach that samples a committee of size c for the protocol to enhance the scalability. Such an optimization can be applied to almost all other protocols. The computational complexity of some protocols in the table is not summarized. We provide additional details about the analysis in Appendix A.

BatchHAVEN. A concurrent work HAVEN++ [40] shares a lot of similarities with BatchHAVEN. Furthermore, our Breeze protocol exhibits some similarities with a concurrent work by Das et al. [58], which is a conventional ACSS protocol with $O(n^2)$ messages. In contrast, Breeze is a bAVSS-PO protocol with $O(n)$ messages. Finally, it is worth mentioning that Breeze can be used to improve multiparty computation protocols. For example, we can replace the VSS protocols in the *DKG-DL* protocol in [59] with Breeze to simplify the protocol under $n \geq 3t+1$ setting. In particular, we can remove the expensive homomorphic encryption needed for [59].

Batched VSS and packed VSS. Batched AVSS allows the dealer to share many secrets in parallel [36]–[38, 40]. hbACSS [37] proposes a batched ACSS protocol leveraging Bulletproofs [41]. Recently, Shoup and Smart (SS) proposed a lightweight batched AVSS using only lightweight cryptographic primitives, i.e., hash functions [36]. Compared to hbACSS, SS is less communication-efficient, which might be less interesting in wide-area networks. Our approach of supporting batching for Breeze is inspired by hbACSS. Compared to hbACSS, we reduce the computational cost by a factor of B for batch verification.

Another relevant notion is packed secret sharing, which packs many secrets in one sharing [60]. Packed VSS does not suit our needs as all secrets need to be reconstructed at once.

Polynomial Commitment. Polynomial commitment is a cryptographic technique for committing to a polynomial and allowing others to verify its properties (e.g., evaluations at specific points) without disclosing the polynomial itself [61]. The notion was first introduced by Kate, Zaverucha, Goldberg [62] and was found useful in AVSS protocols. In particular, polynomial commitment allows each node to validate the share it receives from the dealer [18,33,37,55]. In this paper, we used a batched polynomial commitment scheme to build Breeze (i.e., our bAVSS-PO protocol).

Dynamic BFT and reconfiguration for SMR. Reconfiguration for crash fault-tolerant (CFT) protocols or synchronous BFT protocols is relatively easy. Many CFT protocols have studied efficient reconfiguration that does not degrade the overall performance [63]–[65]. Prior synchronous randomness beacon protocols such as RandPiper [19] and OptRand [29] also support reconfiguration for BFT-SMR.

BFT-SMaRt is a partially synchronous BFT protocol that supports reconfiguration [66], where membership requests are treated as a special type of client requests. It adopts the workflow of PBFT [45] so it achieves $O(n^2)$ messages. It was mentioned in Dyno [44] that by simplifying treating the *membership requests* as regular client requests, a partially synchronous BFT protocol may have subtle liveness issues. In this work, we follow the definitions by Dyno and provide a new dynamic BFT protocol that can be viewed as dynamic HotStuff [46]. As HotStuff achieves $O(n)$ messages and the communication pattern allows blocks to be produced periodically, it is a better fit for DRB. To the best of our knowledge, dynamic HotStuff has not been studied in the literature.

Randomness Beacon. Chainlink [67] is an industrial solution. The randomness beacon is generated using the verifiable random function (VRF) and thus requires a trusted setup.

A recent work Hashrand [30] uses batched asynchronous weak verifiable secret sharing (BAwVSS) to construct a DRB protocol. The BAwVSS primitive shares some similarities with our bAVSS-PO, where it is weaker than AVSS and supports batching. However, the weak commitment property of BAwVSS leads to a Monte-Carlo beacon and there is no guarantee on an output every round. Also, as pointed out by a recent work [68], the BAwVSS construction is flawed.

DRB can be viewed as a more lightweight version of distributed key generation (DKG) although one can directly use a combination of DKG and threshold cryptosystem to generate randomness beacon [27,32]. Informally, DRB is more lightweight because we only need the aggregated secrets.

III. System Model and Building Blocks

We consider a system consisting of a finite set of nodes $\{P_1, P_2, \cdots\}$, where a fraction of them may fail arbitrarily (Byzantine failures) and they are controlled by an adversary $\mathcal{A}$. Nodes that are not Byzantine are called *honest* or *correct* nodes. The set of nodes that participate in the system may change dynamically. We assume a standard public-key infrastructure, i.e., each node P_i has a public/private key pair (pk_i, sk_i) and the public key pk_i is known by all nodes. We use a collision-resistant hash function $H()$ mapping a message of arbitrary length to a fixed-length output.

We follow the notations in Dyno [44] for our definition of a reconfigurable system. In particular, we use *configuration* to denote the successive membership of the system. The configuration number c grows monotonically, beginning with 0. In each configuration c, there is a fixed set of nodes M_c and we call these nodes *members* of configuration c. Each node changes its configuration via configuration installation. Every correct node is aware of the initial configuration M_0 (i.e., the set of nodes when the system is started).

Let n_c denote the number of nodes in M_c (i.e., $|M_c|$) and t_c denote the number of c-faulty nodes in M_c. Dyno defines two assumptions regarding the relationship of n_c and t_c. We assume the *standard assumption* and additionally require a new *one-correct* assumption as summarized below.

(a) **Standard assumption:** For any configuration M_c, $t_c \leq \lfloor \frac{|M_c|-1}{3} \rfloor$.
(b) **One-correct assumption:** There exists at least one correct replica in M_0 that never leaves the system.

We may omit the subscript for n_c and t_c when no ambiguity arises. For instance, when we describe our bAVSS-PO protocol, we use n and t to denote the number of nodes and the number of faulty nodes in each instance.

We assume that all nodes have an agreement of a common reference string (CRS): $crs = (\mathbb{G}, \mathbf{g}, h)$, which contains a cyclic group $\mathbb{G}$, a vector of generators $\mathbf{g}$ and another single generator h.

We consider the partially synchronous model [69], where there exists an unknown global stabilization time (GST) such that after GST, messages sent between two correct nodes arrive within a fixed delay.

Randomness beacon. We consider a distributed randomness beacon protocol that operates in *epochs* and each epoch consists of multiple *rounds*. By default, the protocol is expected to generate a beacon output in every round. In our approach, a batch of secrets is shared once per epoch, and a randomness beacon is generated in each round. We adopt terminology from previous studies [18, 20] and refine the security objectives of the randomness beacon protocol as follows.

- **Unpredictability.** Let $r_1, r_2, ..., r_i$ be the beacon outputs generated so far. The protocol is unpredictable if for every future beacon output r_j such that $j > i+1$, for any PPT adversary $\mathcal{A}$ as mentioned above and $i \geq 1$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:
$$\mathsf{Pr}[(j, r'_j) \leftarrow \mathcal{A}(r_1, r_2, ..., r_i) : r'_j = r_j] \leq \mathsf{negl}(\lambda)$$
- **Bias-resistance.** Let $r_i(k)$ be the k^{th} bit of the beacon output r_i, $|r_i|$ be the number of bits of r_i, and $\mathcal{A}(r_1, r_2, ..., r_{i-1})$ be the adversary having access to all the beacon outputs generated so far. The randomness beacon protocol is bias-resistant if for any PPT adversary $\mathcal{A}$ where $i \geq 1$, and $k = 1, ..., |r_i|$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:
$$\left| \mathsf{Pr}\left[\begin{array}{c} (k, r'_i(k)) \leftarrow \mathcal{A}(r_1, r_2, ..., r_{i-1}) : \\ r'_i(k) = r_i(k) \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)$$
$$|\mathsf{Pr}[r_i(k) = 0] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$$
- **Liveness.** A beacon output is generated in every round except with a negligible probability.
- **Public verifiability.** The randomness beacon protocol is publicly verifiable if any users that do not participate in the protocol can verify the beacon outputs.

Notably, the bias-resistance and unpredictability properties emphasize on different properties. In particular, bias-resistance focuses on preventing adversaries from "actively" influencing the output, while unpredictability prevents "passive" guessing of the output by adversaries.

We consider the public verifiability property an *optional* property for a DRB protocol. Indeed, if the protocol serves as a standalone DRB service, public verifiability is necessary. In contrast, if the protocol only provides randomness beacons to participants of the protocol (e.g., providing on-chain randomness beacon), public verifiability is not necessary.

We summarize the most frequently used notations in Table II.

A. Building Blocks

Dynamic Byzantine fault-tolerant state machine replication (BFT-SMR or BFT for short) and types of requests. In BFT, a node *delivers or commits requests* submitted by clients. A node then sends a reply to the corresponding client. The client computes a final response based on the reply messages. There are two types of requests: the regular request that consists

TABLE II: Notations used in this work.

Notation	Description
$\mathbb{G}$	Cyclic group
$\|M\|$	Number of elements in set M
B	Batch size
P_i	Party/Node i
$\mathbf{a}$	Vector
$\mathbf{a} \cdot \mathbf{b}$	Inner product of $\mathbf{a}$ and $\mathbf{b}$
$\mathbf{a}[i]$	The i^{th} element of vector $\mathbf{a}$
$\mathbf{s}[i, j]$	The element in the i^{th} row and j^{th} column of matrix $\mathbf{s}$
$\mathbf{s}[i, :]$	The i^{th} row of matrix $\mathbf{s}$
$\mathbf{s}[:, j]$	The j^{th} column of matrix $\mathbf{s}$
$\hat{S}$	Polynomial commitment of a set of polynomials, a vector
$H(\cdot)$	Hash function

of *normal* transactions; the membership request that consists of requests for membership changes. A membership request might be either a JOIN request (for adding a new node) or a LEAVE request (for removing a node). In our case, a valid JOIN request contains the public key of the sender as a unique identifier. Requests are also called *transactions* and we use them interchangeably in this work. We consider dynamic BFT in our work to build the reconfigurable randomness beacon. The correctness of dynamic BFT is defined below.

- **Agreement:** If a correct node in epoch e delivers a request rq, then every correct node in the same epoch e eventually delivers rq.
- **Enhanced total order:** If a correct node in epoch e delivers a request with a sequence number k, and another correct node delivers a request rq' in epoch e' with the same sequence number, then $e = e'$ and $rq = rq'$.
- **Liveness:** If a correct client submits a request rq, then eventually a correct node in some configuration delivers rq.
- **Consistent delivery:** A correct client submitting rq will deliver a correct response which is consistent with the state in some configuration where rq is delivered.

Batched asynchronous verifiable secret sharing with partial output (bAVSS-PO). We introduce a new primitive bAVSS-PO and we first present the non-batched version AVSS-PO. Similar to conventional AVSS, an AVSS-PO protocol also consists of two stages: sharing and reconstruction. In the sharing stage, a node, also called a dealer, shares a secret s to all nodes. In the reconstruction stage, nodes interact to reconstruct secret s. The sharing stage of AVSS-PO is *verifiable* and any node that completes the sharing stage receives some validation data that can be verified by any node. Additionally, every node produces some *partial* output (i.e., in our case a secret share from the dealer) during the sharing stage. Once a node receives the validation data proving that some node has completed the sharing state, the node can verify whether its partial output is *valid*. A secure AVSS-PO protocol satisfies the following properties.

- **Privacy.** If a correct dealer shared s using ID.d, then before any correct node starts reconstruction for ID.d, $\mathcal{A}$ has no information about s.
- **Liveness.** 1) If the dealer P_d is correct throughout the sharing stage, then all correct nodes complete the sharing stage; 2) If at least $t+1$ correct nodes validate their partial outputs and start reconstruction for ID.d, then every correct node P_i reconstructs some z_i for ID.d.
- **Correctness.** Once $t+1$ correct nodes have completed the sharing for ID.d, there exists a fixed value z such that the following holds: 1) if the dealer shared s using ID.d and is correct throughout the sharing stage, then $z = s$ and 2) if a correct node P_i reconstructs z_i for ID.d, then $z_i = z$.
- **Commitment.** Once some node completes the sharing stage, the following holds: 1) at least $t+1$ correct nodes that receive the validation data can validate their partial outputs; 2) there exists a secret s, such that in the reconstruction stage, if any honest node outputs s', $s' = s$.

bAVSS-PO is the batched version of AVSS-PO. Namely, the dealer shares a batch of B shares $\{s_1, \cdots, s_B\}$ in one instance and all the secrets are independent to each other.

bAVSS-PO vs. identifiable abort. Identifiable abort of threshold cryptosystems (or multiparty computation) [37, 59] refers to the capability of aborting the protocol under the existence of corrupted nodes. In bAVSS-PO, if PO is not valid, the notion is akin to identifiable abort. We do not emphasize it in our notion as it is not *needed* for our DRB.

Batched polynomial commitment. We extend Bulletproofs [41] to build polynomial commitments for batching and use them to build our protocol. Following Bulletproofs, our scheme requires a common reference string (CRS). Formally, a polynomial commitment P for a batch of secrets consists of three sub-algorithms:

- $\mathrm{BatchCommit}(crs, \{S_i | i \in [1, B]\}, p) \rightarrow \hat{S}$ is given the common reference string crs, B polynomials $\{S_i | i \in [1, B]\}$, and the degree p. It outputs a commitment vector $\hat{S}$.
- $\mathrm{BatchEval}(crs, \{S_i | i \in [1, B]\}, \hat{S}, (y_1, y_2, \ldots, y_m), p) \rightarrow (V, \Phi)$ is given the common reference string crs, B polynomials $\{S_i | i \in [1, B]\}$, polynomial commitment $\hat{S}$, evaluation points arranged as vectors $(y_1, y_2, \ldots, y_m)$, the degree of the polynomials p. It outputs the evaluations arranged as a matrix V, and the corresponding proof vector Φ.
- $\mathrm{BatchVerifyEval}(crs, \hat{S}, \mathbf{v}, \mathbf{y}, \phi, p) \rightarrow \mathsf{true} \backslash \mathsf{false}$ takes as input the common reference string crs, a commitment $\hat{S}$, evaluation point $\mathbf{y}$, evaluation proof ϕ, and the degree of S_i, denoted by p. It outputs a Boolean.

Let $\mathbb{G}$ be a cyclic group of a prime order q with a generator g. Given a polynomial $R(x)$ of degree p: $R(x) = r_0 + r_1 x + \cdots + r_p x^p$, where the coefficients $r_i \in \mathbb{Z}_q$, Bulletproofs commitment for $R(x)$ is $\prod_{i=0}^{\mathrm{p}} g_i^{r_i}$. A Bulletproofs commitment for $R(x)$ uniquely determines $R(x)$.

We also need the binding and hiding property of commitments [55]:

- **Binding.** Let $crs \leftarrow \mathrm{Setup}(1^\kappa, \mathbb{F}, D)$. For any PPT adversary $\mathcal{A}(crs)$ that outputs a batching commitment $\hat{S}$, the degree p of the polynomials, and two evaluations $e = \langle \mathbf{v}, \mathbf{y}, \phi \rangle$ and $e' = \langle \mathbf{v}', \mathbf{y}', \phi' \rangle$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\Pr\left[\begin{array}{c} (\hat{S}, e, e', p+1) \leftarrow \mathcal{A}(crs) : \\ \mathbf{y} = \mathbf{y}' \wedge \mathbf{v} \neq \mathbf{v}' \\ \wedge\, \mathrm{BatchVerifyEval}(crs, \hat{S}, \mathbf{v}, \mathbf{y}, \phi, p) \\ \wedge\, \mathrm{BatchVerifyEval}(crs, \hat{S}, \mathbf{v}', \mathbf{y}', \phi', p) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

- **Hiding.** Let $crs \leftarrow \text{Setup}(1^{\kappa}, \mathbb{F}, D)$, p be an arbitrary integer less than D, and $I \subset \mathbb{F}$ be an arbitrary set of evaluation points with $|I| \leq p$. Randomly choose a batch of polynomials $\{S_i | i \in [1, B]\} \leftarrow \mathbb{F}[x]$ of degree p and construct its commitment $\hat{S} = \text{BatchCommit}(\text{pp}, \{S_i | i \in [1, B]\}, p)$. For all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\Pr\left[\begin{array}{c} E = \text{BatchEval}(crs, \{S_i | i \in [1, B]\}, \hat{S}, I, p, n, |I|) : \\ (x, \mathbf{v}) \leftarrow \mathcal{A}(crs, \hat{S}, E) \\ \mathbf{v}[k] = S_k(x), k \in [1, n] \wedge x \notin I \end{array}\right]$$

$\leq \mathsf{negl}(\lambda)$

In this paper, given a batch of polynomials $\{S_i | i \in [1, B]\}$, we use a matrix $\mathbf{s}$ to denote the vectors of coefficients of them, and use $\mathbf{s}[i, :]$ to denote the i^{th} row of $\mathbf{s}$, which is the coefficient vector of S_i.

We also present the definitions of other related primitives in our full paper [70].

IV. Overview of Our Approach

Rondo follows the generic workflow of conventional randomness beacon protocols: in the commitment phase, each node first disseminates its secret shares via an AVSS protocol; in the agreement phase, a BFT-SMR protocol is executed to agree on a *common subset* of the secrets; in the reconstruction phase, nodes reconstruct the secrets corresponding to the common subset and then output the beacon values.

The concrete workflow of Rondo is illustrated in Figure 1. Namely, once every B rounds (i.e., an epoch), each node queries a Breeze instance to share a batch of B secrets. In the first round of every epoch, Rondo-BFT agrees on a set of validation data generated in $t + 1$ Breeze instances. The validation data can be used for honest nodes to validate whether their partial outputs (i.e., the secret shares) in Breeze instances are valid. In other rounds of the epoch, Rondo-BFT simply agrees on some default data (e.g., the round number). Finally, after an agreement is reached in every round, nodes reconstruct one secret in the reconstruction phase.

Our contributions focus on both AVSS and the BFT-SMR, as summarized below.

Breeze. We propose Breeze, a bAVSS-PO protocol and bAVSS-PO is a new variant of AVSS as defined in Sec. III-A. As mentioned in the introduction, bAVSS-PO does not achieve the conventional *completeness* property of AVSS and instead achieves a weaker *commitment* property. Our insight is that the weaker property is sufficient for randomness beacon but allows us to build a more efficient protocol. Namely, any correct replica that completes the sharing stage receives the validation data. The commitment property ensures that once the validation data exists, the secret is "fixed". However, even if the validation data exists, some nodes may still not terminate the sharing stage. Any correct node that has not terminated the sharing stage can validate its secret share after it receives the validation data. In our DRB protocol, the validation data can be obtained in the agreement phase. In this way, we can make the commitment phase more efficient.

The communication paradigm of Breeze is inspired by consistent broadcast (CBC) [39]: nodes only need to interact with the dealer, allowing the sharing stage to achieve an optimal message complexity of $O(n)$. Namely, after receiving the secret share from the dealer, each node that validates its share sends a digital signature to the dealer. After the dealer collects $2t+1$ signatures, some secret s shared by the dealer is *fixed*. If correct nodes reconstruct the secret, they will output the same s so our protocol achieves the commitment property. The $2t + 1$ signatures serve as the validation data to prove the existence of the fixed secret.

Our Breeze protocol allows a dealer to share a batch of B secrets $\{s_1, \cdots, s_B\}$ at once. A crucial property is that the reconstruction is flexible. Namely, each secret could be separately recovered by more than $t + 1$ honest nodes. In this way, the commitment phase only needs to be executed once every epoch and the amortized latency of beacon outputs becomes very close to the latency of the BFT-SMR.

The technique of support batching draws inspiration from hbACSS [37], leveraging Bulletproofs for verifying evaluations of the polynomials. These evaluations are represented as inner products of coefficients and evaluation points. To efficiently compute and prove a batch of polynomials of degree p at multiple evaluation points, Bulletproofs' inner-product argument is utilized:

$$[P_i \leftrightarrow P_j] : \{\hat{S}[m] = \prod_{k=0}^{p} g_k^{\mathbf{s}[m,k]} \wedge v = \langle \mathbf{s_{m,:}}, \mathbf{y_{:,j}} \rangle, m \in [1, B]\}$$

Here, $i, j \in [1, n]$ and $m \in [1, B]$. In this setup, the prover P_i (acting as the dealer) shares B polynomials $S_m \mid m \in [1, B]$, while each verifier P_j (a node) verifies all polynomials at a designated evaluation point y_j.

The advantage of using Bulletproofs, also as mentioned in hbACSS, is that its commitment size is constant and the proof size is logarithmic. The underlying idea of Bulletproofs is using a so-called *folding* technique to recursively halve the degree of the polynomials to one.

Nevertheless, the drawback of the above approach is that each polynomial requires a halving computation so the computational cost is high. We therefore provide a more computation-efficient verification mechanism. The idea is to use a group of random numbers to sum the polynomials up to a combined one. In this way, the subsequent computation can be done based on a combined polynomial instead of B polynomials. Thus, our approach can eliminate a B factor of the computational complexity. Meanwhile, we found some subtle mistakes in the pseudocode of hbACSS and we fix them in our work.

Dynamic BFT. In Dyno [44], it was mentioned that a partially synchronous BFT may suffer from liveness issues when treating membership requests as regular client requests. A dynamic PBFT is then provided. In contrast, Rondo-BFT can be viewed as a *dynamic* HotStuff protocol. In Rondo-BFT, we adopted some techniques in Dyno to address the liveness issues. Furthermore, when we transform HotStuff into a dynamic BFT, we face some unique safety and liveness challenges.

The safety challenge arises because the linear message complexity of HotStuff makes it extremely challenging to achieve the *agreement* property, a *required* property for building dynamic BFT [44]. Indeed, the leader is the only node that

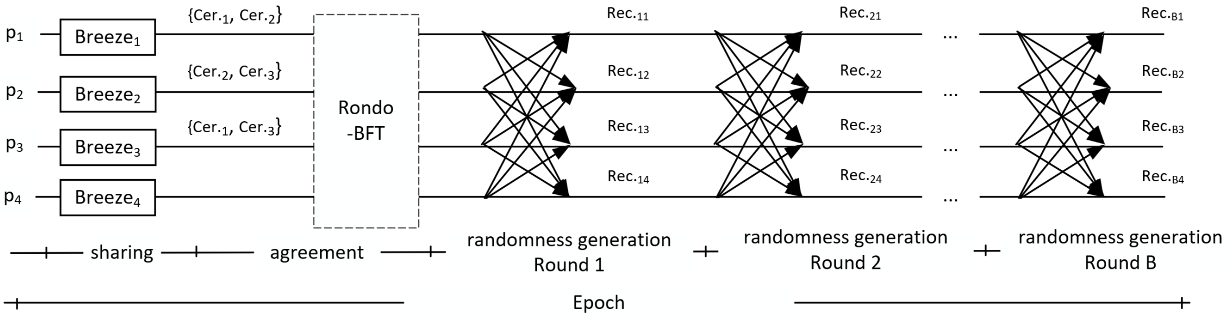


Fig. 1: The Rondo protocol.

collects the quorum certificates (a sufficiently large number of matching votes, also called QC or certificates) and the certificates are crucial for correct replicas to deliver client requests. For instance, when a membership request for adding a new node to the system is delivered, a faulty leader may choose to send the certificate to nodes in the system but not the new node. In this way, the new node will not join the system. To overcome this challenge, we carefully add some constraints for nodes that request to join or leave the system and nodes may need to forward their received messages to other nodes. Therefore, the agreement property is achieved by correct nodes nearly simultaneously for membership requests (i.e., all correct nodes deliver the membership requests with a bounded network delay). Meanwhile, agreement is eventually achieved for regular requests.

The liveness challenge mainly arises during view changes. This is because multiple nodes from different configurations may *compete* for becoming a new leader, again due to the fact that agreement is challenging. To handle this problem, we additionally introduce some procedures in the normal-case operation. In particular, before any correct node installs a new configuration, it needs to collect a view-change certificate vc signed by at least $2t + 1$ nodes in the current configuration. Accordingly, more than $t + 1$ correct nodes in the previous configuration are aware of the new configuration. Finally, only one leader will be selected once a view change occurs.

V. Rondo

We show the pseudocode of our randomness beacon protocol Rondo in Figure 2. The protocol consists of three phases: the *commitment* phase where nodes disseminate their secret shares via n parallel Breeze instances; the *agreement* phase where nodes agree on a common subset (denoted as CS_e) via BFT-SMR; the *reconstruction* phase where nodes reconstruct the secrets.

A. The Main Workflow

```
▷ Initialization.
01:  - At the beginning of Rondo, set epoch $e \leftarrow 0$, round $r \leftarrow 1$.
02:  - At the beginning of round $r$, if $r > B$ then
03:      - Set $r \leftarrow r \mod B$, $e \leftarrow e + 1$.
04:      - Let $CS_e$ denote common subset and $CS_e \leftarrow \emptyset$.
05:      - Let $S_e \leftarrow [\perp]$ be the shares received from other nodes.
06:  - Let $M_e$ be the members of epoch $e$.
▷ The commitment phase.
07:  - if $P_i \in M_e$ and $r \mod B = 1$ then
08:      - Select the random secrets $\mathbf{s^i} \leftarrow (s_1^{(i)}, \cdots, s_B^{(i)})$.
09:      - Query the $\mathsf{Breeze}_i.\mathsf{Share}(in, share, \mathbf{s^i})$ for $\mathsf{Breeze}_i$
           and participate in other Breeze instances
10:  - Upon completing the sharing stage of $\mathsf{Breeze}_j$
11:      - $S_e \leftarrow S_e \cup \{\mathsf{cer}_{e,j}\}$, in which $\mathsf{cer}_{e,j}$ is the validation data
           for $\mathsf{Breeze}_j$.
▷ The agreement phase.
12:  - if $r \mod B = 1$ then
13:      - Wait until $|S_e| \geq t+1$, $\mathsf{RondoBFT.Propose}_i(e, r, (S_e, m))$
           and let $S'_e \leftarrow S_e$    //$m$ consists of additional input data
14:      - Introduce the following predicate to the input $S'_e$ of
           Rondo-BFT: $S'_e$ is valid only if $S_e$ consists of at least
           $t + 1$ valid certificates.
15:      - Wait until $\mathsf{RondoBFT.Return}_i(e, r, (S'_e, M, *))$
16:      - $CS_e \leftarrow S'_e$, $M_{e+1} \leftarrow M$.
17:  - else
18:      - $\mathsf{RondoBFT.Propose}_i(e, r, (-, m))$
19:  - Wait until $\mathsf{RondoBFT.Return}_i(e, r, (-, *))$
▷ The reconstruction phase.
20:  - if $r \mod B = 1$ then
21:      - For $j \in CS_e$, invoke $\mathrm{ParVerify}(cer_{e,j}, C, \mathsf{set}_i^{e,j})$
22:      - For $k \in [1, B]$
23:          - Aggregate the shares received from nodes in $CS_e$
24:          - $r_k^{(i)} \leftarrow \sum_{j \in CS_e} s_k^{(j)}$.
25:  - Upon a reconstruction request for round $r$:
26:      - Output $\mathsf{Breeze.Reconstruct}(in, reconstruct, r, r_k^{(i)})$.
```

Fig. 2: The pseudocode of Rondo for node P_i, round r and epoch e. The $\mathrm{ParVerify}$ is functions provided by Breeze.

Rondo proceeds in epochs and each epoch has B rounds. The commitment phase is executed once every epoch where n bAVSS-PO is executed. The batch size of bAVSS-PO is B. The agreement and reconstruction phases are executed in every round. Meanwhile, membership requests (i.e., JOIN or LEAVE requests) can be issued at any time to add new nodes to the system or remove existing nodes from the system. However, they are only processed in the first round of every epoch in the agreement phase.

Specifically, each phase proceeds as follows. First, every node executes the commitment phase once every B rounds (i.e., an epoch). In particular, node P_i starts an bAVSS-PO instance Breeze_i and shares B secrets $\mathbf{s^i}$ (lines 07-11). Second, the agreement phase is executed in every round r and there are two cases.

- If r is the first round of an epoch, every node waits until $n - t$ Breeze instances have completed and then proposes

(S_e, m) to Rondo-BFT (lines 12-13). Here, S_e consists of the validation data for Breeze instances. Jumping ahead, we use cer_i to denote the validation data for Breeze_i. Additionally, m denotes the input data to Rondo-BFT. Every node then waits for the output of Rondo-BFT (line 15).

- If r is not the first round of an epoch, every node simply proposes $(-, m)$ to Rondo-BFT and then waits for the output of Rondo-BFT (lines 18-19).

There are two cases for m mentioned above. If we use Rondo as a dedicated DRB protocol, the field m is either $\perp$ or some membership requests. Alternatively, if Rondo is a system that generates on-chain randomness beacon, m may additionally include a batch of regular requests.

Finally, the reconstruction phase is by default triggered in every round. In each round r, nodes reconstruct the $(r \bmod e)^{th}$ secret among the B secrets, where e is the epoch number. As $|CS_e| \geq t+1$, the reconstruction phase in fact reconstructs the secrets shared in $t+1$ Breeze instances. We thus require each node to first verify its shares based on CS_e via the $\mathrm{ParVerify}$ function (line 21, to be described in Sec. VI) and then aggregate its shares in CS_e (lines 22-24). In this way, we only need one reconstruct instance to reconstruct all the secrets corresponding to CS_e (lines 25-26).

As our protocol supports reconfiguration of the nodes, we additionally use M_e to denote the members of epoch e. To simplify our protocol and not *waste* the secrets shared by Breeze, we only support membership requests at the boundary of epochs. Namely, any membership requests are included in the m field as input to Rondo-BFT in the first round of an epoch (line 13). After Rondo-BFT outputs the results that involve membership requests, nodes update their M_{e+1} accordingly (line 16). Starting from epoch $e+1$, nodes use M_{e+1} to identify the set of nodes in the system. We leave the discussion about the details in Sec. VII when we present Rondo-BFT. We show the proof for Rondo in Appendix B and proof for Rondo-BFT in our full paper [70].

B. Discussion

In the reconstruction phase, every node aggregates the shares in CS_e. As our bAVSS-PO does not achieve the *completeness* property, some honest nodes may not receive shares. In this case, nodes need to exchange their shares to ensure that all correct nodes can receive the shares corresponding to CS_e, which incurs $O(\lambda n^3)$ communication. Alternatively, we can build an optimized solution, inspired by [58]. In particular, at the end of the commitment phase, every dealer sends the *missing shares and proofs* (the shares for those nodes that do not have a signature in the validation data) to the leader of the Rondo-BFT. In Rondo-BFT, every correct node first validates the missing shares before voting. Such an approach ensures that every honest node has a valid share from each dealer in CS_e before the reconstruction phase and does not increase the communication complexity of our approach.

VI. BREEZE: COMPUTATION-EFFICIENT bAVSS-PO

In this section, we present the workflow of our Breeze protocol using the batched polynomial commitment (defined in Sec. III) in a black-box manner and demonstrate how our proposed concept of "partial output (PO)" operates within the protocol.

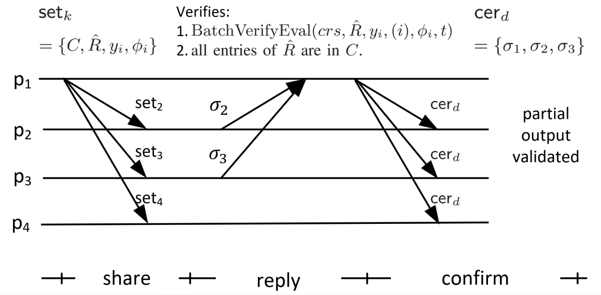

set$_k$
$= \{C, \hat{R}, y_i, \phi_i\}$
Verifies:
1. $\mathrm{BatchVerifyEval}(crs, \hat{R}, y_i, (i), \phi_i, t)$
2. all entries of $\hat{R}$ are in C.
cer$_d$
$= \{\sigma_1, \sigma_2, \sigma_3\}$
p$_1$
p$_2$
p$_3$
p$_4$
set$_2$
set$_3$
set$_4$
σ_2
σ_3
cer$_d$
cer$_d$
cer$_d$
partial output validated
share
reply
confirm

Fig. 3: The Breeze protocol.

A. The Main Workflow

The sharing stage (Figure 4). The sharing stage involves three communication phases: share, reply, and $\mathsf{confirm}$, also as shown in Figure 3. In the share phase, the dealer P_d first generates B secrets $\{s_1, \cdots, s_B\}$ and the shares of them. Additionally, P_d generates correctness proofs for the shares using our new batching polynomial commitment $\mathrm{BatchCommit}$ and $\mathrm{BatchEval}$. To guarantee the consistency of the shares, the dealer P_d also creates a vector commitment C for the shares by querying the vCom function (line 04). We instantiate the vector commitment using Merkle Tree.

In the reply phase, if node P_i validates the share it receives from P_d, P_i sends a digital signature for C to P_d. At least $2t+1$ matching signatures form a *certificate* cer_d. After collecting cer_d, P_d sends cer_d as the validation data to the nodes.

The share each node receives from P_d is called a *partial output* for the batch of secrets. Additionally, cer_d is used as the *validation data* for the sharing stage. Upon receiving cer_d, each node can verify the validity of its partial output. Since nodes may not receive cer_d during the sharing stage (e.g., when P_d is faulty), we also define a $\mathrm{ParVerify}$ function for each node to independently verify the validity of its partial output. Namely, each node may trigger the $\mathrm{ParVerify}$ function even if the node does not complete the sharing stage.

Reconstruction stage (Figure 5). Each reconstruction procedure recovers one secret in the batch of B secrets and we use idx as the index of the secret to be reconstructed. Specifically, each node P_i may hold the shares $\{R_k(i) | k \in [1, B]\}$ as the partial output of the sharing stage. Before participating in the reconstruction stage, each node needs to validate its partial output by invoking the $\mathrm{ParVerify}$ function (if it has not done so yet). If so, P_i sends its idx^{th} share $R_{idx}(i)$ to all nodes. After receiving $t+1$ valid shares, P_i interpolates the corresponding points and recovers the secret $R_{idx}(0)$.

B. Instantiating Batched Polynomial Commitment

The batched polynomial commitment scheme consists of three functions: $\mathrm{BatchCommit}$, $\mathrm{BatchEval}$ and $\mathrm{BatchVerifyEval}$.

Batching commitments (Figure 6). The $\mathrm{BatchCommit}$ function takes input the common reference string crs, the set of polynomials $\{S_i | i \in [1, B]\}$, the degree of the polynomials p, and outputs a commitment $\hat{S}$ arranged as a vector.

```
▷ Initialization: Set $flag_1 \leftarrow$ false.                                  //for validation of the partial output
▷ The share phase.
    - Upon receiving (ID.$d$, in, share, $s_1, \cdots, s_B$), as a dealer (i.e., $P_i = P_d$):
        - For $k \in [1, B]$, randomly choose a recovery polynomial $R_k \in \mathbb{F}[x]$ of degree $t$ s.t. $R_k(0) = s_k$.
        - Compute $\hat{R} \leftarrow \mathrm{BatchCommit}(crs, \{R_k | k \in [1, B]\}, t)$.
        - Compute $C \leftarrow \mathrm{vCom}(\hat{R}[1], \cdots, \hat{R}[B])$.                    //create a root commitment
        - Compute $(Y, \Phi) \leftarrow \mathrm{BatchEval}(crs, \{R_k | k \in [1, B]\}, \hat{R}, (1, \ldots, n), t)$.   //evaluates of the polynomials
        - For $k \in [1, n]$, $y_k \leftarrow Y[:, k]$, $\phi_k \leftarrow \Phi[k]$.
        - For $k \in [1, n]$, send (ID.$d$, send, set$_k$) to node $P_k$, where set$_k = \{C, \hat{R}, y_k, \phi_k\}$.
▷ The reply phase.                                                                //executed by any node
    - Upon receiving (ID.$d$, send, set$_d$) from the dealer $P_d$ for the first time:
        - Verify that $\mathrm{BatchVerifyEval}(crs, \hat{R}, y_i, (i), \phi_i, t)$ returns true.
        - Verify that all entries of $\hat{R}$ are in $C$.
        - Set $flag_1 \leftarrow$ true and create a digital signature $\sigma_i$ for $C$.
        - Send (ID.$d$, reply, $\sigma_i$) to dealer $P_d$.
▷ The confirm phase.
    - if $P_i$ is the dealer (i.e. $P_i = P_d$) then
        - Let cer$_i$ be the set of signatures received from other nodes, cer$_i \leftarrow [\bot]$.
        - Upon receiving (ID.$d$, reply, info$_{j,i}$) from $P_j$ for the first time:
            - If $\sigma_j$ is a valid signature for $C$, then cer$_i \leftarrow$ cer$_i \cup \{\sigma_j\}$.
        - if $|$cer$_i| \geq 2t + 1$ then
            - Send (ID.$d$, confirm, $C$, cer$_i$) to all nodes.
            - Output (ID.$d$, out, shared, set$_i$).
    - Upon receiving (ID.$d$, confirm, $C$, cer$_d$) from $P_d$ for the first time where cer$_d$ is a valid certificate for $C$   //executed by any node
            - if $flag_1 =$ true then                                            //the partial output is validated
                - Output (ID.$d$, out, shared, set$_i$)
▷ func $\mathrm{ParVerify}($cer$_d, C,$ set$_i)$:
    - if cer$_d$ is a valid certificate for $C$ and $C =$ set$_i.C$ and $\mathrm{BatchVerifyEval}(crs, \hat{R}, y_k, (i), \phi_i, t)$ then
        - Output true.                                                         //the partial output is validated
    - Output false.
```

Fig. 4: The sharing stage of Breeze, for node P_i and tag ID.d. A certificate cer$_d$ is a valid certificate for C if cer$_d$ consists of $2t+1$ digital signatures for C.

```
- Upon Breeze.Reconstruct (in, share, $s$, $idx$, $R_{idx}(i)$):                       //reconstruct the $idx^{th}$ secret.
    - Send (reconstruct-share, $C$, $\hat{R}[idx]$, $y_i[idx]$, $\phi_i[idx]$) to node $P_j$, for $j \in [1, n]$.
- Upon receiving (reconstruct-share, $C$, $\hat{R}[idx]$, $y_m[idx]$, $\phi_m[idx]$) from node $P_m$:
    - if $\hat{R}[idx]$ is in $C$ and $\mathrm{BatchVerifyEval}(crs, \hat{R}[idx], y_m[idx], (m), \phi_m[idx], t)$ returns true then
        - if $t + 1$ valid reconstruct-share messages with the same $C$ are received then
            - Interpolate $R[idx]$ from the $t + 1$ valid points.
            - Output (out, reconstructed, $R_{idx}(0)$)
```

Fig. 5: The reconstruction stage of Breeze for the idx^{th} secret in the batch. Code is shown for node P_i and tag ID.d.

```
func $\mathrm{BatchCommit}(crs, \{S_i | i \in [1, B]\}, p)$:
 - For $j \in [1, B]$, compute $\hat{S}[j] = \prod_{i=1}^{p+1} g_i^{\mathbf{s}[j,i]}$, where $\mathbf{s}[j, :]$
   are the coefficients vector of $S_j$.
 - Output $\hat{S}$
```

Fig. 6: The $\mathrm{BatchCommit}$ function.

To do so, for each S_i, compute Bulletproofs of S_i, and then set $\hat{S}[i]$ as the Bulletproofs. Accordingly, $\hat{S}$ is a vector with n components.

Batching evaluation (Figure 7). The $\mathrm{BatchEval}$ function takes input the common reference string crs, a set of polynomials $\{S_i | i \in [1, B]\}$, the polynomial commitment $\hat{S}$, evaluation points $(y_1, y_2, \ldots, y_m)$, and the degree of the polynomial p. For simplicity, we use the evaluation points in their vector forms, namely $[1, y_i^1, y_i^2, \ldots, y_i^p]$. The function outputs (V, Φ), where V is the evaluation matrix, and Φ is the proof vector. The function proceeds as follows.

- Line 01-02: Select a blind mask $\mathbf{d}$ to mask the coefficients of the input polynomial.
- Line 03-04: Compute the evaluation matrices V and V_D using inner products and combine the input polynomials into a single polynomial.
- Line 05-06: Commit $\sigma + \mathbf{d}$ and compute the corresponding evaluation matrix V'.
- Line 07-09: Calculate the parameters required for $\mathrm{InnerProductProof}$.
- Line 10-12: Recursively invoke $\mathrm{InnerProductProof}$ to generate a correctness proof for the inner product and append the commitment $\hat{D}$ and evaluation V_D to the proof.
- Line 13-16: If the input contains only one coefficient, include this coefficient in the proof and outputs.
- Line 17-19: Handle cases where the number of coefficients is odd by excluding the last entry and updating corresponding parameters.
- Line 20-34: Halve the input vectors and generate sub-statements for inner product recursion. $\mathrm{BatchEval}$ then recursively invokes $\mathrm{InnerProductProof}$.

Batching verification (Figure 8). The function verifies the

```
**func** BatchEval($crs, \{S_i | i \in [1, B]\}, \hat{S}, (y_1, y_2, \cdots, y_n), p$):
01:  - Parse $crs$ as $(\mathbb{G}, \mathbf{g}, h)$.
02:  - Randomly select a blind mask $\mathbf{d}$, then set $\hat{D}$ as the
       commitment of $\mathbf{d}$.
03:  - $V \leftarrow \mathbf{s} \cdot \mathbf{y}, V_D \leftarrow \mathbf{d} \cdot \mathbf{y}$.
04:  - Compute $\boldsymbol{\sigma} \leftarrow \sum_{i=1}^{B} \gamma_i \mathbf{s}[i, :]$, in which $\gamma_i$ is the random
       number generated by Fiat-Shamir heuristic.
05:  - Compute $\widehat{SD} \leftarrow \mathrm{BatchCommit}(crs, \boldsymbol{\sigma} + \mathbf{d}, p)$ as a
       commitment of the polynomial covered with blind mask.
06:  - Compute $V' \leftarrow \sum_{i=1}^{B} \gamma_i V[i, :] + V_D$ as the corresponding
       evaluations.
07:  - For $i \in [1, n]$, generate the inner product statement $stmt_i$
       and the challenges $z_i \leftarrow H(stmt_i)$.
08:  - Compute $\widehat{SD}'[i] \leftarrow \widehat{SD} h^{z_i \cdot V'[i]}$.
09:  - $\mathbf{h}^{\mathbf{z}} \leftarrow (h^{z_1}, \ldots, h^{z_n})$.
10:  - $\Phi \leftarrow \mathrm{InnerProductProof}((\mathbb{G}, \mathbf{g}, \mathbf{h}^{\mathbf{z}}), \widehat{SD}', \mathbf{y}, p, \boldsymbol{\sigma} + \mathbf{d})$.
11:  - For $i \in [1, n]$, $\Phi[i] \leftarrow \Phi[i] |||| \hat{D} || V_D[i]$.
12:  - Output $(V, \Phi)$.

**func** InnerProductProof($(\mathbb{G}, \mathbf{g}, \mathbf{h}), A, \mathbf{y}, p, \mathbf{a}$):
13:  - Initialize the proof and a flag, $\Phi \leftarrow \perp, flag \leftarrow 0$
14:  - **if** $p = 1$ **then**
15:      - Affix the proof $\Phi$ with the only entry of the vector.
16:      - Output $\Phi$.
17:  - **if** $p$ is odd **then**
18:      - Cut off the last entry of $\mathbf{a}$ and let $\tilde{a}$ denote the negative
value of this entry. Set this entry into the proof. Also, cut off the last
entry of $\mathbf{y}$ and $\mathbf{g}$.
19:      - Set $flag \leftarrow 1, p \leftarrow p - 1$, $A[j, i] \leftarrow A[j, i] g_p^{\tilde{a}[j]} h^{\tilde{a}[j]\mathbf{y}[p, i]}$.
20:  - Let $p \leftarrow p/2$, then halve $\mathbf{a}, \mathbf{y}, \mathbf{g}$ into sub-matrices: let $\mathbf{a}_L, \mathbf{y}_L$,
       $\mathbf{g}_L$ be the left halves and $\mathbf{a}_R, \mathbf{y}_R, \mathbf{g}_R$ be the right halves.
21:  - Set the following local parameters: $\mathbf{c}_L \leftarrow \mathbf{a}_L \cdot \mathbf{y}_R$,
       $\mathbf{c}_R \leftarrow \mathbf{a}_R \cdot \mathbf{y}_L$, $L[i] \leftarrow \mathbf{g}_R^{\mathbf{a}_L} h^{z_i \mathbf{c}_L[i]}$, $R[i] \leftarrow \mathbf{g}_L^{\mathbf{a}_R} h^{z_i \mathbf{c}_R[i]}$.
22:  - $\Phi[i] \leftarrow \Phi[i] || (L[i], R[i])$.
23:  - Build a Merkle Tree over all transcript as follows:
24:      - **if** $flag = 1$ **then**
25:        - $leaf_i \leftarrow H(\mathbf{g}, p, h^{z_i}, \mathbf{y}[:, i], \mathbf{A}[i], L[i], R[i], \tilde{a})$
26:        - $z \leftarrow roothash$, $b_i \leftarrow MerkleBranch(i)$.
27:      - **else**
28:        - $leaf_i \leftarrow H(\mathbf{g}, p, h^{z_i}, \mathbf{y}[:, i], \mathbf{A}[i], L[i], R[i])$.
29:        - $z \leftarrow roothash$, $b_i \leftarrow MerkleBranch(i)$.
30:  - $\Phi[i] \leftarrow \Phi[i] || (z, b_i)$.
31:  - $A'[i] \leftarrow L^{z^2}[i] A[i] R^{z^{-2}}[i]$, $\mathbf{g}' \leftarrow \mathbf{g}_L^{z^{-1}} \mathbf{g}_R^{z^1}$.
32:  - $\mathbf{y}' \leftarrow z^{-1} \mathbf{y}_L + z^1 \mathbf{y}_R$, $\mathbf{a}' \leftarrow z^1 \mathbf{a}_L + z^{-1} \mathbf{a}_R$.
33:  - $crs' = (\mathbb{G}, \mathbf{g}', h)$.
34:  - Output $\Phi = \Phi || \mathrm{InnerProductProof}(crs', A', \mathbf{y}', p', \mathbf{a}')$.
```

Fig. 7: The BatchEval function.

correctness of the output of the BatchEval function.

- Line 01-05: Generate the statement in the same way as that in BatchEval.
- Line 06-22: Parse the parameters from the proof and repeat the computation in BatchEval to verify the proof.

```
**func** BatchVerifyEval($crs, \hat{S}, \mathbf{v}, \mathbf{y}, \phi, p$):
01:  - Parse $crs$ as $(\mathbb{G}, \mathbf{g}, h)$.
02:  - Generate the inner product statement $stmt = (\widehat{SD}, \mathbf{y}, \mathbf{v}')$
03:  - $z \leftarrow H(stmt)$
04:  - Compute $\widehat{SD}' \leftarrow \widehat{SD} h^{zv'}$.
05:  - OutputVerifyInnerProduct($(\mathbb{G}, \mathbf{g}, h^z), \widehat{SD}', \mathbf{y}, p, \phi$).

**func** VerifyInnerProduct($(\mathbb{G}, \mathbf{g}, h), A, \mathbf{y}, p, \phi$):
06:  - **if** $p = 1$ **then**
07:      - Parse $\mathbf{a}$ from $\phi$.
08:      - Output $(A = g_0^{\mathbf{a}} h^{\mathbf{a}y})$.
09:  - **if** $p$ is odd **then**
10:      - Parse $\tilde{a}$ from $\phi$ and update the parameters $\phi, A, \mathbf{y}, \mathbf{g}, p$.
11:      - Parse $(L, R)$ from $\phi$.
12:      - Compute $leaf \leftarrow H(\mathbf{g}, p, h^{z_i}, \mathbf{y}, A, L, R, \tilde{a})$.
13:  - **else**
14:      - Parse $(L, R)$ from $\phi$.
15:      - Compute $leaf \leftarrow H(\mathbf{g}, p, h^{z_i}, \mathbf{y}, A, L, R)$.
16:  - Parse $c \leftarrow roothash$ and $b$ from $\phi$.
17:  - Verify whether the Merkle Tree commitment is $(leaf, c, b)$.
       Otherwise, output false.
18:  - Halve $\mathbf{g}$ into two sub-matrix $\mathbf{g}_L$ and $\mathbf{g}_R$.
19:  - $A' \leftarrow L^{c^2} A R^{c^{-2}}$, $\mathbf{g}' \leftarrow \mathbf{g}_L^{c^{-1}} \mathbf{g}_R^{c^1}$.
20:  - $y' \leftarrow c^{-1} \mathbf{y}_L + c^1 \mathbf{y}_R$.
21:  - $crs' = (\mathbb{G}, \mathbf{g}', h)$
22:  - Output VerifyInnerProduct($crs', A', \mathbf{y}', p', \phi$).
```

Fig. 8: The BatchVerifyEval function

C. System Optimization in Rondo

When Breeze is used in Rondo, we further optimize the reconstruction stage to reduce the computational overhead. The idea is that the reconstruction stage reconstructs $t+1$ secrets (that belong to the common subset CS_e) at a time and every node computes the Lagrange interpolation as follows:

$$R_{idx} = \sum_{i=1}^{t+1} S_{idx,i}(i) \prod_{j \neq i, j \in CS_e} \frac{-j}{i-j}$$

This costs $O(n^2)$ multiplication per round. Hence, in each epoch, each node needs to perform $O(n^2 B)$ multiplications.

An interesting fact is that the product term is determined by the common subset CS_e and CS_e is fixed before the reconstruction stage (for any secret) begins. Therefore, we can pre-compute this product once in every epoch. Using this trick, each node only has to perform $O(n^2)$ multiplication during reconstruction, B times lower than the trivial case.

D. Discussion

In our Breeze protocol, the PO is the secret share each node receives from the dealer, and the validation data includes $2t+1$ digital signatures. Each node obtains the validation data during the agreement phase, so the completeness property is not required for secret sharing.

The correctness of Breeze is ensured by the batched polynomial commitment and validation data (signatures). The commitment guarantees accurate polynomial evaluations. The validation data ensures at least $t+1$ honest nodes verify their shares (PO) originate from the same polynomial. The secret is thus "fixed". The proof is in Appendix C.

Breeze achieves $O(\lambda n(B + \log n))$ communication if we instantiate the certificate using aggregate signatures. If we set B as $O(\log n)$, Breeze achieves $O(\lambda n)$ amortized communication. Our result is the most optimal result so far, as existing protocols achieve $O(\lambda n)$ amortized communication by setting B as $O(n)$ [38,40].

Our $\mathrm{BatchEval}$ function can be viewed as a computation-efficient function of that in hbACSS. Specifically, $\mathrm{BatchEval}$ combines B polynomials into a single polynomial (line 04 of Figure 7). It uses B random numbers to create a linear combination of input polynomials, reducing computational complexity by a factor of B compared to hbACSS.

Unlike hbACSS, Breeze resolves subtle issues. First, the *hbPolyCommit-Core* algorithm [37, Fig.1] in hbACSS uses an incorrect crs, making share verification impossible. This is fixed in line 09 of Figure 7. Second, hbACSS's $\mathrm{InnerProductProof}$ fails to handle even-sized input vectors in its Merkle tree construction. We introduce a $flag$ parameter (lines 22-28) to address this.

Our batched polynomial commitment supports generic bAVSS protocols. For example, integrating it with HAVEN [55] yields BatchHAVEN, which assumes only a discrete logarithm (DLog) in the random oracle model and avoids PKI. Details are in our full paper [70].

VII. Rondo-BFT: Dynamic BFT

We present Rondo-BFT, a dynamic HotStuff protocol [46]. HotStuff is a leader-based partially synchronous protocol. The normal-case operation in HotStuff involves four phases: propose, prepare, pre-commit, and commit. Following the notion of Rondo, membership changes only occur at the boundary of the epochs. Each epoch has B *rounds* and up to B blocks can be delivered in an epoch. The protocol is viewed-based and there is only one leader in each *view* v and configuration. We use $\text{LEADER}(v, M)$ to denote the leader of view v given members of a configuration M. A *view change* mechanism might be started to elect a new leader. In this section, we sketch the workflow of the normal-case protocol of Rondo-BFT. We highlight the difference from HotStuff (which is designated to handle membership requests) in green. The implementation-level details of our protocol (normal-case and view change) and the proofs can be found in our full paper [70].

A. Rondo-BFT-Specific Data Structures

Blocks and branches. A block is in the form $b = [pl, view, height, rq]$, where pl is the hash of the parent block of b, $view$ is the view number, $height$ is the number of blocks on the branch led by b, rq is a batch of client requests. The first block on a branch is set as an empty *genesis block* with height 0. For a block b, we use $b.x$ to denote the element x of b. If block b satisfies $b.height \in [eB+1, eB+B]$, then b is a block in epoch e. If b is the first block in epoch e, i.e., $b.height \mod B = 1$, then $b.rq$ may contain membership requests. Otherwise, $b.rq$ only contains regular requests. In addition, $b.parent$ denotes the parent block of b.

Quorum certificates (QC) and view-change certificate. A QC consists of $2t + 1$ matching votes for a message m, which is instantiated by an aggregate signature in our work. There are three types of QCs in the normal-case operation: $prepareQC$, $precommitQC$, $commitQC$. Meanwhile, view-change certificates are generated during view changes. A view change certificate vc for view v consists of new-view messages in view v from more than two-thirds of nodes in a configuration. Based on vc, each node P_i checks whether it is the correct new leader and then enters the new view. Any node can verify the identity of P_i via vc. Here, vc is also called a view-change certificate for P_i.

Rank of QCs and blocks. We use the notion of *rank* [71]–[73] in our work. Given a QC qc for a block, the *rank*(qc) function does not explicitly return a value. Instead, we only care if the rank of a block is higher than that of another one. In our protocol, ranks equal heights by default. During view changes, a block with a higher view has a higher rank than other blocks. The rank of qc is the same as that of *block*(qc).

Local state. Each node maintains the following state parameters: (i) the current view number $cview$; (ii) a quorum certificate $highQC$, i.e., the highest $prepareQC$ for which the node has sent a pre-commit message; (iii) a $lockedQC$, i.e., the highest QC for which the node has sent a commit message; (iv) $confirmQC$, i.e., the $commitQC$ for the latest delivered block; (v) the last voted block lv; (vi) the current system configuration M and its epoch e; and (vii) the *temporary membership* of a configuration TM.

B. The Protocol

Normal-case operation (Figure 9). In the propose phase, the leader P_L proposes a new block b with requests $(*, m)$ (also specified in the API in Figure 2). Here $b.height = (e-1)B+r$, $r \in [1, B]$, and b extends the block of the $highQC$ of P_L. If $r = 1$, membership requests might be included in m. We distinguish JOIN and LEAVE requests here. For LEAVE requests, the leader directly includes the requests in $b.rq$. For each JOIN request $(\mathsf{join}, pk_\varepsilon)$, the leader assigns an id ε to the new node and then includes a message $(\mathsf{add}, \varepsilon, pk_\varepsilon)$ in $b.rq$. Meanwhile, P_i adds P_ε to its *temporary membership* TM. After the requests are packed, the leader sends $(\mathsf{prepare}, cview, b, highQC)$ to TM.

Upon receiving a $(\mathsf{prepare}, cview, b, qc)$ message from the leader, P_i checks whether one of the conditions is satisfied for block b. One noteworthy requirement we introduce is highlighted as condition ii). This check requires that b be the first block proposed for view v, b extends *block*$(lockedQC)$ and *block*$(lockedQC)$ has already been delivered. Such a change is crucial for our protocol to achieve liveness. After the conditions are checked, P_i sends a partially signed prepare message for b to the leader P_L. If $r = 1$ and b consists of some $(\mathsf{add}, \varepsilon, pk_\varepsilon)$ request, P_i helps P_ε complete the state transfer via a catchup message. We ignore the details on state transfer as it largely follows previous work [44].

After collecting $2t + 1$ matching prepare messages, the signatures in the prepare messages form a $prepareQC$. P_L updates its $highQC$ as $prepareQC$ and enters the pre-commit phase. Such a procedure is repeated in the commit phase, similar to that in HotStuff. We highlight the changes we made here. First, in the pre-commit phase, the leader broadcasts the pre-commit message to all nodes in $TM \cup M_{e-1}$, including the previous configuration, existing configuration, and the new nodes to be added. If P_i is a node in M_{e-1} but not in M_e (i.e., P_i requests to leave the system), it waits until receiving the pre-commit message in epoch e and delivering all the blocks in epoch $e - 1$ before leaving the system. In addition, if P_i is a node in M_e, after receiving the pre-commit message for the first block in epoch e, P_i forwards this message to all nodes in

Propose phase: $\triangleright$ Upon $\mathsf{RondoBFT.Propose}_r(e, r, (*, m))$, P_L proposes a new block b with a height of $(e-1)B+r$ including requests $(*, m)$ If $r = 1$, for LEAVE requests in m, P_i directly packs them in $b.rq$; for any JOIN request in m, P_L assigns and ID ε to the new node P_ε, includes $(\mathsf{add}, \varepsilon, pk_\varepsilon)$ in $b.rq$, and included P_ε in its TM. Then P_L sends b to all the nodes in TM via a $(\mathsf{prepare}, v, b, qc)$ message, where qc is the $highQC$ of P_L.

Prepare phase: $\triangleright$ Upon receiving block b from P_L, each node P_i checks whether i) b is consistent with HotStuff specifications using qc; or ii) b is the first block proposed for view v, b extends $block(lockedQC)$ and $block(lockedQC)$ is delivered. If one of i) and ii) holds, P_i sends a prepare message for b to P_L. If $r = 1$, for any ADD request in b proposed by P_ε, P_i adds P_ε to its TM and sends local state to P_ε via a catchup message.

Pre-Commit phase: $\triangleright$ Upon recieving $2t + 1$ prepare messages for block b from nodes in M_e, P_L forms a $prepareQC$ qc and sends qc via a pre-commit message to $TM \cup M_{e-1}$. $\triangleright$ Upon receiving a pre-commit message for b from P_L, each node P_i updates its $highQC$. If $P_i \in M_e$, P_i sends a pre-commit message for b to P_L. If $r = 1$, P_i also forwards the received message to all the leaving nodes in $M_{e-1} - M$. If $P_i \in M_{e-1} - M$, then P_i directly leave the system.

Commit phase: $\triangleright$ Upon receiving $2t + 1$ pre-commit messages for b from nodes in M_e, P_L forms a $precommitQC$ qc and broadcasts qc in commit messages to TM. $\triangleright$ Upon receiving a pre-commit message for b, each node P_i in M_e updates its $lockedQC$ and sends a commit message for b to P_L.

Decide phase: $\triangleright$ Upon receiving $2t + 1$ commit messages for b from nodes in M_e, P_L forms a $commitQC$ qc and sends qc in decide messages to all the nodes in TM. $\triangleright$ Upon receiving a decide message for b from P_L, each node P_i delivers block b as that in HotStuff and updates its $confirmQC$. If $r = 1$, P_i computes M_{e+1} according to membership requests in b and $\mathsf{RondoBFT.Return}_{e,r}(*, M_{e+1}, m)$. If $r = B$, then P_i forward the decide message to M_{e+1}. If $P_i \in M_{e+1}$, after receiving commit messages for b from $2t + 1$ notes in M_e, P_i updates its view to $v + 1$ and installs M_{e+1} as current configuration.

Fig. 9: Normal-case operation of Rondo-BFT in view v, epoch e, and round r. M_e denote the configuration for epoch e. P_L denote $\textsc{Leader}(v, M_e)$. By slightly abusing notation, we assume $|M_e| = 3t + 1$ in the pseudocode.

M_{e-1}. As mentioned in Sec. IV, such changes ensure that the agreement property for the membership requests is achieved by all correct nodes nearly simultaneously. Second, in the decide phase, the leader sends the commit message to all nodes in TM. If P_i is a node in M_e, after receiving the decide message for block b, P_i delivers the branch led by b as that in HotStuff. In particular, if b consists of membership requests, P_i outputs the new configuration according to the client requests. If r is the last round of an epoch, P_i sends a $(\mathsf{decide}, cview, b, qc)$ message to $M_e \cup M_{e+1}$, including the current and the next configuration. The idea is again for correct nodes to achieve agreement nearly simultaneously so nodes can correctly install a new configuration. If P_i is a node in the new configuration M_{e+1}, after receiving the decide message for the last block in epoch e from $2t + 1$ nodes in M_e, P_i delivers the block and then installs the new configuration (which also match lines 2-3 in Figure 2).

The view change protocol. Compared to HotStuff, we made two major changes for Rondo-BFT to support membership requests. First, inspired by Dyno, we introduce a novel message forwarding process. When view change happens in epoch e and view v, P_i broadcasts its $highQC$ and $confirmQC$ via a new-view message to TM. Upon receiving a new-view message m from node P_j, P_i checks whether P_j has installed the latest configuration. If the highest QC contained in m is from a configuration $M_{e'}$ such that $e' < e$, then P_i forwards the message to the current leader $\textsc{Leader}(v{+}1, M_e)$. Otherwise, P_i updates its local state according the latest QCs. This forwarding process ensures that all correct nodes are finally aware of the view change and the latest system configuration. Second, the new leader needs to provide a view-change certificate signed by $2t + 1$ nodes. Note in normal cases, a correct node only installs a new configuration after $2t + 1$ nodes in the current configuration have agreed on the last block of the current epoch. Thus, during view change, only one leader can collect a valid view-change certificate and proposes new blocks, addressing the liveness issue caused by *competing leaders*.

Discussion. Rondo-BFT achieves $O(n)$ message complexity in normal-case operation and has $O(n^2)$ messages only during configuration changes and view changes. Namely, nodes need to forward their messages to all nodes across configurations and views. This is "unavoidable" as we need the agreement property by correct nodes nearly simultaneously. As we use aggregate signatures to instantiate quorum certificates, Rondo-BFT achieves $O(\lambda n^2 \log n)$ communication.

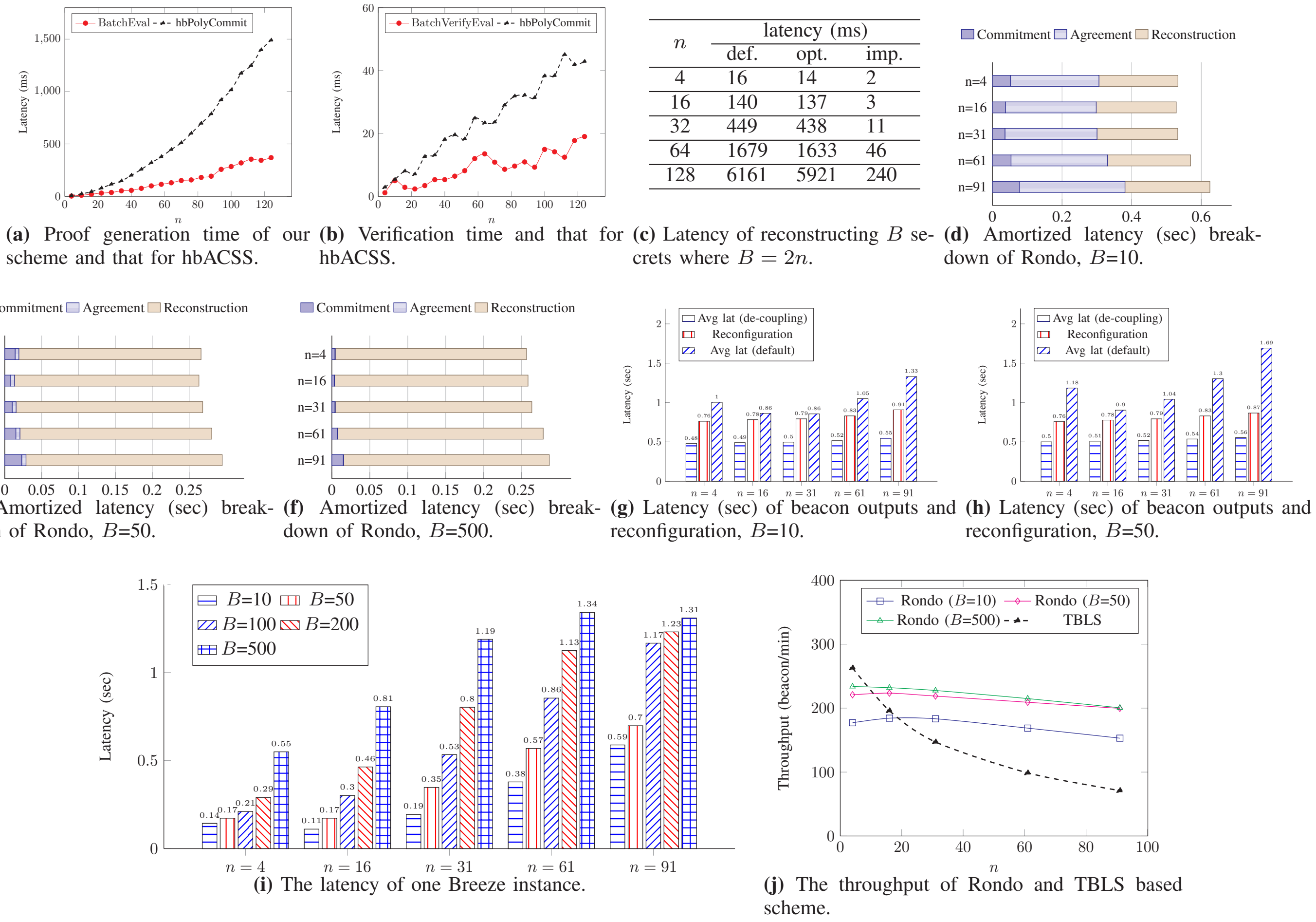
VIII. Implementation and Evaluation

We implement Rondo in Golang, including both Breeze and Rondo-BFT. We use the kyber crypto library[1] to implement our batched polynomial commitment scheme. In the same library, we also implement the batch verification functions of hbACSS [37] to compare the computational cost. We use gRPC as the communication library. We use HMAC to realize the authenticated channel and SHA256 as the hash function. Our implementation involves around 13,000 new LOC for the system and about 1,000 LOC for evaluation.

We evaluate the performance of our protocols on Amazon EC2 using up to 91 virtual machines (VMs). We use *c5.4xlarge* instances for our evaluation. The c5.xlarge instance has 16 virtual CPUs and 32GB memory. We distribute the nodes evenly in four different regions: us-west-2 (Oregon, US), us-east-2 (Ohio, US), ap-southeast-1 (Singapore), and eu-west-1 (Ireland). We vary the network size and batch size B in our evaluation. We use n to denote the network size where $n = 3t + 1$ nodes are launched in each experiment.

For performance evaluation, we focus on the computational cost of our batched polynomial commitment scheme and the throughput of Rondo. To assess the computational cost, we compare our scheme with the hbPolyCommit function in

[1] The kyber crypto library: https://github.com/dedis/kyber



(a) Proof generation time of our scheme and that for hbACSS.

(b) Verification time and that for hbACSS.

n	latency (ms)		
	def.	opt.	imp.
4	16	14	2
16	140	137	3
32	449	438	11
64	1679	1633	46
128	6161	5921	240

(c) Latency of reconstructing B secrets where $B = 2n$.

(d) Amortized latency (sec) breakdown of Rondo, B=10.

(e) Amortized latency (sec) breakdown of Rondo, B=50.

(f) Amortized latency (sec) breakdown of Rondo, B=500.

(g) Latency (sec) of beacon outputs and reconfiguration, B=10.

(h) Latency (sec) of beacon outputs and reconfiguration, B=50.

(i) The latency of one Breeze instance.

(j) The throughput of Rondo and TBLS based scheme.

Fig. 10: Evaluation results.

hbACSS. For the DRB protocol, we compare Rondo with that using threshold Boneh-Lynn-Shacham (TBLS) signature using the TBLS implementation in kyber. Indeed, some industrial-level randomness beacon protocols such as Drand[2] and Dfinity[3] also use TBLS. Such a scheme requires a DKG setup, which might be expensive. In our evaluation, we ignore the latency for DKG and only assess the performance of generating a randomness beacon using TBLS.

Batched polynomial commitment. We first compare the computational cost of hbPolyCommit and our $\mathsf{BatchEval}$ function. As no network communication is needed, we evaluate the performance on one machine with Intel i7 CPU with 2.70GHz and 16GB memory. Figure 10a illustrates the proof generation time with different n, where B is set as $2n$. Our $\mathsf{BatchEval}$ function consistently outperforms hbPolyCommit. When n is greater than 100, $\mathsf{BatchEval}$ is 4x-5x faster than hbPolyCommit. In addition, the latency of $\mathsf{BatchEval}$ does not increase as significantly when n grows.

Meanwhile, we show the verification time of the $\mathsf{BatchVerifyEval}$ function and hbPolyCommit in Figure 10b. Our $\mathsf{BatchVerifyEval}$ function is 2x-2.5x faster than hbPolyCommit in all the experiments we conducted.

Reconstruction. We set $B = 2n$ and evaluate the latency of reconstructing B secrets. As shown in Figure 10c, our optimized reconstruction phase improves the performance of the default reconstruction approach. Besides, the improvement is more significant as n and B grow, which matches our analysis in VI-C.

Performance of Breeze. We evaluate the performance of Breeze on EC2 by varying B and n. As shown in Figure 10i, given certain n, the latency grows as B grows. This is expected as a higher computation cost is involved when B grows. Meanwhile, given certain B, the latency grows as n grows. This is because higher communication is involved. All of our experiments were completed within 1.3 seconds. The highest latency happens when $n = 61$ and $B = 500$ (The latency is slightly higher than that for $n = 91$ and $B = 500$, and we believe this is caused by the variance of network delay), so the amortized latency for sharing each secret is only 2.68 ms.

Performance of reconfiguration. We evaluate Rondo under reconfiguration. In particular, we add/remove one node at a time and report the latency of the protocol. As shown in

[2] Drand: https://github.com/drand/drand

[3] Dfinity: https://github.com/dfinity-side-projects/random-beacon

Figure 10g-10h, the latency of reconfiguration grows slightly when n increases. In all of our experiments, the latency of reconfiguration is only slightly longer than the average latency of generating beacon outputs.

Performance of Rondo in the de-coupling mode. We implement a de-coupled mode of Rondo, where the commitment phase is executed in parallel with the other phases. We summarize the results for $B = 10$ and 50 in Figure 10g-10h. As the commitment phase is executed in parallel, the average latency of generating beacon outputs is further reduced compared to the default mode, where de-coupling is not implemented.

Performance of Rondo. We compare the performance of Rondo (default mode) and that using TBLS. For Rondo, we assess the latency of each epoch of Rondo (which involves running the commitment phase with n Breeze instances, Rondo-BFT for B rounds, and reconstruction of secrets in each round) and then report the throughput accordingly. In practice, the performance can be further optimized, e.g., the commitment phase can be executed prior to the beginning of an epoch.

We report the throughput of Rondo with different B in Figure 10j. As B grows, the throughput of our Rondo also increases. In our case, the throughput is very close to the peak when B=50. For a network with more than 4 nodes, Rondo consistently outperforms that using TBLS and Rondo does not require DKG. Furthermore, unlike almost all previous works [18, 19, 29, 31, 32, 34] (and TBLS) where the throughput degrades significantly as n grows, the throughput of Rondo does not degrade much so our protocol offers better scalability.

We show the amortized latency breakdown (for generating each beacon) in Figure 10d-10f. Our results show that when B is small, the bottlenecks of the system are the agreement and reconstruction phases. While as B grows, the overhead is dominated by the reconstruction phase. The results show that our Breeze and Rondo-BFT protocols are efficient and are not the bottleneck of the system.

IX. Conclusion

We present Rondo, a scalable distributed randomness beacon protocol and the first reconfiguration-friendly protocol in the partially synchronous model. We propose a new primitive called batched asynchronous verifiable secret sharing with partial output (bAVSS-PO) and a bAVSS-PO protocol Breeze, which is both communication-efficient and computation-efficient. To support reconfiguration, we propose Rondo-BFT, a dynamic Byzantine fault-tolerant protocol that supports efficient reconfiguration. We show that Rondo achieve both high throughput and better scalability than prior works.

Acknowledgment

We thank Sourav Das for the feedback about the paper. This work was supported in part by the National Key R&D Program of China under 2022YFB2701700, Beijing Natural Science Foundation under M23015, the National Natural Science Foundation of China under 92267203, a research grant from the Ant Group, and the Shuimu Tsinghua Scholar Program of Tsinghua University. Sisi is also with Zhongguancun Laboratory and Shandong Institute of Blockchain.

References

[1] M. O. Rabin, "Transaction protection by beacons," *Journal of Computer and System Sciences*, vol. 27, no. 2, pp. 256–267, 1983.

[2] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *SOSP*. ACM, 2017, pp. 51–68.

[3] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual international cryptology conference*. Springer, 2017, pp. 357–388.

[4] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *S&P*. IEEE, 2018, pp. 583–598.

[5] B. Adida, "Helios: Web-based open-audit voting." in *USENIX security symposium*, vol. 17, 2008, pp. 335–348.

[6] A. Mostefaoui, H. Moumen, and M. Raynal, "Signature-free asynchronous byzantine consensus with t< n/3 and o (n 2) messages," in *PODC*. ACM, 2014, pp. 2–9.

[7] A. Mostéfaoui and M. Raynal, "Signature-free asynchronous byzantine systems: from multivalued to binary consensus with t< n/3, o(n^2) messages, and constant time," *Acta Informatica*, vol. 54, no. 5, pp. 501–520, 2017.

[8] S. Duan, X. Wang, and H. Zhang, "Practical signature-free asynchronous common subset in constant time," in *CCS*, 2023.

[9] H. Zhang and S. Duan, "PACE: Fully parallelizable BFT from reproposable byzantine agreement," in *CCS*, 2022.

[10] S. Goel, M. Robson, M. Polte, and E. Sirer, "Herbivore: A scalable and efficient protocol for anonymous communication," Cornell University, Tech. Rep., 2003.

[11] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, "Vuvuzela: Scalable private messaging resistant to traffic analysis," in *SOSP*, 2015, pp. 137–152.

[12] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, "Dissent in numbers: Making strong anonymity scale," in *OSDI*, 2012, pp. 179–182.

[13] R. Dingledine, N. Mathewson, P. F. Syverson *et al.*, "Tor: The second-generation onion router." in *USENIX security symposium*, vol. 4, 2004, pp. 303–320.

[14] J. Bonneau, J. Clark, and S. Goldfeder, "On bitcoin as a public randomness source," *Cryptology ePrint Archive*, 2015.

[15] K. Chatterjee, A. K. Goharshady, and A. Pourdamghani, "Probabilistic smart contracts: Secure randomness on the blockchain," in *ICBC*. IEEE, 2019, pp. 403–412.

[16] M. Haahr, "Random. org: True random number service," *School of Computer Science and Statistics, Trinity College, Dublin, Ireland. Website (http://www. random. org). Accessed*, vol. 10, 2023.

[17] J. Kelsey, L. T. Brandão, R. Peralta, and H. Booth, "A reference for randomness beacons: Format and protocol version 2," National Institute of Standards and Technology, Tech. Rep., 2019.

[18] S. Das, V. Krishnan, I. M. Isaac, and L. Ren, "Spurt: Scalable distributed randomness beacon with transparent setup," in *S&P*. IEEE, 2022, pp. 2502–2517.

[19] A. Bhat, N. Shrestha, Z. Luo, A. Kate, and K. Nayak, "Randpiper–reconfiguration-friendly random beacons with quadratic communication," in *CCS*, 2021, pp. 3502–3524.

[20] K. Choi, A. Manoj, and J. Bonneau, "Sok: Distributed randomness beacons," *S&P*, 2023.

[21] I. Cascudo and B. David, "Scrape: Scalable randomness attested by public entities," in *ACNS*. Springer, 2017, pp. 537–556.

[22] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. R. Weippl, "Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness," in *NDSS*, 2021.

[23] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "Hydrand: Efficient continuous distributed randomness," in *S&P*. IEEE, 2020, pp. 73–89.

[24] J. Camenisch, M. Drijvers, T. Hanke, Y.-A. Pignolet, V. Shoup, and D. Williams, "Internet computer consensus," in *PODC*, 2022, pp. 81–91.

[25] A. Cherniaeva, I. Shirobokov, and O. Shlomovits, "Homomorphic encryption random beacon," *Cryptology ePrint Archive*, 2019.

[26] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *S&P*, 2017, pp. 444–460.

[27] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[28] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[29] A. Bhat, N. Shrestha, A. Kate, and K. Nayak, "Optrand: Optimistically responsive reconfigurable distributed randomness." in *NDSS*, 2023.

[30] A. Bandarupalli, A. Bhat, S. Bagchi, A. Kate, and M. Reiter, "Hashrand: Efficient asynchronous random beacon without threshold cryptographic setup," *Cryptology ePrint Archive*, 2023.

[31] R. Bacho, C. Lenzen, J. Loss, S. Ochsenreither, and D. Papachristoudis, "Grandline: Adaptively secure dkg and randomness beacon with (almost) quadratic communication complexity." 2023.

[32] "Drand - a distributed randomness beacon daemon, 2020, https://github.com/drand/drand."

[33] I. Cascudo and B. David, "Scrape: Scalable randomness attested by public entities," in *ACNS*. Springer, 2017, pp. 537–556.

[34] J. Camenisch, M. Drijvers, T. Hanke, Y.-A. Pignolet, V. Shoup, and D. Williams, "Internet computer consensus," in *PODC*, 2022, pp. 81–91.

[35] R. Bacho and J. Loss, "Adaptively secure (aggregatable) pvss and application to distributed randomness beacons," in *CCS*, 2023, pp. 1791–1804.

[36] V. Shoup and N. P. Smart, "Lightweight asynchronous verifiable secret sharing with optimal resilience," *Journal of Cryptology*, vol. 37, no. 3, p. 27, 2024.

[37] T. Yurek, L. Luo, J. Fairoze, A. Kate, and A. Miller, "hbacss: How to robustly share many secrets," in *NDSS*, 2022.

[38] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, and G. Stern, "Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation," in *CRYPTO*, 2023.

[39] M. K. Reiter, "Secure agreement protocols: Reliable and atomic group multicast in rampart," in *CCS*, 1994, pp. 68–80.

[40] N. Alhaddad, M. Varia, and Z. Yang, "Haven++: Batched and packed dual-threshold asynchronous complete secret sharing with applications," *Cryptology ePrint Archive*, 2024.

[41] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *S&P*. IEEE, 2018, pp. 315–334.

[42] R. Vassantlal, E. Alchieri, B. Ferreira, and A. Bessani, "Cobra: Dynamic proactive secret sharing for confidential bft services," in *S&P*, 2022, pp. 1335–1353.

[43] S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song, "Churp: dynamic-committee proactive secret sharing," in *CCS*, 2019, pp. 2369–2386.

[44] S. Duan and H. Zhang, "Foundations of dynamic bft," in *IEEE Symposium on Security and Privacy (SP)*, 2022.

[45] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *TOCS*, vol. 20, no. 4, pp. 398–461, 2002.

[46] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *PODC*, 2019.

[47] M. Ben-Or, R. Canetti, and O. Goldreich, "Asynchronous secure computation," ser. STOC '93, 1993.

[48] R. Canetti and T. Rabin, "Fast asynchronous byzantine agreement with optimal resilience," in *STOC*, vol. 93. Citeseer, 1993, pp. 42–51.

[49] A. Choudhury and A. Patra, "An efficient framework for unconditionally secure multiparty computation," *IEEE Transactions on Information Theory*, vol. 63, no. 1, pp. 428–468, 2016.

[50] Z. Beerliová-Trubíniová and M. Hirt, "Simple and efficient perfectly-secure asynchronous mpc," in *Asiacrypt*. Springer, 2007, pp. 376–392.

[51] A. Patra, A. Choudhury, and C. Pandu Rangan, "Efficient asynchronous verifiable secret sharing and multiparty computation," *Journal of Cryptology*, vol. 28, pp. 49–109, 2015.

[52] M. Ben-Or, B. Kelmer, and T. Rabin, "Asynchronous secure computations with optimal resilience," in *PODC*. ACM, 1994, pp. 183–192.

[53] A. Patra, A. Choudhary, and C. P. Rangan, "Efficient statistical asynchronous verifiable secret sharing with optimal resilience," in *ICITS*. Springer, 2009, pp. 74–92.

[54] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl, "Asynchronous verifiable secret sharing and proactive cryptosystems," in *CCS*, 2002, pp. 88–97.

[55] N. Alhaddad, M. Varia, and H. Zhang, "High-threshold avss with optimal communication complexity," in *FC*, 2021, pp. 479–498.

[56] S. Das, Z. Xiang, and L. Ren, "Asynchronous data dissemination and its applications," in *CCS*, 2021, pp. 2705–2721.

[57] H. Zhang, S. Duan, C. Liu, B. Zhao, X. Meng, S. Liu, Y. Yu, F. Zhang, and L. Zhu, "Practical asynchronous distributed key generation: Improved efficiency, weaker assumption, and standard model," in *DSN*, 2023, pp. 568–581.

[58] S. Das, Z. Xiang, A. Tomescu, A. Spiegelman, B. Pinkas, and L. Ren, "Verifiable secret sharing simplified," Cryptology ePrint Archive, Paper 2023/1196, 2023, https://eprint.iacr.org/2023/1196.

[59] H. W. Wong, J. P. Ma, and S. S. Chow, "Secure multiparty computation of threshold signatures made more efficient," in *NDSS*, 2024.

[60] M. Franklin and M. Yung, "Communication complexity of secure computation," in *STOC*, 1992, pp. 699–710.

[61] Z. Zhang, W. Li, Y. Guo, K. Shi, S. S. Chow, X. Liu, and J. Dong, "Fast RS-IOP multivariate polynomial commitments and verifiable secret sharing," in *USENIX Security*, 2024.

[62] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *ASIACRYPT*. Springer, 2010, pp. 177–194.

[63] L. Lamport, "The part-time parliament," *TOCS*, vol. 16, no. 2, pp. 133–169, 1998.

[64] J. R. Lorch, A. Adya, W. J. Bolosky, R. Chaiken, J. R. Douceur, and J. Howell, "The SMART way to migrate replicated stateful services," in *EuroSys*, Y. Berbers and W. Zwaenepoel, Eds., 2006, pp. 103–115.

[65] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *ATC*, 2014, pp. 305–319.

[66] J. Sousa, E. Alchieri, and A. Bessani, "State machine replication for the masses with bft-smart," in *DSN*, 2014, pp. 355–362.

[67] L. Breidenbach, C. Cachin, B. Chan, A. Coventry, S. Ellis, A. Juels, F. Koushanfar, A. Miller, B. Magauran, D. Moroz *et al.*, "Chainlink 2.0: Next steps in the evolution of decentralized oracle networks," *Chainlink Labs*, vol. 1, pp. 1–136, 2021.

[68] V. Shoup, "A theoretical take on a practical consensus protocol," *Cryptology ePrint Archive*, 2024.

[69] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of ACM*, vol. 32, no. 2, pp. 288–323, 1988.

[70] X. Meng, X. Sui, Z. Yang, K. Rong, W. Xu, S. Chen, Y. Yan, and S. Duan, "Rondo: Scalable and reconfiguration-friendly randomness beacon," *Cryptology ePrint Archive*, 2024.

[71] R. Gelashvili, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, and Z. Xiang, "Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback," in *FC*, 2022.

[72] X. Sui, S. Duan, and H. Zhang, "Marlin: Two-phase BFT with linearity," in *DSN*, 2022.

[73] S. Duan, H. Zhang, X. Sui, B. Huang, C. Mu, G. Di, and X. Wang, "Dashing and star: Byzantine fault tolerance using weak certificates," in *Eurosys*, 2024.

[74] G. Asharov and Y. Lindell, "A full proof of the bgw protocol for perfectly secure multiparty computation," *Journal of Cryptology*, vol. 30, no. 1, pp. 58–151, 2017.

Appendix A
Analysis of Computational Complexity

As introduced before, some protocols do not introduce their computational complexity in the paper. In this section, we use GRandline [31] as an example to illustrate how we analyze the computational complexity. The complexity for other protocols in Table I can be conducted in a similar way. Besides, in this paper, all the protocols count the complexity of exponentiation, except Hashrand [30], which counts hash functions.

GRandline [31] has five phases. First, nodes executes a distributed key generation protocol to obtain a key pair $(PK_i, SK_i) := (g^{f(i)}, h^{f(i)})$ for a hidden polynomial f. Then, in the commitment phase, each party P_i locally samples an α_i uniformly at random and sends $cm_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$ to all nodes. After that, in the third phase, i.e. beacon epoch, each node creates a partial signature $\sigma_i := (g_r^{\alpha_i}, e(g_r, SK_i))$, in which $g_r = H(r)$, r is the epoch number. Each node then sends it partial signature to each other. In the reconstruction phase, nodes collect $t+1$ valid signatures from distinct parties and use Lagrange interpolation to combine them. Finally, nodes compute the hash of the signature and use it as the beacon output.

The bottleneck of the computational complexity of GRandline is the commitment phase and the beacon epoch, since nodes have to compute exponentiation in these two phases. Besides, each node has to compute $2n$ exponentiation to calculate $cm_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$ in the commitment phase and n exponentiation to calculate $g_r^{\alpha_i}$ in the beacon epoch. Therefore, the computational complexity of this protocol is $O(n)$.

APPENDIX B
PROOF OF RONDO

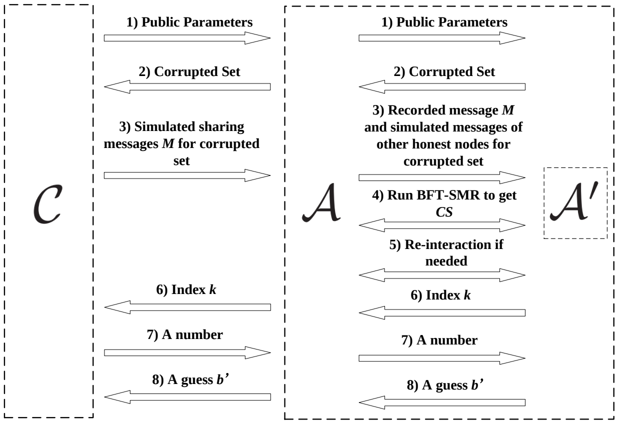
Following previous work [18], we capture the unpredictability and bias-resistance properties based on the indistinguishability game defined below.

Definition 1. *Rondo satisfies* ***indistinguishability*** *in an epoch if for any PPT adversary $\mathcal{A}$ corrupting at most t nodes, $\mathcal{A}$ wins the following game $\mathcal{G}_{beacon}$ against a challenger $\mathcal{C}$ with negligible probability.*

1) The challenger $\mathcal{C}$ generates public parameters and then sends them to the adversary $\mathcal{A}$.
2) $\mathcal{A}$ selects a set of at most t nodes and corrupts them, then sends the identities of these nodes to $\mathcal{C}$.
3) $\mathcal{C}$ and $\mathcal{A}$ interact to execute Rondo as follows:
   - $\mathcal{C}$ sends inputs to the honest nodes and simulates their operations as per the protocol in the sharing phase, while $\mathcal{A}$ has the ability to misbehave.
   - $\mathcal{A}$ aggregates the messages it receives and runs the BFT-SMR protocol with $\mathcal{C}$. In the protocol, $\mathcal{A}$ is able to observe messages sent between every pair of nodes and reorder them.
   - After at least one honest node outputs a common subset CS, $\mathcal{C}$ halt the execution.
4) $\mathcal{A}$ sends an index k to $\mathcal{C}$ indicating that it will make a guess of the beacon in the k^{th} round.
5) Then, $\mathcal{C}$ samples a bit $b \in \{0, 1\}$. Subsequently, if $b = 0$, $\mathcal{C}$ sends the corresponding beacon output of CS for the k^{th} round to $\mathcal{A}$. Otherwise, $\mathcal{C}$ sends a real random number to $\mathcal{A}$.
6) $\mathcal{A}$ makes a guess b'.
7) $\mathcal{A}$ wins the game if and only if $b = b'$.

Theorem 1. *Assuming the DLog assumption, Rondo satisfies indistinguishability.*

Proof: According to the analysis in Appendix C, assuming DLog, Breeze achieves privacy. Now we show that the indistinguishability of Rondo can be reduced to the privacy of Breeze. Concretely, if there exists a PPT adversary $\mathcal{A}'$ who can win the indistinguishability game $\mathcal{G}_{beacon}$ of Rondo with overwhelming possibility, we can construct a PPT adversary $\mathcal{A}$ to break the privacy of Breeze. This is achieved by acting as an adversary in the game $\mathcal{G}_{bavss-po}$ of bAVSS-PO and a challenger in the game of randomness beacon, as shown below.



$\mathcal{C}$
$\mathcal{A}$
$\mathcal{A}'$
1) Public Parameters
2) Corrupted Set
3) Simulated sharing messages *M* for corrupted set
6) Index *k*
7) A number
8) A guess *b'*
1) Public Parameters
2) Corrupted Set
3) Recorded message *M* and simulated messages of other honest nodes for corrupted set
4) Run BFT-SMR to get *CS*
5) Re-interaction if needed
6) Index *k*
7) A number
8) A guess *b'*

Fig. 11: Depiction of the reduction proof for beacon indistinguishability

1) $\mathcal{A}$ interacts with the challenger $\mathcal{C}$ in the game $\mathcal{G}_{bhacss}$ of Breeze to get the public parameters. Then $\mathcal{A}$ sends the parameters to $\mathcal{A}'$.
2) $\mathcal{A}'$ selects a set of at most t nodes and corrupts them. Then $\mathcal{A}'$ sends the identities of these nodes to $\mathcal{A}$, who then provide the identities to $\mathcal{C}$.
3) $\mathcal{C}$ uniformly selects a batch of random numbers as the dealer's input and simulates all the honest nodes' operations as per the protocol in the sharing phase. Then $\mathcal{A}$ records the messages received by the corrupted parties, denoted by M. Furthermore, it acts like the challenger in the game $\mathcal{G}_{beacon}$ of the randomness beacon. Notably, $\mathcal{A}$ sends the inputs to all honest nodes except for only one specified node, denoted as **a**. For node **a**, $\mathcal{A}$ does not send to **a** a real input but simulates what it has to broadcast in the sharing stage by directly sending M, the message $\mathcal{A}$ has recorded.
4) $\mathcal{A}'$ aggregate the messages it receives from $\mathcal{A}$ and run the BFT-SMR protocol with $\mathcal{A}$. $\mathcal{A}'$ is able to observe messages sent between every pair of nodes during this protocol and reorder them.
5) When at least one honest node outputs a CS, $\mathcal{A}$ halts the execution. If **a** $\notin CS$, $\mathcal{A}$ generates a new randomness beacon instance and re-interacts with $\mathcal{A}'$ as before until **a** $\in CS$.
6) $\mathcal{A}'$ sends an index k to $\mathcal{A}$, who transfers this to $\mathcal{C}$.
7) Then, $\mathcal{A}$ receives a number from $\mathcal{C}$ and transfers it to $\mathcal{A}'$. After that, $\mathcal{A}$ waits for $\mathcal{A}'$ to respond.
8) $\mathcal{A}$ responds to $\mathcal{C}$ whatever $\mathcal{A}'$ sends to it.

We point out that in step 3), those recorded information only consists of the messages that the corrupted nodes received. However, these are enough for the simulation since $\mathcal{A}$ does not need to simulate the view of the honest nodes. Furthermore, in step 5), the iteration cannot be done infinite times but at most polynomial times. This is adequate for $\mathcal{A}$, who can directly output 1 to $\mathcal{C}$ without doing steps 6)-8). The probability of this situation is at most $\frac{1}{(n-t)^{poly()}}$ which is a negligible function.

Thus, if Rondo fails to achieve indistinguishability, $\mathcal{A}$ can break the privacy of Breeze, a violation of our assumption. ■

Figure 11 depicts this procedure.

Theorem 2. *Assuming the DLog assumption, Rondo achieves unpredictability.*

Proof: From Theorem 1, Rondo achieves indistinguishability. Assume $\exists$ a PPT adversary $\mathcal{A}'$, corrupting at most t nodes, $i \geq 1, j > i+1$, such that for any negligible function $\mathsf{negl}(\lambda)$,

$$\Pr[(j, r'_j) \leftarrow \mathcal{A}(r_1, r_2, ..., r_i) : r'_j = r_j] > \mathsf{negl}(\lambda).$$

Then we can construct a PPT adversary $\mathcal{A}$ utilizes $\mathcal{A}'$ as a sub-algorithm to win the game $\mathcal{G}_{beacon}$ in Definition 1 as follows:

Firstly, $\mathcal{A}$ interacts with $\mathcal{C}$ as per the game $\mathcal{G}_{beacon}$ in step 1), then it invokes $\mathcal{A}'$ with current information it receives from $\mathcal{C}$, including previous beacon output as well as public parameters. Moreover, $\mathcal{A}$ corrupts the nodes that $\mathcal{A}'$ chooses to corrupt and sends the identities of them to $\mathcal{C}$. After that, $\mathcal{A}$ interact with $\mathcal{C}$ as per the step 3), then sends the information it received from $\mathcal{C}$ to $\mathcal{A}'$, who will return a 2-tuple (j, r'_j). Subsequently, $\mathcal{A}$ sends j to $\mathcal{C}$ in step 4). In step 6), $\mathcal{A}$ needs to make a guess. Here, if the number $\mathcal{A}$ receives from $\mathcal{C}$ is equal to r'_j, let $b' = 0$. Otherwise, let $b' = 1$. Here, it is obvious that $\mathcal{A}$ is PPT. Additionally, we have

$$|\Pr[b = b'] - \frac{1}{2}| > \mathsf{negl}(\lambda).$$

As there does not exist such $\mathcal{A}$, Rondo is unpredictable. ■

Theorem 3. *Assuming the DLog assumption, Rondo achieves bias-resistance.*

Proof: From Theorem 1, Rondo achieves indistinguishability. Assume a PPT adversary $\mathcal{A}'$, $i \geq 1$, $k \in [1, |r_i|]$ for any negligible function $\mathsf{negl}(\lambda)$,

$$\left|\Pr\left[\begin{array}{c}(k, r'_i(k)) \leftarrow \mathcal{A}(r_1, r_2, ..., r_{i-1}) : \\ r'_i(k) = r_i(k)\end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$$

or

$$|\Pr[r_i(k) = 0] - \frac{1}{2}| > \mathsf{negl}(\lambda).$$

Then we can construct a PPT adversary $\mathcal{A}$ utilizes $\mathcal{A}'$ as a sub-algorithm to win the game $\mathcal{G}_{beacon}$ in Definition 1 as follows:

Firstly, $\mathcal{A}$ interacts with $\mathcal{C}$ as per the game $\mathcal{G}_{beacon}$ in step 1), then it invokes $\mathcal{A}'$ with current information it receives from $\mathcal{C}$, including previous beacon output as well as public parameters. Moreover, $\mathcal{A}$ corrupts the nodes that $\mathcal{A}'$ chooses to corrupt and sends the identities of them to $\mathcal{C}$. After that, $\mathcal{A}$ interacts with $\mathcal{C}$ as per step 3), then sends the information it received from $\mathcal{C}$ to $\mathcal{A}'$, who will return a 2-tuple $(k, r'_{i+1}(k))$. Subsequently, $\mathcal{A}$ sends $i+1$ to $\mathcal{C}$ in step 4). Furthermore, in step 6), $\mathcal{A}$ needs to make a guess. If the k^{th} bit of the number it receives from $\mathcal{C}$ is equal to $r'_{i+1}(k)$, let $b = 0$. Otherwise, let $b = 1$. It is obvious that $\mathcal{A}$ is PPT. Additionally, we have

$$|\Pr[b = b'] - \frac{1}{2}| > \mathsf{negl}(\lambda).$$

As there does not exist such $\mathcal{A}$, Rondo is bias-resistant. ■

Theorem 4. *Rondo achieves liveness.*

Proof: In each epoch e there are at least $t+1$ correct nodes acting as dealers of Breeze instances. Without loss of generality, we assume $\{\mathsf{Breeze}_i, i \in [1, t+1]\}$ are $t+1$ correct instances. Subsequently, according to the liveness property of bAVSS-PO, each cer_i of Breeze_i is eventually received by all correct nodes and then be proposed in Rondo-BFT. Then, due to the liveness and agreement properties of our Rondo-BFT, every correct node will have the same common subset CS_e. Additionally, at least one cer_k of a correct instance Breeze_k is contained in CS_e. Finally, due to the liveness and commitment of bAVSS-PO, the batch of secrets shared by Breeze_k can be reconstructed by all the correct nodes. Thus, in each round of epoch e, there is a beacon output. ■

Theorem 5. *Rondo achieves public verifiability.*

Proof: As our Rondo-BFT protocol is signature-based, a set of $2t+1$ matching signatures in the commit phase in the protocol can be used to prove the identities of CS_e in each epoch. ■

Appendix C
Proof of Breeze

Lemma 1. *(Schwartz-Zippel) Let $f(x) \in \mathbb{F}[x]$ be a nonzero polynomial of total degree at most p and $\mathbb{F}$ is a finite set. Then if a is randomly chosen from $\mathbb{F}$, then we have:*

$$\Pr[f(a) = 0] \leq \frac{p}{|\mathbb{F}|}$$

Proof: The proof of this lemma directly comes from the fundamental theorem of algebra. ■

Lemma 2. *Let $\mathcal{P}^{s,t}$ denotes the set of all degree-t univariate polynomials over $\mathbb{F}$ whose constant term is s, where $|\mathcal{P}^{s,t}| = |\mathbb{F}|$. For any set of distinct non-zero elements $\alpha_1, ..., \alpha_n \in \mathbb{F}$, any pair of values s, s', any subset $\mathcal{C} \subset \{P_1, \ldots, P_n\}$ where $|\mathcal{C}| = \ell \leq t$ and every $\vec{y} \in \mathbb{F}^{\ell}$, it holds that:*

$$\begin{aligned} & \Pr_{f(x) \in_R \mathcal{P}^{s,t}} \left[\vec{y} = \left(\{f(\alpha_i)\}_{P_i \in \mathcal{C}}\right)\right] \\ = & \Pr_{g(x) \in_R \mathcal{P}^{s',t}} \left[\vec{y} = \left(\{g(\alpha_i)\}_{P_i \in \mathcal{C}}\right)\right] \\ = & \frac{1}{|\mathbb{F}|^{\ell}} \end{aligned}$$

where $f(x)$ and $g(x)$ are chosen uniformly and independently from the set of polynomials $\mathcal{P}^{s,t}$ and $\mathcal{P}^{s',t}$, respectively.

Proof: Correctness of the lemma follows [74, Claim 3.2]. ■

Informally, Lemma 2 shows that for each ACSS instance, the distribution of the shares seen by the adversary is independent of the underlying secret.

Corollary 1. *For our Breeze protocol, the distributions of the shares of different Breeze instances seen by the adversary are independent to each other.*

Lemma 3. *Assuming the DLog assumption and a collision-resistant hash function, our polynomial commitment is binding and hiding for polynomials.*

Proof: Firstly, when taking a batch of polynomials $\{S_i, i \in [1, B]\}$ with coefficients $\{\mathbf{s}_i = (s_{i,0}, ..., s_{i,deg(S)}), i \in [1, B]\}$ as the input, the output of $\mathrm{BatchCommit}$ is a vector, in which every entry is a normal Bulletproofs of S_i with the form $\prod_{j=0}^{\deg(S)} g_j^{\mathbf{s}_{i,j}}$. Thus $\mathrm{BatchCommit}$ holds the same properties as Bulletproofs. Secondly, in the $\mathrm{BatchEval}$ algorithm, it evaluates the underlying polynomials. For each S, we use $\mathbf{d}$ as a blind mask to cover S. Due to the randomness of $\mathbf{d}$, it achieves a hiding property. Moreover, we generate a group of random numbers γ to sum the polynomials up. Then according to Lemma 1, if the verifier validates that $\boldsymbol{\sigma} \cdot \mathbf{y} = \sum_{i=1}^{B} \gamma_i \mathbf{v}[i] + V_D$ by validating the proof, then with overwhelming probability, it holds that $s_i \cdot y = \mathbf{v}[i], i \in [1, B]$. Hence, our polynomial commitment scheme achieves the hiding property. ■

Theorem 6. *Assuming the DLog assumption, our Breeze protocol is a secure bAVSS-PO protocol.*

Proof: We show that Breeze achieves liveness, correctness, commitment, and privacy.

Liveness-1. If the dealer P_d is correct, all nodes will receive the same root commitment C and polynomial commitments from P_d. Additionally, each node receives B points for B recovery polynomial. It is then not difficult to see that every correct node will eventually send a reply message to the dealer. Consequently, the dealer P_d will receive at least $2t + 1$ valid reply message and be able to send cer_d to all nodes. Therefore, in the confirm phase, each honest node will receive a cer_d for C and complete the sharing.

Liveness-2. If at least $t + 1$ correct nodes validate their partial outputs and start reconstruction for ID.d, the correct nodes must have received a valid cer_d for C. The cer_d of C contains at least $2t + 1$ valid signatures, so at least $t + 1$ correct nodes have received valid shares in the sharing stage. Thus, every correct node will receive at least $t + 1$ valid secret shares, C and idx from other correct nodes, and thereby reconstruct the corresponding $z_i = R[idx](0)$.

Correctness-1. Assume that a correct dealer shared a batch of secrets $\{s_1, ..., s_B\}$. The share polynomial evaluations at all $\{R_k(i)\}_{k \in [1,B]}$ lie on a batch of degree t polynomials. If at least $t + 1$ correct nodes complete the sharing, correct nodes will have a common root commitment C. Due to the binding property of $\mathrm{BatchCommit}$, it must hold $z_i = s_i$, for $i \in [1, B]$.

Correctness-2. If a correct node P_i reconstructs z_i, it must have received at least $t + 1$ valid shares with the same vector commitment C. Any set of $t + 1$ points uniquely determines a polynomial of degree t. As nodes have agreed on the commitment C by validating the partial outputs, the commitments for different subsets of points must be the same due to the binding property of vector commitment. As $\mathrm{BatchCommit}$ is also binding, the reconstruction is therefore unique.

Commitment-1. Our cer_d is used as the validation data. If cer_d is formed, at least $t+1$ correct nodes have received the partial outputs in the sharing stage. Thus, after receiving the validation data, these $t+1$ correct nodes can validate their partial outputs by invoking $\mathrm{ParVerify}$.

Commitment-2. First, it is clear that in the reconstruction stage, correct nodes will output if and only if they receive at least $t + 1$ valid shares. Therefore, at least $t + 1$ correct nodes have validated their partial outputs with the same C. Thereby, due to the binding property of $\mathrm{BatchCommit}$, at least $t+1$ correct nodes jointly hold $t + 1$ valid shares of the same sharing polynomial R_{idx}. Accordingly, the sharing polynomials are determined and every correct node output $R_{idx}[0]$.

Privacy. From Lemma 3, the commitments that the adversary $\mathcal{A}$ received in the sharing phase, leak no information about the shares. Subsequently, the information that the adversary $\mathcal{A}$ could learn is only the shares of the corrupted nodes, which are independent with the underlying secrets due to Lemma 2 and Corollary 1. Hence, the adversary $\mathcal{A}$ has no ascendance in distinguishing the underlying secrets with a batch of random numbers when the dealer is honest. ■

Theorem 7. *Breeze achieves $O(n)$ message complexity and $O(\lambda n)$ amortized communication complexity.*

Proof: The protocol achieves a message complexity of $O(n)$, as nodes in the system only need to communicate with the dealer. In the share phase, the dealer sends n messages, each of size $O(\lambda(B + \log n))$ so the communication is $O(\lambda n(B + \log n))$. In the reply phase, each node sends a signature to the dealer so the communication is $O(\lambda n)$. In the confirm phase, the dealer sends a certificate to the nodes. This costs $O(\lambda n^2)$ communication. If we instantiate the certificate, the length of the certificate is $O(\lambda n \log n)$. Breeze then achieves $O(\lambda n(B + \log n))$. We can set B as $O(\log n)$ to achieve an optimal $O(\lambda n)$ amortized communication, as the amortized communication of Breeze is $O(\lambda(n + \frac{1}{B}\log n + \frac{1}{B}n^2))$.

Alternatively, if we instantiate the certificate using digital signatures, the amortized communication is $O(\lambda(n + \frac{1}{B}\log n + \frac{1}{B}n\log n))$ and we need to set $B = O(n)$ to get $O(\lambda n)$ amortized communication.

For the computational complexity, we calculate three kinds of computation: scalar multiplication, hash operation, and exponentiation. In $\mathrm{BatchEval}$, the most significant part is in line 03, which costs $O(nBp)$ scalar multiplication to obtain the evaluation V. For the hash operation, it is to compute the Fiat-Shamir heuristic and the Merkle Tree, which consumes $O(B + n)$ and $O(n \log p)$ respectively. Hence we have $O(B + n \log p)$ hash operation in total. Furthermore, the exponentiation required are $\mathrm{BatchCommit}$, computation of L and R in line 20 and to compute A' in line 30. It is clear that $\mathrm{BatchCommit}$ for B polynomials with degree t costs $O(Bt)$. Besides, the sizes of L, R and A' are $O(p)$, and for each entry we need $O(1)$ exponentiation. We have $O(\log p)$ iteration, thus in total, it takes $O(p \log p)$ exponentiation. Therefore in total it costs $O(Bt + p \log p)$ exponentiation. ■