DyCAPS: Asynchronous Dynamic-committee Proactive Secret Sharing

Bin Hu, Zongyang Zhang, *Member, IEEE*, Han Chen, You Zhou, Huazu Jiang, Jianwei Liu, *Senior Member, IEEE*

***Abstract*—Dynamic-committee proactive secret sharing (DPSS) enables the refresh of secret shares and the alternation of shareholders without changing the secret. Such a proactivization functionality makes DPSS a promising technology for long-term key management and committee governance. In non-asynchronous networks, CHURP (CCS '19) and COBRA (S&P '22) have achieved best-case square and cubic communication cost, respectively, *w.r.t.* the number of shareholders. However, the overhead of asynchronous DPSS remains high. This gap hinders asynchronous protocols from evolving to the dynamic setting, such as BFT systems and threshold cryptography services.**

In this paper, we fill this gap and propose DyCAPS, an efficient asynchronous DPSS protocol with a cubic communication cost. DyCAPS supports the transfer of both low- and high-threshold secret shares among dynamic committees with the same communication and computation complexity. Experimental results show that proactivization between two disjoint committees of 4 (resp., 64) members takes 1.3 (resp., 51) seconds. Moreover, DyCAPS is designed to be compatible with asynchronous BFT protocols without increasing the asymptotic communication cost. Given a payload of 5–10 MB per node, DyCAPS achieves member change in Dumbo2 (CCS '20) at around 10% temporary throughput degradation, with the committee size varying from 4 to 22.

***Index Terms*—proactive secret sharing, dynamic committee, asynchronous, BFT.**

I. Introduction

Proactive secret sharing (PSS) [1], [2], [3] is an extension of the well-known Shamir's secret sharing [4]. In PSS, a dealer shares a secret among a committee, and the secret shares are refreshed periodically by the committee, without changing or revealing the original secret. Recently, there has been a trend to reconsider the design and applications of dynamic-committee PSS (DPSS) [5], [6]. DPSS further empowers the shareholder committee to reconfigure its composition over time through a so-called *handoff* protocol. This feature makes DPSS a promising technology for long-term key management and committee governance, as membership alternation is inevitable in practice.

Moreover, DPSS gains increased significance, given that many state-of-the-art Byzantine Fault Tolerant (BFT) consensus protocols [7], [8], [9] rely on threshold signatures [10] for reduced communication cost. As pointed out by Duan and Zhang [11], dynamic-committee BFT protocols are in great demand real-world applications. By enabling the transfer of secret key shares among different committees, DPSS introduces a compelling approach to facilitate the transformation of these protocols into dynamic settings.

Researchers have achieved high performance in pure or partially-synchronous networks. In these settings, there is a time bound for message delivery, so that misbehaving nodes can be identified efficiently. The state-of-the-art synchronous DPSS protocol, CHURP [5], consumes $O(\kappa n^2)$ bits of communication in the presence of a semi-honest adversary, where κ denotes the security parameter, and n is the committee size. When faced with Byzantine faults, CHURP's communication cost grows to $O(\kappa n^3)$ bits, which is still the asymptotically best among existing schemes. As for the partially-synchronous solutions, COBRA [6] achieves $O(\kappa n^3)$ and $O(\kappa n^4)$ bits of communication in the best and worst cases, respectively.

However, to the best of our knowledge, there is little related research in asynchrony, which assumes no time bound of network latency. Cachin et al. [12] propose the first asynchronous PSS protocol with $O(\kappa n^4)$ communication complexity, but it only supports a static committee. Zhou et al. [13] are the first to achieve asynchronous DPSS, whereas the communication cost grows exponentially, far from real-world implementation. Until recently, Shanrang [14] brings down the communication complexity of asynchronous DPSS to $O(\kappa n^3 \log n)$ in an all-honest scenario, at the expense of non-optimal fault tolerance $t < n/4$, where t is the reconstruction threshold. When misbehaviors are detected, its cost blows up to $O(\kappa n^4)$.

In this paper, we are aimed to design an efficient and BFT-friendly asynchronous DPSS protocol with optimal fault tolerance $t < n/3$. Such a protocol may strengthen the robustness of long-lived systems, including BFT protocols [15], [8], decentralized autonomous organizations [16], and threshold-cryptography-as-a-service systems [17]. To achieve our goal, there are several challenges to be tackled.

Challenges. The first challenge is to withstand an adversary that corrupts at most $2t$ nodes during a handoff, t in each of the old and new committees. CHURP [5] addresses this problem by introducing a bivariate sharing polynomial. During a handoff, the reconstruction threshold is temporarily raised from t to $2t$ to prevent secret leakage. Then, the new committee collectively generates a common random polynomial to refresh the temporal shares. Finally, the threshold is switched back to t. We follow CHURP's strategy and take a step to support asynchronous networks.

The second challenge is to accommodate CHURP to asynchrony, while retaining the $O(\kappa n^3)$ communication complexity in Byzantine cases. CHURP and other non-asynchronous DPSS protocols rely on challenge-response mechanisms to

Bin Hu, Zongyang Zhang, Han Chen, You Zhou, and Jianwei Liu are with School of Cyber Science and Technology, Beihang University, China. Emails: {hubin0205, zongyangzhang, chenhan1123, youzhou, liujianwei}@buaa.edu.cn.

Huazu Jiang is with Shen Yuan Honors College, Beihang University, China. Email: anjhz@buaa.edu.cn.

Zongyang Zhang is the corresponding author.

make progress. This strategy does not work in asynchronous networks, because an honest node cannot distinguish whether the absence of messages is due to unbounded network latency or malicious behaviors. To adapt to asynchrony, we employ voting to certify honest behaviors, instead of challenging the misbehaving nodes. Additionally, we carefully design the asynchronous randomness generation procedure to maintain the communication complexity at $O(\kappa n^3)$ bits, which is the same as the Byzantine-case performance of CHURP.

The third challenge is to build a BFT-friendly DPSS protocol. Most DPSS protocols are proposed as individual tools, resulting in additional communication and latency costs per handoff. We design our protocol to fully utilizes the components of asynchronous BFT protocols, so that a handoff can be done simultaneously within a round of Byzantine consensus, thereby achieving lower extra overhead.

Contributions. Our contributions are as follows.

- We propose DyCAPS, the first efficient asynchronous DPSS protocol with $O(\kappa n^3)$ communication complexity. In the worst-case scenario, DyCAPS has the same communication complexity as CHURP [5], and it outperforms COBRA [6] — both CHURP and COBRA assume non-asynchrony.
- We derive hDyCAPS, a variant that supports the transfer of high-threshold secret shares among dynamic committees. This variant has the same complexity as DyCAPS.
- We design DyCAPS to be compatible with existing asynchronous BFT protocols. We implement DyCAPS and integrate it into Dumbo2 [8], supporting dynamic membership without increasing the asymptotic communication cost. The implementation is open-source.
- We evaluate DyCAPS on Amazon EC2 t2.medium instances. The handoff between two disjoint committees of equal size takes 1.3 and 51 seconds, respectively, for $n=4$ and $n=64$. Given $4\leq n\leq 22$ and a payload of 5–10MB per node, DyCAPS achieves member change in Dumbo2 at the cost of around 10% temporary throughput degradation.

Concurrent work. A recent work of Yurek et al. [18], referred to as LongLive, also supports high-thrshold DPSS and achieves $O(\kappa n^3)$ communication complexity. The distinguishing features between DyCAPS and LongLive are as follows:

- In design goals, LongLive is aimed at batch amortization, whereas DyCAPS is designed to refresh a single share.
- In technology, LongLive uses the resharing technique [19], [12], whereas DyCAPS utilizes bivariate polynomials [5] to accommodate dynamic committees.
- In performance, DyCAPS exhibits stable performance in handling high- and low-threshold shares, thanks to the bivariate polynomials, whereas LongLive experiences significantly higher latency in the high-threshold case when compared to the low-threshold case.

II. Preliminaries

A. Notations

We use $[n]$ to denote the ordered set $\{1, ..., n\}$, where $n \in \mathbb{N}^*$. Arbitrary-length tuples are denoted as $\langle\cdot\rangle$. Sets are denoted with upper-case calligraphic letters, e.g., $\mathcal{S}$. We refer to the size of $\mathcal{S}$ as $|\mathcal{S}|$. Small capital letters are used to denote the message type, e.g., Com.

TABLE I
Notations

Notation	Description
κ	Security parameter
s	Secret value
e	Epoch number
$\mathcal{C}^e$	Shareholder committee in epoch e
P_i^e	i-th node in committee $\mathcal{C}^e$
n_e	Size of committee $\mathcal{C}^e$
t_e	Maximum number of corrupted nodes in epoch e
$B(x,y), B'(x,y)$	Old and new bivariate sharing polynomial
$C_{\phi(x)}$	Polynomial commitment to $\phi(x)$
$w_{\phi(i)}$	Witness for the evaluation of $\phi(x)$ at $x=i$
σ_m	Digital signature of message m
$\sigma^*_{m,i}$	Signature share produced by P_i on message m
$\mathrm{FLG}_{\mathrm{cntx}}$	Flag, where cntx denotes the context
$\emptyset$	Empty set

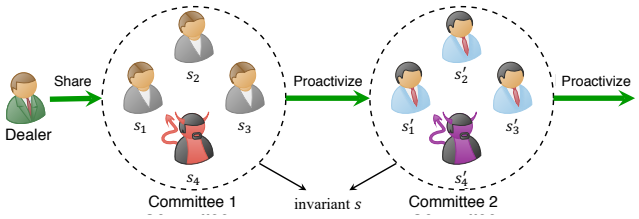

Fig. 1. System model of DPSS. The secret s stays invariant during proactivization, and the nodes are allowed to join or leave the committee over time.

Some special representations are used for particular meanings, as listed in Table I. We use κ as the security parameter, representing the length of signaturs and hash values. The secret value is denoted as s. Epoch number is referred to as e, where $e \in \mathbb{N}^*$. The e-th committee is denoted as $\mathcal{C}^e = \{P_i^e\}_{i\in[n_e]}$, where P_i^e is the i-th node and n_e is the committee size. We use t_e as the maximum number of nodes an adversary can corrupt in epoch e. $C_{\phi(x)}$ denotes the commitment to a polynomial $\phi(x)$, and $w_{\phi(i)}$ is the witness for the evaluation of $\phi(x)$ at $x = i$. The letter σ_m denotes a digital signature of message m, whereas $\sigma^*_{m,i}$ is a signature share produced by P_i. Flags are represented as FLG, with subscripts denoting the context. An empty set is denoted as $\emptyset$.

B. System Model

Our DPSS protocol involves a dealer and several committees, as shown in Figure 1. The dealer is responsible for initializing the secret and distributing shares among the nodes in the first committee. Afterward, the shares are refreshed periodically without the help of the dealer. The proactivization is executed between every two committees, such that the committee members may vary from overlapped to completely disjoint, whereas the secret value s stays invariant over time.

Epochs and secure erasure. We follow Schultz-MPSS [20] and define epochs according to local events. A node is active in epoch e if it holds the secret share for this epoch. Between epochs e and $e+1$, the committees $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ collaboratively execute a handoff protocol to transfer and refresh the shares. To counter with Alexandru et al.'s impossibility result of asynchronous DPSS [21], we assume a handoff is invoked after every honest node in $\mathcal{C}^e$ has obtained a valid share from the

prior handoff or sharing. When an honest node leaves epoch e, it erases the sensitive information related to epoch e.

Network. We assume asynchrony, where an adversary controls the order of messages, but the messages will be delivered eventually. Besides, nodes are fully connected by authenticated and private channels. We further assume these channels are forward-secure, as demonstrated in [20].

Adversary. We assume a probabilistic polynomial-time (PPT) mobile adversary, who adaptively corrupts at most t_e nodes in committee $\mathcal{C}^e$, where $t_e < n_e/3$. The corrupted nodes stay malicious throughout this epoch, and they can misbehave arbitrarily. The adversary can release one node and corrupt another in the next epoch.

Trusted setup. We require a one-time trusted setup to initialize the public paramters for KZG polynomial commitment [22]. This can be done by a trusted third party or a distributed ceremony [23], [24], [25].

C. Building Blocks

Reliable broadcast (RBC) [26], [27] allows a node (dealer) to reliably broadcast a message, ensuring that all honest nodes deliver the same message, or none delivers any message. An RBC protocol satisfies the following properties.

- *Agreement.* If any two honest nodes have outputs, then their outputs are the same.
- *Totality.* If an honest node outputs, then all honest nodes output.
- *Validity.* If the dealer is honest and inputs v, then all honest nodes output v.

Multi-valued validated Byzantine agreement (MVBA) [28], [29], [9] allows a group of nodes to agree on a valid proposal. An MVBA protocol satisfies the following properties.

- *External validity.* If an honest node outputs v, then v satisfies the external predicate P_{MVBA}, i.e., $P_{\mathsf{MVBA}}(v) = 1$.
- *Agreement.* If two honest nodes have outputs, then their outputs are the same.
- *Termination.* If all honest nodes input valid proposals that satisfy P_{MVBA}, then every honest node outputs.

KZG commitment [22] is polynomial commitment scheme whose output is a single group element. We mainly use the following four algorithms.

- $pp \leftarrow \mathsf{KZG.Setup}(t, 1^\kappa)$: this algorithm sets up the public parameters. It takes as inputs a degree t and a security parameter κ in unary form. The output is $O(t)$-sized public parameters pp. We sometimes omit pp for simplicity.
- $C_\phi \leftarrow \mathsf{KZG.Commit}(\phi(x), pp)$: this algorithm commits to a polynomial. It takes as inputs a polynomial $\phi(x) \in \mathbb{Z}_p[x]$ and public parameters pp. The output is a commitment C_ϕ.
- $\langle \phi(i), w_{\phi(i)} \rangle \leftarrow \mathsf{KZG.CreateWitness}(\phi(x), i, pp)$: this algorithm creates a witness for a polynomial evaluation. It takes as inputs a polynomial $\phi(x)$, an index i, and public parameters pp. The output is an evaluation $\phi(i)$ and a witness $w_{\phi(i)}$.
- $0/1 \leftarrow \mathsf{KZG.VerifyEval}(C_\phi, i, v, w_{\phi(i)}, pp)$: this algorithm verifies a polynomial evaluation. It takes as inputs a commitment C_ϕ, an index i, an evaluation v, a witness $w_{\phi(i)}$, and public parameters pp. It outputs 1 iff $v = \phi(i)$.

The KZG scheme satisfies the following properties except with negligible probability.

- *Strong correctness.* An adversary cannot commit to a t'-degree polynomial such that $t' > t$, where t is the input to $\mathsf{KZG.Setup}$.
- *Evaluation binding.* An adversary cannot generate two witnesses, $w_{\phi(i)}$ and $w'_{\phi(i)}$, that both pass $\mathsf{KZG.VerifyEval}$.
- *Hiding.* Given t evaluation-witness tuples $\langle \phi(i), w_{\phi(i)} \rangle$ and the commitment C_ϕ to a t-degree polynomial $\phi(x)$, an adversary cannot determine $\phi(i')$ for any unqueried i'.
- *Homomorphism.* The commitment to $\phi(x) = \phi_1(x) + \phi_2(x)$ can be computed as $C_\phi = C_{\phi_1} C_{\phi_2}$. Similarly, $w_{\phi(i)} = w_{\phi_1(i)} w_{\phi_2(i)}$ holds for $\phi(i) = \phi_1(i) + \phi_2(i)$.

Threshold signature [10] allows a quorum of nodes to jointly compose a full signature. It consists of the following five algorithms.

- $\langle tpk, tvk_i, tsk_i \rangle_{i\in[n]} \leftarrow \mathsf{TS.KeyGen}(t, n, 1^\kappa)$: this algorithm generates threshold key pairs. It takes as inputs a threshold t, a committee size n, and a security parameter κ in unary form. The output is a threshold public key tpk, a set of threshold verification keys $\{tvk_i\}_{i\in[n]}$, and a set of threshold secret keys $\{tsk_i\}_{i\in[n]}$. Each node P_i is assigned with $\langle tpk, \{tvk_i\}_{i\in[n]}, tsk_i \rangle$. We sometimes omit tpk and tvk_i for simplicity.
- $\sigma^*_{m,i} \leftarrow \mathsf{TS.SigShare}_t(m, tsk_i)$: this algorithm generates a signature share. The input is a message m and a threshold secret key tsk_i. The output is a signature share $\sigma^*_{m,i}$.
- $0/1 \leftarrow \mathsf{TS.VerifySh}_t(m, tvk_i, \sigma^*_{m,i})$: this algorithm verifies a signature share. It takes as inputs a message m, a threshold verifier key tvk_i, and a signature share $\sigma^*_{m,i}$. It outputs 1 iff $\sigma^*_{m,i}$ is correctly generated via $\mathsf{TS.SigShare}_t(m, tsk_i)$.
- $\sigma_m \leftarrow \mathsf{TS.Combine}_t(m, \{\sigma^*_{m,i}\}_{i\in I})$: this algorithm generates a full signature from signature shares. It takes as inputs a message m and a share set $\{\sigma^*_{m,i}\}_{i\in I}$, where $I \subset [n]$ and $|I| > t$. The output is a full signature σ_m.
- $0/1 \leftarrow \mathsf{TS.Verify}(m, tpk, \sigma_m)$: this algorithm verifies a full signature. It takes as inputs a message m, a threshold public key tpk, and a signature σ_m. It outputs 1 iff σ_m is valid.

We require the following properties of a threshold signature scheme, which hold with an overwhelming probability.

- *Unforgeability.* Given t corrupted nodes, an adversary cannot forge a valid signature of any unqueried message m.
- *Robustness.* Any $t+1$ valid signature shares of message m yield a valid full signature.

III. Technical Overview

A. Definition and Design Goal

Definition. A DPSS protocol consists of the following three sub-protocols: sharing, handoff, and reconstruction.

- $\langle s_i, \pi_i \rangle_{P_i^1 \in \mathcal{C}^1} \leftarrow \mathsf{DPSS.Share}(t, n, s, 1^\kappa)$: this sub-protocol shares a secret to the initial committee $\mathcal{C}^1$. It takes as inputs a threshold t, a committee size n, a secret value s, and a security parameter κ in unary form. Each node $P_i^1 \in \mathcal{C}^1$ output a share-proof tuple $\langle s_i, \pi_i \rangle$.
- $\langle s'_j, \pi'_j \rangle_{P_j^{e+1} \in \mathcal{C}^{e+1}} \leftarrow \mathsf{DPSS.Handoff}(\langle s_i, \pi_i \rangle_{P_i^e \in \mathcal{C}^e})$: this sub-protocol allows the new committee $\mathcal{C}^{e+1}$ to obtain

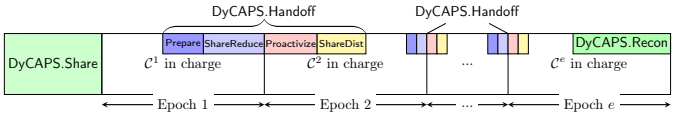

Fig. 2. Life cycle of DyCAPS. DyCAPS.Share is invoked at first, and then DyCAPS.Handoff is executed repeatedly. DyCAPS.Recon is called at the end of the life cycle, if necessary.

refreshed shares from the old committee $\mathcal{C}^e$. Each old node $P_i^e \in \mathcal{C}^e$ inputs a share-proof tuple $\langle s_i, \pi_i \rangle$, and each new node $P_j^{e+1} \in \mathcal{C}^{e+1}$ outputs a refreshed tuple $\langle s'_j, \pi'_j \rangle$.

- $v \leftarrow$ DPSS.Recon$(t, \langle s_i, \pi_i \rangle_{i \in I})$: this sub-protocol reconstructs the secret. It takes as inputs a threshold t and a set of share-proof tuples $\langle s_i, \pi_i \rangle_{i \in I}$, where $I \subset [n]$ and $|I| > t$. The output is a reconstructed secret v.

Design goal. We are aimed to design an efficient asynchronous DPSS.Handoff protocol that satisfies the following properties except with negligible probability.

- *Termination.* If all honest nodes in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ invoke the handoff protocol, then all honest nodes terminate in it.
- *Correctness.* The secret value s remains invariant across the executions of the handoff protocol.
- *Secrecy.* A PPT adversary gains no advantage in extracting the secret value s than random sampling during the handoff.

B. DyCAPS Overview

Aimed at the above design goals, we propose DyCAPS. The life cycle of DyCAPS is depicted in Figure 2. Before epoch 1, DyCAPS.Share is invoked to distribute secret shares among the initial committee $\mathcal{C}^1$, while DyCAPS.Handoff is executed periodically between adjacent epochs for share transfer. In the end, DyCAPS.Recon collects shares from the latest committee and reconstructs the secret (if necessary).

In the rest of this paper, we focus on DyCAPS.Handoff, the core of DyCAPS that is constantly invoked. The specific instantiations of DyCAPS.Share and DyCAPS.Recon are delayed to Appendix VII and Appendix VII-B, respectively. In the following, we refer the nodes in the old (resp., new) committee as old (resp., new) nodes. For ease of expression, we assume $n_e = n_{e+1} = n$, $t_e = t_{e+1} = t$, and $n \geq 3t+1$.

We use bivariate polynomials and adopt the dimension-switching technique [5] to prevent the mobile adversary. The reconstruction threshold is temporarily raised from t to $2t$ during DyCAPS.Handoff, so that an adversary learns no information about the secret even with $2t$ corrupted nodes. Specifically, the secret s is shared via a $\langle t, 2t \rangle$-degree sharing polynomial $B(x, y)$, where $B(0, 0) = s$. In the normal state, $t+1$ *full shares*, $B(*, y)$, are needed to deal with the inquiries, e.g., generating a signature or decrypting a ciphertext. The *reduced shares*, $B(x, *)$, are temporarily used during the handoff, where $2t + 1$ of them are needed for the inquiries.

A typical handoff protocol includes three phases:

1) Raise the threshold to $2t$ and produce reduced shares.

2) Generate a random bivariate zero-polynomial $Q(x, y)$ to refresh the reduced shares, such that $Q(0, 0) = 0$.

3) Revert the threshold to t and restore refreshed full shares.

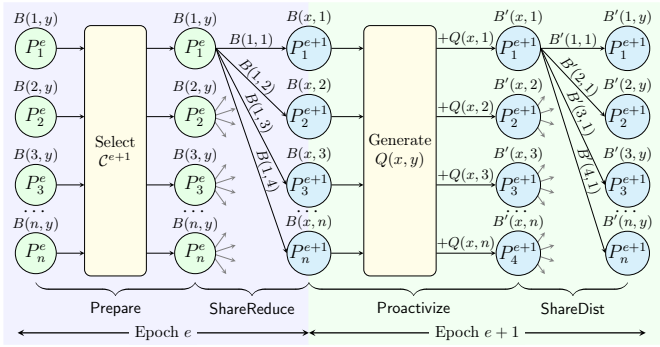

Fig. 3. Overview of the e-th DyCAPS.Handoff between epoch e and $e+1$. The polynomial above each node refers to the share it currently holds.

These phases are referred to as ShareReduce, Proactivize, and ShareDist in CHURP [5]. We introduce an additional Prepare phase in DyCAPS.Handoff, leaving space for selecting new committees and miscellaneous pre-computations. An overview of the four phases are shown in Figure 3.

Prepare. In this phase, a new committee $\mathcal{C}^{e+1}$ is selected, and private channels are established among all nodes in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$. Public parameters are also delivered to $\mathcal{C}^{e+1}$.

ShareReduce. In this phase, full shares are converted to reduced shares. Every new node P_i^{e+1} waits for $t + 1$ valid messages from distinct old nodes to interpolate a t-degree polynomial $B(x, i)$ as its reduced share. The old (local) epoch ends after this phase, when old nodes finish their tasks and new nodes obtain the reduced shares.

Proactivize. In this phase, the new committee jointly generate a $\langle t, 2t \rangle$-degree random bivariate zero-polynomial $Q(x, y)$, such that $Q(0, 0) = 0$. Each node P_i^{e+1} obtains a t-degree polynomial $Q(x, i)$, which is then added to its reduced share $B(x, i)$, making the new share independent of the old one.

ShareDist. In this phase, reduced shares are converted back to full shares. Each node P_i^{e+1} in the new committee waits for $2t + 1$ valid messages from distinct new nodes to interpolate the refreshed full share $B'(i, y)$.

IV. Bivariate Zero-Polynomial Generation

Before elaborating the details of the four phases in our DyCAPS.Handoff, we first propose an efficient bivariate zero-polynomial generation protocol, GenBivariateZeroPoly, which is the key component in the Proactivize phase.

This protocol is invoked by a committee $\mathcal{C} = \{P_i\}_{i \in [n]}$. It takes as inputs a committee size n, a threshold t, and a y-dimension degree t_y, where $t \leq t_y \leq 2t$. Each node P_i outputs a t-degree polynomial $Q(x, i)$ and the commitments to the others' polynomials $\{C_{Q(x,k)}\}_{k \in [n]}$. The bivariate polynomial $Q(x, y)$ is $\langle t, t_y \rangle$-degree, such that $Q(0, 0) = 0$. In the context of Proactivize described in Section III-B, we use $t_y = 2t$.

Properties. The bivariate zero-polynomial generation protocol satisfies the following properties:

- *Termination.* If all honest nodes invoke the protocol, then every honest node P_i outputs $Q(x, i)$ and $\{C_{Q(x,k)}\}_{k \in [n]}$.
- *Correctness.* The output of each honest node is a share of a $\langle t, t_y \rangle$-degree polynomial $Q(x, y)$, where $Q(0, 0) = 0$.
- *Secrecy.* A PPT adversary obtains no extra information about $Q(x, i)$ for any uncorrupted P_i.

A. Strawman Protocols

Satisfying the above properties is not hard within a non-asynchronous network, but it becomes challenging when faced with asynchrony. We illustrate this observation via two strawman schemes. The first strawman assumes a non-asynchronous network, where a timeout exists. The second is in an asynchronous network but fails to satisfy the termination peroperty.

Strawman I. In this strawman protocol, each node P_i first generates a random $\langle t, t_y\rangle$-degree zero-polynomial $Q_i(x, y)$, such that $Q_i(0, 0) = 0$. Then, P_i invokes an RBC instance to broadcast n encrypted polynomials[1], $\{\mathsf{Enc}_j(Q_i(x, j))\}_{j\in[n]}$, and a set of commitments and witnesses certifying the correctness of $Q_i(x, y)$. Each node waits for the outputs of these RBC instances and decrypts the polynomials. If any decrypted polynomial is invalid, an honest node raises a verifiable challenge by revealing its decryption secret key. In this way, either the challenger or the challenged node will be identified as malicious. Depending on the verification results, the new committee decides a candidate set $\mathcal{U}$, such that all malicious nodes are excluded. Finally, each P_i computes its random share $Q(x, i) = \sum_{P_j\in\mathcal{U}} Q_j(x, i)$.

Analysis. Informally, this strawman protocol satisfies the three properties mentioned above:

- *Termination.* The verifiable challenges ensure that all honest nodes receive valid information from all nodes in $\mathcal{U}$, which is used to compute the outputs.
- *Correctness.* The verification of decrypted polynomials ensures that each node in $\mathcal{U}$ proposes a $\langle t, t_y\rangle$-degree zero-polynomial, so the sum of these polynomials is also a $\langle t, t_y\rangle$-degree zero-polynomial.
- *Secrecy.* The t polynomials $Q(x, *)$ controlled by a PPT adversary are insufficient to interpolate $Q(x, i)$ if P_i is not corrupted, since the y-dimension degree of $Q(x, y)$ is t_y, where $t_y \geq t$.

However, the challenge mechanism is not applicable in an asynchronous network—the honest nodes may not raise or receive challenges in time. Therefore, we need other methods to determine the candidate set $\mathcal{U}$.

Strawman II. In this strawman protocol, we relax the network assumption and advance to the asynchronous network. Inspired by asynchronous BFT consensus protocols [15], [8], we use voting to replace the challenges.

Similar to Strawman I, each node P_i to first generates a $\langle t, t_y\rangle$-degree zero-polynomial $Q_i(x, y)$. However, we no longer need to reliably broadcast the encrypted shares. Instead, each P_i sends $Q_i(x, j)$ to every P_j for $j \in [n]$, and broadcasts the commitments $\{C_{Q_i(x,j)}\}_{j\in[n]}$ via RBC. If P_i receives a valid polynomial from P_j, it multicasts a signature share $\sigma^*_{j,i}$ as the vote for P_j, denoting that it has received a valid share from P_j. Upon receiving $n - t$ votes for the same P_j, P_i forms a full signature σ_j and includes $\langle j, \sigma_j\rangle$ in a proposal set V_i. When there are $t + 1$ elements in V_i, P_i inputs it to MVBA, from which all honest nodes output the same set $\widetilde{V}$. The candidate set $\mathcal{U}$ is then denoted as $\{P_j | \langle j, \sigma_j\rangle \in \widetilde{V}\}$.

Analysis. This protocol fails to satisfy the termination property. A malicious node P_m may get included in $\mathcal{U}$, if it obtains $n - t$ votes. In the worst case, only $t+1$ honest nodes receive valid shares from P_m and vote for it, whereas the other honest nodes receive no information from P_m. That is, an honest node P_j may not receive $Q_m(x, j)$ to compute $Q(x, j)$. Moreover, the $t + 1$ shares from the honest nodes who have voted for P_m are insufficient to restore $Q_m(x, j)$, since its y-dimension degree is t_y, where $t_y \geq t$.

[1] $\mathsf{Enc}_j(m)$ encrypts a message m using P_j's encryption public key.

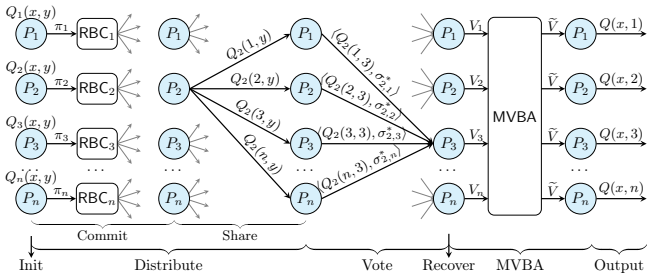

Fig. 4. Workflow of GenBivariateZeroPoly. Each P_i first generates a random polynomial $Q_i(x, y)$ and broadcasts its commitment π_i by RBC_i. Then, P_i sends $Q_i(j, y)$ to each P_j. When P_j receives a valid $Q_i(j, y)$, it sends an evaluation on $Q_i(j, y)$ to the others, along with a vote $\sigma^*_{i,j}$ for P_i. Every P_i composes a proposal V_i containing $t + 1$ full signatures and inputs it to MVBA, and computes $Q(x, i)$ according to the output $\widetilde{V}$ of MVBA.

B. Our Bivariate Zero-Polynomial Generation Protocol

In this formal protocol, we enrich the information in each sharing message, so that $t+1$ honest nodes can help the others restore their missing shares, if some nodes in $\mathcal{U}$ misbehave.

High-level overview. We let each node P_i share its local polynomial by sending $Q_i(j, y)$ to P_j instead of $Q_i(x, j)$. In this way, every P_j obtains partial information on $Q_i(x, j)$ for $i \in [n]$, which ensures all nodes can obtain shares from the selected nodes. This modification brings in an additional round of communication to obtain the desired random share $Q_i(x, j)$. The workflows and procedures of GenBivariateZeroPoly are shown in Figure 4 and Figure 5, respectively.

Init. Upon receiving (n, t, t_y) as input, node P_i initializes several empty sets, including a commitment set π_i, two flag sets $\mathrm{FLG}_{\mathrm{com}}$ and $\mathrm{FLG}_{\mathrm{rec}}$, two buffers $\mathcal{S}_{\mathrm{rec}}$ and $\mathcal{S}_{\sigma^*}$, and an MVBA input set V_i.

Share. Node P_i samples a t_y-degree random polynomial $F_i(y)$, such that $F_i(0) = 0$. For each $\ell \in [t_y + 1]$, P_i reshares $F_i(\ell)$ by generating a t-degree random polynomial $Q_i(x, \ell)$, such that $Q_i(0, \ell) = F_i(\ell)$. Then, P_i sends to each P_j, $j \in [n]$, a SHARE message, which contains $t_y + 1$ evaluation-witness tuples $\langle Q_i(j, \ell), w_{Q_i(j,\ell)}\rangle_{\ell\in[t_y+1]}$.

Commit. For each $\ell \in [t_y + 1]$, node P_i commits to $F_i(\ell)$, $Q_i(x, \ell)$, and $Z_{i,\ell}(x) = Q_i(x, \ell) - F_i(\ell)$, and creates a witness for $Z_{i,\ell}(0) = 0$. P_i gathers the commitments and witnesses in π_i and broadcast it using a COM message via RBC_i.

Verify. Upon outputting π_j from RBC_j, node P_i extracts $\{g^{F_j(\ell)}\}_{\ell\in[t_y+1]}$ from π_j and interpolates $g^{F_j(0)}$. P_i confirms $F_j(0) = 0$ by checking $g^{F_j(0)} = 1$. Then, for each $\ell \in [t_y + 1]$, P_i verifies $Z_{j,\ell}(0) = 0$ using $\langle C_{Z_{j,\ell}}, w_{Z_{j,\ell}(0)}\rangle$. Finally, P_i checks $C_{Q_{j,\ell}} = C_{Z_{j,\ell}} g^{F_j(\ell)}$, which ensures $Q_j(0, \ell) = F_j(\ell)$. If any verification fails, the COM message is discarded. Finally, P_i interpolates $C_{Q_{j,k}}$ for $k \in [n]$ and sets $\mathrm{FLG}_{\mathrm{com}}[j] = 1$.

GenBivariateZeroPoly(n, t, t_y)

1: **upon** invocation by P_i **do**
2: **upon** receiving (n, t, t_y) as input **do** ▷ Init
3: $\pi_i \leftarrow \emptyset$
4: $\text{FLG}_{\text{com}}[1, ..., n] \leftarrow \{0, ..., 0\}$
5: $\text{FLG}_{\text{rec}}[1, ..., n] \leftarrow \{0, ..., 0\}$
6: $\mathcal{S}_{\text{rec}}[1, ..., n] \leftarrow \{\emptyset, ..., \emptyset\}$
7: $\mathcal{S}_{\sigma^*}[1, ..., n] \leftarrow \{\emptyset, ..., \emptyset\}$
8: $V_i[1, ..., n] \leftarrow \{\emptyset, ..., \emptyset\}$
9: sample a t_y-degree polynomial $F_i(y)$, where $F_i(0) = 0$ ▷ Share
10: **for each** $\ell \in [t_y + 1]$ **do**
11: sample a t-degree polynomial $Q_i(x, \ell)$, where $Q_i(0, \ell) = F_i(\ell)$
12: **for each** $j \in [n]$ **do**
13: **for each** $\ell \in [t_y + 1]$ **do**
14: $\langle Q_i(0, \ell), w_{Q_i(j,\ell)} \rangle \leftarrow \mathsf{KZG.CreateWitness}(Q_i(x, \ell), j)$
15: send $\langle \text{SHARE}, \langle Q_i(j, \ell), w_{Q_i(j,\ell)} \rangle_{\ell \in [t_y+1]} \rangle$ to P_j
16: **for each** $\ell \in [t_y + 1]$ **do** ▷ Commit
17: $Z_{i,\ell}(x) \leftarrow Q_i(x, \ell) - F_i(\ell)$
18: $C_{Q_{i,\ell}} \leftarrow \mathsf{KZG.Commit}(Q_i(x, \ell))$
19: $C_{Z_{i,\ell}} \leftarrow \mathsf{KZG.Commit}(Z_{i,\ell}(x))$
20: $\langle Z_{i,\ell}(0), w_{Z_{i,\ell}(0)} \rangle \leftarrow \mathsf{KZG.CreateWitness}(Z_{i,\ell}(x), 0)$
21: $\pi_i \leftarrow \pi_i \cup \langle \ell, g^{F_i(\ell)}, C_{Q_{i,\ell}}, C_{Z_{i,\ell}}, w_{Z_{i,\ell}(0)} \rangle$
22: call RBC_i with input $\langle \text{COM}, \pi_i \rangle$

23: **upon** outputting $\langle \text{COM}, \pi_j \rangle$ from RBC_j **do** ▷ Verify
24: parse π_j as $\langle \ell, g^{F_j(\ell)}, C_{Q_{j,\ell}}, C_{Z_{j,\ell}}, w_{Z_{j,\ell}(0)} \rangle_{\ell \in [t_y+1]}$
25: interpolate $g^{F_j(0)}$ from $\{g^{F_j(\ell)}\}_{\ell \in [t_y+1]}$
26: **if** $g^{F_j(0)} \neq 1$ **then**
27: discard this message
28: **for each** $\ell \in [t_y + 1]$ **do**
29: **if** $\mathsf{KZG.VerifyEval}(C_{Z_{j,\ell}}, 0, 0, w_{Z_{j,\ell}(0)}) = 0 \vee C_{Q_{j,\ell}} \neq C_{Z_{j,\ell}} g^{F_j(\ell)}$ **then**
30: discard this message
31: interpolate $C_{Q_{j,k}}$ from $\{C_{Q_{j,\ell}}\}_{\ell \in [t_y+1]}$ for all $k \in [n]$
32: $\text{FLG}_{\text{com}}[j] \leftarrow 1$

33: **upon** receiving $\langle \text{SHARE}, \langle Q_j(i, \ell), w_{Q_j(i,\ell)} \rangle_{\ell \in [t_y+1]} \rangle$ from P_j **do** ▷ Vote
34: **wait** until $\text{FLG}_{\text{com}}[j] = 1$ **do**
35: extract $\{C_{Q_{j,\ell}}\}_{\ell \in [t_y+1]}$ from π_j
36: **if** $\forall \ell \in [t_y + 1], \mathsf{KZG.VerifyEval}(C_{Q_{j,\ell}}, i, Q_j(i, \ell), w_{Q_j(i,\ell)}) = 1$ **then**
37: $\sigma^*_{j,i} \leftarrow \mathsf{TS.SigShare}_{n-t}(j, tsk_i)$
38: **for each** $k \in [n]$ **do**
39: interpolate $Q_j(i, k)$ from $\{Q_j(i, \ell)\}_{\ell \in [t_y+1]}$
40: interpolate $w_{Q_j(i,k)}$ from $\{w_{Q_j(i,\ell)}\}_{\ell \in [t_y+1]}$
41: send $\langle \text{RECOVER}, j, Q_j(i, k), w_{Q_j(i,k)}, \sigma^*_{j,i} \rangle$ to P_k

42: **upon** receiving $\langle \text{RECOVER}, k, Q_k(j, i), w_{Q_k(j,i)}, \sigma^*_{k,j} \rangle$ from P_j **do** ▷ Recover
43: **wait** until $\text{FLG}_{\text{com}}[k] = 1$ **do**
44: **if** $\mathsf{TS.VerifySh}_{n-t}(k, \sigma^*_{k,j}) = 1$ **then**
45: **if** $\mathsf{KZG.VerifyEval}(C_{Q_{k,i}}, j, Q_k(j, i), w_{Q_k(j,i)}) = 1$ **then**
46: $\mathcal{S}_{\text{rec}}[k] \leftarrow \mathcal{S}_{\text{rec}}[k] \cup (j, Q_k(j, i))$
47: **if** $|\mathcal{S}_{\text{rec}}[k]| \geq t + 1$ **then**
48: interpolate t-degree $Q_k(x, i)$ from $\mathcal{S}_{\text{rec}}[k]$
49: $\text{FLG}_{\text{rec}}[k] \leftarrow 1$
50: $\mathcal{S}_{\sigma^*}[k] \leftarrow \mathcal{S}_{\sigma^*}[k] \cup \sigma^*_{k,j}$
51: **if** $|\mathcal{S}_{\sigma}[k]| \geq n - t$ **then**
52: $\sigma_k \leftarrow \mathsf{TS.Combine}_{n-t}(k, \mathcal{S}_{\sigma^*}[k])$
53: $V_i[k] \leftarrow \langle k, \sigma_k \rangle$
54: **wait** until there are $t + 1$ full signatures in V_i **do** ▷ MVBA
55: call MVBA with input $\langle \text{MVBA.IN}, V_i \rangle$
56: // $P_{\mathsf{MVBA}}(V)$: *there are* $t + 1$ *valid signatures in* V

57: **upon** outputting $\langle \text{MVBA.OUT}, \widetilde{V} \rangle$ from MVBA **do** ▷ Output
58: $\mathcal{U} \leftarrow \{P_j \mid \langle j, \sigma_j \rangle \in \widetilde{V}\}$
59: **wait** until $\text{FLG}_{\text{rec}}[j] = 1$ for all $P_j \in \mathcal{U}$ **do**
60: $Q(x, i) \leftarrow \sum_{P_j \in \mathcal{U}} Q_j(x, i)$
61: **for each** $k \in [n]$ **do**
62: $C_{Q(x,k)} \leftarrow \prod_{P_j \in \mathcal{U}} C_{Q_{j,k}}$
63: **output** $\langle Q(x, i), \{C_{Q(x,k)}\}_{k \in [n]} \rangle$

Fig. 5. Procedures of $\mathsf{GenBivariateZeroPoly}(t, n, t_y)$ as P_i. It takes as inputs a threshold t, a committee size n, and a y-dimension degree t_y. The output of P_i is a t-degree polynomial $Q(x, i)$ and a set of commitments $\{C_{Q(x,k)}\}_{k \in [n]}$. $Q(x, y)$ is a $\langle t, t_y \rangle$-degree zero-polynomial.

Vote. Upon receiving a SHARE message from P_j, node P_i verifies it *w.r.t.* the commitment set π_j delivered from RBC_j. If all verifications pass, it generates $\sigma^*_{j,i} = \mathsf{TS.SigShare}_{n-t}(j, tsk_i)$ as a vote for P_j. Afterward, for each $k \in [n]$, P_i computes an evaluation-witness tuple $\langle Q_j(i, k), w_{Q_j(i,k)} \rangle$ and sends it to P_k within a RECOVER message, along with the vote $\sigma^*_{j,i}$.

Recover. Upon receiving $t+1$ valid RECOVER messages from distinct P_j with the same index k, node P_i recovers a t-degree polynomial $Q_k(x, i)$ by interpolation. P_i also waits for $n - t$ valid votes to compose a full signature σ_k and stores it in the MVBA input set V_i.

MVBA. When there are $t+1$ elements in V_i, node P_i inputs V_i into MVBA. The external predicate $P_{\mathsf{MVBA}}(V)$ requires that there are $t + 1$ valid full signatures in V.

Output. Upon outputting $\widetilde{V}$ from MVBA, node P_i denotes the candidate set $\mathcal{U}$ as $\{P_j \mid \langle j, \sigma_j \rangle \in \widetilde{V}\}$. Then, P_i outputs its random share $Q(x, i) = \sum_{P_j \in \mathcal{U}} Q_j(x, i)$ and the commitments $C_{Q(x,k)} = \prod_{P_j \in \mathcal{U}} C_{Q_{j,k}}$ for every $k \in [n]$.

Remark. We use the share-recover technique in this protocol, which is widely used in asynchronous complete secret sharing (ACSS) protocols [30], [31], [32]. However, unlike the mentioned ACSS protocols that interleave the sharing messages into Bracha-style RBC [26], we separate the RBC and sharing messages to allow more efficient RBC implementations (e.g., the work of Das et al. [27]). As a trade-off, we have to add threshold signatures in the sharing procedures. That is, only an honest node obtaining a full signature, rather than a share, guarantees that all honest nodes will obtain valid shares.

Performance. We employ the RBC and MVBA protocols by Das et al. [27] and Guo et al. [9], respectively. Each node first sends out n $O(\kappa n)$-bit SHARE messages. Then, n RBC instances are invoked, each consuming $O(n|m| + \kappa n^2)$ bits, with an input size $|m| = O(\kappa n)$. Next, each node generates n^2 $O(\kappa)$-sized RECOVER messages. Finally, the MVBA instance consumes $O(n^2|V| + \kappa n^2)$ bits, where the proposal size $|V| = O(\kappa n)$. Overall, the communication complexity of this protocol is $O(\kappa n^3)$. Besides, as the RBC and MVBA protocols both have $O(1)$ expected rounds, the round complexity of our GenBivariateZeroPoly protocol is also $O(1)$.

C. Security Analysis

We sketch the proof that the GenBivariateZeroPoly protocol satisfies the termination, correctness, and secrecy properties by Theorem 1, 2, and 3, respectively. Without loss of generality, we denote the malicious nodes as $\{P_m\}_{m \in [t]}$.

Theorem 1. *If all honest nodes invoke* GenBivariateZeroPoly, *then every honest node* P_i *output* $Q(x, i)$ *and* $\{C_{Q(x,k)}\}_{k \in [n]}$.

Proof. In the following, we first prove that an honest node P_i will proceed to the end of MVBA, and then P_i obtains the random polynomials $Q_j(x, i)$ generated by every $P_j \in \mathcal{U}$.

The worst situation for P_i is that the corrupted nodes do not send any messages to it. In this case, P_i only receives n COM messages from RBC (line 23, Figure 5, same below) and $n - t$ SHARE messages from the honest nodes (line 33). Then, all honest nodes will send RECOVER messages to each other

(line 41). These RECOVER messages are sufficient for P_i to interpolate the t-degree polynomial $Q_h(x,i)$ and compose σ_h for every $h \in [n] \backslash [t]$ (lines 46–51). Therefore, P_i is guaranteed to form a valid proposal V_i as the input to MVBA, even without any private message from the corrupted nodes.

Similarly, every honest node has a valid input for MVBA. Due to the termination of MVBA, each honest node obtains a common output $\widetilde{V}$ and a candidate set $\mathcal{U}$ (line 57). Next, we use two cases to prove that every honest node P_i outputs $Q(x,i)$ and $\{C_{Q(x,k)}\}_{k\in[n]}$ after MVBA.

Case 1: The nodes in $\mathcal{U}$ are all honest. Then, as mentioned above, P_i obtains $Q_h(x,i)$ for all $h \in [n] \backslash [t]$, so it can compute $Q(x,i) = \sum_{P_h \in \mathcal{U}} Q_h(x,i)$. P_i also computes $\{C_{Q(x,k)}\}_{k\in[n]}$ using the commitments that are derived from the COM messages (line 31).

Case 2: Some malicious nodes get included in $\mathcal{U}$. In the worst case, an honest P_i receives no private message from each malicious node $P_m \in \mathcal{U}$. However, $P_m \in \mathcal{U}$ means that there is a valid signature σ_m in $\widetilde{V}$, corresponding to $n-t$ signature shares (line 50). Hence, at least $t+1$ honest nodes have voted for P_m. These nodes have received valid COM and RESHARE messages from RBC_m and P_m, respectively. The totality of RBC ensures that P_i receive the same COM message, which is used to derive the commitments $\{C_{Q(x,k)}\}_{k\in[n]}$ (line 61). The $t+1$ honest nodes receiving valid SHARE messages from P_m will distribute the evaluations on $Q_m(*,y)$ by the RECOVER messages (line 41). Consequently, P_i receives $t+1$ points to interpolate $Q_m(x,i)$, which is then used to compute $Q(x,i)$.

In either case, every P_i outputs a random share $Q(x,i)$ and the commitments $\{C_{Q(x,k)}\}_{k\in[n]}$. □

Theorem 2. *The output of each honest node is a share of a $\langle t, t_y\rangle$-degree polynomial $Q(x,y)$, where $Q(0,0) = 0$.*

Proof. We first prove $Q(x,y)$ is a $\langle t, y\rangle$-degree polynomial. Since $Q(x,y) = \sum_{P_j \in \mathcal{U}} Q_j(x,y)$, and every honest node $P_h \in \mathcal{U}$ generates a $\langle t, t_y\rangle$-degree polynomial, we only need to prove that the polynomial $Q_m(x,y)$ generated by any malicious node $P_m \in \mathcal{U}$ is also $\langle t, t_y\rangle$-degree.

In the *Commit* procedure, P_m broadcasts t_y+1 commitments to $\{Q_m(x,\ell)\}_{\ell\in[t_y+1]}$. Due to the strong correctness of KZG commitments, the degree of each $Q_m(x,\ell)$ is bounded by t. On the other hand, these t_y+1 commitments bound the y-dimension degree of $Q_m(x,y)$ by t_y. If an evaluation $Q_m(i,j)$ in a RECOVER message is validated by a commitment $C_{Q_{m,j}}$, which is interpolated from $\{C_{Q_{m,\ell}}\}_{\ell\in[t_y+1]}$, then the polynomial $Q_m(i,y)$ is of t_y degree. Altogether, the degree of $Q_m(x,y)$ is $\langle t, t_y\rangle$.

On the other hand, the t_y+1 commitments $\{g^{F_m(\ell)}\}_{\ell\in[t_y+1]}$ in each COM message ensure $Q_m(0,0) = F_m(0) = 0$ for each $P_m \in \mathcal{U}$ (line 26), so we also have $Q(0,0) = 0$. □

Theorem 3. *A PPT adversary obtains no extra information about $Q(x,i)$ for any uncorrupted P_i.*

Proof. Firstly, if the adversary wants to compute P_i's share $Q(x,i)$ by $\sum_{P_j \in \mathcal{U}} Q_j(x,i)$, it needs to obtain $Q_j(x,i)$ for all $P_j \in \mathcal{U}$. Since $|\mathcal{U}| = |\widetilde{V}| = t+1$ (line 55), the candidate set $\mathcal{U}$ contains at least one honest node. For any honest node $P_h \in \mathcal{U}$, the adversary cannot recover the t-degree polynomial $Q_h(x,i)$ with t points on it. Hence, the adversary only has access to t polynomials $\{Q(x,m)\}_{m\in[t]}$.

Secondly, since the y-dimension degree of $Q(x,y)$ is t_y (Theorem 2), where $t_y \geq t$, the t shares $\{Q(x,m)\}_{m\in[t]}$ also reveal no information about $Q(x,i)$.

Hence, if a PPT adversary does not corrupt P_i, it obtains no extra information about $Q(x,i)$. □

ShareReduce

1: **upon** invocation by P_i^e **do**
2: **upon** first entering this phase **do** ▷ Reduce
3: **for each** $j \in [n]$ **do**
4: interpolate $w_{B(i,j)}$ from $\{w_{B(i,\ell)}\}_{\ell\in[2t+1]}$
5: $B(i,j) \leftarrow B(i,y)|_{y=j}$
6: send $\langle$REDUCE$, B(i,j), w_{B(i,j)}\rangle$ to P_j^{e+1}
7: erase all sensitive information and go off-line

8: **upon** invocation by P_i^{e+1} **do**
9: **upon** receiving $t+1$ valid REDUCE messages **do** ▷ Interpolate
10: interpolate a t-degree polynomial $B(x,i)$
11: enter the Proactivize phase

Fig. 6. Procedures of ShareReduce as P_i^e or P_i^{e+1}.

V. THE HANDOFF PROTOCOL IN DYCAPS

Now we are ready to elaborate on the four phases of our DyCAPS.Handoff protocol, which is executed between two committees $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ for $e \in \mathbb{N}^*$. We continue to assume $n_e = n_{e+1} = n$, $t_e = t_{e+1} = t$, and $n \geq 3t+1$. The adjustment of n and t is introduced in Section IX.

A. Preparation Phase

In the Prepare phase, a new committee is selected, and public parameters are transferred to each new node. We do not restrict the relationship between the new and old committees, but we do have a limit on the size and threshold of the new committee (see Section IX).

After the committee selection, the nodes in both old and new committees establish private channels with each other. Once a channel is established, each old node transfers the public parameters to new nodes, including the KZG parameters (also called the powers of tau) and the commitments to the reduced shares $\{C_{B(x,j)}\}_{j\in[n]}$. The new nodes confirm these parameters if there are $t+1$ consistent messages. Afterward, the nodes enter the Proactivize phase.

B. Share Reduction Phase

In the ShareReduce phase, each new node P_i^{e+1} obtains a t-degree polynomial $B(x,i)$ as its reduced share. The specific procedures are depicted in Figure 6.

Before detailed descriptions, we remark that every old node P_i^e has obtained $2t+1$ witnesses[2] $\{w_{B(i,\ell)}\}_{\ell\in[2t+1]}$ from either the initial secret sharing DyCAPS.Share ($e=1$) or the prior handoff DyCAPS.Handoff ($e>1$). Hence, P_i^e may interpolate other witnesses from these elements, thanks to the homomorphism of KZG scheme.

Reduce. For all $j \in [n]$, every old node P_i^e sends a REDUCE message to P_j^{e+1}, containing an evaluation $B(i,j)$ and a

[2]The witness $w_{B(i,\ell)}$ is for the evaluation of $B(x,\ell)$ at $x=i$.

Proactivize

1: **upon** invocation by P_i^{e+1} **do**
2: **upon** first entering this phase **do** ▷ Refresh
3: $\mathcal{S}_{\text{com}} \leftarrow \emptyset$
4: $\langle Q(x,i), \{C_{Q(x,k)}\}_{k\in[n]}\rangle \leftarrow \mathsf{GenBivariateZeroPoly}(n,t,2t)$
5: $B'(x,i) \leftarrow B(x,i) + Q(x,i)$
6: $C_{B'(x,i)} \leftarrow \mathsf{KZG.Commit}(B'(x,i))$
7: **for each** $k \in [n]$
8: $C_{B'(x,k)} \leftarrow C_{B(x,k)} C_{Q(x,k)}$
9: $\mathcal{S}_{\text{com}} \leftarrow \mathcal{S}_{\text{com}} \cup C_{B'(x,k)}$
10: enter the ShareDist phase

Fig. 7. Procedures of Proactivize as P_i^{e+1}.

witness $w_{B(i,j)}$. Afterward, P_i^e erases its memory and goes offline, denoting the end of epoch e.

Interpolate. Each new node P_i^{e+1} waits for $t+1$ valid REDUCE messages and interpolates its reduced share $B(x,i)$. This denotes the start of epoch $e+1$. Afterward, P_i^{e+1} enters the Proactivize phase.

C. Proactivization Phase

In the Proactivize phase, the reduced shares are refreshed by a common bivariate zero-polynomial $Q(x,y)$. The procedures are shown in Figure 7.

Refresh. Node P_i^{e+1} invokes the GenBivariateZeroPoly protocol with input $(n,t,2t)$ and waits to output a random share $Q(x,i)$ and a set of commitments $\{C_{Q(x,k)}\}_{k\in[n]}$. The reduced share is refreshed as $B'(x,i) = B(x,i) + Q(x,i)$. Then, for each $k \in [n]$, P_i^{e+1} computes the new commitment $C_{B'(x,k)} = C_{B(x,k)} C_{Q(x,k)}$, where the old commitment $C_{B(x,k)}$ is obtained in the Prepare phase. Afterward, it enters the ShareDist phase.

Remark. The reduced shares $\{B(x,i)\}_{i\in[n]}$ obtained in the ShareReduce phase can be used as threshold secret keys to generate threshold signatures in the *Vote* and *MVBA* procedures in GenBivariateZeroPoly (Figure 5). In this way, only a one-time trusted setup or distributed key generation (DKG) is needed to initialize the signing keys for the first committee. That is, we avoid trusted setup or DKG for each subsequent new committee. We refer to this property as "*no-extra-DKG*."

D. Share Distribution Phase

In ShareDist, reduced shares are converted to full shares. The procedures are shown in Figure 8.

Distribute. For each $j \in [n]$, node P_i^{e+1} generates an evaluation-witness tuple $\langle B'(j,i), w_{B'(j,i)}\rangle$ and sends it to P_j^{e+1} within a DIST message.

Interpolate. Node P_i^{e+1} waits for $2t+1$ valid DIST messages to interpolate the refreshed full share $B'(i,y)$ and $2t+1$ commitment-witness tuples $\langle C_{B'(x,\ell)}, w_{B'(i,\ell)}\rangle_{\ell\in[2t+1]}$. Next, P_i^{e+1} multicasts a SUCCESS message to notify the other nodes.

Success. Upon receiving $2t+1$ SUCCESS messages, node P_i^{e+1} outputs $B'(i,y)$ and $\langle C_{B'(x,\ell)}, w_{B'(i,\ell)}\rangle_{\ell\in[2t+1]}$. Afterward, it enters the normal state.

VI. Performance and Security Analysis

A. Performance Analysis

In Prepare, each old node sends the public parameters to the new committee. The $O(\kappa n)$-sized KZG parameters and

ShareDist

1: **upon** invocation by P_i^{e+1} **do**
2: **upon** first entering this phase **do** ▷ Distribute
3: $\mathcal{S}_{B'} \leftarrow \emptyset$
4: **for each** $j \in [n]$ **do**
5: $\langle B'(j,i), w_{B'(j,i)}\rangle \leftarrow \mathsf{KZG.CreateWitness}(B'(x,i), j)$
6: send $\langle \text{DIST}, B'(j,i), w_{B'(j,i)}\rangle$ to P_j^{e+1}
7: **upon** receiving $\langle \text{DIST}, B'(i,j), w_{B'(i,j)}\rangle$ from P_j^{e+1} **do** ▷ Interpolate
8: retrieve $C_{B'(x,j)}$ from $\mathcal{S}_{\text{com}}$ in Proactivize
9: **if** $\mathsf{KZG.VerifyEval}(C_{B'(x,j)}, i, B'(i,j), w_{B'(i,j)}) = 1$ **then**
10: $\mathcal{S}_{B'} \leftarrow \mathcal{S}_{B'} \cup \langle j, C_{B'(x,j)}, B'(i,j), w_{B'(i,j)}\rangle$
11: **if** $|\mathcal{S}_{B'}| \geq 2t+1$ **then**
12: interpolate $B'(i,y)$ and $\langle C_{B'(x,\ell)}, w_{B'(i,\ell)}\rangle_{\ell\in[2t+1]}$ from $\mathcal{S}_{B'}$
13: multicast SUCCESS
14: **upon** receiving $2t+1$ SUCCESS **do** ▷ Success
15: **output** $\langle B'(i,y), \langle C_{B'(x,\ell)}, w_{B'(i,\ell)}\rangle_{\ell\in[2t+1]}\rangle$

Fig. 8. Procedures of ShareDist as P_i^{e+1}.

the n commitments to the reduced shares lead to a communication complexity of $O(\kappa n^3)$ bits. In ShareReduce, every old node spreads n REDUCE messages, each containing two constant-sized elements. Therefore, the communication cost of the ShareReduce phase is $O(\kappa n^2)$ bits. In Proactivize, the communication only occurs in the the GenBivariateZeroPoly protocol, which consumes $O(\kappa n^3)$ bits of communication. In ShareDist, each node multicasts two constant-sized messages, DIST and SUCCESS, leading to an overall communication cost of $O(\kappa n^2)$ bits. Altogether, DyCAPS.Handoff achieves a communication complexity of $O(\kappa n^3)$ bits.

Besides, since the GenBivariateZeroPoly protocol has $O(1)$ expected rounds, and all other message transferring procedures in the four phases of DyCAPS.Handoff consume constant rounds, the overall round complexity is also $O(1)$.

B. Security Analysis

We give proof sketches of the security of DyCAPS.Handoff, which involves termination, correctness, and secrecy. Without loss of generality, we denote the malicious nodes in committees $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ as $\{P_m^e\}_{m\in[t]}$ and $\{P_m^{e+1}\}_{m\in[t]}$, respectively.

Lemma 4. *If all honest nodes in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ invoke* DyCAPS.Handoff*, then all honest old nodes in $\mathcal{C}^e$ terminate* DyCAPS.Handoff.

Proof. The old committee $\mathcal{C}^e$ is only active in the Prepare and ShareReduce phases. So, we prove that honest nodes terminate in these two phases.

In Prepare, all honest nodes will connect to at least $2(n-t)$ nodes, after which they send public parameters to the new committee and enter the ShareReduce phase.

In ShareReduce, the old committee only needs to send out REDUCE messages and has no expected output. Hence, we prove that every honest node in $\mathcal{C}^e$ has enough information to generate the REDUCE messages (lines 3–4, Figure 6).

As we mentioned in Section II-B, the e-th DyCAPS.Handoff is called after the completion of DyCAPS.Share ($e = 1$) or the prior DyCAPS.Handoff ($e > 1$), from which every honest node P_i^e outputs $2t+1$ commitment-witness pairs $\langle C_{B(x,\ell)}, w_{B(i,\ell)}\rangle_{\ell\in[2t+1]}$. Hence, every honest node P_i^e can interpolate all required witnesses to compose the REDUCE messages, thus terminating the e-th DyCAPS.Handoff. □

Lemma 5. *If all honest nodes in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ invoke* DyCAPS.Handoff*, then all honest new nodes in $\mathcal{C}^{e+1}$ terminate* DyCAPS.Handoff.

Proof. We prove that every honest new node P_i^{e+1} terminates in all phases in the e-th DyCAPS.Handoff.

In Prepare, each P_i^{e+1} is guaranteed to connect to at least $n-t$ nodes in both old and new committees and obtain the public parameters from any $t+1$ honest old nodes. Afterward, it enters the ShareReduce phase.

In ShareReduce, each P_i^{e+1} receives at least $2t+1$ valid REDUCE messages from the honest old nodes (by Lemma 4). These messages are sufficient for P_i^{e+1} to interpolate its reduced share $B(x,i)$ and enters the Proactivize phase.

In Proactivize, each P_i^{e+1} is guaranteed to output a random polynomial $Q(x,i)$ and a commitment set $\{C_{Q(x,k)}\}_{k\in[n]}$ from GenBivariateZeroPoly (by Theorem 1). Hence, P_i^{e+1} can refresh its reduced share and generate $\{C_{B'(x,j)}\}_{j\in[n]}$ from the old commitments $\{C_{B(x,j)}\}_{j\in[n]}$ and $\{C_{Q(x,k)}\}_{k\in[n]}$. After that, P_i^{e+1} enters the ShareDist phase.

In ShareDist, each P_i^{e+1} can generate the DIST messages using the refreshed share $B'(x,i)$ obtained in Proactivize. P_i^{e+1} is guaranteed to receive at least $2t+1$ DIST messages from the honest nodes, which are used to interpolate the full share $B'(i,y)$ and the commitment-witness pairs $\langle C_{B'(x,\ell)}, w_{B'(i,\ell)}\rangle_{\ell\in[2t+1]}$. Then, every honest node multicasts a SUCCESS message. Hence, P_i^{e+1} is ensured to receive at least $2t+1$ SUCCESS messages and terminate ShareDist.

In summary, every honest new node P_i^{e+1} terminates DyCAPS.Handoff. □

Theorem 6 (Termination)**.** *If all honest nodes in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ invoke* DyCAPS.Handoff*, then all honest nodes in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ terminate* DyCAPS.Handoff.

Proof. Combining Lemma 4 and Lemma 5, we conclude that the honest nodes in both old and new committees terminate DyCAPS.Handoff. □

Theorem 7 (Correctness)**.** *The secret s stays invariant during* DyCAPS.Handoff.

Proof. As the Prepare phase does not involve the secret s, we only need to prove that the secret stays invariant within the other three phases.

In ShareReduce, an honest new node P_i^{e+1} accepts a REDUCE message from P_j^e iff the evaluation $B(j,i)$ passes the KZG verification (line 9, Figure 6). The commitment $C_{B(x,i)}$ used to verify $B(j,i)$ is transferred in the Prepare phase, and it is endorsed by $t+1$ nodes. Hence, t corrupted nodes cannot convince an honest node with a different commitment. Due to the binding property of KZG commitment, the interpolated share $B(x,i)$ is bound with the commitment $C_{B(x,i)}$.

In Proactivize, a reduced share is refreshed as $B'(x,i) = B(x,i) + Q(x,i)$, where $i \in [n]$. By Theorem 1 and Theorem 2, each honest node P_i^{e+1} outputs a random share $Q(x,i)$, such that $Q(0,0)=0$ and $Q(x,y)$ is a $\langle t,2t\rangle$-degree polynomial. Hence, the secret $s = B'(0,0) = B(0,0)+Q(0,0)$ stays invariant, and every honest node P_i^{e+1} obtains a new reduced share $B'(x,i)$, where $B'(x,y)$ is $\langle t,2t\rangle$-degree. Every node also computes the commitments to the new reduced shares $\{C_{B'(x,j)}\}_{j\in[n]}$.

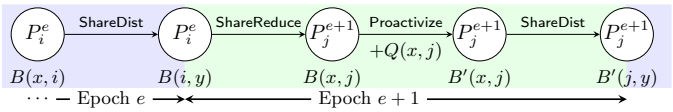

Fig. 9. Shares held by node P_i^e and P_j^{e+1} in adjacent epochs.

In ShareDist, if $2t+1$ points within the DIST messages pass the KZG verification, then by the binding property of KZG commitments, the interpolated $B'(i,y)$ is ensured to be a full share of $B'(x,y)$. Namely, each node obtains a share of $s = B'(0,0)$, which further implies that the secret s is not changed up to the end of ShareDist. □

Theorem 8 (Secrecy)**.** *A PPT adversary gains no advantage in extracting the secret s than random sampling during* DyCAPS.Handoff.

Proof. The shares held by P_i^e and P_j^{e+1} are depicted in Figure 9. At the end of epoch $e+1$, each P_i^e holds $B(x,i)$ and $B(i,y)$, and each P_j^{e+1} has $B(x,j)$, $B'(x,j)$, and $B'(j,y)$.

By Theorem 3, a PPT adversary cannot obtain the common random polynomial $Q(x,y)$. Since the refreshed polynomial is computed as $B'(x,y) = B(x,y)+Q(x,y)$, the polynomials $B'(x,y)$ and $B(x,y)$ are independent in the adversary's view.

Without loss of generality, we focus on polynomial $B(x,y)$. As denotd in Figure 9, the adversary has access to $2t$ reduced shares and t full shares. These shares correspond to $2t^2+3t$ independent evaluations. Since the polynomial $B(x,y)$ has $(t+1)(2t+1)$ unknown coefficients, these evaluations are insufficient to determine the coefficient $s = B(0,0)$. □

VII. SHARING AND RECONSTRUCTION PROTOCOL

We present an asynchronous complete secret sharing (ACSS) protocol DyCAPS.Share and a reconstruction protocol DyCAPS.Recon compatible with our handoff protocol. Our DyCAPS.Share is derived from eAVSS-SC [30]. Due to limited space, the security analysis is omitted, which can be derived from the proofs in [30].

A. Details of Our Sharing Protocol

In DyCAPS.Share, a dealer P_d shares a secret s among a committee $\mathcal{C} = \{P_i\}_{i\in[n]}$. As the dealer may simply withhold the messages to block the sharing, we require that if any honest node outputs a valid share from DPSS.Share, then all honest nodes output valid shares at the end of DyCAPS.Share. This is referred to as the *completeness* property [33].

Before DyCAPS.Share, a trusted setup is required to initialize the powers of tau for KZG commitment scheme [22]. Upon termination of DyCAPS.Share, each node P_i outputs a full share $B(i,y)$ and a set of commitments and witnesses $\{C_{B(x,k)}, w_{B(i,k)}\}_{k\in[n]}$, where $B(x,y)$ is a $\langle t,2t\rangle$-degree bivariate sharing polynomial. The procedures for the dealer P_d and the node $P_i \in \mathcal{C}$ are shown in Figure 10 and Figure 11, respectively.

DyCAPS.Share (as dealer P_d)

1: **upon** receiving s as input **do** ▷ Init
2: $\pi \leftarrow g^s$
3: sample a $2t$-degree polynomial $F(y)$, where $F(0) = s$
4: **for each** $\ell \in [2t+1]$ **do**
5: sample a t-degree polynomial $B(x, \ell)$, where $B(0, \ell) = F(\ell)$
6: $Z_\ell(x) \leftarrow B(x, \ell) - F(\ell)$ ▷ Commit
7: $C_{B(x,\ell)} \leftarrow \mathsf{KZG.Commit}(B(x, \ell))$
8: $C_{Z_\ell} \leftarrow \mathsf{KZG.Commit}(Z_\ell(x))$
9: $w_{Z_\ell(0)} \leftarrow \mathsf{KZG.CreateWitness}(Z_\ell(x), 0)$
10: $\pi \leftarrow \pi \cup \langle \ell, g^{F(\ell)}, C_{B(x,\ell)}, C_{Z_\ell}, w_{Z_\ell(0)} \rangle$
11: **for each** $i \in [n]$ **do** ▷ Send
12: **for each** $\ell \in [2t+1]$ **do**
13: $w_{B(i,\ell)} \leftarrow \mathsf{KZG.CreateWitness}(B(x, \ell), i)$
14: send $\langle \textsc{Send}, \pi, \{B(i, \ell), w_{B(i,\ell)}\}_{\ell \in [2t+1]} \rangle$ to P_i

Fig. 10. Procedures of DyCAPS.Share as dealer P_d.

DyCAPS.Share (as node $P_i \in \mathcal{C}$)

1: **upon** invocation by $P_i \in \mathcal{C}$
2: $\mathcal{S}_{\text{full}} \leftarrow \emptyset$ ▷ Init
3: $\text{FLG}_{\text{ready}} \leftarrow 0$
4: **upon** receiving $\langle \textsc{Send}, \pi', \{B(i, \ell), w_{B(i,\ell)}\}_{\ell \in [2t+1]} \rangle$ from P_d **do** ▷ Echo
5: verify π' as lines 24–28 in Proactivize
6: verify $\{B(i, \ell), w_{B(i,\ell)}\}_{\ell \in [2t+1]}$ *w.r.t.* π'
7: **if** all above verifications return true **then**
8: $\pi \leftarrow \pi'$
9: interpolate a $2t$-degree polynomial $B^*(i, y)$ from $\{B(i, \ell)\}_{\ell \in [2t+1]}$
10: interpolate $\{w_{B^*(i,j)}\}_{j \in [n]}$ from $\{w_{B(i,\ell)}\}_{\ell \in [2t+1]}$
11: multicast $\langle \textsc{Echo}, \pi \rangle$
12: **upon** receiving $n - t$ $\langle \textsc{Echo}, \pi' \rangle$ or $t + 1$ $\langle \textsc{Ready}, \pi', * \rangle$ **do** ▷ Ready
13: **if** $\text{FLG}_{\text{ready}} = 0$ **then**
14: **if** $\pi' = \pi$ **then**
15: send $\langle \textsc{Ready}, \pi', \texttt{share}, B^*(i, j), w_{B^*(i,j)} \rangle$ to each $P_j \in \mathcal{C}$
16: **else**
17: $\pi \leftarrow \pi'$
18: discard $B^*(i, y)$ and $\{w_{B^*(i,\ell)}\}_{\ell \in [n]}$ obtained in lines 9–10
19: multicast $\langle \textsc{Ready}, \pi', \texttt{noShare} \rangle$
20: $\text{FLG}_{\text{ready}} \leftarrow 1$
21: **upon** receiving $n - t$ $\langle \textsc{Ready}, \pi', * \rangle$ **do** ▷ Distribute
22: **wait** until there are $t + 1$ valid Ready messages with `share` tag **do**
23: interpolate a t-degree polynomial $B(x, i)$
24: send $\langle \textsc{Distribute}, B(j, i), w_{B(j,i)} \rangle$ to each $P_j \in \mathcal{C}$
25: **upon** reiceiving $\langle \textsc{Distribute}, B(i, j), w_{B(i,j)} \rangle$ from P_j **do** ▷ Recover
26: **wait** until $\text{FLG}_{\text{ready}} = 1$ **do**
27: interpolate $\{C_{B(x,k)}\}_{k \in [n]}$ from π
28: **if** $\mathsf{KZG.VerifyEval}(C_{B(x,j)}, B(i, j), w_{B(i,j)}) = 1$ **then**
29: $\mathcal{S}_{\text{full}} \leftarrow \mathcal{S}_{\text{full}} \cup \langle j, B(i, j), w_{B(i,j)} \rangle$
30: **if** $|\mathcal{S}_{\text{full}}| \geq 2t + 1$ **then**
31: interpolate a $2t$-degree polynomial $B(i, y)$ and $\{w_{B(i,k)}\}_{k \in [n]}$
32: **output** $\langle B(i, y), \{C_{B(x,k)}, w_{B(i,k)}\}_{k \in [n]} \rangle$

Fig. 11. Procedures of DyCAPS.Share as node $P_i \in \mathcal{C}$.

Dealer's operation. Given an input s, the dealer P_d first initializes a proof set π, which originally contains only g^s. Then, P_d samples a $2t$-degree random polynomial $F(y)$, where $F(0) = s$. This $F(y)$ is extended to $2t + 1$ random polynomials $B(x, \ell)$ of degree t, where $B(0, \ell) = F(\ell)$ for every $\ell \in [2t + 1]$.

To prove the correctness of the sharing polynomial, P_d generates $\langle g^{F(\ell)}, C_{B(x,\ell)}, C_{Z_\ell}, w_{Z_\ell(0)} \rangle$ for $\ell \in [2t + 1]$, where $g^{F(\ell)}$, $C_{B(x,\ell)}$, and C_{Z_ℓ} are the commitments to $F(\ell)$, $B(x, \ell)$, and $Z_\ell(x) = B(x, \ell) - F(\ell)$, respectively, and $w_{Z_\ell(0)}$ is the witness for $Z_\ell(0) = 0$. All these elements are included in π.

Finally, P_d sends $\langle \textsc{Send}, \pi, \{B(i, \ell), w_{B(i,\ell)}\}_{\ell \in [2t+1]} \rangle$ to each $P_i \in \mathcal{C}$. At this point, the dealer P_d has finished all the tasks, and the remaining procedures are conducted by the committee members $\{P_i\}_{i \in [n]}$.

Node's operation. The operation for each node $P_i \in \mathcal{C}$ are divided into several procedures, as described in the following.

Init. Each node P_i initializes an empty buffer $\mathcal{S}_{\text{full}}$ and a flag $\text{FLG}_{\text{ready}} = 0$.

Echo. Upon receiving the Send message from the dealer P_d, node P_i verifies the commitments in π following similar steps as in Proactivize. P_i also verifies the evaluation-witness pairs. If all verifications return true, P_i sets π as π'. Then, P_i interpolates a $2t$-degree polynomial $B^*(i, y)$ and the corresponding witnesses $\{w_{B^*(i,j)}\}_{j \in [n]}$. Finally, P_i multicasts $\langle \textsc{Echo}, \pi \rangle$.

Ready. Upon receiving $n - t$ Echo messages or $t + 1$ Ready messages with the same π', node P_i checks whether $\pi = \pi'$. If so, P_i sends $\langle \textsc{Ready}, \pi', \texttt{share}, B^*(i, \ell), w_{B^*(i,j)} \rangle$ to each $P_j \in \mathcal{C}$. Otherwise, P_i resets π as π', discards the temporary share $B^*(i, y)$ and witnesses $\{w_{B^*(i,\ell)}\}_{P_\ell \in \mathcal{C}}$ obtained in lines 9–10, and multicasts $\langle \textsc{Ready}, \pi', \texttt{noShare} \rangle$.

Distribute. Node P_i collects $n - t$ Ready messages, among which least $t + 1$ contain valid shares. Then, P_i interpolates a t-degree polynomial $B(x, i)$ and sends an evaluation-witness tuple $\langle B(j, i), w_{B(j,i)} \rangle$ to every $P_j \in \mathcal{C}$ within a Distribute message.

Recover. Node P_i first waits for the finalization of π, from wihch it computes the commitment set $\{C_{B(x,k)}\}_{k \in [n]}$. P_i waits for the Distribute messages and verifies them against the commitments. When P_i have collected $2t + 1$ valid Distribute messages, it interpolates a $2t$-degree polynomial $B(i, y)$ as its full share. P_i also interpolates the witness set $\{w_{B(i,k)}\}_{k \in [n]}$ from these messages. The output of P_i is $\langle B(i, y), \{C_{B(x,k)}, w_{B(i,k)}\}_{k \in [n]} \rangle$.

Performance. The procedures above consumes $O(\kappa n^2)$ bits of communication and terminates in $O(1)$ expected rounds.

B. Details of Our Reconstruction Protocol

A dealer-based DyCAPS.Recon protocol only involves one round of communication between the dealer (or a client) and committee members. When a dealer invokes DyCAPS.Recon, every node P_i in the current committee sends an evaluation $B(i, 0)$ to the dealer. The dealer collects at least $2t + 1$ evaluations and runs an online error-correcting (OEC) algorithm to reconstruct the secret $s = B(0, 0)$.

In case there is no dealer, the nodes may milticast their shares, and each of them waits for sufficient evaluations to reconstruct the secret via OEC.

Performance. In either case, the communication complexity of DyCAPS.Recon does not exceed $O(\kappa n^2)$, and the round complexity is $O(1)$.

VIII. Implementation and Evaluation

A. Implementation

We implement[3] DyCAPS using Golang v1.18, partially adopted from CHURP's implementation [34]. We use KZG commitments and BLS threshold signatures [10] implemented by go-KZG [35] and drand/kyber [36]. We also implement Das et al.'s RBC [27] and Guo et al.'s sMVBA [9], which is built upon the Reed-Solomon library by Klaus Post [37].

[3]Open-sourced at https://github.com/DyCAPSTeam/DyCAPS

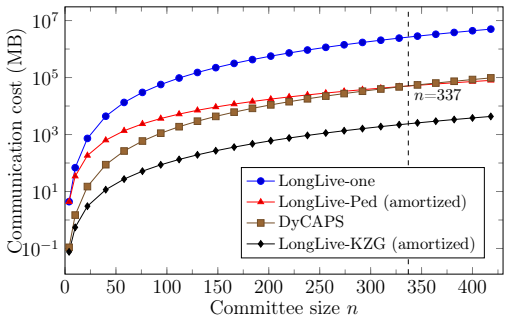

Fig. 12. Per secret communication cost of DyCAPS and LongLive [18] in log scale. The batch size is $n - 2t$ (if applicable).

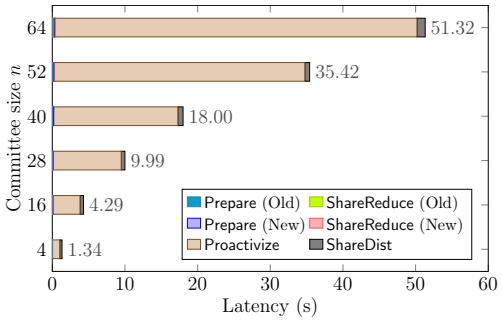

Fig. 13. Latency of DyCAPS for each handoff. The latencies of the old and new committees are accumulated here for simplicity.

We use the pairing-friendly bls12-381 curve [38]. The full and reduced shares are defined over a polynomial ring $\mathbb{F}_p[x]$ with a 256-bit prime p. Besides, we use SHA256 for hashing.

B. Evaluation

We deploy DyCAPS on Amazon EC2 t2.medium instances from 8 regions, each with 2 vCPUs and 4GB memory. Every instance serves as a node. Experiments, each repeated 5 times, are conducted between two honest committees of equal size. When there is no ambiguity, we use DyCAPS and LongLive to denote the handoff process.

Communication cost. We first compare the concrete communication cost of DyCAPS with LongLive [18], a concurrent work that also achieves $O(\kappa n^3)$ communication complexity. LongLive includes three variants: LongLive-one, LongLive-Ped, and LongLive-KZG, where the first two employ Pedersen commitments and the third uses KZG commitments. Besides, the latter two variants support batching. Figure 12 shows the per secret cost of DyCAPS and LongLive for each handoff, where the batch size is $n - 2t$. The results demonstrate that the concrete cost of DyCAPS is around 2% of LongLive-one. This is because LongLive-one uses heavy zero-knowledge proofs of Paillier encryptions (10KB per proof [18]). Given a batch size $B = n - 2t$, the amortized overhead of LongLive-Ped beats DyCAPS when $n \geq 337$. Although the amortized LongLive-KZG outperforms DyCAPS for $O(n)$-sized batch sizes, DyCAPS incurs a smaller communication cost when the batch size $B = 1$ (approximately 77% of LongLive-KZG). Namely, in terms of communication cost, DyCAPS is more efficient when the handoff involves only a single secret.

Latency. The latency of DyCAPS refers to the average time between the initiation of handoff by the old nodes and the moment the new nodes output refreshed full shares. The results are shown in Figure 13. Remarkably, there is an overlap between the latencies of the old and new committees, which are accumulated in this figure for simplicity. A handoff between two smallest committees ($n = 4$) takes 1.34 seconds. When the committee scales to 64 nodes, the latency grows to 51.32 seconds. We observe that the latency is dominated by the Proactivize phase, i.e., the GenBivariateZeroPoly protocol.

To further identify the bottleneck, we measure the step-by-step latency of Proactivize by executing the procedures sequentially. The results show that the growth of latency is mainly caused by the $O(n^2)$ KZG verifications (pairings) in the *Verify*, *Vote*, and *Recover* procedures, accounting for 18%, 18%, and 39% of the latency of Proactivize, respectively, when the committee size $n = 64$.

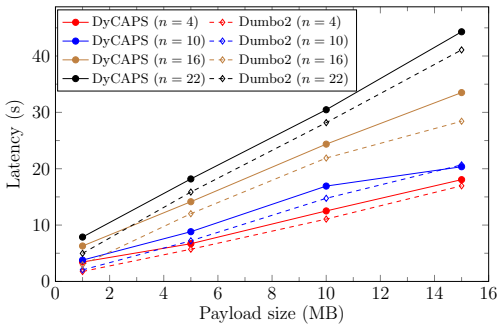

Fig. 14. Latency of DyCAPS and Dumbo2 with different payload sizes.

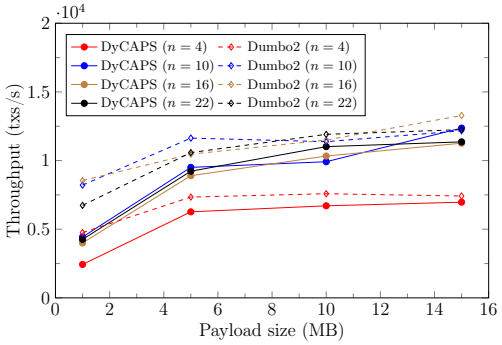

Fig. 15. Throughput of DyCAPS and Dumbo2. The transaction size is 250 bytes. For $n = 10$, the latency of DyCAPS is slightly smaller than Dumbo2, we attribute this to network fluctuation and ignore this biased point.

Throughput. Observe that the GenBivariateZeroPoly protocol in DyCAPS.Handoff includes all gadgets of Dumbo2 [8], including RBC, threshold signatures, and MVBA. Hence, DyCAPS may serve as a dynamic BFT consensus protocol, where the transaction payloads and polynomial commitments are sent within the same RBC instances, so that the membership adjustment is achieved concurrently during the BFT consensus. We evaluate DyCAPS and Dumbo2 with different payload sizes, employing the same building blocks. In both protocols, we set the output size $|\widetilde{V}| = t + 1$. Besides, we use $2t + 1$ as the threshold of the signatures in Dumbo2. The latency and throughput are depicted in Figure 14 and Figure 15, respectively.

As the payload size grows, the performance of DyCAPS become comparable with Dumbo2. Given a payload of 15MB per node, the extra latency of DyCAPS is about 6% ($n = 4$)

and 15% ($n=16$) when compared to Dumbo2. In general, a 5–10MB payload is sufficient for Dumbo2 to enjoy dynamic membership, at the cost of around 10% temporary throughput degradation introduced by DyCAPS. Such a payload size is typical in state-of-the-art BFT protocols (e.g., Dumbo2 and sDumbo [9]) to achieve peak throughput.

IX. Change of Committee Size and Threshold

Changing the committee size and threshold is a common demand for long-lived systems, for both security and flexibility considerations. In the following, we consider the scenarios where the committee size and threshold are updated independently. We denote $n_e=n$, $n_{e+1}=n'$, $t_e=t$, and $t_{e+1}=t'$.

Changing committee size. Given a fixed threshold t, we only need to replace the variable n with n' in the description of DyCAPS.Handoff when referring to the size of new committee. Besides, when the new size n' is too small, i.e., $n' < 3t+1$, a reduction of threshold (see below) is needed before decreasing the committee size.

Increasing threshold. Given a fixed committee size n, to lift the threshold from t to t', where $t' > t$, we need to raise the degree of sharing polynomial from $\langle t, 2t\rangle$ to $\langle t', 2t'\rangle$. An intuitive solution is generating a $\langle t', 2t'\rangle$-degree zero-polynomial $Q(x,y)$ and adding it to $B(x,y)$. However, this solution enables an adversary to obtain $t+t' > 2t$ reduced shares of $B(x,y)$ during the handoff (see Figure 9), leading to the leakage of the secret $s = B(0,0)$. To fix this problem, we let the old committee perform an additional round of handoff towards itself, raising the y-dimension degree of the sharing polynomial to $2t'$.

In this additional round, the old nodes invoke the GenBivariateZeroPoly protocol with input $(n, t, 2t')$, and the other operations remain the same[4]. In this way, each old node P_i^e obtains a t-degree reduced share $B_{\text{tmp}}(x,i)$ and a $2t'$-degree full share $B_{\text{tmp}}(i,y)$ at the end. Afterward, the old committee starts the regular DyCAPS.Handoff, in which the old nodes hand over the reduced shares to the new committee, and the new nodes generate a $\langle t', 2t'\rangle$-degree zero-polynomial $Q(x,y)$ and refresh $B_{\text{tmp}}(x,y)$ to $B'(x,y)$. An adversary with $t+t' < 2t'$ reduced shares of $B_{\text{tmp}}(x,y)$ cannot reconstruct the secret $s = B_{\text{tmp}}(0,0)$.

The additional handoff within the old committee implicitly requires $n > 3t'$. If this is not the case, one might increase the committee size n before increasing the threshold.

Decreasing threshold. To reduce the threshold from t to $t' = t-d$, where $0 < d < t$, we follow prior schemes [20], [5] and introduce d virtual nodes, whose full shares are exposed to all nodes. In this way, the degree of $B'(x,y)$ remains $\langle t, 2t\rangle$, while $t'+1$ extra full shares are needed to perform threshold operations or reconstruct the secret.

The modifications are as follows. In ShareReduce, each old node additionally sends d points on its full share to the new nodes, so that every new node obtains the reduced shares of d virtual nodes. This will not influence the secrecy, because the adversary only has access to $t+t'+d = 2t$ reduced shares. In Proactivize, all honest nodes (including the virtual ones) vote for the virtual nodes, whose contributions are $Q_v(x,y) = 0$. In this way, the MVBA instance terminates even if the corrupted nodes withhold the inputs. Finally in ShareDist, the messages towards the virtual nodes are multicasted so that every node can interpolate the full shares of the virtual nodes.

[4]Generating a $\langle t, 2t'\rangle$-degree polynomial $Q(x,y)$ requires higher-degree KZG public parameters. We adopt the extended KZG scheme by Maram et al. [5] to address this problem.

X. High-threshold DyCAPS

In cryptographic applications, there are demands on high-threshold secret sharing, where $2t$ shares are needed to reconstruct the secret. For example, high-threshold signatures are widely used in MVBA [39], [29], [40], [9] and BFT consensus protocols [15], [8], [7]. In the following, we introduce a high-threshold variant of DyCAPS (referred to as hDyCAPS).

Observe that the reduced shares are already high-threshold. We may treat these reduced shares as regular shares and find a way to transfer them among dynamic committees. In hDyCAPS, we reorder the four phases of DyCAPS.Handoff, letting the ShareDist phase to occur before ShareReduce.

Specifically, each old node Pi^e initially holds a reduced share $B(x,i)$. To transfer the shares to a new committee, the old committee first performs ShareDist to obtain temporary full shares $\{B(i,y)\}_{i\in[n]}$. Then, both committees proceed with the ShareReduce and Proactivize phases to transfer and refresh the reduced shares. At the end of hDyCAPS.Handoff, each new node obtains a refreshed share from the Proactivize phase.

When a threshold $2t-d$ is needed, where $0 < d < t$, we may follow the strategy of threshold decrease in Section IX. Namely, we may introduce d virtual nodes in hDyCAPS, such that the reduced shares of virtual nodes are publicly known.

Remark. The reordering of phases does not influence the communication and computation complexities. Namely, hDyCAPS and DyCAPS have similar performance.

XI. Discussion

A. Related Work

Table II shows the performance of related DPSS protocols.

Non-asynchronous DPSS. Schultz-MPSS [20] realizes DPSS with $O(\kappa n^4)$ bits of communication complexity. Although it claims to support asynchrony, its underlying network model has recently been classified to be partially synchronous [15].

CHURP [5] and COBRA [6] are two state-of-the-art DPSS protocols in non-asynchronous networks. CHURP works in synchrony and has a communication cost of $O(\kappa n^2)$ bits in an all-honest setting. If any node misbehaves, CHURP falls into a pessimistic path that consumes $O(\kappa n^3)$ bits of communication. Goyal et al. [41] propose a DPSS protocol that supports batching, and it has the same asymptotic communication complexity as CHURP in the single-secret setting. COBRA achieves $O(\kappa n^3)$ communication complexity in a partially synchronous network, but its worst-case complexity grows to $O(\kappa n^4)$ due to continuous view-changes.

Asynchronous DPSS. APSS [13] is the first asynchronous DPSS. However, it has only theoretical value due to its exponential communication.

TABLE II
RELATED DPSS PROTOCOLS.

Reference	Async.	Fault tolerance	High-threshold	Best-case[1] comm. cost	Worst-case comm. cost	No trusted setup	No extra[3] DKG
Schultz-MPSS [20]	×	$t < n/3$	×	$O(\kappa n^4)$	$O(\kappa n^5)$	√	√
CHURP [5]	×	$t < n/2$	√	$O(\kappa n^2)$	$O(\kappa n^3)$	×	√
Goyal et a. [41]	×	$t < n/2$	√	$O(\kappa n^2)$	$O(\kappa n^3)$	×	√
COBRA [6]	×	$t < n/3$	×	$O(\kappa n^3)$	$O(\kappa n^4)$	×	√
APSS [13]	√	$t < n/3$	×	$\exp(n)$	$\exp(n)$	√	√
Shanrang [14]	√	$t < n/4$	×	$O(\kappa n^3 \log n)$	$O(\kappa n^4)$	×	√
LongLive-Ped [18]	√	$t < n/3$	√	$O(\kappa n^3)$	$O(\kappa n^3)$	√[2]	×
LongLive-KZG [18]	√	$t < n/3$	×	$O(\kappa n^3)$	$O(\kappa n^3)$	×	×
DyCAPS (this work)	√	$t < n/3$	√	$O(\kappa n^3)$	$O(\kappa n^3)$	×	√

[1] In the best case, all nodes behave honestly. In the worst case, there are t corrupted nodes behaving maliciously.
[2] Without a trusted setup, the zero-knowledge proofs in LongLive-Ped [18] introduce a large constant factor.
[3] For every new committee, LongLive requires an DKG setup to prepare the threshold signing keys for the common coin in MVBA. The other protocols either use bivariate polynomials (Shanrang [14] and DyCAPS) or do not rely on threshold signatures (APSS [13] and all non-asynchronous protocols [20], [5], [41], [6]), so only a one-time execution of DKG for the initial committee is needed.

Research on asynchronous DPSS has been revived recently. Shanrang [14] uses the well-known asynchronous BFT protocol Honeybadger [15] to deal with asynchrony, but it comes with a communication cost of $O(\kappa n^3 \log n)$ bits and tolerates $t < n/4$ corrupted nodes, which is sub-optimal. LongLive [18] follows the reshare-then-agree paradigm proposed in the static PSS protocol by Cachin et al. [12]. LongLive achieves the same $O(\kappa n^3)$ communication complexity as ours, but it focuses on the amortized cost of refreshing multiple secrets. Besides, both LongLive and hDyCAPS support high-threshold DPSS, but LongLive relies on expensive zero-knowledge proofs of encryptions, whereas our hDyCAPS uses the dimension switching technique, thus achieving a much lower concrete communication cost. Additionally, LongLive requires distributed key generation (DKG) for each new committee to participate in the MVBA procedure, whereas our DyCAPS only require a one-time DKG for the initial committee, and the reduced shares can serve as the threshold signing keys for the subsequent committees.

B. Applications of DyCAPS

Flexible committees for BFT protocols. In state-of-the-art BFT protocols [15], [8], [7], the committee is usually fixed, and the change of committee members may require a temporal halt of the protocol. As DyCAPS has a similar workflow to the BFT protocols, it may facilitate the membership adjustment of BFT systems without halting the service. That is, the transactions and inquiries are handled simultaneously during the handoff. Moreover, DyCAPS allows the BFT system to maintain a consistent key pair for signing transactions, regardless of the change of committees. In this way, external verifiers will be relieved of the burden of recording historical public keys. Namely, they can use a single public key to verify the transactions signed by any committee that is in charge at a certain epoch.

Decentralized identity (DID). The blossom of decentralized applications (DApps) on blockchains [42] has drawn public attention to DID [43], [44], [45], which manages personal assets and credentials on the blockchain. With DyCAPS, users may share the secret keys of DIDs among personal devices or cloud services. The shares are refreshed periodically, and users may replace any devices or service providers without compromising the security of their DIDs.

Threshold cryptography as a service. As recently pionted out by Benhamouda et al. [17], threshold cryptographic services are attractive in many fields, including private cloud storage [46], [6], document certification, random beacons [47], [48], and cross-chain bridges [49]. Most scenarios above might encounter the demand of dynamic committee and the challenge of asynchrony in practice. DyCAPS takes a step to takle the dynamic problems and may promote these services.

XII. CONCLUSION

In this paper, we propose DyCAPS, an efficient asynchronous DPSS protocol with a communication complexity of $O(\kappa n^3)$ bits. DyCAPS supports high-threshold DPSS and can be integrated into existing BFT protocols. Due to its robustness in asynchrony, DyCAPS is suitable for long-term key management and committee governance. DyCAPS may facilitate the evolution of committee-based systems into a dynamic setting. DyCAPS has potential applications in systems, decentralized autonomous organizations, and threshold cryptographic services, as well as personal use for managing secret keys and decentralized identities.

ACKNOWLEDGMENTS

The authors thank Ren Zhang, Yanpei Guo, Qitong Liu, and Bingyu Yan for their helpful suggestions.

REFERENCES

[1] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks (extended abstract)," in *PODC 1991*. ACM, 1991, pp. 51–59.
[2] C. U. Günther, S. Das, and L. Kokoris-Kogias, "Practical asynchronous proactive secret sharing and key refresh," *IACR Cryptol. ePrint Arch.*, p. 1586, 2022. [Online]. Available: https://eprint.iacr.org/2022/1586
[3] B. H. Falk, D. Noble, and T. Rabin, "Proactive secret sharing with constant communication," in *TCC 2023*, ser. LNCS. Springer, 2023.
[4] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[5] 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*. ACM, 2019, pp. 2369–2386.

[6] R. Vassantlal, E. Alchieri, B. Ferreira, and A. Bessani, "Cobra: Dynamic proactive secret sharing for confidential bft services," in *SP 2022*. IEEE Computer Society, 2022, pp. 1528–1528.

[7] M. Yin, D. Malkhi, M. K. Reiter, G. Golan-Gueta, and I. Abraham, "Hotstuff: BFT consensus with linearity and responsiveness," in *PODC 2019*. ACM, 2019, pp. 347–356.

[8] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster asynchronous BFT protocols," in *CCS 2020*. ACM, 2020, pp. 803–818.

[9] B. Guo, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Speeding dumbo: Pushing asynchronous BFT closer to practice," in *NDSS 2022*, 2022, pp. 1–18.

[10] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *PKC 2003*, ser. LNCS, vol. 2567. Springer, 2003, pp. 31–46.

[11] S. Duan and H. Zhang, "Foundations of dynamic BFT," in *SP 2022*. IEEE, 2022, pp. 1317–1334.

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

[13] L. Zhou, F. B. Schneider, and R. van Renesse, "APSS: proactive secret sharing in asynchronous systems," *ACM Trans. Inf. Syst. Secur.*, vol. 8, no. 3, pp. 259–286, 2005.

[14] Y. Yan, Y. Xia, and S. Devadas, "Shanrang: Fully asynchronous proactive secret sharing with dynamic committees," 2022. [Online]. Available: https://eprint.iacr.org/2022/164

[15] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *CCS 2016*. ACM, 2016, pp. 31–42.

[16] S. Wang, W. Ding, J. Li, Y. Yuan, L. Ouyang, and F. Wang, "Decentralized autonomous organizations: Concept, model, and applications," *IEEE Trans. Comput. Soc. Syst.*, vol. 6, no. 5, pp. 870–878, 2019.

[17] F. Benhamouda, S. Halevi, H. Krawczyk, A. Miao, and T. Rabin, "Threshold cryptography as a service (in the multiserver and YOSO models)," in *CCS 2022*. ACM, 2022, pp. 323–336.

[18] T. Yurek, Z. Xiang, Y. Xia, and A. Miller, "Long live the honey badger: Robust asynchronous DPSS and its applications," 2023.

[19] Y. Desmedt and S. Jajodia, "Redistributing secret shares to new access structures and its applications," 1997. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.55.2968&rep=rep1&type=pdf

[20] D. A. Schultz, B. Liskov, and M. D. Liskov, "MPSS: mobile proactive secret sharing," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 4, pp. 34:1–34:32, 2010.

[21] A. B. Alexandru, E. Blum, J. Katz, and J. Loss, "State machine replication under changing network conditions," in *ASIACRYPT 2022*, ser. LNCS, vol. 13791. Springer, 2022, pp. 681–710.

[22] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *ASIACRYPT 2010*, ser. LNCS, vol. 6477. Springer, 2010, pp. 177–194.

[23] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza, "Secure sampling of public parameters for succinct zero knowledge proofs," in *SP 2015*. IEEE Computer Society, 2015, pp. 287–304.

[24] V. Nikolaenko, S. Ragsdale, J. Bonneau, and D. Boneh, "Powers-of-tau to the people: Decentralizing setup ceremonies," *IACR Cryptol. ePrint Arch.*, 2022. [Online]. Available: https://eprint.iacr.org/2022/1592

[25] S. Das, Z. Xiang, and L. Ren, "Powers of tau in asynchrony," 2022. [Online]. Available: https://eprint.iacr.org/2022/1683

[26] G. Bracha, "Asynchronous byzantine agreement protocols," *Inf. Comput.*, vol. 75, no. 2, pp. 130–143, 1987.

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

[28] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *CRYPTO 2001*, ser. LNCS, vol. 2139. Springer, 2001, pp. 524–541.

[29] I. Abraham, D. Malkhi, and A. Spiegelman, "Asymptotically optimal validated asynchronous byzantine agreement," in *PODC 2019*. ACM, 2019, pp. 337–346.

[30] M. Backes, A. Datta, and A. Kate, "Asynchronous computational VSS with reduced communication complexity," in *CT-RSA 2013*, ser. LNCS, vol. 7779. Springer, 2013, pp. 259–276.

[31] E. Kokoris-Kogias, D. Malkhi, and A. Spiegelman, "Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures," in *CCS 2020*. ACM, 2020, pp. 1751–1767.

[32] N. Alhaddad, M. Varia, and H. Zhang, "High-threshold AVSS with optimal communication complexity," in *FC 2021*, ser. LNCS, vol. 12675. Springer, 2021, pp. 479–498.

[33] A. Patra, A. Choudhary, and C. P. Rangan, "Efficient statistical asynchronous verifiable secret sharing with optimal resilience," in *ICITS 2009*, ser. LNCS, vol. 5973. Springer, 2009, pp. 74–92.

[34] CHURPTeam, "Churp," 2019. [Online]. Available: https://github.com/CHURPTeam/CHURP

[35] protolambda, "go-KZG library," 2023. [Online]. Available: https://github.com/protolambda/go-kzg

[36] Drand, "Drand/kyber library," 2023. [Online]. Available: https://github.com/drand/kyber

[37] K. Post, "Reed-solomon library," 2023. [Online]. Available: https://github.com/klauspost/reedsolomon

[38] kilic, "bls12-381 library," 2023. [Online]. Available: https://github.com/kilic/bls12-381

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

[40] Y. Lu, Z. Lu, Q. Tang, and G. Wang, "Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited," in *PODC 2020*. ACM, 2020, pp. 129–138.

[41] V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song, "Storing and retrieving secrets on a blockchain," in *PKC 2022*, ser. LNCS, vol. 13177. Springer, 2022, pp. 252–282.

[42] B. Hu, Z. Zhang, J. Liu, Y. Liu, J. Yin, R. Lu, and X. Lin, "A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems," *Patterns*, vol. 2, no. 2, p. 100179, 2021.

[43] D. Maram, H. Malvai, F. Zhang, N. Jean-Louis, A. Frolov, T. Kell, T. Lobban, C. Moy, A. Juels, and A. Miller, "Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability," in *SP 2021*. IEEE, 2021, pp. 1348–1366.

[44] dotbit, "Your decentralized identity for web3.0 life," 2022. [Online]. Available: https://www.did.id/

[45] ConsenSys, "Serto: trust with control," 2022. [Online]. Available: https://www.serto.id/

[46] A. N. Bessani, E. A. P. Alchieri, M. Correia, and J. da Silva Fraga, "Depspace: a byzantine fault-tolerant coordination service," in *EuroSys 2008*. ACM, 2008, pp. 163–176.

[47] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *SP 2017*. IEEE Computer Society, 2017, pp. 444–460.

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

[49] Y. Li, J. Weng, M. Li, W. Wu, J. Weng, J. Liu, and S. Hu, "Zerocross: A sidechain-based privacy-preserving cross-chain solution for monero," *J. Parallel Distributed Comput.*, vol. 169, pp. 301–316, 2022.