DyCAPS: Asynchronous Proactive Secret Sharing for Dynamic Committees

Bin Hu[†], Zongyang Zhang[†], Han Chen[†], You Zhou[†], Huazu Jiang[‡], Jianwei Liu[†]
[†]*School of Cyber Science and Technology, Beihang University*
[‡]*SHENYUAN Honors College, Beihang University*
Email: {hubin0205, zongyangzhang, chenhan1123, youzhou, anjhz, liujianwei}@buaa.edu.cn

***Abstract*—Dynamic-committee proactive secret sharing (DPSS) enables the update of secret shares and the alternation of shareholders, which makes it a promising technology for long-term key management and committee governance. However, there is a huge gap in communication costs between the state-of-the-art asynchronous and non-asynchronous DPSS schemes. In this paper, we fill this gap and propose the first practical DPSS scheme, DyCAPS, with a cubic communication cost *w.r.t.* the number of shareholders.**

DyCAPS can be efficiently integrated into existing asynchronous BFT-based blockchains to support the member change in BFT committees, without increasing the overall asymptotic communication cost. The experimental results show that DyCAPS introduces acceptable latency during the reconfiguration of the committees.

1. Introduction

Proactive secret sharing (PSS) [1], [2] is an extension of the well-known Shamir's secret sharing [3]. In PSS, a user (also called a dealer) shares a secret among a committee, after which the shares are updated periodically in a distributed manner. In recent years, there has been a trend to reconsider the design and applications of dynamic-committee PSS (DPSS) schemes, first proposed by Desmedt and Jajodia [4]. DPSS allows the committee to adjust its member, size, and threshold over time. This dynamic feature makes DPSS a promising technology for long-term key management and committee governance. The design of DPSS schemes has gained additional significance thanks to the development of BFT-based blockchains. As recently pointed out by Duan and Zhang [5], dynamic-committee BFT protocols are in great demand in real-world applications. DPSS schemes may effectively solve the problem of committee authentication, where the member change will not influence the committee's public keys.

However, there is a huge gap in communication costs between DPSS schemes under different network assumptions. Researchers have achieved high performance in pure or partially synchronous networks. In these settings, there exists a time bound for the message delivery, so that the misbehaving parties can be identified efficiently. The state-of-the-art synchronous DPSS scheme, CHURP [6], requires $O(\kappa n^2)$ bits of communication cost under the semi-honest adversary model, where κ is the security parameter, and n is the committee size. When faced with a malicious adversary, CHURP consumes $O(\kappa n^3)$ bits of communication, which is still the asymptotically best among existing schemes. As for the partially synchronous solutions, COBRA [7] achieves $O(\kappa n^3)$ bits of communication cost in the best case, but the cost degenerates to $O(\kappa n^4)$ in the worst case. However, to the best of our knowledge, there is little research on DPSS schemes in asynchronous networks, which only assumes that the messages will be delivered eventually. Zhou et al. [8] achieve asynchronous DPSS at the cost of $\exp(n)$ bits of communication, which only has theoretical meaning and is far from real-world implementation. The lack of efficient asynchronous DPSS schemes may hinder the asynchronous BFT-based systems, especially the blockchain systems [9], [10], from adapting to the dynamic setting.

Migrating the non-asynchronous DPSS schemes to asynchrony is non-trivial, as most of them [6], [7], [11] rely on a challenge-response mechanism to make progress. Such a strategy is inapplicable in asynchronous networks, because an honest party cannot determine whether the absence of responses is due to the unbounded network latency or malicious behaviors.

In this paper, we propose DyCAPS, an efficient asynchronous DPSS scheme with optimized communication cost.

Contributions. Our contributions are as follows.

- We propose DyCAPS, the first practical asynchronous DPSS scheme with $O(\kappa n^3)$ communication cost, closing the communication cost gap between asynchronous and non-asynchronous schemes. In the worst case, our asymptotic cost beats that of COBRA [7], which works in a partially synchronous network.
- We give a formal definition of asynchronous DPSS and prove that our DyCAPS satisfies the required properties.
- We implement DyCAPS and achieve dynamic-committee asynchronous BFT protocol without increasing the asymptotic communication cost. The implementation is open-source at https://github.com/DyCAPSTeam/DyCAPS.
- We present an experimental evaluation of DyCAPS on 64 Amazon EC2 t2.medium instances distributed from 8 regions. The results show that given a large payload (several MB), the latency of DyCAPS is comparable with the static-committee BFT protocol Dumbo2 [10].

Organizations. In the rest of this paper, we give the preliminaries in Section 2. The formal description of DyCAPS is shown in Section 3, and the security and performance analysis is conducted in Section 4. We show the implementation results in Section 5 and describe the adjustment of committee size and threshold in Section 6. The discussion and conclusion are in Section 7 and Section 8, respectively.

2. Preliminaries

Before demonstrating our main protocol, we introduce the notations, system model, and building blocks in Section 2.1, Section 2.2, and Section 2.3, respectively.

2.1. Notations

We use $[n]$ to denote the set $\{1, ..., n\}$, where $n \in \mathbb{N}^*$. Arbitrary-length tuples are denoted as $\langle\cdot\rangle$. Sets are mostly denoted with upper-case calligraphic letters, e.g., $\mathcal{S}$. We refer to the size of $\mathcal{S}$ as $|\mathcal{S}|$. Besides, we use small capital letters to denote the message type, e.g., COM. As for the operations, we use left arrows to assign values to variables.

Some special representations are used for particular meanings, as listed in Table 2, Appendix A. We use κ as the security parameter. The secret value is denoted as s. We denote the epoch number as e, where $e \in \mathbb{N}^*$. The committee in the e-th epoch is denoted as $\mathcal{C}^e = \{P_i^e\}_{i\in[n_e]}$, where P_i^e is the i-th member and n_e is the committee size. We use t_e as the maximum number of parties the adversary can corrupt in epoch e. The letter σ denotes digital signatures. Flags are referred to as FLG, with a subscript denoting its context, e.g., FLG_{com} is the commitment flag. We use C_ϕ to denote the commitment to the polynomial $\phi(x)$, and $w_{\phi(i)}$ is the witness of the evaluation of $\phi(x)$ at $x = i$.

2.2. System Model

The core of a DPSS scheme is the handoff protocol, where two committees jointly refresh the shares for the next epoch. After the handoff, we require that honest parties erase their secret information of the current epoch, which is a common assumption for PSS schemes [6], [7], [11]. Otherwise, it would be trivial for the adversary to recover the secret through the information about previous epochs.

Network model. We assume an asynchronous and authenticated peer-to-peer (P2P) network. Asynchrony implies that the adversary controls the order of the messages, but the messages will be delivered eventually. The authenticated channel means that each peer knows the source of a message, and the adversary cannot see the contents of the messages or speak on behalf of an uncorrupted party. Finally, the P2P network indicates that the connections are private, and there is no broadcast channel. We further assume that the P2P channels are forward-secure, as demonstrated in [11], to prevent the adversary from decrypting old messages upon new corruptions.

Adversary model. We assume a mobile adversary who adaptively corrupts at most t_e parties in committee $\mathcal{C}^e$, such that $t_e < n_e/3$. The corrupted parties stay malicious throughout this epoch, and they can misbehave arbitrarily, including sending invalid messages and withholding responses. Moreover, the adversary is assumed to be computationally bounded.

Trusted setup. We require a trusted setup to initialize the polynomial commitment scheme [12] (see Section 2.3), which is one of the key ingredients in DyCAPS to achieve cubic communication cost.

2.3. Building Blocks

We use several building blocks in DyCAPS, as described in the following.

Reliable broadcast (RBC) [13], [14], [15] ensures that all honest parties consistently receive a message from an honest sender. Formally, an RBC protocol has the following properties:

- *Agreement.* If any two honest parties have outputs, their outputs are the same.
- *Totality.* If an honest party outputs, all honest parties will output.
- *Validity.* If the sender is honest, all honest parties will output the sender's input.

Multi-valued validated asynchronous Byzantine agreement (MVBA) [16], [17], [18] allows each participant to input a proposal and agree on a validated input *w.r.t.* an external predicate P_{MVBA}. Formally, an MVBA protocol satisfies the following properties:

- *Agreement.* If any two honest parties have outputs, their outputs are the same.
- *Totality.* If $n-t$ honest parties each have a validated input, all honest parties will output.
- *External validity.* The output of an honest party is validated *w.r.t.* the external predicate P_{MVBA}.
- *Quality.* If an honest party outputs v, the probability that v is proposed by the adversary is at most $1/2$.

KZG commitment [12] is an efficient polynomial commitment scheme whose output is a single group element. We mainly use four algorithms: KZG.Setup, KZG.Commit, KZG.CreateWitness, and KZG.VerifyEval. For simplicity of expression, we will omit the commitment public key cpk in these algorithms in our protocol.

- $\{\langle p, \mathbb{G}, \mathbb{G}_T, e, g\rangle, cpk\} \leftarrow \mathsf{KZG.Setup}(t, 1^\kappa)$: this algorithm sets up the public parameters for the commitments. It takes as inputs the degree bound t and the security parameter κ in unary form. The output is a bilinear group $\langle p, \mathbb{G}, \mathbb{G}_T, e, g\rangle$ and a commitment key pair $\langle cpk, csk\rangle$. The commitment secret key csk is then securely erased, and all subsequent computations are based on the commitment public key cpk.
- $C_\phi \leftarrow \mathsf{KZG.Commit}(\phi(x), cpk)$: this algorithm commits to a polynomial. It takes as inputs a polynomial $\phi(x) \in \mathbb{Z}_p[x]$ and the commitment public key cpk. The output is a commitment C_ϕ to $\phi(x)$.
- $\langle \phi(i), w_{\phi(i)}\rangle \leftarrow \mathsf{KZG.CreateWitness}(\phi(x), i, cpk)$: this algorithm creates a witness to an evaluation of the polynomial. It takes as inputs a polynomial $\phi(x)$, an index

i, and the commitment public key cpk. The output is an evaluation $\phi(i)$ and a witness $w_{\phi(i)}$.

- $0/1 \leftarrow \mathsf{KZG.VerifyEval}(C_\phi, i, v, w_{\phi(i)}, cpk)$: this algorithm verifies an evaluation on the polynomial. It takes as inputs a commitment C_ϕ, an index i, an evaluation v, a witness $w_{\phi(i)}$, and the commitment public key cpk. It outputs 1 iff $v = \phi(i)$.

The KZG scheme satisfies the following properties:

- *Correctness.* Given the same public parameters, the output of $\mathsf{KZG.CreateWitness}$ always passes $\mathsf{KZG.VerifyEval}$.
- *Strong correctness.* The adversary cannot commit to a t'-degree polynomial such that $t' > t$, where t is the input to $\mathsf{KZG.Setup}$.
- *Evaluation binding.* Given the same cpk, the adversary cannot generate two different witnesses, $w_{\phi(i)}$ and $w'_{\phi(i)}$, such that both of them pass $\mathsf{KZG.VerifyEval}$.
- *Hiding.* Given t evaluation-witness pairs $\langle i, \phi(i), w_{\phi(i)} \rangle$ and the commitment C_ϕ to a t-degree $\phi(x)$, the adversary cannot determine $\phi(i')$ with non-negligible advantage for any unqueried i'.
- *Homomorphism.* Given the same public parameters, the commitment to $\phi(x) = \phi_1(x) + \phi_2(x)$ can be calculated as $C_\phi = C_{\phi_1} C_{\phi_2}$. Similarly, we have $w_{\phi(i)} = w_{\phi_1(i)} w_{\phi_2(i)}$.

Threshold signature [19], [20] allows a quorum of parties to construct a full signature jointly. A threshold signature scheme consists of five algorithms: $\mathsf{TS.KeyGen}$, $\mathsf{TS.SigShare}$, $\mathsf{TS.VerifySh}$, $\mathsf{TS.Combine}$, and $\mathsf{TS.Verify}$. For simplicity of expression, we will omit the threshold public key tpk and verification keys vpk_i in these algorithms in our protocol.

- $\{\langle tpk, tvk_i, tsk_i \rangle_{i\in[n]}\} \leftarrow \mathsf{TS.KeyGen}(t, n, 1^\kappa)$: this algorithm generates the threshold key pairs. It takes as inputs the threshold t, the total number of parties n, and the security parameter κ in unary form. The output is a threshold public key tpk, a set of threshold verifier keys $\{tvk_i\}_{i\in[n]}$, and a set of threshold secret keys $\{tsk_i\}_{i\in[n]}$. Each P_i is assigned with $\langle tpk, tvk_i, tsk_i \rangle$.
- $\sigma^*_{i,m} \leftarrow \mathsf{TS.SigShare}(m, tsk_i)$: this algorithm generates a signature share. The input is a threshold secret key tsk_i and a message m, and the output is a signature share $\sigma^*_{i,m}$.
- $1/0 \leftarrow \mathsf{TS.VerifySh}(m, tvk_i, \sigma^*_{i,m})$: this algorithm verifies P_i's signature share. It takes as inputs a message m, P_i's threshold verifier key tvk_i, and a signature share $\sigma^*_{i,m}$. It outputs 1 iff the share is correctly generated by P_i's threshold secret key tsk_i for m.
- $\sigma \leftarrow \mathsf{TS.Combine}(m, \{\sigma^*_{i,m}\}_{i\in I,|I|>t})$: this algorithm generates a full signature with enough signature shares. It takes as inputs a message m and a set of valid signature shares $\{\sigma^*_{i,m}\}_{i\in I,|I|>t}$ of the message m, where I is the index set. The output is a full signature σ.
- $0/1 \leftarrow \mathsf{TS.Verify}(m, tpk, \sigma)$: this algorithm verifies a signature. It takes as the input the message m, the threshold public key tpk, and the full signature σ. It outputs 1 iff the signature is validated by tpk.

A threshold signature scheme satisfies the following properties:

- *Unforgeability.* A computationally bounded adversary cannot forge a valid full signature with only t threshold secret keys.
- *Robustness.* All honest parties will obtain a valid full signature if they have at least $t+1$ valid signature shares.
- *Correctness.* A signature share $\sigma^*_{i,m}$ generated by an honest P_i always passes the share verification $\mathsf{TS.VerifySh}$. Besides, a full signature generated by an honest P_i always passes $\mathsf{TS.Verify}$.

3. The DyCAPS Protocol

We first provide a formal definition of a DPSS scheme and the properties we are aimed to satisfy in Section 3.1. Then we give an overview of DyCAPS in Section 3.2. The technical details of DyCAPS are represented in the rest of this section.

3.1. Definition of DPSS

A typical secret sharing scheme consists of two protocols, sharing and reconstruction, where the secret s is shared and reconstructed, respectively. To achieve periodic updates of secret shares and support dynamic committees, we add a handoff protocol, as shown in Definition 1.

Definition 1 (Dynamic-committee Proactive Secret Sharing, DPSS). *A DPSS scheme consists of three protocols: DPSS.Share, DPSS.Handoff, and DPSS.Recon.*

- $\{\langle s_i, \pi_i \rangle_{P_i\in\mathcal{C}}\} \leftarrow \mathit{DPSS.Share}(t, n, s, 1^\kappa)$*: this protocol shares the secret among the participants. It takes as inputs the threshold t, the total number of parties n, the secret value s, and the security parameter κ in unary form. Each P_i in the original committee $\mathcal{C}$ will receive a tuple $\langle s_i, \pi_i \rangle$, where s_i is the share, and π_i is the proof of correctness.*
- $\{\langle s'_j, \pi'_j \rangle_{P_j\in\mathcal{C}^{e+1}}\} \leftarrow \mathit{DPSS.Handoff}(\{\langle s_i, \pi_i \rangle_{P_i\in\mathcal{C}^e}\})$*: this protocol allows the old committee $\mathcal{C}^e$ to transfer the secret shares to the new committee $\mathcal{C}^{e+1}$, during which the shares are refreshed. The input is the share-proof tuples $\langle s_i, \pi_i \rangle$ held by each old party P^e_i. Each new party P^{e+1}_j will receive a new tuple $\langle s'_j, \pi'_j \rangle$.*
- $v \leftarrow \mathit{DPSS.Recon}(\{\langle s_i, \pi_i \rangle_{i\in I}\})$*: this protocol reconstruct the secret. It takes as inputs at least $t+1$ valid share-proof tuples $\{\langle s_i, \pi_i \rangle_{i\in I,|I|>t}\}$, where I is the index set. The output is the reconstructed secret v.*

Note that the concrete implementations of $\mathsf{DPSS.Share}$ and $\mathsf{DPSS.Recon}$ depend on the application scenarios. For example, if a client uses DPSS to store a long-term secret, it trivially serves as a trusted dealer to distribute and reconstruct the secret. In another case where a committee jointly generates and maintains a secret key, a decentralized version of $\mathsf{DPSS.Share}$ is needed, and $\mathsf{DPSS.Recon}$ may become unnecessary since the secret will never be restored due to privacy concerns.

A DPSS shceme satisfies the following properties:

- *Termination.* If any protocol is invoked by at least $n-t$ honest parties, all honest parties will output.

- *Correctness.* If an honest dealer inputs s to DPSS.Share and v is the output of DPSS.Recon, we have $v = s$, where an arbitrary number of executions of DPSS.Handoff are allowed before the reconstruction.
- *Secrecy.* The adversary gains no extra knowledge about the secret s other than public information.

For a dealer-based DPSS.Share protocol, the termination only applies when the adversary decides to deliver the shares to the committee [21], as the dealer may withhold the messages or send invalid messages to all honest parties. In this case, we use the *liveness* and *totality* properties instead:

- *Liveness.* If the dealer is honest, all honest parties will complete DPSS.Share.
- *Totality.* If an honest party completes DPSS.Share, all honest parties will complete DPSS.Share.

As DPSS.Handoff involves two committees, its termination requires at least $n_e - t_e$ and $n_{e+1} - t_{e+1}$ honest parties in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$, respectively. In the rest of this section, we assume the old and new committees are of the same size for simplicity, i.e., $n_e = n_{e+1} = n$, $t_e = t_{e+1} = t$, and $n = 3t + 1$. The adjustment of the committee size and threshold is delayed to Section 6.

3.2. DyCAPS Overview

The life cycle of DyCAPS consists of one invocation of DyCAPS.Share, unlimited executions of DyCAPS.Handoff, and one call (if any) of DyCAPS.Recon, as depicted in Figure 1.

In the rest of this section, we focus on DyCAPS.Handoff, which is invoked constantly. The details of DyCAPS.Share and DyCAPS.Recon are delayed to Appendix B and Appendix C, respectively.

To prevent the mobile adversary, we use $\langle t, 2t \rangle$-degree bivariant polynomials and adopt the dimension-switching technique [6]. Informally, the degree of sharing polynomial is temporarily raised from t to $2t$ during DyCAPS.Handoff, so that the adversary learns no information about the secret even with $2t$ corrupted parties. Specifically, the secret s is shared via a polynomial $B(x, y)$, where $B(0, 0) = s$. We refer to $B(x, y)$ as the sharing polynomial. In the normal stage, $t+1$ full shares, $B(*, y)$, are needed to deal with the inquiries, e.g., generating a signature or decrypting a message. In turn, the reduced shares, $B(x, *)$, are temporarily used during the handoff, where $2t + 1$ of them are needed for the inquiries.

Intuitively, the handoff protocol includes three phases: 1) raise the threshold to $2t$ and produce reduced shares, 2) refresh the reduced shares using a jointly generated bivariant polynomial, and 3) switch back the threshold to t and obtain refreshed full shares. These three phases are referred to as ShareReduce, Proactivize, and ShareDist by Maram et al. [6]. We adopt their naming but explicitly add a Prepare phase, leaving space for selecting new committees and miscellaneous pre-computations. The four phases of DyCAPS.Handoff are shown in Figure 2.

Prepare. In this phase, a new committee $\mathcal{C}^{e+1}$ is selected, and P2P channels are established among all members in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$. The public parameters are also delivered to $\mathcal{C}^{e+1}$ at the same time.

ShareReduce. In this phase, the full shares are converted to reduced shares to withstand the mobile adversary. In the beginning, each party P_i^e holds a $2t$-degree polynomial as its full share. Then, each P_i^e sends a point on its full share to P_j^{e+1}. Next, P_j^{e+1} waits for $t+1$ valid points to interpolate a t-degree polynomial as its reduced share.

Proactivize. In this phase, the new committee members jointly generate random shares to refresh the reduced shares. Specifically, the parties share their local randomness, i.e., bivariate polynomials, and agree on a candidate set $\mathcal{Q}$. Utilizing the building blocks in Section 2.3, we ensure that every honest party will obtain the deserved random share from the candidates in $\mathcal{Q}$. Finally, the random shares are added to the reduced shares, making the refreshed shares independent of the old ones.

ShareDist. In this phase, the new committee converts the new reduced shares to the full shares. Specifically, parties send points on their new reduced shares to each other. Each party interpolates the refreshed full shares using the received points. At this time, the new committee enters the normal state and uses full shares to handle the inquiries.

The specific steps of these four phases are illustrated in the rest of this section.

3.3. Preparation

In the Prepare phase, the new committee is selected, and the public parameters are transferred to the new committee. The new committee might be the same as or disjoint from the old one. We do not restrict the selection method, but we do have a limit on the new committee size and threshold (see Section 6).

After the committee selection, the parties in both committees establish P2P channels with each other. As the adversary may corrupt up to t peers in each committee, we tolerate at most $2t$ unsuccessful connections. Once a channel is established, the old party transfers the public parameters to the new party, including the commitment public key cpk and commitments of the reduced shares. Each new party confirms these parameters after it has received $t+1$ messages with the same contents.

When an honest party has established at least $n - t$ P2P connections with each committee, it enters the ShareReduce phase. The P2P connection requests are still appropriately handled in the subsequent phases, allowing the slow but honest parties to connect to the others.

Remarkably, the steps above are taken concurrently when the old committee $\mathcal{C}^e$ is in charge. Therefore, we do not count this phase into the overall communication cost.

3.4. Share Reduction

In the ShareReduce phase, each new committee member waits to receive a t-degree polynomial $B(x, *)$ as its reduced share. The specific procedures are depicted in Figure 3.

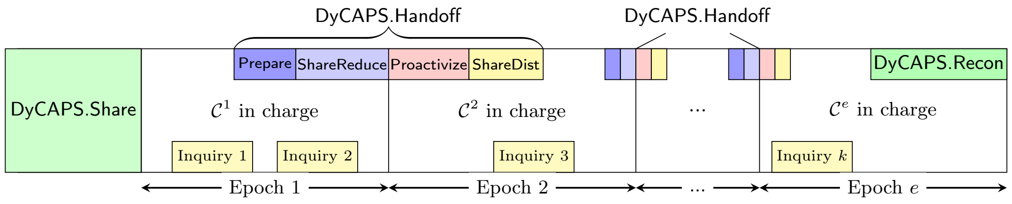


Figure 1. Life cycle of DyCAPS. DyCAPS.Share is invoked at first, and then DyCAPS.Handoff is executed periodically. DyCAPS.Recon may be called at the end of the life cycle (if necessary). The committee in charge handles the inquiries regardless of the handoff.

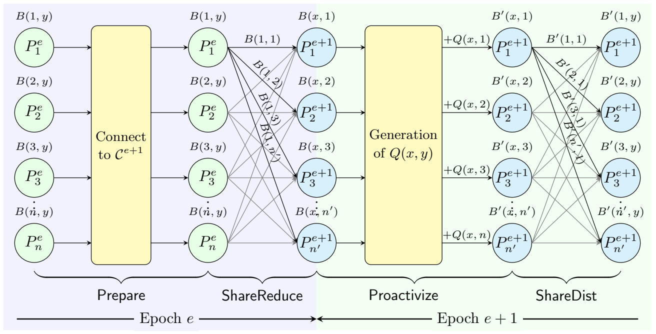


Figure 2. Overview of DyCAPS.Handoff. The polynomial above each party refers to the share it currently holds.

```
ShareReduce

1:  Upon invocation by $P_i^e$ do                                  ▷ Reduce
2:    For each $P_j^{e+1} \in \mathcal{C}^{e+1}$ do
3:      Interpolate $C_{B(x,j)}$ from $\{\langle \ell, C_{B(x,\ell)}\rangle\}_{\ell\in[2t+1]}$
4:      Interpolate $w_{B(i,j)}$ from $\{\langle \ell, w_{B(i,\ell)}\rangle\}_{\ell\in[2t+1]}$
5:      $B(i,j) \leftarrow B(i,y)|_{y=j}$
6:      Send $\langle \text{REDUCE}, C_{B(x,j)}, B(i,j), w_{B(i,j)}\rangle$ to $P_j^{e+1}$
7:    Erase the memory and go off-line

8:  Upon invocation by $P_i^{e+1}$ do                              ▷ Interpolate
9:    Upon receiving $t+1$ valid $\langle \text{REDUCE}, C_{B(x,i)}, B(*,i), w_{B(*,i)}\rangle$ do
10:     Interpolate $B(x,i)$
11:     Enter the Proactivize phase
```

Figure 3. Procedures of ShareReduce. Each old party $P_i^e \in \mathcal{C}^e$ obtains the set $\{\langle C_{B(x,\ell)}, w_{B(i,\ell)}\rangle\}_{\ell\in[2t+1]}$ from DyCAPS.Share or the prior DyCAPS.Handoff.

Each party in the old committee has at least $2t+1$ tuples[1] $\{\langle C_{B(x,\ell)}, w_{B(i,\ell)}\rangle\}_{\ell\in[2t+1]}$ from either DyCAPS.Share or the prior DyCAPS.Handoff. With these tuples, P_i^e may interpolate any other commitments and witnesses. For example, given $B(x,j) = \sum_{\ell\in[2t+1]} \lambda_{\ell,j} B(x,\ell)$, we have $C_{B(x,j)} = \prod_{\ell\in[2t+1]} C_{B(x,\ell)}^{\lambda_{\ell,j}}$ and $w_{B(i,j)} = \prod_{\ell\in[2t+1]} w_{B(i,\ell)}^{\lambda_{\ell,j}}$ where $\{\lambda_{\ell,j}\}_{\ell\in[2t+1]}$ are the Lagrange coefficients. The interpolated tuples are used to validate the upcoming messages in subsequent steps.

The specific procedures of ShareReduce involve both old and new committees, as stated in the following.

1. The witness $w_{B(i,k)}$ is corresponding to $B(x,k)$ at $x=i$, rather than $B(i,y)$ at $y=k$. Throughout this paper, we only use commitments and witnesses of the reduced shares $B(x,*)$.

Reduce. Each old committee member P_i^e sends $\langle \text{REDUCE}, C_{B(x,j)}, B(i,j), w_{B(i,j)}\rangle$ to every new party P_j^{e+1}, where $C_{B(x,j)}$ and $w_{B(i,j)}$ are interpolated from $2t+1$ available commitment-witness pairs. Afterward, P_i^e erases its memory and goes offline.

Interpolate. Each new committee member P_i^{e+1} waits for $t+1$ valid REDUCE messages containing the same commitment $C_{B(x,i)}$. Using the polynomial evaluations in these messages, P_i^{e+1} interpolates its reduced share $B(x,i)$ and enters the Proactivize phase.

3.5. Proactivization

In the Proactivize phase, the reduced shares are refreshed by a jointly generated random polynomial. To keep the secret value $s = B(0,0)$ invariant, we need a $\langle t, 2t\rangle$-degree random polynomial $Q(x,y)$, such that $Q(0,0) = 0$. Each party obtains a random share $Q(x,*)$ and adds it to the reduced share $B(x,*)$. From a high level, the sharing polynomial is refreshed as $B'(x,y) = B(x,y) + Q(x,y)$.

This phase mainly includes two steps, collecting randomness and agreeing on a candidate set $\mathcal{Q}$. The contributions from the members in $\mathcal{Q}$ add up to the common random polynomial $Q(x,y)$. There are four requirements for the joint generation:

1) The agreement on $\mathcal{Q}$ eventually terminates.
2) Every honest party P_i^{e+1} eventually receives all necessary information to compute its random share $Q(x,i)$.
3) At least one honest party is included in $\mathcal{Q}$ so that the adversary cannot manipulate the randomness of $Q(x,y)$.
4) The adversary obtains no extra information about $Q(x,i)$ for any uncorrupted P_i^{e+1}.

The first two requirements ensure the correctness of the proactivization. The third and fourth requirements guarantee the randomness and secrecy of $Q(x,y)$, respectively.

Meeting these requirements is straightforward in non-asynchronous networks, but it becomes challenging when faced with asynchrony. We illustrate this observation with two strawman schemes before putting forward our solution. The first strawman assumes a non-asynchronous network, while the second is in the asynchronous model but fails to meet the four requirements.

Strawman I. We start from a primary scheme in the non-asynchronous setting. In this setting, there exists a timeout, so we may decide the candidate set $\mathcal{Q}$ through a challenge-response procedure.

Proactivize

```
1:  Upon invocation by $P_i^{e+1}$ with input InitProactivize do
2:    Upon receiving InitProactivize from $P_i^{e+1}$ do                ▷ Init
3:      $\pi_i \leftarrow \emptyset$
4:      $FLG_{\text{com}}[1, ..., n] \leftarrow \{0, ..., 0\}$
5:      $FLG_{\text{rec}}[1, ..., n] \leftarrow \{0, ..., 0\}$
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:      Generate a $2t$-degree polynomial $F_i(y)$, where $F_i(0) = 0$
10:     For each $\ell \in [2t+1]$ do
11:       Generate a $t$-degree polynomial $Q_i(x, \ell)$, where $Q_i(0, \ell) = F_i(\ell)$
12:     Send Commit to $P_i^{e+1}$
13:     Send Reshare to $P_i^{e+1}$

14:   Upon receiving Commit from $P_i^{e+1}$ do                         ▷ Commit
15:     For each $\ell \in [2t+1]$ do
16:       $Z_{i,\ell}(x) \leftarrow Q_i(x, \ell) - F_i(\ell)$
17:       $C_{Q_{i,\ell}} \leftarrow \mathsf{KZG.Commit}(Q_i(x, \ell))$
18:       $C_{Z_{i,\ell}} \leftarrow \mathsf{KZG.Commit}(Z_{i,\ell}(x))$
19:       $w_{Z_{i,\ell}(0)} \leftarrow \mathsf{KZG.CreateWitness}(Z_{i,\ell}(x), 0)$
20:       $\pi_i \leftarrow \pi_i \cup \{\ell, C_{Q_{i,\ell}}, C_{Z_{i,\ell}}, w_{Z_{i,\ell}(0)}, g^{F_i(\ell)}\}$
21:     Call $\mathsf{RBC}_{1,i}$ with input $\langle \text{Com}, \pi_i \rangle$

22:   Upon receiving $\langle \text{Com}, \pi_j \rangle$ from $\mathsf{RBC}_{1,j}$ do          ▷ Verify
23:     Parse $\pi_j$ as $\{\ell, C_{Q_{j,k}}, C_{Z_{j,\ell}}, w_{Z_{j,\ell}(0)}, g^{F_j(\ell)}\}_{\ell \in [2t+1]}$
24:     If $\prod_{m=1}^{2t+1} (g^{F_j(m)})^{\lambda_{m,0}^{2t}} \neq 1$ then   // $\lambda_{m,0}^{2t}$ is the Lagrange coefficient
25:       Discard this message and revert
26:     For each $\ell \in [2t+1]$ do
27:       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
28:         Discard this message and revert
29:     For each $P_\ell^{e+1} \in \mathcal{C}^{e+1}$ do
30:       $C_{Q_{j,\ell}} \leftarrow \prod_{m=1}^{2t+1} C_{Q_{j,m}}^{\lambda_{m,\ell}^{2t}}$        // $\lambda_{m,\ell}^{2t}$ is the Lagrange coefficient
31:     $FLG_{\text{com}}[j] \leftarrow 1$

32:   Upon receiving Reshare from $P_i^{e+1}$ do                        ▷ Reshare
33:     For each $P_j^{e+1} \in \mathcal{C}^{e+1}$ do
34:       For each $\ell \in [2t+1]$ do
35:         $w_{Q_i(j,\ell)} \leftarrow \mathsf{KZG.CreateWitness}(Q_i(x, \ell), j)$
36:       Send $\langle \text{Reshare}, \{Q_i(j, \ell), w_{Q_i(j,\ell)}\}_{\ell \in [2t+1]} \rangle$ privately to $P_j^{e+1}$

37:   Upon receiving $\langle \text{Reshare}, \{Q_j(i, \ell), w_{Q_j(i,\ell)}\}_{\ell \in [2t+1]} \rangle$ from $P_j^{e+1}$ do   ▷ Vote
38:     Upon $FLG_{\text{com}}[j] = 1$ then
39:       If $\forall \ell \in [2t+1], \mathsf{KZG.VerifyEval}(C_{Q_{j,\ell}}, i, Q_j(i, \ell), w_{Q_j(i,\ell)}) = 1$ then
40:         $\sigma_{j,i}^* \leftarrow \mathsf{TS.SigShare}(j, tsk_i)$
41:       For each $P_\ell^{e+1} \in \mathcal{C}^{e+1}$ do
42:         $Q_j(i, \ell) \leftarrow \sum_{m=1}^{2t+1} \lambda_{m,\ell}^{2t} Q_j(i, m)$
43:         $w_{Q_j(i,\ell)} \leftarrow \prod_{m=1}^{2t+1} w_{Q_j(i,m)}^{\lambda_{m,\ell}^{2t}}$
44:         // $\lambda_{m,\ell}^{2t}$ is the Lagrange coefficient
45:         Send $\langle \text{Recover}, j, Q_j(i, \ell), w_{Q_j(i,\ell)}, \sigma_{j,i}^* \rangle$ privately to $P_\ell^{e+1}$

46:   Upon receiving $\langle \text{Recover}, k, Q_k(j, i), w_{Q_k(j,i)}, \sigma_{k,j}^* \rangle$ from $P_j^{e+1}$ do   ▷ Recover
47:     Upon $FLG_{\text{com}}[k] = 1 \wedge \mathsf{TS.VerifySh}(k, \sigma_{k,j}^*) = 1$ then
48:       If $\mathsf{KZG.VerifyEval}(C_{Q_{k,i}}, j, Q_k(j, i), w_{Q_k(j,i)}) = 1$ then
49:         $\mathcal{S}_{\text{rec}}[k] \leftarrow \mathcal{S}_{\text{rec}}[k] \cup \langle j, Q_k(j, i) \rangle$
50:         If $|\mathcal{S}_{\text{rec}}[k]| \geq t+1$ then
51:           Interpolate $t$-degree $Q_k(x, i)$ from $\mathcal{S}_{\text{rec}}[k]$
52:           $FLG_{\text{rec}}[k] \leftarrow 1$
53:         $\mathcal{S}_\sigma[k] \leftarrow \mathcal{S}_\sigma[k] \cup \langle j, \sigma_{k,j}^* \rangle$
54:         If $|\mathcal{S}_\sigma[k]| \geq 2t+1$ then
55:           $\sigma_k \leftarrow \mathsf{TS.Combine}(k, \{\sigma_{k,j}^*\}_{(j, \sigma_{k,j}^*) \in \mathcal{S}_\sigma[k]})$
56:           $V_i[k] \leftarrow \langle k, \sigma_k \rangle$

57:   Upon there are $t+1$ full signatures in $V_i$ do                  ▷ MVBA
58:     Call MVBA with input $\langle \text{MVBA.In}, V_i \rangle$
59:     // $P_{MVBA}$ requires $|\widetilde{V}| = t+1 \wedge \forall \langle \ell, \sigma_\ell \rangle \in \widetilde{V}$, $\mathsf{TS.Verify}(\ell, \sigma_\ell) = 1$

60:   Upon reiceiving $\langle \text{MVBA.Out}, \widetilde{V} \rangle$ from MVBA do         ▷ Refresh
61:     $\mathcal{Q} \leftarrow \{P_j^{e+1} | \langle j, \sigma_j \rangle \in \widetilde{V}\}$
62:     Upon $FLG_{\text{rec}}[j] = 1$ for all $\langle j, \sigma_j \rangle \in \widetilde{V}$ do
63:       $Q(x, i) \leftarrow \sum_{P_j^{e+1} \in \mathcal{Q}} Q_j(x, i)$
64:       $B'(x, i) \leftarrow B(x, i) + Q(x, i)$
65:       For each $P_\ell^{e+1} \in \mathcal{C}^{e+1}$ do
66:         $C_{Q(x,\ell)} \leftarrow \prod_{P_j^{e+1} \in \mathcal{Q}} C_{Q_{j,\ell}}$
67:       Enter the ShareDist phase
```

Figure 4. Procedures of Proactivize.

Specifically, we let each party P_i^{e+1} initialize $\mathcal{Q}$ as $\mathcal{C}^{e+1}$ and locally generate a random $\langle t, 2t \rangle$-degree polynomial $Q_i(x, y)$, such that $Q_i(0, 0) = 0$. Next, each P_i^{e+1} invokes an RBC instance to broadcast n encrypted[2] polynomials $\mathsf{Enc}_j(Q_i(x, j))$, where $j \in [n]$, along with n commitments to these polynomials.

Then, each party waits for the output of n RBC instances and decrypts its corresponding polynomial. An honest party will raise a challenge if the polynomial is invalid or a timeout occurs. The challenged party has to respond within a limited time. The remaining parties verify the challenges and responses from both parties, either of which will be identified as malicious and excluded from $\mathcal{Q}$. Finally, each P_i^{e+1} obtains a random share $Q(x, i) = \sum_{P_j^{e+1} \in \mathcal{Q}} Q_j(x, i)$.

***Analysis*.** This strawman scheme satisfies the four requirements mentioned above:

1) All challenges will arrive in time, so the honest parties will eventually agree on $\mathcal{Q}$.
2) Due to the binding property of commitments, all malicious parties that fail to provide correct information will be challenged and excluded from $\mathcal{Q}$.
3) All honest parties will stay included in $\mathcal{Q}$ even faced with malicious challenges.
4) The adversary has at most $2t$ polynomials, which are insufficient to interpolate $Q(x, y)$.

This strawman scheme is straightforward, but the n RBC instances consume $O(\kappa n^4)$ bits of communication. Mram et al. [6] reduce the input size of each RBC instance to $O(\kappa n)$ by dividing the generation of $Q(x, y)$ into two steps, with $2t+1$ RBC instances in each step, where the separated two steps both relies on challenge-response procedures.

The challenge-response procedure is not applicable in an asynchronous network, as the honest parties may not raise or receive challenges in time. Therefore, we need other methods to determine the candidate set $\mathcal{Q}$.

Strawman II. In this strawman scheme, we relax the network assumption and advance to the asynchronous network. Inspired by asynchronous BFT protocols [9], [10], we use votes to avoid the challenge-response procedure. The voting results are decided by an MVBA instance, so that all honest parties will agree on the candidate set $\mathcal{Q}$.

Similar to Strawman I, we require each party to generate and share a local bivariate polynomial. However, we no

2. The algorithm $\mathsf{Enc}_j(m)$ encrypts a message m using P_j's encryption public key epk_j.

longer need to reliably broadcast the encrypted messages because there are no challenges and responses to be verified. Instead, each P_i^{e+1} broadcasts $2t+1$ polynomial commitments $C_{Q_{i,\ell}} = \mathsf{KZG.Commit}(Q_i(x,\ell))$ via RBC, where $\ell \in [2t+1]$, and the polynomials $Q_i(x,\ell)$ are sent to each P_ℓ^{e+1} privately. These commitments are sufficient to derive the remaining commitments to $Q_i(x,j)$.

After sending the polynomials and commitments, we let each party use threshold signatures to vote for the correct parties. Specifically, each P_i^{e+1} multicasts a signature share $\sigma_{j,i}^* = \mathsf{TS.SigShare}(j, tsk_i)$, denoting that it has received a valid polynomial from P_j^{e+1}. Upon receiving $2t+1$ signature shares for the same j, P_i^{e+1} forms a full signature σ_j. P_i^{e+1} waits for $t+1$ full signatures and formulates V_i as the input to the MVBA instance, which ensures that each party obtains the same output set $\widetilde{V}$, where $|\widetilde{V}| = t+1$. The candidate set $\mathcal{Q}$ is then denoted as $\{P_j^{e+1} | \langle j, \sigma_j \rangle \in \widetilde{V}\}$.

Analysis. This strawman scheme satisfies the first and third requirements, due to the totality of MVBA and $|\widetilde{V}| = t+1$.

However, a malicious party P_m^{e+1} may get included in $\mathcal{Q}$ if it obtains $2t+1$ votes. In the worst case, there are only $t+1$ honest parties that have voted for P_m^{e+1} and hold $Q_m(x,*)$. These $t+1$ polynomials are insufficient to recover the other $Q_m(x,j)$, whose y-dimension is $2t$-degree. Therefore, not all honest parties will receive the final share of $Q(x,y)$. Namely, this strawman scheme fails to meet the second requirement.

Our scheme. In this formal scheme, we enrich the information contained in each sharing message, so that the honest parties can help the others restore their shares.

Specifically, we make a dimension switch and let each P_i^{e+1} send a y-dimension polynomial $Q_i(*,y)$ instead of $Q_i(x,*)$. In this way, every party obtains partial information on every random share $Q_i(x,*)$. This modification brings in additional overhead to switch back the dimension.

We do not directly commit to $Q_i(*,y)$, as we need the commitments $C_{Q_i(x,*)}$ at the end of $\mathsf{DyCAPS.Handoff}$. Each $Q_i(*,y)$ is represented by $2t+1$ points and witnesses, which are verifiable *w.r.t.* $C_{Q_i(x,*)}$. The procedures of $\mathsf{Proactivize}$ are described in Figure 4. For ease of understanding, we also present the message flows of this phase in Figure 5.

Init. Each P_i^{e+1} initializes several empty sets, including a proof set π_i, two flag sets FLG_{com} and FLG_{rec}, two buffers $\mathcal{S}_{\text{rec}}$ and $\mathcal{S}_\sigma$, and an MVBA input set V_i. Then, P_i^{e+1} generates a $2t$-degree random polynomial $F_i(y)$, where $F_i(0) = 0$. Next, P_i^{e+1} reshares $2t+1$ points on $F_i(y)$ via t-degree random polynomials $Q_i(x,\ell)$, such that $Q_i(0,\ell) = F_i(\ell)$, $\ell \in [2t+1]$.

Commit. Each P_i^{e+1} generates a correctness proof $\pi_i = \{\ell, C_{Q_{i,\ell}}, C_{Z_{i,\ell}}, w_{Z_{i,\ell}(0)}, g^{F_i(\ell)}\}_{\ell \in [2t+1]}$, where $C_{Q_{i,\ell}}$ and $C_{Z_{i,\ell}}$ are the commitments to $Q_i(x,\ell)$ and $Z_{i,\ell}(x) = Q_i(x,\ell) - F_i(\ell)$, respectively, $w_{Z_{i,\ell}(0)}$ is the witness of $Z_{i,\ell}(0) = 0$, and $g^{F_i(\ell)}$ is the commitment to $F_i(\ell)$. Finally, P_i^{e+1} broadcasts $\langle \text{COM}, \pi_i \rangle$ via $\mathsf{RBC}_{1,i}$.

Verify. Upon receiving $\langle \text{COM}, \pi_j \rangle$ from $\mathsf{RBC}_{1,j}$, P_i^{e+1} verifies that the committed resharing polynomials $Q_j(x,\ell)$ are formulated correctly, where $\ell \in [2t+1]$. Specifically, P_i^{e+1} first verifies that $F_j(0) = 0$ by checking $\prod_{m=1}^{2t+1} (g^{F_j(m)})^{\lambda_{m,0}^{2t}} = 1$, where $\{\lambda_{m,0}^{2t}\}$ are Lagrange coefficients. Then, P_i^{e+1} checks $Q_j(0,\ell) = F_j(\ell)$ through $\mathsf{KZG.VerifyEval}(C_{Z_{j,\ell}}, 0, 0, w_{Z_{j,\ell}(0)}) = 1$ and $C_{R_{j,\ell}} = C_{Z_{j,\ell}} g^{F_j(\ell)}$. If any verification fails, the COM message will be discarded, and the changes related to this message will be reverted. Finally, P_i^{e+1} interpolates $C_{Q_{j,\ell}}$ for each $P_\ell^{e+1} \in \mathcal{C}^{e+1}$, and sets $FLG_{\text{com}}[j] = 1$.

Reshare. P_i^{e+1} sends $\langle \text{RESHARE}, \{Q_i(j,\ell), w_{Q_i(j,\ell)}\}_{\ell \in [2t+1]} \rangle$ to each $P_j^{e+1} \in \mathcal{C}^{e+1}$, where $w_{Q_i(j,\ell)}$ is the witness. This step is executed concurrently with the *Commit* procedure.

Vote. Upon receiving a RESHARE message from P_j^{e+1}, P_i^{e+1} first verifies it *w.r.t.* the proof π_j, which is the output from $\mathsf{RBC}_{1,j}$. Then, it formulates a signature share $\sigma_{j,i}^* = \mathsf{TS.SigShare}(j, tsk_i)$ as the vote for P_j^{e+1}. Afterward, the contents in the RESHARE message are split and relayed to the others. Specifically, P_i^{e+1} calculates an evaluation-witness tuple $\langle Q_j(i,\ell), w_{Q_j(i,\ell)} \rangle$ and sends it to each $P_\ell^{e+1} \in \mathcal{C}^{e+1}$ within a RECOVER message. The vote $\sigma_{j,i}^*$ is also included in this message.

Recover. Upon receiving $t+1$ valid RECOVER messages with the same index k, where $\mathsf{TS.VerifySh}(k, \sigma_{k,j}^*) = 1$ and $\mathsf{KZG.VerifyEval}(C_{Q_{k,i}}, *, Q_k(*,i), w_{Q_k(*,i)}) = 1$, P_i^{e+1} recovers the k-th shares by interpolating a t-degree polynomial $Q_k(x,i)$. P_i^{e+1} also waits for $2t+1$ valid votes and composes a full signature $\sigma_k = \mathsf{TS.Combine}(k, \{\sigma_{k,j}^*\}_{k \in I})$, where I contains the indexes of the collected votes. The full signatures are stored in the MVBA input set V_i.

MVBA. Upon filling the input set V_i with $t+1$ full signatures, P_i^{e+1} inputs V_i into MVBA. The external predicate P_{MVBA} requires the output size $|\widetilde{V}| = t+1$ and the full signatures within $\widetilde{V}$ are all valid. The candidate set is then referred to as $\mathcal{Q} = \{P_j^{e+1} | \langle j, \sigma_j \rangle \in \widetilde{V}\}$.

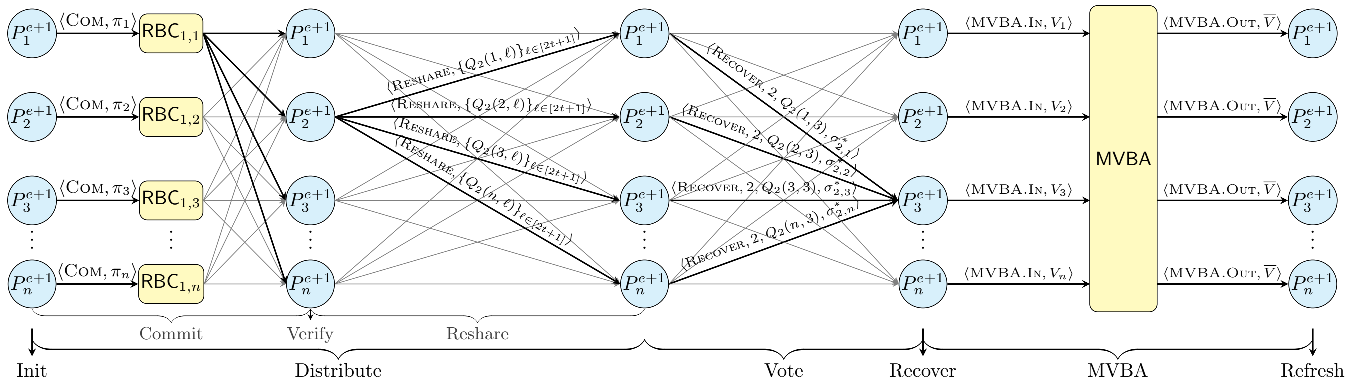
Refresh. Upon receiving $\widetilde{V}$ from MVBA, P_i^{e+1} calculates its random share $Q(x,i) = \sum_{P_j^{e+1} \in \mathcal{Q}} Q_j(x,i)$. The reduced share is thus refreshed as $B'(x,i) = B(x,i) + Q(x,i)$. Finally, P_i^{e+1} calculates the commitments $C_{Q(x,\ell)} = \prod_{P_j^{e+1} \in \mathcal{Q}} C_{Q_{j,\ell}}$ for all $P_\ell \in \mathcal{C}^{e+1}$ and enters the next phase.

3.6. Share Distribution

In the $\mathsf{ShareDist}$ phase, the reduced shares are converted to full shares. The procedures are shown in Figure 6.

Init. P_i^{e+1} initializes two empty buffers $\mathcal{S}_{\text{com}}$ and $\mathcal{S}_{B'}$. Next, P_i^{e+1} sends the COMMITNEW and DISTRIBUTE instructions to itself.

Commit. Upon receiving the COMMITNEW instruction, P_i^{e+1} commits to the new reduced share $B'(x,i)$ and broadcasts $\langle \text{NEWCOM}, C_{B'(x,i)} \rangle$ via $\mathsf{RBC}_{2,i}$.



P_1^{e+1}
$\langle \text{Com}, \pi_1 \rangle$
$\mathsf{RBC}_{1,1}$
P_2^{e+1}
$\langle \text{Com}, \pi_2 \rangle$
$\mathsf{RBC}_{1,2}$
P_3^{e+1}
$\langle \text{Com}, \pi_3 \rangle$
$\mathsf{RBC}_{1,3}$
P_n^{e+1}
$\langle \text{Com}, \pi_n \rangle$
$\mathsf{RBC}_{1,n}$
$\langle \text{Reshare}, \{Q_2(1,\ell)\}_{\ell\in[2t+1]} \rangle$
$\langle \text{Reshare}, \{Q_2(2,\ell)\}_{\ell\in[2t+1]} \rangle$
$\langle \text{Reshare}, \{Q_2(3,\ell)\}_{\ell\in[2t+1]} \rangle$
$\langle \text{Reshare}, \{Q_2(n,\ell)\}_{\ell\in[2t+1]} \rangle$
$\langle \text{Recover}, 2, Q_2(1,3), \sigma^*_{2,1} \rangle$
$\langle \text{Recover}, 2, Q_2(2,3), \sigma^*_{2,2} \rangle$
$\langle \text{Recover}, 2, Q_2(3,3), \sigma^*_{2,3} \rangle$
$\langle \text{Recover}, 2, Q_2(n,3), \sigma^*_{2,n} \rangle$
$\langle \text{MVBA.In}, V_1 \rangle$
$\langle \text{MVBA.In}, V_2 \rangle$
$\langle \text{MVBA.In}, V_3 \rangle$
$\langle \text{MVBA.In}, V_n \rangle$
MVBA
$\langle \text{MVBA.Out}, \overline{V} \rangle$
Commit
Verify
Reshare
Init
Distribute
Vote
Recover
MVBA
Refresh

Figure 5. Message flow of Proactivize within epoch $e+1$. All parties are assumed to be honest here. In the Vote stage, the emphasized Recover messages received by P_3^{e+1} refer to the responses to P_2^{e+1}'s Reshare messages. The witnesses in these messages are ommitted for cleraity of expression.

Distribute. Upon receiving the Distribute instruction, P_i^{e+1} sends $\langle \text{ShareDist}, B'(j,i), w_{B'(j,i)} \rangle$ to each P_j^{e+1}, where $w_{B'(j,i)}$ is the witness of $B'(j,i)$.

Verify. Upon receiving the NewCom message from $\mathsf{RBC}_{2,j}$, P_i^{e+1} verifies that the sender P_j^{e+1} uses the common random polynomial $Q(x,y)$ to fresh its share. Specifically, P_i^{e+1} checks $C_{B'(x,j)} = C_{B(x,j)} C_{Q(x,j)}$, which indicates that $B'(x,j) = B(x,j) + Q(x,j)$. If the verification fails, this NewCom message will be ignored.

Interpolate. P_i^{e+1} waits for $2t+1$ valid ShareDist messages to interpolate the full share $B'(i,y)$. Next, P_i^{e+1} multicasts a Success message to notify the other parties.

Success. Upon having sent the Success message, P_i^{e+1} waits for another $2t$ Success messages and then enters the normal state.

```
ShareDist
Upon invocation by $P_i^{e+1}$ with input InitDist do
    Upon receiving InitDist from $P_i^{e+1}$ do                                ▷ Init
        $\mathcal{S}_{\text{com}} \leftarrow \emptyset$
        $\mathcal{S}_{B'} \leftarrow \emptyset$
        Send CommitNew to $P_i^{e+1}$
        Send Distribute to $P_i^{e+1}$

    Upon receiving CommitNew from $P_i^{e+1}$ do                               ▷ Commit
        $C_{B'(x,i)} \leftarrow \mathsf{KZG.Commit}(B'(x,i))$
        Call $\mathsf{RBC}_{2,i}$ with input $\langle \text{NewCom}, C_{B'(x,i)} \rangle$

    Upon receiving Distribute from $P_i^{e+1}$ do                              ▷ Distribute
        For each $P_j^{e+1} \in \mathcal{C}^{e+1}$ do
            $\langle B'(j,i), w_{B'(j,i)} \rangle \leftarrow \mathsf{KZG.CreateWitness}(B'(x,i), j)$
            Send $\langle \text{ShareDist}, B'(j,i), w_{B'(j,i)} \rangle$ privately to $P_j^{e+1}$

    Upon receiving $\langle \text{NewCom}, C_{B'(x,j)} \rangle$ from $\mathsf{RBC}_{2,j}$ do       ▷ Verify
        If $C_{B'(x,j)} = C_{B(x,j)} C_{Q(x,j)}$ then
            $\mathcal{S}_{\text{com}} \leftarrow \mathcal{S}_{\text{com}} \cup \langle j, C_{B'(x,j)} \rangle$

    Upon receiving $\langle \text{ShareDist}, B'(i,j), w_{B'(i,j)} \rangle$ from $P_j^{e+1}$ do   ▷ Interpolate
        Upon $\langle j, C_{B'(x,j)} \rangle \in \mathcal{S}_{\text{com}}$ then
            If $\mathsf{KZG.VerifyEval}(C_{B'(x,j)}, i, B'(i,j), w_{B'(i,j)}) = 1$ then
                $\mathcal{S}_{B'} \leftarrow \mathcal{S}_{B'} \cup \langle j, C_{B'(x,j)}, B'(i,j), w_{B'(i,j)} \rangle$
                If $|\mathcal{S}_{B'}| \geq 2t+1$ then
                    Interpolate $2t$-dgree $B'(i,y)$ from $\mathcal{S}_{B'}$
                    Multicast Success

    Upon having sent Success and receiving $2t+1$ Success do                   ▷ Success
        Enter the normal state
```

Figure 6. Procedures of ShareDist.

4. Security and Performance Analysis

Due to limited space, we only analyze the security and performance of DyCAPS.Handoff here. The analysis of DyCAPS.Share and DyCAPS.Recon are delayed to Appendix B and Appendix C, respectively.

4.1. Security Analysis

For simplicity of expression, we continue to assume $n_e = n_{e+1} = n$ and $t_e = t_{e+1} = t$. Moreover, without loss of generality, we denote the malicious members in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ as $\{P_m^e\}_{m\in[t]}$ and $\{P_m^{e+1}\}_{m\in[t]}$, respectively.

Termination. The termination of DyCAPS.Handoff relies on the following four lemmas. Lemma 1 states that the old committee will terminate in the first two phases. Lemma 2, Lemma 3, and Lemma 4 show that the new committee will terminate in the four phases.

Lemma 1. *The honest parties in $\mathcal{C}^e$ will terminate within the e-th execution of DyCAPS.Handoff, given that at least $n-t$ honest parties from $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ are active, respectively.*

Proof. Within the e-th DyCAPS.Handoff, the old committee $\mathcal{C}^e$ is only active during Prepare and ShareReduce. In the Prepare phase, all honest old parties will connect to at least $2(n-t)$ parties, after which they send the public parameters and enter the ShareReduce phase. In the ShareReduce phase, the honest old parties only need to send messages to the new committee. The $2t+1$ commitment-witness pairs required to generate such messages come from the last execution of DyCAPS.Handoff ($e \geq 2$) or DyCAPS.Share ($e = 1$, see Appendix B), both of which are guaranteed to terminate. Therefore, the honest old parties will always send the required messages and terminate. □

Lemma 2. *The honest parties in $\mathcal{C}^{e+1}$ will terminate in the* Prepare *and* ShareReduce *phases within the e-th execution of* DyCAPS.Handoff*, given that at least $n-t$ honest parties from $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ are active, respectively.*

Proof. In the Prepare phase, each new committee member is guaranteed to connect to at least $2(n-t)$ parties and receive the public parameters from the honest old parties. Afterward, it will enter the ShareReduce phase.

In the ShareReduce phase, each honest new party will receive at least $n-t$ valid SHAREREDUCE messages, including the one P_i^{e+1} sends to itself. These messages are sufficient for P_i^{e+1} to interpolate the reduced share $B(x,i)$ and enter the next phase. ☐

Lemma 3. *The honest parties in $\mathcal{C}^{e+1}$ will terminate in the* Proactivize *phase within the e-th execution of* DyCAPS.Handoff*, given that at least $n-t$ honest parties from $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ are active, respectively.*

Proof. For an honest party P_i^{e+1} to terminate in Proactivize, it has to obtain enough points to interpolate the random polynomials $Q_j(x,i)$ for each $P_j^{e+1} \in \mathcal{Q}$. In the following, we first prove that an honest P_i^{e+1} will always proceed to the end of the MVBA instance, and then we prove that each P_i^{e+1} will obtain the random share $Q(x,i)$ after MVBA.

The worst situation for P_i^{e+1} is that the corrupted parties will not send any private message to it. In this case, the $n-t$ honest parties will each receive $n-t$ COM and RESHARE messages from the other peers. Then, they will vote for each other by the RECOVER messages. These RECOVER messages are sufficient for an honest P_i^{e+1} to interpolate $n-t$ polynomials $Q_j(x,i)$. Therefore, an honest P_i^{e+1} is guaranteed to collect $t+1$ full signatures and form a valid proposal V_i as the input to the MVBA instance. Due to the totality of MVBA, each honest party will obtain the output $\widetilde{V}$ and thus formulate the candidate set $\mathcal{Q}$.

After the termination of MVBA, each honest party will eventually calculate the random share $Q(x,*) = \sum_{P_j^{e+1}\in\mathcal{Q}} Q_j(x,*)$. We prove this statement in two cases.

Case 1: If the members in $\mathcal{Q}$ are all honest, P_i^{e+1} will receive at least $n-t$ RECOVER messages for each $P_j^{e+1} \in \mathcal{Q}$, which are sufficient to interpolate $Q_j(x,i)$. Consequently, P_i^{e+1} will compute $Q(x,i) = \sum_{P_j^{e+1}\in\mathcal{Q}} Q_j(x,i)$, refresh the reduced shares, and terminate.

Case 2: If any malicious P_m^{e+1} is included in $\mathcal{Q}$, in the worst case, P_i^{e+1} will not receive any private message from P_m^{e+1}. On the other hand, $P_m^{e+1} \in \mathcal{Q}$ means $\langle m, \sigma_m \rangle \in \widetilde{V}$, where σ_m corresponds to $2t+1$ signature shares. Hence, there are at least $t+1$ honest parties voting for P_m^{e+1}. These parties only vote for P_m^{e+1} when they receive valid COM and RESHARE messages from $\mathsf{RBC}_{1,m}$ and P_m^{e+1}, respectively. Due to the totality of RBC, P_i^{e+1} will also receive the COM message from $\mathsf{RBC}_{1,m}$. Besides, if at least $t+1$ honest parties receive the RESHARE messages from P_m^{e+1}, they will distribute the points on Q_m through the RECOVER messages. P_i^{e+1} will receive these $t+1$ shares to interpolate $Q_m(x,i)$. Therefore, P_i^{e+1} is able to compute $Q(x,i)$, refresh the reduced shares, and terminate.

In conclusion, P_i^{e+1} will terminate in either case without directly receiving messages from the corrupted peers. ☐

Lemma 4. *The honest parties in $\mathcal{C}^{e+1}$ will terminate in the* ShareDist *phase within the e-th execution of* DyCAPS.Handoff*, given that at least $n-t$ honest parties from $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ are active, respectively.*

Proof. In the final ShareDist phase, since all honest parties have refreshed their reduced shares in the Proactivize phase, an honest P_i^{e+1} will receive at least $n-t$ valid SHAREDIST messages to interpolate its full share. Similarly, each P_i^{e+1} will obtain at least $n-t$ SUCCESS, after which it terminates and enters the normal state. ☐

Theorem 5 (Termination of DyCAPS.Handoff)**.** *All honest parties will terminate in the e-th execution of* DyCAPS.Handoff*, given that at least $n-t$ honest parties from $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ are active, respectively.*

Proof. By Lemma 1, the honest parties from the old committee $\mathcal{C}^e$ will terminate. By Lemma 2, Lemma 3, and Lemma 4, the honest parties from the new committee $\mathcal{C}^{e+1}$ will terminate. Combining these lemmas, we conclude that all honest parties will terminate in the e-th execution of DyCAPS.Handoff. ☐

Correctness. 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, as shown by Lemma 6, Lemma 7, and Lemma 8, respectively.

Lemma 6. *The secret s stays invariant during the e-th execution of* ShareReduce*, given the conditions in Theorem 9.*

Proof. In this phase, the new committee members wait for enough SHAREREDUCE messages from the old committee and interpolate the reduced shares. These messages are verified against the commitment-witness pairs. An honest P_i^{e+1} will accept the tuple $\langle j, C_B(x,j), B(*,j), w_{B(*,j)} \rangle$ iff it has received at least $t+1$ messages containing the same commitment $C_B(x,j)$ and the evaluations all pass the verification. Therefore, the t corrupted parties in $\mathcal{C}^e$ cannot convince the honest parties with a different commitment C' to any other polynomial.

As shown by Lemma 2, each honest P_i^{e+1} will interpolate $B(x,i)$, whose commitment is attested by $t+1$ SHAREREDUCE messages. Due to the binding property of the commitment, the polynomial stays invariant in the ShareReduce phase, and so does the secret value $s = B(0,0)$. ☐

Lemma 7. *The secret s stays invariant during the e-th execution of* Proactivize*, given the conditions in Theorem 9.*

Proof. In this phase, the reduced shares are refreshed as $B'(x,*) = B(x,*) + Q(x,*)$. In Lemma 3, we have proved that each party will receive a random share $Q(x,*)$. In this part, we prove that the shares $Q(x,*)$ are consistent with the same $Q(x,y)$, where $Q(0,0)=0$ and $Q(x,y)$ is a $\langle t,2t\rangle$-degree polynomial for $x,y\in[n]$.

Firstly, the consistency of the candidate set $\mathcal{Q}$ is guaranteed by the agreement of MVBA. Therefore, to prove the consistency of $Q(x,*)=\sum_{P_j\in\mathcal{Q}}Q_j(x,*)$, we only need to show that the local shares $Q_m(x,*)$ generated by malicious parties $P_m^{e+1}\in\mathcal{Q}$ are consistently interpolated by the honest parties.

Considering a malicious party $P_m^{e+1}\in\mathcal{Q}$ that proposes an illegal $Q_m^*(x,y)$, whose y-dimension degree is greater than $2t$, it is only allowed to broadcast $2t+1$ commitments via $\mathsf{RBC}_{1,m}$. The remaining commitments are interpolated by the receivers. Due to the binding property, these $2t+1$ commitments fix a $\langle t,2t\rangle$-degree shadow polynomial $\hat{Q}_m(x,y)$ in the view of honest parties. If P_m^{e+1} sends an invalid point *w.r.t.* the commitments to $\hat{Q}_m(x,y)$, the receivers will not accept it. Therefore, the guaranteed outputs in Lemma 3 are actually the shares of the $\langle t,2t\rangle$-degree shadow polynomial $\hat{Q}_m(x,y)$. Namely, the honest parties will receive consistent random shares from $P_m^{e+1}\in\mathcal{Q}$, and the common random polynomial $Q(x,y)$ is also guaranteed to be $\langle t,2t\rangle$-degree.

Besides, within each COM message, the $2t+1$ commitments $\{g^{F_i(\ell)}\}_{\ell\in[2t+1]}$ ensure $Q_i(0,0)=F_i(0)=0$, so we have $Q(0,0)=\sum_{P_i^{e+1}\in\mathcal{Q}}Q_i(0,0)=0$.

In summary, the secret $s=B'(0,0)=B(0,0)+Q(0,0)$ stays invariant, and each honest party will receive a consistent new full share $B'(*,y)$. $\square$

Lemma 8. *The secret s stays invariant during the e-th execution of ShareDist, given the conditions in Theorem 9.*

Proof. In this phase, each party broadcasts the commitment to its new reduced share via an RBC instance. Each new commitment $C_{B'(x,j)}$ is verified *w.r.t.* the old commitment $C_{B(x,j)}$ and the random polynomial's commitment $C_{Q(x,j)}$. If $2t+1$ points within the SHAREDIST messages pass the evaluation verification *w.r.t.* $C_{B'(x,j)}$, the interpolated $B'(i,y)$ is indeed a full share of $B'(x,y)$. Hence, this phase also does not change the secret $s=B'(0,0)$. $\square$

Theorem 9 (Correctness of DyCAPS.Handoff)**.** *The secret s stays invariant throughout the e-th execution of DyCAPS.Handoff, at the presence of a mobile adversary corrupting at most t parties in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$, respectively.*

Proof. By Lemma 6, Lemma 7, and Lemma 8, the secret s stays invariant in all four phases. Therefore, we conclude that the correctness of DyCAPS.Handoff holds throughout the e-th execution of DyCAPS.Handoff for any $e\geq 1$. $\square$

Secrecy. To prove the secrecy of DyCAPS.Handoff, we first prove by Lemma 10 that the refreshed shares are independent of the old ones.

Lemma 10. *The mobile adversary cannot obtain extra information about the new share $B'(x,i)$ from any $t+1$ old shares $B(x,j)$, where $j\neq i$, if it does not corrupt P_i^{e+1}.*

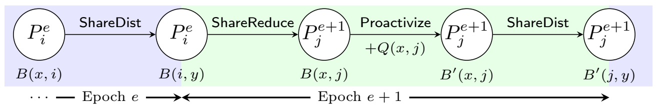


Figure 7. Shares held by P_i^e and P_j^{e+1} in adjacent handoff.

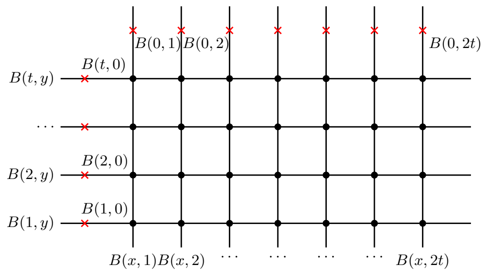


Figure 8. The adversary holds $2t^2+3t$ evaluations on $B(x,y)$.

Proof. The refreshed share $B'(x,i)$ for P_i^{e+1} is derived from the old share $B(x,i)$ by adding a random polynomial $Q(x,i)$. To compute $Q(x,i)=\sum_{P_j^{e+1}\in\mathcal{Q}}Q_j(x,i)$, the adversary needs to obtain $Q_j(x,i)$ for all $P_j^{e+1}\in\mathcal{Q}$. As $|\mathcal{Q}|=t+1$, at least one honest party is included in $\mathcal{Q}$. Given an honest party $P_h^{e+1}\in\mathcal{Q}$, the adversary obtains at most t points on the t-degree polynomial $Q_h(x,i)$, revealing no information about $Q_h(x,i)$.

On the other hand, the adversary obtains t polynomials $Q(x,m)$, where $\{P_m^{e+1}\}_{m\in[t]}$ are the corrupted parties. Since $Q(x,y)$ is of degree $\langle t,2t\rangle$ (see Lemma 7), these t polynomials reveal no information about $Q(x,i)$ for any uncorrupted P_i^{e+1}.

In conclusion, if the adversary does not corrupt P_i^{e+1}, it obtains no information about the polynomial $Q(x,i)$, so $B'(x,i)$ is independent of $B(x,i)$. $\square$

Theorem 11 (Secrecy of DyCAPS.Handoff)**.** *The adversary gains no extra knowledge about the secret s other than the public information during the repeated executions of DyCAPS.Handoff.*

Proof. The shares held by P_i^e and P_j^{e+1} are depicted in Figure 7. Specifically, 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} holds $B(x,j)$, $B'(x,j)$, and $B'(j,y)$.

By Lemma 10, we have that the bivariant polynomials in different epochs are independent. Hence, without loss of generality, we focus on polynomial $B(x,y)$. The adversary has access to $2t$ reduced shares $B(x,*)$ and t full shares $B(*,y)$. These polynomials correspond to $2t^2+3t$ independent evaluations (see Figure 8). Since $B(x,y)$ has $(t+1)(2t+1)$ coefficients, these evaluations are insufficient to interpolate $s=B(0,0)$. Therefore, the adversary gains no extra information about the secret s. $\square$

4.2. Performance Analysis

We evaluate the performance of DyCAPS by the communication complexity, which is measured in bits. Due to a similar reason in Section 4.1, we focus on the performance of DyCAPS.Handoff here. As we have remarked in Section 3.3, the communication cost in the Prepare phase is not counted, so we start with the ShareReduce phase.

In the ShareReduce phase, messages are only transferred from the old committee to the new committee. Specifically, an old member spreads n SHAREREDUCE messages, each containing three constant-sized elements. Therefore, the communication cost of this phase is $O(\kappa n^2)$ bits.

In the Proactivize phase, communication only takes place within the new committee. In the beginning, each P_i^{e+1} sends n $O(\kappa n)$-sized RESHARE messages to the other peers. Then, each party invokes an RBC instance with an $O(\kappa n)$-sized input. Each RBC instance costs $O(\kappa n^2)$ bits due to the latest scheme of Das et al. [15], which achieves $O(n|m| + \kappa n^2)$ bits of communication complexity, where $|m|$ is the input size. Next, each party sends out n^2 $O(\kappa)$-sized RECOVER messages. Finally, using sMVBA [18], which realizes $O(n^2|m| + \kappa n^2)$ communication complexity, the MVBA instance takes $O(\kappa n^3)$ bits of communication[3]. To summarize, the Proactivize phase consumes $O(\kappa n^3)$ bits of communication.

In the ShareDist phase, each party invokes an RBC instance with an $O(\kappa)$-sized input. Besides, two constant-sized messages, SHAREDIST and SUCCESS, are sent to each other. Overall, this phase consumes $O(\kappa n^3)$ bits of communication.

Altogether, DyCAPS.Handoff achieves $O(\kappa n^3)$ bits of communication complexity.

5. Implementation and Evaluation

We implement DyCAPS and deploy it on Amazon AWS for evaluation. The initial shares are generated through DyCAPS.Share (see Appendix B). The evaluations are focused on DyCAPS.Handoff, which is our main contribution.

5.1. Implementation

We implement DyCAPS using Golang v1.18 in around 5,500 lines of codes, part of which are adopted from the CHURP implementation [23]. Our implementation is built upon the GMP [24] and PBC [25] libraries. We use KZG commitments [12] and BLS threshold signatures [26] as black blocks. The P2P communication is realized through TCP sockets. Moreover, we implement Das et al.'s RBC [15] and sMVBA [18], which are of independent interest. The source code of DyCAPS is available at https://github.com/DyCAPSTeam/DyCAPS.

The commitments and signatures are on the same elliptic curve, $y^2 = x^3 + x$ over $\mathbb{F}_q$, where q is of 512 bits. The bivariant polynomials are defined over the polynomial ring $\mathbb{F}_p[x]$ for a 256-bit prime p.

3. If we apply Dumbo-MVBA [22] framework to sMVBA, the communication cost will be reduced to $O(n|m| + \kappa n^2)$, but this modification will not influence the overall asymptotic complexity of DyCAPS.

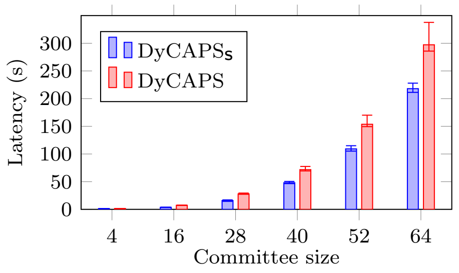


Figure 9. Latency evaluation of DyCAPS.Handoff. DyCAPS$_\mathsf{s}$ refers to the simplified version, where KZG verifications are omitted.

5.2. Evaluation

We deploy DyCAPS on 64 Amazon EC2 t2.medium instances from 8 regions. Every instance serves as a committee member. For ease of measuring, our experiments are conducted within an honest and static committee.

Latency. The latency of DyCAPS.Handoff is defined as the time for each party to obtain the refreshed full shares. Observe that one of the main bottlenecks of DyCAPS.Handoff is the $O(n^2)$ verifications of the incoming RECOVER messages. To separate the computational costs and network latency, we further evaluate a simplified version, DyCAPS$_\mathsf{s}$, where the KZG verifications are omitted.

The results are shown in Figure 9. For the smallest committee ($n = 4$), the latency of DyCAPS.Handoff is around 1.36 seconds. The latency grows to around 300 seconds when the committee is scaled to 64 members. Such latency is induced by both asynchronous communication and local computation. By comparing DyCAPS and DyCAPS$_\mathsf{s}$, we conclude that the local computation accounts for 30-50% of the total latency.

Dynamic BFT. Observe that the message flow of DyCAPS.Handoff includes all procedures of Dumbo2 [10], including RBC, threshold signatures, and MVBA. Therefore, DyCAPS.Handoff may serve as a dynamic BFT protocol, where the transaction payloads are sent along with the commitments. We evaluate the latency of DyCAPS.Handoff and Dumbo2 with different payload sizes. Remarkably, the performance of Dumbo2 is better than reported in [10], because we use more efficient RBC and MVBA schemes. Besides, Golang has better runtime performance than Python [27].

As shown in Figure 10, DyCAPS.Handoff requires several seconds even when there is no payload, but as the payload size grows, the overall latency of Dumbo2 catches up with DyCAPS.Handoff. Given a payload size of 20 MB and a committee size $n = 22$, the latency overhead of DyCAPS.Handoff is less than 15% when compared to Dumbo2 latency. When n is small, this overhead is even smaller, i.e., 9% at $n = 10$ and 6% at $n = 4$. In conclusion, our implementation equips Dumbo2 with the ability of proactivization, and Dumbo2's overall latency is only slightly influenced.

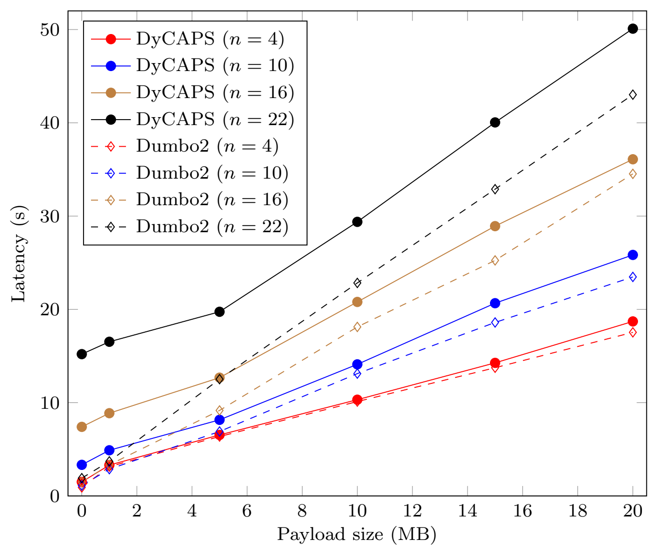


Figure 10. Latency of DyCAPS.Handoff and Dumbo2 [10] with different payload sizes.

6. Change of Size and Threshold

The change of the committee size and threshold is a common demand for long-term systems, for both security and flexibility considerations. To support such changes, we introduce several modifications to the scheme in Section 3.

6.1. Change of Size

Given a fixed threshold, the change of committee size is already taken into consideration and supported by the scheme in Section 3. However, we do have a limit on the committee size when $n' < n$. That is, we require $n' > 3t$ to ensure the security properties. If the old size n has reached the lower bound, i.e., $n = 3t+1$, a reduction of t is needed before decreasing n to n', as shown in Section 6.2.

6.2. Change of Threshold

The increase and decrease of the threshold require different techniques, as described in the following.

Increasing threshold. To increase the threshold from t to t', where $t' > t$, we require that the refreshed polynomial $B'(x, y)$ is of degree $\langle t', 2t' \rangle$. An intuitive solution is directly generating a $\langle t', 2t' \rangle$-degree $Q(x, y)$ and adding it to $B(x, y)$. However, this method enables the adversary to recover the secret $s = B(0, 0)$ with $t + t' > 2t$ reduced shares $B(x, *)$. To fix this problem, we let the old committee locally perform an additional round of DyCAPS.Handoff, raising the y-dimension degree to $2t'$.

In this additional round, the sharing polynomial $B(x, y)$ held by $\mathcal{C}^e$ is refreshed to $B_{\text{tmp}}(x, y)$, which is of $\langle t, 2t' \rangle$-degree. This round only involves the old committee, who already has the reduced share $B(x, *)$ from the last handoff (or the initial sharing), so the Prepare and ShareReduce phases are omitted. In Proactivize, each P_i^e generates $2t'+1$ random polynomials $Q_i(x, \ell)$ of degree t, where $\ell \in [2t' + 1]$. The remaining operations are the same as in Section 3.5 and Section 3.6. By Lemma 3 and Lemma 4, each P_i^e will obtain a t-degree reduced share $B_{\text{tmp}}(x, i)$ and a $2t'$-degree full share $B_{\text{tmp}}(i, y)$. Afterward, the old committee starts the regular DyCAPS.Handoff and hands over the reduced shares to the new committee, which subsequently generates $Q(x, y)$ of degree $\langle t', 2t' \rangle$ and refresh $B_{\text{tmp}}(x, y)$ to $B'(x, y)$. In this way, the adversary, who obtains $t + t' < 2t'$ reduced shares, cannot recover the secret.

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

Decreasing threshold. Reducing the sharing polynomial's degree is non-trivial. We follow prior schemes [6], [11] and introduce virtual parties. Specifically, given a new threshold $t' = t - d$, where $t > d > 0$, we set d virtual parties, whose full shares are exposed to all members. In this way, the degree of $B'(x, y)$ remains $\langle t, 2t \rangle$, while $t + 1 - d$ full shares from non-virtual parties are needed to perform the threshold operations.

In this case, the Prepare phase remains the same. In the ShareReduce phase, each old party additionally sends d points on its full share, so that every new party obtains the reduced shares of d virtual parties. This will not influence the secrecy, because the adversary only has access to $t + t' + d = 2t$ reduced shares. In the Proactivize phase, all honest parties (including the virtual ones) will vote for the virtual parties, whose contributions are $Q_v(x, y) = 0$. In this way, the MVBA instance will terminate even if the corrupted parties withhold the inputs, as shown in Lemma 3. Finally, in the ShareDist phase, the messages towards the virtual parties are multicasted so that every party can interpolate the full shares of the virtual parties.

7. Discussion

Here, we first list the related works in Section 7.1. Then we discuss the application scenarios of DyCAPS in Section 7.2.

7.1. Related Work

Table 1 concludes the performance and properties of several related DPSS schemes.

Non-asynchronous DPSS. Desmedt and Jajodia [4] are the first to consider dynamic committees in PSS. However, their work assumes a semi-honest adversary. Wong et al. [30] extend [4] to withstand a malicious adversary, but they introduce another impractical assumption that the new members are all honest.

Schultz-MPSS [11] achieves DPSS with a communication cost of $O(\kappa n^4)$ bits. Although Schultz-MPSS claims to support asynchrony, its underlying PBFT [31] protocol has been identified as partially synchronous in recent works [9].

TABLE 1. RELATED DPSS SCHEMES. THE COMMUNICATION COST IS CALCULATED IN BITS[1].

Reference	Async.	Adversary	Threshold	Best-case[2] comm. cost	worst-case comm. cost	Trusted setup[3]	PKE required
Schultz-MPSS [11]	$\times$	Mobile	$t < n/3$	$O(\kappa n^4)$	$O(\kappa n^5)$	$\times$	$\surd$
Opt-CHURP [6]	$\times$	Mobile & semi-honest	$t < n/2$	$O(\kappa n^2)$	N/A	$\surd$	$\times$
Exp-CHURP-A [6]	$\times$	Mobile	$t < n/2$	N/A	$O(\kappa n^3)$	$\surd$	$\surd$
COBRA [7]	$\times$	Mobile	$t < n/3$	$O(\kappa n^3)$	$O(\kappa n^4)$	$\surd$	$\surd$
APSS [8]	$\surd$	Mobile	$t < n/3$	$\exp(n)$	–	$\times$	$\times$
Shanrang [28]	$\surd$	Mobile	$t < n/4$	$O(\kappa n^3 \log n)$	N/A	$\surd$	$\surd$
Yurek et al. [29]	$\surd$	Mobile	$t < n/3$	$O(\kappa n^3)$	–	$\times$[4]	$\surd$
DyCAPS (this work)	$\surd$	Mobile	$t < n/3$	$O(\kappa n^3)$	–	$\surd$	$\times$

[1] n: comittee size, t: number of corrupted parties, κ: security parameter, N/A: not applicable, –: same as the best case.
[2] In the best case, all parties behave honestly. In the worst case, there are t corrupted parties behaving maliciously.
[3] The trusted setup does not include the initial share distribution, which is replaceable by distributed key generation.
[4] Given no trusted setup, the NIZK proofs of correct PKE in [29] may introduce a large constant factor.

CHURP [6] and COBRA [7] are two state-of-the-art DPSS schemes in synchronous and partially synchronous networks, respectively. CHURP achieves a communication cost of $O(\kappa n^2)$ bits in the optimistic case. However, if any party misbehaves, CHURP falls to the pessimistic path and requires $O(\kappa n^3)$ bits of communication[4]. COBRA achieves an overhead of $O(\kappa n^3)$ bits, but its worst-case communication grows to $O(\kappa n^4)$ when faced with t continuous malicious leaders.

Asynchronous DPSS. Zhou et al. [8] propose the first asynchronous dynamic-committee PSS. They use the XOR-based secret sharing, which results in exponential communication costs. We reduce the communication cost to $O(\kappa n^3)$ bits, making asynchronous DPSS more practical.

We have noticed two concurrent works by Yan et al. [28] and Yurek et al. [29]. Yan et al. [28] propose Shanrang, which uses two rounds of Honeybadger [9] to deal with the asynchronous network, with a communication cost of $O(\kappa n^3 \log n)$. Besides, Shanrang only tolerates $t < n/4$ corrupted parties. Yurek et al. [29] achieve the same asymptotic overhead as ours, and they focus on the amortized cost of multiple secrets. Their scheme uses public-key encryptions (PKE) and relies on non-interactive zero-knowledge (NIZK) proofs to guarantee the correctness of PKE. Although there are constant-sized NIZK proofs [32], [33], they are much larger than KZG commitments (approximately 1 KB vs. 256 bits). Moreover, the prover and verifier time may dominate the computational cost of each party. Therefore, our scheme will be more efficient in some scenarios where only several secrets are shared, e.g., permissioned blockchains. On the other hand, when the secret batch becomes large, Yurek et al.'s work [29] will be more practical.

Asynchronous verifiable secret sharing (AVSS). The reshare-recover procedures within our Proactivize phase are similar to Backes et al.'s eAVSS-SC [34]. The main difference is that we separate the RBC and resharing messages, so that we may choose more efficient RBC schemes for implementation. As a trade-off, these procedures cannot be extracted as a standalone DPSS.Share scheme because the separation of messages will influence the totality property (see Definition 1).

Asynchronous distributed key generation (ADKG). The core of asynchronous PSS schemes is an ADKG protocol, where common randomness is jointly generated and added to the original shares. DyCAPS has the same asymptotic efficiency as the state-of-the-art ADKG scheme [35], but each participant will obtain a random polynomial instead of a random element.

4. We replace the broadcast channel (referred to as bulletin board) with Das et al.'s RBC [15] to calculate CHURP's communication overhead.

7.2. Applications of DyCAPS

As DyCAPS supports a more flexible committee, it may promote the applications of several long-term systems, such as committee-based blockchains and decentralized identity.

Flexible committees for blockchains. Most committee-based blockchains use BFT protocols [9], [10], [36] to order the transactions, where the BFT committee is usually fixed. Using DyCAPS, the committee management will be more flexible. Adjusting committee members, size, and threshold may strengthen the system's long-term security against a mobile adversary. DyCAPS is also suitable for proof-of-stake (PoS) blockchains, where the committee is selected according to the users' stakes and changes over time. With DyCAPS, the PoS committee may use the same key pairs to sign the blocks. In this way, the blockchain users will be relieved of the burden of recording historical public keys to verify the blocks.

Decentralized identity (DID). The blossom of decentralized applications (DApps) on blockchains [37] has triggered the public's interest in DID [38], [39], [40], which refers to the on-chain assets and credentials. To manage a DID, a user may refer to DyCAPS to lower the risk of losing or exposing the secret key.

8. Conclusion

In this paper, we propose DyCAPS, an efficient asynchronous DPSS scheme with $O(\kappa n^3)$ bits of communication cost. DyCAPS ensures its termination and correctness in asynchrony and guarantees the privacy of the secret shares. Due to its robustness in asynchrony, DyCAPS is suitable for long-term key management and committee governance. DyCAPS may facilitate committee-based systems to evolve to the dynamic setting, especially for decentralized autonomous organizations and blockchains.

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, pp. 51–59.

[2] V. Nikov and S. Nikova, "On proactive secret sharing schemes," in *SAC 2004*, ser. LNCS, vol. 3357. Springer, pp. 308–325.

[3] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[4] 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

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

[6] 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, pp. 2369–2386.

[7] 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.

[8] 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.

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

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

[11] 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.

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

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

[14] K. Nayak, L. Ren, E. Shi, N. H. Vaidya, and Z. Xiang, "Improved extension protocols for byzantine broadcast and agreement," in *DISC 2020*, ser. LIPIcs, vol. 179. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 28:1–28:17.

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

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

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

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

[19] 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, pp. 31–46.

[20] A. Tomescu, R. Chen, Y. Zheng, I. Abraham, B. Pinkas, G. Golan-Gueta, and S. Devadas, "Towards scalable threshold cryptosystems," in *SP 2020*. IEEE, pp. 877–893.

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

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

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

[24] "The GNU multiple precision (GMP) arithmetic library," 2021. [Online]. Available: https://gmplib.org/

[25] "Go wrapper for the pairing based cryptography (PBC) library," 2018. [Online]. Available: https://github.com/Nik-U/pbc

[26] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *J. Cryptol.*, vol. 17, no. 4, pp. 297–319, 2004.

[27] D. Lion, A. Chiu, M. Stumm, and D. Yuan, "Investigating managed language runtime performance," in *USENIX ATC 2022*, pp. 835–852.

[28] 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

[29] T. Yurek, Z. Xiang, Y. Xia, and A. Miller, "Long live the honey badger: Robust asynchronous dpss and its applications," Cryptology ePrint Archive, Paper 2022/971, 2022. [Online]. Available: https://eprint.iacr.org/2022/971

[30] T. M. Wong, C. Wang, and J. M. Wing, "Verifiable secret redistribution for archive systems," in *First International IEEE Security in Storage Workshop, 2002. Proceedings*. IEEE, pp. 94–105.

[31] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *OSDI 1999*. USENIX Association, pp. 173–186.

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

[33] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," Cryptology ePrint Archive, Paper 2018/046, 2018. [Online]. Available: https://eprint.iacr.org/2018/046

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

[35] 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, pp. 444–460.

[36] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *EuroSys 2018*. ACM, 2018, pp. 30:1–30:15.

[37] 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.

[38] 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, pp. 1348–1366.

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

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

Appendix A. Notations

TABLE 2. Frequently used notations in this paper

Notation	Description
κ	Security parameter
s	Secret value
e	Epoch number
$\mathcal{C}^e$	The committee in epoch e
P_i^e	The i-th party in $\mathcal{C}^e$
n_e	Size of $\mathcal{C}^e$
t_e	Maximum number of corrupted parties in $\mathcal{C}^e$
C_ϕ	Commitment to some polynomial $\phi(x)$
$w_{\phi(i)}$	Witness to the evaluation of $\phi(x)$ at $x = i$
σ_i^*, σ	Signature share from P_i and full signature, respectively
$\emptyset$	Empty set

Appendix B. DyCAPS.Share

In this section, we present a leader-based instantiation of DyCAPS.Share. The existence of a dealer is necessary for some applications. For example, a dealer may delegate its secret key to a group of people to do threshold cryptographic operations. We leave the dealer-free version for future research.

DyCAPS.Share requires a trusted setup to initialize the KZG commitments. We further assume the commitment public key cpk and the public parameters are available for all members. We slightly modify Backes et al.'s eAVSS-SC scheme [34] to support a $\langle t, 2t\rangle$-degree bivariate polynomial. The procedures are shown in Figure 11.

Init. The initialization for the dealer P_d and the peers P_i is different. Specifically, P_d initializes a proof set π, which originally contains only g^s. Then, P_d generates a $2t$-degree random polynomial $F(y)$, where $F(0) = s$. $F(y)$ is further extended to $2t+1$ random polynomials $B(x, \ell)$ of degree t, such that $B(0, \ell) = F(\ell)$ for each $\ell \in [2t + 1]$. As for the peers, each of them only needs to initialize an empty buffer $\mathcal{S}_{\text{full}}$ and a flag $FLG_{\text{ready}} = 0$.

Commit. To prove the correctness of *Init*, P_d sets $\pi = \{g^s, \{\ell, C_{B_\ell}, C_{Z_\ell}, w_{Z_\ell(0)}, g^{F(\ell)}\}_{\ell\in[2t+1]}\}$, where C_{B_ℓ} and C_{Z_ℓ} are the commitments to $B(x, \ell)$ and $Z_\ell(x) = B(x, \ell) - F(\ell)$, respectively, and $w_{Z_\ell(0)}$ is the witness of $Z_\ell(0) = 0$. The main difference from Proactivize in Section 3.5 is that we include g^s in π.

Send. P_d sends $\langle \text{SEND}, \pi, \{B(i, \ell), w_{B(i,\ell)}\}_{\ell\in[2t+1]}\rangle$ 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 peers.

Echo. Upon receiving the SEND message from the dealer, P_i verifies that the polynomials $B(x, \ell)$ are correctly formulated, where $\ell \in [2t + 1]$, following similar verification steps as in Proactivize. P_i also verifies that the evaluation-witness pairs *w.r.t.* the commitments in π'. If all verifications return true, P_i sets π as π'. Then, P_i interpolates a $2t$-degree polynomial $B^*(i, y)$. The witnesses $\{w_{B^*(i,j)}\}_{P_j\in\mathcal{C}}$ are also interpolated from π'. Finally, P_i multicasts $\langle \text{ECHO}, \pi'\rangle$.

Ready. Upon receiving $n-t$ ECHO messages or $t+1$ READY messages with the same π', P_i checks whether $\pi = \pi'$ holds. If so, P_i sends $\langle \text{READY}, \pi', \text{SHARE}, B^*(i, \ell), w_{B^*(i,\ell)}\rangle$ to each $P_\ell \in \mathcal{C}$. Otherwise, P_i resets π as π' and discards the interpolated $\{w_{B^*(i,\ell)}\}_{P_\ell\in\mathcal{C}}$ and $B^*(i, y)$. In the latter case, P_i multicasts $\langle \text{READY}, \pi', \text{NOSHARE}\rangle$.

Distribute. 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 one point on this polynomial to every $P_\ell \in \mathcal{C}$ via $\langle \text{DISTRIBUTE}, B(\ell, i), w_{B(\ell,i)}\rangle$.

Recover. P_i collects $2t+1$ valid DISTRIBUTE messages and interpolates a $2t$-degree polynomial $B(i, y)$, which is the full share of s.

Analysis. Following Backes et al.'s scheme [34], we may easily conclude that the liveness, totality, and secrecy properties hold until the *Ready* stage. We prove that the additional steps in *Distribute* and *Recover* do not influence these properties. The proof of correctness is delayed to Appendix C, as it involves DyCAPS.Recon.

Theorem 12 (Liveness of DyCAPS.Share)**.** *If the dealer is honest, all honest parties will complete DyCAPS.Share in the presence of t corrupted parties.*

Proof. Following the proof of liveness in [34], we may conclude that all honest parties will interpolate a t-degree polynomial $B(x, i)$ in the *Distribute* stage. Then, every honest party P_i will send a DISTRIBUTE message to the others, and P_i will receive at least $n - t$ such messages, which are sufficient to interpolate $B(i, y)$. □

Theorem 13 (Totality of DyCAPS.Share)**.** *If an honest party completes DyCAPS.Share, all honest parties will complete DyCAPS.Share.*

Proof. If an honest party completes DyCAPS.Share, it has received $2t+1$ valid DISTRIBUTE messages, where $t+1$ of them are from honest parties. In turn, an honest party P_i only sends the DISTRIBUTE messages when it has interpolated $B(x, i)$. Following the proof of totality in [34] (referred to as agreement in this work), if an honest party P_i interpolates $B(x, i)$, all honest parties will obtain the reduced shares and send the DISTRIBUTE messages. As a result, all honest parties will receive at least $n - t$ valid messages, which are sufficient to interpolate the full shares $B(*, y)$. □

Theorem 14 (Secrecy of DyCAPS.Share)**.** *The adversary gains no extra knowledge about the secret s other than the public information during DyCAPS.Share.*

```
DyCAPS.Share

1: Upon invocation by $P_d$ with input $\langle$InitShare$, s\rangle$ do
2:   Upon receiving $\langle$InitShare$, s\rangle$ from $P_d$ do                ▷ Init
3:     $\pi \leftarrow g^s$
4:     Generate a $2t$-degree polynomial $F(y)$, where $F(0) = s$
5:     For each $\ell \in [2t+1]$ do
6:       Generate a $t$-degree polynomial $B(x, \ell)$, where $B(0, \ell) = F(\ell)$
7:     Send Commit to $P_d$

8:   Upon receiving Commit from $P_d$ do                                          ▷ Commit
9:     For each $\ell \in [2t+1]$ do
10:      $Z_\ell(x) \leftarrow B(x, \ell) - F(\ell)$
11:      $C_{B_\ell} \leftarrow$ KZG.Commit$(B(x, \ell))$
12:      $C_{Z_\ell} \leftarrow$ KZG.Commit$(Z_\ell(x))$
13:      $w_{Z_\ell(0)} \leftarrow$ KZG.CreateWitness$(Z_\ell(x), 0)$
14:      $\pi \leftarrow \pi \cup \{j, C_{B_\ell}, C_{Z_\ell}, w_{Z_\ell(0)}, g^{F(\ell)}\}$
15:    Send Send to $P_d$

16:  Upon receiving Send from $P_d$ do                                            ▷ Send
17:    For each $P_i \in \mathcal{C}$ do
18:      For each $\ell \in [2t+1]$ do
19:        $w_{B(i,\ell)} \leftarrow$ KZG.CreateWitness$(B(x, \ell), i)$
20:      Send $\langle$Send$, \pi, \{B(i, \ell), w_{B(i,\ell)}\}_{\ell \in [2t+1]}\rangle$ privately to $P_i$

21: Upon invocation by $P_i \in \mathcal{C}$ with input InitShare do
22:   Upon receiving InitShare from $P_i$ do                                      ▷ Init
23:     $\mathcal{S}_{\text{full}} \leftarrow \emptyset$
24:     $FLG_{\text{ready}} \leftarrow 0$

25:   Upon receiving $\langle$Send$, \pi', \{B(i, \ell), w_{B(i,\ell)}\}_{\ell \in [2t+1]}\rangle$ from $P_d$ do   ▷ Echo
26:     Verify $\pi'$ as line 24-28 in Proactivize // $\{g^{F(\ell)}\}_{\ell \in [2t+1]}$ are verified w.r.t. $g^s$
27:     Verify $\{B(i, \ell), w_{B(i,\ell)}\}_{\ell \in [2t+1]}$ w.r.t. $\pi$
28:     $\pi \leftarrow \pi'$
29:     Interpolate a $2t$-degree polynomial $B^*(i, y)$ from $\{\langle \ell, B(i, \ell)\rangle\}_{\ell \in [2t+1]}$
30:     Interpolate $\{w_{B^*(i,j)}\}_{P_j \in \mathcal{C}}$ from $\{w_{B(i,\ell)}\}_{\ell \in [2t+1]}$
31:     Multicast $\langle$Echo$, \pi'\rangle$

32:   Upon receiving $n - t$ $\langle$Echo$, \pi'\rangle$ or $t + 1$ $\langle$Ready$, \pi', *\rangle$ do   ▷ Ready
33:     If $FLG_{\text{ready}} = 0$ then
34:       If $\pi' = \pi$ then
35:         Send $\langle$Ready$, \pi',$ Share$, B^*(i, \ell), w_{B^*(i,\ell)}\rangle$ to each $P_\ell \in \mathcal{C}$
36:       Else
37:         $\pi \leftarrow \pi'$
38:         Discard $\{w_{B^*(i,\ell)}\}_{P_\ell \in \mathcal{C}}$ and $B^*(i, y)$
39:         Multicast $\langle$Ready$, \pi',$ noShare$\rangle$
40:       $FLG_{\text{ready}} \leftarrow 1$

41:   Upon receiving $\langle$Ready$, \pi', *\rangle$ from $n - t$ parties do        ▷ Distribute
42:     Upon there are $t + 1$ valid Ready-Share messages do
43:       Interpolate $B(x, i)$
44:       Send $\langle$Distribute$, B(\ell, i), w_{B(\ell,i)}\rangle$ to each $P_\ell \in \mathcal{C}$

45:   Upon reiceiving $\langle$Distribute$, B(i, j), w_{B(i,j)}\rangle$ from $P_j$ do   ▷ Recover
46:     Upon $FLG_{\text{ready}} = 1$ do
47:       Interpolate $C_{B_j}$ from $\pi$
48:       If KZG.VerifyEval$(C_{B_j}, B(i, j), w_{B(i,j)}) = 1$ then
49:         $\mathcal{S}_{\text{full}} \leftarrow \mathcal{S}_{\text{full}} \cup (j, B(i, j))$
50:         If $|\mathcal{S}_{\text{full}}| \geq 2t + 1$ then
51:           Interpolate a $2t$-degree polynomial $B(i, y)$ from $\mathcal{S}_{\text{full}}$
```

Figure 11. Procedures of dealer-based DyCAPS.Share. The dealer P_d is responsible to distribute the shares of the secret s among $\mathcal{C} = \{P_i\}_{i \in [n]}$.

Proof. The secrecy is only meaningful when the dealer is honest. In this case, the adversary obtains t Send messages, n Echo messages, $t \times n$ Ready messages, and $t \times n$ Distribute messages. Besides, the final polynomial $B(*, y)$ is the same as $B^*(*, y)$, which is interpolated from the Send message. We refer to both $B(*, y)$ and $B^*(*, y)$ as full shares in the following.

Without loss of generality, we denote the corrupted parties as $\{P_m\}_{m \in [t]}$. The t Send messages held by the adversary correspond to t full shares $B^*(m, y)$, which are insufficient to interpolate $s = B^*(0, 0)$. The n Echo messages only contain public information, revealing no information about s. In the subsequent steps, each P_m obtains n Ready and Distribute messages, respectively. Any $t+1$ Ready messages result in the reduced share $B(x, m)$, and any $2t+1$ Distribute messages lead to the full share $B(m, y)$. Therefore, the adversary will have t reduced shares and t full shares. As $B(x, y)$ is of degree $\langle t, 2t\rangle$, the adversary obtains no information about the secret $s = B(0, 0)$ with these shares. Remarkably, the adversary will obtain another t reduced shares during the first handoff, but $2t$ reduced shares are still insufficient to recover the secret, as proved in Lemma 11. □

Appendix C. DyCAPS.Recon

When a dealer invokes DyCAPS.Recon, the peers send their full shares $B(*, y)$ to the dealer, along with a set of commitments $\{C_{B(\ell,y)}\}_{P_\ell \in \mathcal{C}}$, where $\mathcal{C}$ is the current committee. The dealer collects $t+1$ valid shares and interpolates $B(x, y)$. The reconstructed secret is thus $s = B(0, 0)$.

In case there is no dealer, the peers may broadcast their shares via RBC, and each of them will receive enough shares to recover the secret.

Analysis. In the following, we prove the termination of DyCAPS.Recon and the correctness of DyCAPS. The proof of secrecy is omitted, as the secret is exposed to every party in the dealer-free case.

Theorem 15 (Termination of DyCAPS.Recon)**.** *In either dealer-based or dealer-free cases, if there are at least $n - t$ honest parties during the execution of DyCAPS.Recon, the dealer and the honest parties will receive s, respectively.*

Proof. By Theorem 5, all honest parties will receive a full share $B(*, y)$ after the last execution of DyCAPS.Handoff, where $B(x, y)$ is of degree $\langle t, 2t\rangle$. Therefore, in both dealer-based and dealer-free cases, there will be at least $n-t$ valid full shares, so the invoker(s) of DyCAPS.Recon will recover the secret s using any $t + 1$ full shares. □

Theorem 16 (Correctness of DyCAPS)**.** *If the dealer is honest, the output v of* $\mathsf{DyCAPS.Recon}$ *is the same as the original secret s.*

Proof. By Theorem 9, $\mathsf{DyCAPS.Handoff}$ keeps the secret s invariant. Due to the binding property of the commitments, the reconstruction does not change the polynomials and the corresponding secret value. □