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 without changing the secret. Such a proactivization functionality makes DPSS a promising technology for long-term key management and committee governance. Although non-asynchronous DPSS schemes have achieved cubic communication cost *w.r.t.* the number of shareholders, the overhead of asynchronous DPSS remains exponential. In this paper, we fill this gap and propose DyCAPS, an efficient asynchronous DPSS scheme with a cubic communication cost.**

DyCAPS can be efficiently integrated into asynchronous BFT protocols without increasing the overall asymptotic communication cost. Experimental results show that given a payload of 15 MB per party, DyCAPS achieves member change in Dumbo2 (CCS 2020) at the cost of 5%–22% throughput degradation, when the committee size varies from 4 to 22.

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, and the shares are freshed periodically and distributedly, without influencing the original secret. 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 pointed out by Duan and Zhang [5], dynamic-committee BFT protocols are in great demand in real-world applications. DPSS may also solve the problem of committee authentication, where the member change will not influence the public keys of a committee, saving the efforts to inform the public of new public keys and revoke the old ones.

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 is a time bound for the message delivery, so that the misbehaviors can be identified efficiently. The state-of-the-art synchronous DPSS scheme, CHURP [6], requires $O(\kappa n^2)$ bits of communication in the presence of a semi-honest adversary, 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 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 $\exp(n)$ communication, which is far from real-world implementation. The lack of efficient asynchronous DPSS schemes may hinder the decentralized systems from adapting to the dynamic setting, including blockchain systems [9], [10], decentralized autonomous organizations [11], and threshold-cryptography-as-a-service systems [12].

Migrating the non-asynchronous DPSS schemes to asynchrony is not straightforward, as most of them [6], [7], [13] 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 challenges or responses is due to the unbounded network latency or malicious behaviors.

In this paper, we propose DyCAPS, an efficient and BFT-friendly asynchronous DPSS scheme.

Contributions. Our contributions are as follows.

- We propose DyCAPS, the first efficient 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, DyCAPS beats COBRA [7], which assumes partial synchrony.
- We give a formal definition of asynchronous DPSS and prove the security of DyCAPS.
- We implement DyCAPS and integrate it into an asynchronous BFT protocol, achieving dynamic committee without increasing the asymptotic communication cost.
- We evaluate DyCAPS on Amazon EC2 t2.medium instances from 8 regions. The results show that the proactivization for $n = 4$ and $n = 16$ completes in around 1.5 and 8 seconds, respectively. Given a payload of 15 MB per party and n from 4 to 22, the extra latency overhead

TABLE 1: Notations

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 $\mathcal{C}^e$
$B(x, y)$	Bivariate polynomial for the secret sharing
σ	Digital signature
σ_i^*	Signature share from P_i
$\mathrm{FLG}_{\mathrm{cntx}}$	Flag, where cntx denotes the context
C_ϕ	Commitment to the polynomial $\phi(x)$
$w_{\phi(i)}$	Witness for the evaluation of $\phi(x)$ at $x = i$
$\emptyset$	Empty set

of DyCAPS is around 5%–22% when compared to the static-committee 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 security and performance analysis of DyCAPS is 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

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 1. Specifically, 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_m denotes digital signatures of a message m, whereas $\sigma_{m,i}^*$ is the signature share by P_i. Flags are referred to as FLG, with a subscript denoting its context, e.g., $\mathrm{FLG}_{\mathrm{com}}$ is the commitment flag. We use C_ϕ to denote the commitment to the polynomial $\phi(x)$, and $w_{\phi(i)}$ is the witness for the evaluation of $\phi(x)$ at $x = i$.

2.2. System Model

Network model. We assume an asynchronous network, where an adversary controls the order of the messages, but the messages will be delivered eventually. Besides, the parties are connected by authenticated and private channels. We further assume that these channels are forward-secure, as demonstrated in [13].

Epoch. We follow Schultz-MPSS [13] and define epochs according to the local events of each party. An honest party is active in epoch e if it holds the secret share for this epoch. Between epochs e and $e+1$, the committees $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ collaboratively execute a handoff protocol to refresh the secret shares.

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. Moreover, the adversary is computationally bounded.

Trusted setup. We require a trusted setup to initialize the KZG polynomial commitment scheme [14] (see Section 2.3), which is one of the key ingredients for DyCAPS to achieve cubic communication cost.

Memory erasure. Following existing DPSS schemes [6], [7], [13], we require honest parties to erase their memory before exiting the current epoch. Otherwise, an adversary may obtain the old shares when it corrupts a party that is honest in previous epochs.

2.3. Building Blocks

Reliable broadcast (RBC) [15], [16] ensures that all honest parties deliver the same message, or none delivers any message. An RBC protocol satisfies the following properties.

- *Agreement*[1]. If an honest party outputs v, then all honest parties output v.
- *Validity.* If the leader is honest and input v, then all honest parties output v.

Multi-valued validated asynchronous Byzantine agreement (MVBA) [17], [18], [19] allows each party to input a proposal and agree on a valid proposal *w.r.t.* an external predicate P_{MVBA}. An MVBA protocol satisfies the following properties.

- *Agreement.* If two honest parties have outputs, then their outputs are the same.
- *Termination.* If all honest parties input values satisfying P_{MVBA}, then each honest party outputs a value.
- *External validity.* If an honest party outputs v, then v is valid for P_{MVBA}, i.e., $P_{\mathsf{MVBA}}(v) = 1$.

KZG commitment [14] 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.

- $\{pp\} \leftarrow \mathsf{KZG.Setup}(t, 1^\kappa)$: this algorithm sets up the public parameters for the commitments. It takes as inputs a degree bound t and a security parameter κ in unary form. The output is public parameters *pp*. We sometimes omit *pp* for simplicity.
- $C_\phi \leftarrow \mathsf{KZG.Commit}(\phi(x), pp)$: this algorithm commits to a polynomial. It takes as inputs a polynomial $\phi(x) \in$

1. This property is splitted into consistency and totality in [17].

$\mathbb{Z}_p[x]$ and public parameters pp. The output is a commitment C_ϕ.

- $\langle \phi(i), w_{\phi(i)} \rangle \leftarrow \mathsf{KZG.CreateWitness}(\phi(x), i, pp)$: this algorithm creates a witness for a polynomial evaluation. It takes as inputs a polynomial $\phi(x)$, an index i, and public parameters pp. The output is an evaluation $\phi(i)$ and a witness $w_{\phi(i)}$.
- $0/1 \leftarrow \mathsf{KZG.VerifyEval}(C_\phi, i, v, w_{\phi(i)}, pp)$: this algorithm verifies a polynomial evaluation. It takes as inputs a commitment C_ϕ, an index i, an evaluation v, a witness $w_{\phi(i)}$, and public parameters pp. It outputs 1 iff $v = \phi(i)$.

The KZG scheme satisfies the following properties.

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

Threshold signature [20] allows a quorum of parties to construct a full signature jointly. It consists of five algorithms: TS.KeyGen, TS.SigShare, TS.VerifySh, TS.Combine, and TS.Verify.

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

A threshold signature scheme satisfies the following properties.

- *Unforgeability.* Given t corrupted parties, a computationally bounded adversary cannot forge a valid full signature of of any unqueried message m.
- *Robustness.* In the presence of an adversary who corrupts at most t parties, every honest party eventually gets a valid full signature.

3. The DyCAPS Scheme

3.1. Definition of DPSS

A typical secret sharing scheme consists of two protocols, sharing and reconstruction. We add a handoff protocol to achieve the proactivization of secret shares and support dynamic committees, 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.*

- $\{\langle s_i, \pi_i \rangle_{P_i \in \mathcal{C}}\} \leftarrow \mathsf{DPSS.Share}(t, n, s, 1^\kappa)$*: this protocol shares the secret among the participants. It takes as inputs a threshold t, a committee size n, a secret value s, and a security parameter κ in unary form. The output is a set of share-proof tuples* $\{\langle s_i, \pi_i \rangle_{P_i \in \mathcal{C}}\}$*,*
- $\{\langle s'_j, \pi'_j \rangle_{P_j^{e+1} \in \mathcal{C}^{e+1}}\} \leftarrow \mathsf{DPSS.Handoff}(\{\langle s_i, \pi_i \rangle_{P_i^e \in \mathcal{C}^e}\})$*: this protocol allows the new committee $\mathcal{C}^{e+1}$ to obtain refreshed secret shares from the old committee $\mathcal{C}^e$. The input is old share-proof tuples* $\{\langle s_i, \pi_i \rangle_{P_i^e \in \mathcal{C}^e}\}$*, and the output is refreshed tuples* $\{\langle s'_j, \pi'_j \rangle_{P_j^{e+1} \in \mathcal{C}^{e+1}}\}$*.*
- $v \leftarrow \mathsf{DPSS.Recon}(\{\langle s_i, \pi_i \rangle_{i \in I}\})$*: this protocol reconstructs the secret. It takes as inputs at least $t+1$ valid share-proof tuples* $\{\langle s_i, \pi_i \rangle_{i \in I}\}$*, where I is an index set and $|I| > t$. The output is a reconstructed secret v.*

There may be different versions of DPSS.Share and DPSS.Recon, depending on the application scenarios. For example, if a client uses DPSS to store a long-term secret, it trivially serves as the dealer to distribute and reconstruct the secret. If a committee wants to jointly generate and maintain a secret key, a decentralized version of DPSS.Share is needed, and DPSS.Recon may become unnecessary since the secret will never be restored due to privacy concerns.

An asynchronous DPSS scheme satisfies the following properties. For ease of expression, we assume $n_e = n_{e+1} = n$, $t_e = t_{e+1} = t$, and $n = 3t+1$ for all $e \in \mathbb{N}^*$. Besides, the following properties refer to the dealer-based DPSS.Share.

- *Termination.* 1) If the dealer is honest, then all honest parties terminate DPSS.Share. 2) If an honest party terminates DPSS.Share, then all honest parties terminate DPSS.Share. 3) If at least $n - t$ honest parties in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ invoke DPSS.Handoff, respectively, then all honest parties terminate DPSS.Handoff. 4) If all honest parties invoke DPSS.Recon and all of them have terminated DPSS.Share or DPSS.Handoff, then all honest parties terminate DPSS.Recon.
- *Completeness.* If an honest party obtains a valid share from DPSS.Share, then each honest party obtains a valid share from DPSS.Share.

- *Correctness.* If an honest dealer inputs s to DPSS.Share and v is the output of DPSS.Recon, then $v = s$. An arbitrary number of executions of DPSS.Handoff are allowed before DPSS.Recon.
- *Secrecy.* An adversary gains no advantage in extracting the secret s than random sampling.

3.2. DyCAPS Overview

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

DyCAPS.Share is derived from eAVSS-SC [21], which ensures that every honest party holds a valid secret share at the end of this protocol. DyCAPS.Handoff first communicates the shares among two committees, and then it utilizes the building blocks in Section 2.3 to ensure all honest parties receive consistent shares of a common random polynomial, which is used to refresh the shares. DyCAPS.Recon collects the latest shares from the committee members and recovers the secret.

In the rest of this section, we focus on DyCAPS.Handoff, which is invoked constantly. The other two protocols are delayed to Appendix A and Appendix B.

We use $\langle t, 2t\rangle$-degree bivariant polynomials and adopt the dimension-switching technique [6] to prevent the mobile adversary. The reconstruction threshold 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 sharing polynomial $B(x, y)$, where $B(0, 0) = s$. In the normal state, $t + 1$ full shares, $B(*, y)$, are needed to deal with the inquiries, e.g., generating a signature or decrypting a ciphertext. In turn, the reduced shares, $B(x, *)$, are temporarily used during the handoff, where $2t + 1$ of them are needed for the inquiries.

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 restore refreshed full shares. These phases are referred to as ShareReduce, Proactivize, and ShareDist, respectively, in CHURP [6]. On this basis, we introduce an additional phase named Prepare in DyCAPS, leaving space for selecting new committees and miscellaneous pre-computations. The four phases of DyCAPS.Handoff are shown in Figure 2. Throughout this paper, we refer to $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ as the old and new committees, respetively. Similarly, P_i^e and P_i^{e+1} are called old and new parties, respectively. Note that "old" and "new" refers to the epoch, so P_i^e and P_i^{e+1} might be the same entity.

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, full shares are converted to reduced shares to withstand the mobile adversary. The old parties initially hold $2t$-degree polynomials as their full shares. Then, each old party sends a point on its full share to every new party, who 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 propose their local randomness, i.e., bivariate polynomials, and agree on a candidate set $\mathcal{Q}$. The randomness from the members in $\mathcal{Q}$ comprises a common random polynomial. Each party obtains a share of this polynomial, which is 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, a new committee is selected, and public parameters are transferred to it. We do not restrict the relationship between the new and old committee members, but we do have a limit on the new size and threshold (see Section 6).

After the committee selection, the parties in both old and new committees establish P2P channels with each other. Once a channel is established, each old party transfers the public parameters to new parties, including the commitment public key cpk and the commitments to the reduced shares. The new parties confirm these parameters by $t + 1$ consistent messages. The new committee also calls TS.KeyGen$(t, n, 1^\kappa)$ to generate key pairs for the threshold signature scheme.

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.

3.4. Share Reduction

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

Each party P_i^e has at least $2t + 1$ commitments $C_{B(x,*)}$ and witnesses[2] $w_{B(i,*)}$ from either DyCAPS.Share or the prior DyCAPS.Handoff. With these elements, P_i^e may interpolate any other commitments and witnesses due to the homomorphism of KZG commitments. For example, given $B(x, j) = \sum_{\ell\in[2t+1]} \lambda_{\ell,j} B(x, \ell)$, we have $C_{B(x,j)} =$

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

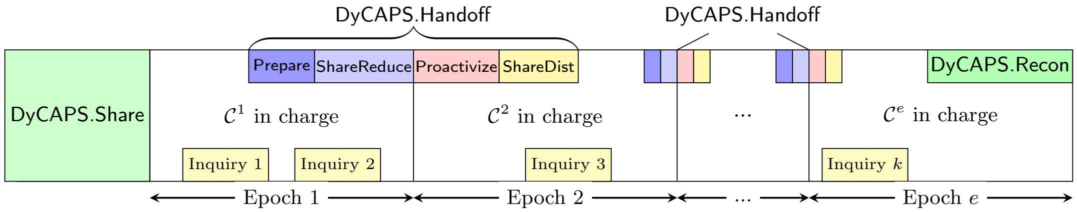

Figure 1: Life cycle of DyCAPS. DyCAPS.Share is invoked at first, and then DyCAPS.Handoff is executed repeatedly. DyCAPS.Recon is called at the end of the life cycle, if necessary. Inquiries are processed regardless of the handoff.

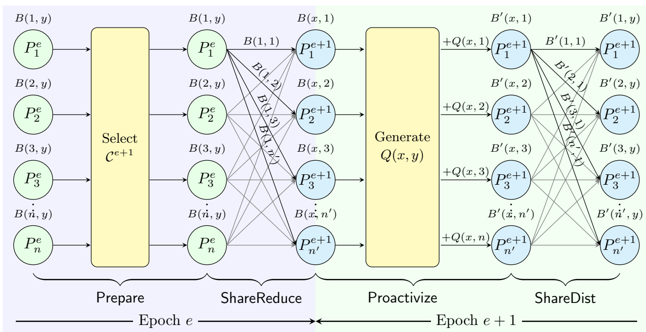

Figure 2: Overview of DyCAPS.Handoff. The polynomial above each party refers to the share it currently holds.

ShareReduce

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

Upon invocation by P_i^{e+1} **do** ▷ Interpolate
  Upon receiving $t+1$ valid $\langle \textsc{Reduce}, C_{B(x,i)}, B(*,i), w_{B(*,i)} \rangle$ **do**
    Interpolate $B(x,i)$
    Enter the Proactivize phase

Figure 3: Procedures of ShareReduce.

$\prod_{\ell \in [2t+1]} C_{B(x,\ell)}^{\lambda_{\ell,j}}$ 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 specific procedures of ShareReduce involve both old and new committees, as stated in the following.

Reduce. Each party P_i^e in the old committee sends a message $\langle \textsc{Reduce}, C_{B(x,j)}, B(i,j), w_{B(i,j)} \rangle$ to every new party P_j^{e+1}, where $C_{B(x,j)}$ and $w_{B(i,j)}$ are interpolated from the aforementioned $2t+1$ commitments and witnesses. Afterward, P_i^e erases its memory and goes offline.

Interpolate. Each party P_i^{e+1} in the new committee waits for $t+1$ valid $\textsc{Reduce}$ 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 $Q(0,0) = 0$. From a high level, the sharing polynomial is refreshed as $B'(x,y) = B(x,y) + Q(x,y)$. This phase only involves the new commitee $\mathcal{C}^{e+1}$, and each new party obtains a random share $Q(x,*)$ and adds it to the reduced share $B(x,*)$.

To generate such a polynomial $Q(x,y)$, each party collects randomness from the others and agrees on a candidate set $\mathcal{Q}$. The randomness proposed by the members in $\mathcal{Q}$ is used to compute $Q(x,y)$. There are four requirements for the joint generation:

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

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

Meeting the above requirements is not hard in a non-asynchronous network, but it becomes challenging when faced with asynchrony. We illustrate this observation via 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, where a timeout exists. In this case, we may decide the candidate set $\mathcal{Q}$ by verifiable challenges.

Specifically, we let each party P_i^{e+1} initialize $\mathcal{Q}$ as $\mathcal{C}^{e+1}$ and generate a random $\langle t, 2t \rangle$-degree polynomial $Q_i(x,y)$, such that $Q_i(0,0) = 0$. Then, each P_i^{e+1} invokes an RBC instance to broadcast n encrypted polynomials[3] $\mathsf{Enc}_j(Q_i(x,j))$, where $j \in [n]$, along with n commitments to these polynomials.

Each P_j^{e+1} waits for the outputs of these n RBC instances and decrypts polynomials $Q_*(x,j)$. An honest party

3. $\mathsf{Enc}_j(m)$ encrypts a message m by P_j's encryption public key epk_j.

Proactivize

Upon invocation by P_i^{e+1} with input InitProactivize **do**
  Upon receiving InitProactivize from P_i^{e+1} **do** ▷ Init
    $\pi_i \leftarrow \emptyset$
    $\mathrm{FLG}_{\mathrm{com}}[1, ..., n] \leftarrow \{0, ..., 0\}$
    $\mathrm{FLG}_{\mathrm{rec}}[1, ..., n] \leftarrow \{0, ..., 0\}$
    $\mathcal{S}_{\mathrm{rec}}[1, ..., n] \leftarrow \{\emptyset, ..., \emptyset\}$
    $\mathcal{S}_{\sigma}[1, ..., n] \leftarrow \{\emptyset, ..., \emptyset\}$
    $V_i[1, ..., n] \leftarrow \{\emptyset, ..., \emptyset\}$
    Generate a $2t$-degree polynomial $F_i(y)$, where $F_i(0) = 0$
    For each $\ell \in [2t+1]$ **do**
      Generate a t-degree polynomial $Q_i(x, \ell)$, where $Q_i(0, \ell) = F_i(\ell)$
    Send Commit to P_i^{e+1}
    Send Reshare to P_i^{e+1}

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

  Upon receiving $\langle \textsc{Com}, \pi_j \rangle$ from $\mathsf{RBC}_{1,j}$ **do** ▷ Verify
    Parse π_j as $\langle \ell, C_{Q_{j,k}}, C_{Z_{j,\ell}}, w_{Z_{j,\ell}(0)}, g^{F_j(\ell)} \rangle_{\ell \in [2t+1]}$
    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*
      Discard this message and revert
    For each $\ell \in [2t+1]$ **do**
      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**
        Discard this message and revert
    For each $P_\ell^{e+1} \in \mathcal{C}^{e+1}$ **do**
      $C_{Q_{j,\ell}} \leftarrow \prod_{m=1}^{2t+1} C_{Q_{j,m}}^{\lambda_{m,\ell}^{2t}}$ // *$\lambda_{m,\ell}^{2t}$ is the Lagrange coefficient*
    $\mathrm{FLG}_{\mathrm{com}}[j] \leftarrow 1$

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

  Upon receiving $\langle \textsc{Reshare}, \{Q_j(i, \ell), w_{Q_j(i,\ell)}\}_{\ell \in [2t+1]} \rangle$ from P_j^{e+1} **do** ▷ Vote
    Upon $\mathrm{FLG}_{\mathrm{com}}[j] = 1$ **then**
      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**
        $\sigma_{j,i}^* \leftarrow \mathsf{TS.SigShare}(j, tsk_i)$
      For each $P_\ell^{e+1} \in \mathcal{C}^{e+1}$ **do**
        $Q_j(i, \ell) \leftarrow \sum_{m=1}^{2t+1} \lambda_{m,\ell}^{2t} Q_j(i, m)$
        $w_{Q_j(i,\ell)} \leftarrow \prod_{m=1}^{2t+1} w_{Q_j(i,m)}^{\lambda_{m,\ell}^{2t}}$
        // *$\lambda_{m,\ell}^{2t}$ is the Lagrange coefficient*
        Send $\langle \textsc{Recover}, j, Q_j(i, \ell), w_{Q_j(i,\ell)}, \sigma_{j,i}^* \rangle$ privately to P_ℓ^{e+1}

  Upon receiving $\langle \textsc{Recover}, k, Q_k(j, i), w_{Q_k(j,i)}, \sigma_{k,j}^* \rangle$ from P_j^{e+1} **do** ▷ Recover
    Upon $\mathrm{FLG}_{\mathrm{com}}[k] = 1 \wedge \mathsf{TS.VerifySh}(k, \sigma_{k,j}^*) = 1$ **then**
      If $\mathsf{KZG.VerifyEval}(C_{Q_{k,i}}, j, Q_k(j, i), w_{Q_k(j,i)}) = 1$ **then**
        $\mathcal{S}_{\mathrm{rec}}[k] \leftarrow \mathcal{S}_{\mathrm{rec}}[k] \cup \langle j, Q_k(j, i) \rangle$
        If $|\mathcal{S}_{\mathrm{rec}}[k]| \geq t + 1$ **then**
          Interpolate t-degree $Q_k(x, i)$ from $\mathcal{S}_{\mathrm{rec}}[k]$
          $\mathrm{FLG}_{\mathrm{rec}}[k] \leftarrow 1$
        $\mathcal{S}_\sigma[k] \leftarrow \mathcal{S}_\sigma[k] \cup \langle j, \sigma_{k,j}^* \rangle$
        If $|\mathcal{S}_\sigma[k]| \geq 2t + 1$ **then**
          $\sigma_k \leftarrow \mathsf{TS.Combine}(k, \{\sigma_{k,j}^*\}_{\langle j, \sigma_{k,j}^* \rangle \in \mathcal{S}_\sigma[k]})$
          $V_i[k] \leftarrow \langle k, \sigma_k \rangle$

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

  Upon reiceiving $\langle \textsc{MVBA.Out}, \widetilde{V} \rangle$ from MVBA **do** ▷ Refresh
    $\mathcal{Q} \leftarrow \{P_j^{e+1} | \langle j, \sigma_j \rangle \in \widetilde{V}\}$
    Upon $\mathrm{FLG}_{\mathrm{rec}}[j] = 1$ for all $\langle j, \sigma_j \rangle \in \widetilde{V}$ **do**
      $Q(x, i) \leftarrow \sum_{P_j^{e+1} \in \mathcal{Q}} Q_j(x, i)$
      $B'(x, i) \leftarrow B(x, i) + Q(x, i)$
      For each $P_\ell^{e+1} \in \mathcal{C}^{e+1}$ **do**
        $C_{Q(x,\ell)} \leftarrow \prod_{P_j^{e+1} \in \mathcal{Q}} C_{Q_{j,\ell}}$
      Enter the ShareDist phase

Figure 4: Procedures of Proactivize.

raises a verifiable challenge if any decrypted polynomial is invalid. Depending on the verification results, either the challenger or the challenged party will be identified as malicious and excluded from $\mathcal{Q}$. Finally, each P_i^{e+1} computes the 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 eventually agree on $\mathcal{Q}$ and terminate.
2) The verifiable accusation procedure ensures that all honest parties receive valid information from parties in $\mathcal{Q}$ via RBC instances.
3) All honest parties stay included in $\mathcal{Q}$ even faced with malicious challengers.
4) The adversary has at most $2t$ polynomials $Q(x, *)$, which are insufficient to interpolate $Q(x, i)$, if P_i^{e+1} is honest.

This strawman scheme is straightforward, but the n RBC instances consume $O(\kappa n^4)$ bits of communication. CHURP [6] reduces the input size of each RBC instance to $O(\kappa n)$ by dividing the generation of $Q(x, y)$ into two steps, each with $2t + 1$ RBC instances. However, the challenge procedure is still required, which is not applicable in an asynchronous network—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 voting to avoid the challenge procedure. The voting results are decided by an MVBA instance, which ensures the agreement of the candidate set $\mathcal{Q}$.

Similar to Strawman I, we require each party to generate and share a local bivariate polynomial. However, we no longer need to reliably broadcast the encrypted messages because there are no challenges 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 $\ell \in [2t+1]$. These commitments are sufficient to derive the commitments to any other $Q_i(x, j)$, where $j \in [n] \backslash [2t+1]$. P_i^{e+1} also sends a polynomial $Q_i(x, \ell)$ to each P_ℓ^{e+1}.

Upon receiving 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, from which all honest parties obtain the same set $\widetilde{V}$, such that $|\widetilde{V}| = t+1$. 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 termination of MVBA and the condition $|\widetilde{V}| = t+1$.

However, a malicious party P_m^{e+1} may get included in $\mathcal{Q}$ if it obtains $2t+1$ votes. In the worst case, only $t+1$ honest parties obtain $Q_m(x,*)$ and vote for P_m^{e+1}, whereas the other honest parties receive no information from P_m^{e+1}. These $t+1$ polynomials are insufficient to recover the other $Q_m(x,*)$, whose y-dimension degree is $2t$. Therefore, some honest parties may not obtain $Q_m(x,*)$, so they cannot compute $Q(x,*)$. 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 even if malicious parties are included in $\mathcal{Q}$.

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

The procedures of $\mathsf{Proactivize}$ are described in Figure 4. We also present the message flows of this phase in Figure 5.

Init. Firstly, each party P_i^{e+1} initializes several empty sets, including a commitment 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 $F_i(0) = 0$. Finally, P_i^{e+1} reshares $F_i(y)$ via $2t{+}1$ random polynomials $Q_i(x,\ell)$, where $Q_i(0,\ell) = F_i(\ell)$, $\ell \in [2t+1]$, and $Q_i(x,\ell)$ is of t degree.

Commit. Each party P_i^{e+1} generates a commitment set $\pi_i = \langle \ell, C_{Q_{i,\ell}}, C_{Z_{i,\ell}}, w_{Z_{i,\ell}(0)}, g^{F_i(\ell)} \rangle_{\ell \in [2t+1]}$, where $C_{Q_{i,\ell}}$ and $C_{Z_{i,\ell}}$ are 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 for $Z_{i,\ell}(0) = 0$, and $g^{F_i(\ell)}$ is the commitment to $F_i(\ell)$. Finally, P_i^{e+1} broadcasts $\langle \textsc{Com}, \pi_i \rangle$ via $\mathsf{RBC}_{1,i}$.

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

Reshare. P_i^{e+1} sends $\langle \textsc{Reshare}, \{Q_i(j,\ell), w_{Q_i(j,\ell)}\}_{\ell \in [2t+1]} \rangle$ 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* step above.

Vote. Upon receiving a $\textsc{Reshare}$ message from P_j^{e+1}, party P_i^{e+1} first verifies it *w.r.t.* the commitment set π_j, which is delivered from $\mathsf{RBC}_{1,j}$. Then, it formulates a signature share $\sigma^*_{j,i} = \mathsf{TS.SigShare}(j, tsk_i)$ as a vote for P_j^{e+1}. Afterward, the contents in the $\textsc{Reshare}$ 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 $\textsc{Recover}$ message. The vote $\sigma^*_{j,i}$ is also included in this message.

Recover. Upon receiving $t+1$ valid $\textsc{Recover}$ messages with the same index k, such that $\mathsf{TS.VerifySh}(k, \sigma^*_{k,j}) = 1$ 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 $\sigma_k = \mathsf{TS.Combine}(k, \{\sigma^*_{k,j}\}_{j \in I})$, 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, party 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 $\mathcal{S}_{B'}$.

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

Distribute. P_i^{e+1} sends $\langle \textsc{ShareDist}, B'(j,i), w_{B'(j,i)} \rangle$ to each P_j^{e+1}, where $w_{B'(j,i)}$ is the witness for $B'(j,i)$. This step is executed concurrently with the *Commit* step above.

Verify. Upon receiving the $\textsc{NewCom}$ 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} verifies $C_{B'(x,j)} = C_{B(x,j)} C_{Q(x,j)}$, which indicates that $B'(x,j) = B(x,j) + Q(x,j)$. If the verification fails, this $\textsc{NewCom}$ message will be ignored.

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

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

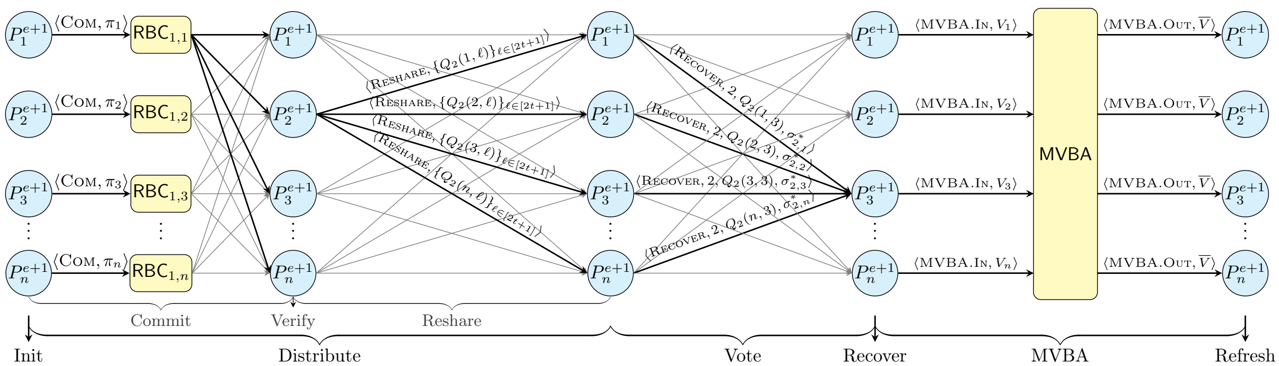

Figure 5: Message flow of Proactivize within epoch $e+1$. In the Vote stage, the emphasized RECOVER messages received by P_3^{e+1} refer to the responses to P_2^{e+1}'s RESHARE messages. The witnesses are ommitted for cleարity of expression.

ShareDist

1: **Upon** invocation by P_i^{e+1} with input INITDIST **do**
2: **Upon** receiving INITDIST from P_i^{e+1} **do** ▷ Init
3: $\mathcal{S}_{\text{com}} \leftarrow \emptyset$
4: $\mathcal{S}_{B'} \leftarrow \emptyset$
5: Send COMMITNEW to P_i^{e+1}
6: Send DISTRIBUTE to P_i^{e+1}

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

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

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

17: **Upon** receiving $\langle \text{SHAREDIST}, B'(i,j), w_{B'(i,j)} \rangle$ from P_j^{e+1} **do** ▷ Interpolate
18: **Upon** $\langle j, C_{B'(x,j)} \rangle \in \mathcal{S}_{\text{com}}$ **then**
19: **If** $\mathsf{KZG.VerifyEval}(C_{B'(x,j)}, i, B'(i,j), w_{B'(i,j)}) = 1$ **then**
20: $\mathcal{S}_{B'} \leftarrow \mathcal{S}_{B'} \cup \langle j, C_{B'(x,j)}, B'(i,j), w_{B'(i,j)} \rangle$
21: **If** $|\mathcal{S}_{B'}| \geq 2t+1$ **then**
22: Interpolate $2t$-dgree $B'(i,y)$ from $\mathcal{S}_{B'}$
23: Multicast SUCCESS

24: **Upon** having sent SUCCESS and receiving $2t+1$ SUCCESS **do** ▷ Success
25: 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 A and Appendix B, respectively.

4.1. Security Analysis

The security of DyCAPS.Handoff involves termination, correctness, and secrecy. For simplicity of expression, we continue to assume $n_e = n_{e+1} = n$ and $t_e = t_{e+1} = t$. Without loss of generality, we denote the malicious and honest parties as $\{P_m^*\}_{m\in[t]}$ and $\{P_h^*\}_{h\in[n]\setminus[t]}$, respectively.

Termination. The termination of DyCAPS consists of four statements (see Section 3.1). Here, we prove the third statement by four lemmas. The remaining proofs are displayed in Appendix A and Appendix B, as they involve the specification of DyCAPS.Share and DyCAPS.Recon.

Lemma 1. *If at least $n-t$ honest parties from $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ invoke DyCAPS.Handoff, respectively, then all honest parties in $\mathcal{C}^e$ terminate DyCAPS.Handoff.*

Proof. The old committee $\mathcal{C}^e$ is only active in the Prepare and ShareReduce phases.

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

In ShareReduce, the honest old parties only need to send messages to the new committee. We now prove that every honest party in $\mathcal{C}^e$ has enough information to generate the REDUCE messages (line 3-4, Figure 3).

We start from $e=1$. Note that DyCAPS.Handoff is called after DyCAPS.Share, so all honest parties have terminated DyCAPS.Share. Hence, an honest party P_i^e must have delivered a commitment set π (line 46, Figure 13) and at least $2t+1$ DISTRIBUTE messages (line 50). The required commitments $C_{B(x,*)}$ and witnesses $w_{B(i,*)}$ are in π (line 14) and DISTRIBUTE messages, respectively (line 44).

For $e \geq 2$, the commitments and witnesses are generated in ShareDist (line 8 and line 17, Figure 6), where the commitments are broadcast via RBC, and the witnesses are included in $2t+1$ valid SHAREDIST messages. Combined with Lemma 3 and Lemma 4 below, the honest parties in $\mathcal{C}^2$ will terminate DyCAPS.Handoff when $e=1$. Hence, the honest parties in $\mathcal{C}^2$ have enough information to generate the REDUCE messages from the prior DyCAPS.Handoff.

By mathematical induction, the honest parties in $\mathcal{C}^e$ terminate DyCAPS.Handoff for all $e \geq 1$. □

Lemma 2. *If at least $n-t$ honest parties from $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ invoke DyCAPS.Handoff, respectively, then all honest parties in $\mathcal{C}^{e+1}$ terminate Prepare and ShareReduce.*

Proof. In Prepare, each honest party in $\mathcal{C}^{e+1}$ is guaranteed to connect to at least $2(n-t)$ parties and deliver the public parameters from any $t+1$ honest old parties. Afterward, it enters the ShareReduce phase.

In ShareReduce, each honest party in $\mathcal{C}^{e+1}$ receives at least $n-t$ valid REDUCE messages. These messages are sufficient for an honest party to interpolate the reduced share $B(x,*)$ and terminate ShareReduce. □

Lemma 3. *If at least $n-t$ honest parties from $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ invoke DyCAPS.Handoff, respectively, then all honest parties in $\mathcal{C}^{e+1}$ terminate Proactivize.*

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

The worst situation for P_i^{e+1} is that the corrupted parties do not send any messages to it. In this case, P_i^{e+1} only receives n COM from RBC (line 22, Figure 4) and $n-t$ RESHARE messages from the honest parties (line 37). Then the honest parties will send RECOVER messages to each other (line 45). Hence, P_i^{e+1} obtains $n-t$ polynomials $Q_h(x,i)$ and full signatures (line 51-55), where $\{P_h^{e+1}\}_{h\in[n]\setminus[t]}$ are honest parties. Therefore, P_i^{e+1} is guaranteed to form a valid proposal V_i as the input to the MVBA instance (line 57), even without any private message from the corrupted parties.

Similarly, every honest party will have a valid proposal and invoke the MVBA instance. Due to the termination of MVBA, each honest party obtains an output $\widetilde{V}$ and candidate set $\mathcal{Q}$ (line 61).

After the termination of MVBA, every honest party can 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, as mentioned above, P_i^{e+1} has obtained $Q_h(x,i)$ for all P_h^{e+1}, $h \in [n]\setminus[t]$, to compute $Q(x,i) = \sum_{P_h^{e+1}\in\mathcal{Q}} Q_h(x,i)$.

Case 2: If any malicious P_m^{e+1} is included in $\mathcal{Q}$, then in the worst case, P_i^{e+1} receives no private message from P_m^{e+1}. However, $P_m^{e+1} \in \mathcal{Q}$ means $\langle m, \sigma_m\rangle \in \widetilde{V}$, where σ_m corresponds to $2t+1$ signature shares (line 55). Hence, at least $t+1$ honest parties have voted for P_m^{e+1} (line 45). These parties have received valid COM and RESHARE messages from $\mathsf{RBC}_{1,m}$ and P_m^{e+1}, respectively. Due to the agreement of RBC, P_i^{e+1} eventually receives the same COM message from $\mathsf{RBC}_{1,m}$. Besides, the $t+1$ honest parties receiving RESHARE messages from P_m^{e+1} will distribute the evaluations $Q_m(*,i)$ via the RECOVER messages (line 45). Consequently, P_i^{e+1} receives $t+1$ points to interpolate $Q_m(x,i)$, which is then used to compute $Q(x,i)$.

In either case, P_i^{e+1} obtains $Q(x,i)$, refreshs the reduced shares, and terminates without directly receiving messages from the corrupted parties. □

Lemma 4. *If at least $n-t$ honest parties from $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ invoke DyCAPS.Handoff, respectively, then all honest parties in $\mathcal{C}^{e+1}$ terminate ShareDist.*

Proof. Due to Lemma 3, all honest parties in $\mathcal{C}^{e+1}$ have refreshed their reduced shares in Proactivize. Therefore, in the ShareDist phase, each P_i^{e+1} receives at least $n-t$ valid SHAREDIST messages to interpolate its refreshed full share $B'(*,y)$. Similarly, each honest party in $\mathcal{C}^{e+1}$ obtains at least $2t+1$ SUCCESS messages and terminates ShareDist. □

Theorem 5 (Termination of DyCAPS.Handoff)**.** *If at least $n-t$ honest parties in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ invoke DyCAPS.Handoff, respectively, then all honest parties in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ terminate DyCAPS.Handoff.*

Proof. By Lemma 1, the honest parties in $\mathcal{C}^e$ terminate DyCAPS.Handoff. By Lemma 2, Lemma 3, and Lemma 4, the honest parties in $\mathcal{C}^{e+1}$ also terminate. Combining these lemmas, we conclude that all honest parties in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$ terminate 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 by Lemma 6, Lemma 7, and Lemma 8, respectively.

Lemma 6. *The secret s stays invariant during ShareReduce, in the presence of a mobile adversary corrupting at most t parties in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$, respectively.*

Proof. In ShareReduce, the new committee members wait for enough REDUCE messages from the old committee and interpolate the reduced shares. An honest P_i^{e+1} accepts $\langle C_B(x,i), B(*,i), w_{B(*,i)}\rangle$ iff it has received at least $t+1$ messages containing the same commitment $C_B(x,i)$ and the evaluations $B(*,i)$ all pass the KZG verifications (line 9, Figure 3). Hence, t corrupted parties cannot convince an honest party with a different commitment.

By Lemma 2, each honest P_i^{e+1} interpolates $B(x,i)$, whose commitment is attested by $t+1$ REDUCE 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 Proactivize, in the presence of a mobile adversary corrupting at most t parties in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$, respectively.*

Proof. In Proactivize, the reduced shares are refreshed as $B'(x,*) = B(x,*) + Q(x,*)$. We have proved by Lemma 3 that each party receives 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 $x, y \in [n]$.

Due to the agreement of MVBA, the honest parties receive the same output $\widetilde{V}$, which leads to the same candidate set $\mathcal{Q}$ (line 61, Figure 4). Hence, to prove the consistency of $Q(x,*) = \sum_{P_j\in\mathcal{Q}} Q_j(x,*)$, we only need to show that the shares $Q_m(x,*)$ generated by malicious parties P_m^{e+1} are consistently interpolated by honest parties, where $P_m^{e+1} \in \mathcal{Q}$.

Suppose the malicious party P_m^{e+1} proposes an illegal polynomial $Q_m^*(x,y)$. Due to the strong correctness of KZG commitments, the x-dimension degree of $Q_m^*(x,y)$ is bounded by t. Hence, only the y-dimension degree of

$Q^*_m(x,y)$ can be manipulated to exceed $2t$. However, P^{e+1}_m is only allowed to broadcast $2t+1$ commitments via $\mathsf{RBC}_{1,m}$ (line 15-21), and the other commitments are interpolated by the receivers (line 30). 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^{e+1}_m sends a point on $Q^*_m(x,y)$ which is invalid *w.r.t.* the commitments to $\hat{Q}_m(x,y)$, the receivers will not accept it (line 39). Therefore, the guaranteed outputs in Lemma 3 are actually the shares of $\hat{Q}_m(x,y)$. Hence, the honest parties will obtain consistent random shares from $P^{e+1}_m \in \mathcal{Q}$, and the common random polynomial $Q(x,y)$ is 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 $Q_i(0,0) = F_i(0) = 0$ (line 24), so we have $Q(0,0) = \sum_{P^{e+1}_i \in \mathcal{Q}} Q_i(0,0) = 0$.

Combining the above proofs, the secret $s = B'(0,0) = B(0,0) + Q(0,0)$ stays invariant, and each honest party obtains a new reduced share $B'(x,*)$ consistently. □

Lemma 8. *The secret s stays invariant during ShareDist, in the presence of a mobile adversary corrupting at most t parties in $\mathcal{C}^e$ and $\mathcal{C}^{e+1}$, respectively.*

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

Theorem 9 (Correctness of DyCAPS.Handoff)**.** *The secret s stays invariant during DyCAPS.Handoff, in 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. □

Secrecy. To prove the secrecy of DyCAPS.Handoff, we first prove by Lemma 10 that an adversary learns no information about a random share $Q(x,i)$ if P^{e+1}_i is not corrupted.

Lemma 10. *If P^{e+1}_i is an honest party, a computationally bounded adversary gains no advantage in extracting the random share $Q(x,i)$ than random sampling.*

Proof. An adversary has access to at most t random shares $Q(x,m)$ and $n\times t$ RESHARE messages, each containing $2t+1$ points $\{Q_i(m,\ell)\}$, where $i\in[n]$ and $m\in[t]$ (line 37, Figure 4). In the following, we prove that the adversary cannot obtain $Q(x,i)$ with the above information.

Firstly, since $Q(x,y)$ is of degree $\langle t,2t\rangle$ (see Lemma 7), the t polynomials $\{Q(x,m)\}_{m\in[t]}$ reveal no information about $Q(x,i)$.

Secondly, if the adversary wants to calculate $Q(x,i)$ from $\sum_{P^{e+1}_j\in\mathcal{Q}} Q_j(x,i)$, it needs to obtain $Q_j(x,i)$ for all $P^{e+1}_j \in \mathcal{Q}$. As $|\mathcal{Q}| = |\widetilde{V}| = t+1$ (line 59), at least one honest party is included in $\mathcal{Q}$. However, for any honest party $P^{e+1}_h \in \mathcal{Q}$, the adversary has at most t points on the t-degree polynomial $Q_h(x,i)$. Hence, the adversary gains no advantage on recovering $Q_h(x,i)$ than random sampling.

Finally, the t polynomials $\{Q(x,m)\}_{m\in[t]}$ and the $t\times n$ points on $Q_j(x,i)$, where $j\in[n]$, are independent, so their combination also reveals no information about $Q(x,i)$.

In conclusion, if the adversary does not corrupt P^{e+1}_i, it has no advantage on extracting the polynomial $Q(x,i)$ than random sampling. □

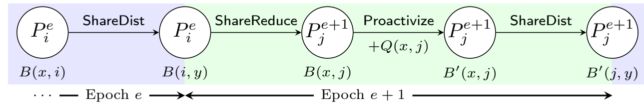

Figure 7: Shares held by P^e_i and P^{e+1}_j in adjacent epochs.

Theorem 11 (Secrecy of DyCAPS.Handoff)**.** *An adversary gains no advantage in extracting the secret s than random sampling during DyCAPS.Handoff.*

Proof. We depict the shares held by P^e_i and P^{e+1}_j in Figure 7. Specifically, at the end of epoch $e+1$, each P^e_i holds $B(x,i)$ and $B(i,y)$, and each P^{e+1}_j holds $B(x,j)$, $B'(x,j)$, and $B'(j,y)$.

By Lemma 10, the adversary cannot obtain the common random polynomial $Q(x,y)$. Since the refreshed polynomial is calculated as $B'(x,y) = B(x,y) + Q(x,y)$, the bivariant polynomials $B'(x,y)$ and $B(x,y)$ are independent in the adversary's view.

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. Since $B(x,y)$ has $(t+1)(2t+1)$ coefficients, these evaluations are insufficient to determine the free coefficient $s = B(0,0)$. Therefore, the adversary gains no extra information about the secret s. □

4.2. Performance Analysis

We evaluate the performance of DyCAPS.Handoff by communication complexity, which is measured in bits.

In Prepare, each old party sends the public parameters to the new committee. The communication is dominated by the $O(\kappa n)$-sized KZG parameters, which lead to $O(\kappa n^3)$ bits of communication.

In ShareReduce, an old party spreads n REDUCE messages, each containing three constant-sized elements. Therefore, the communication cost of this phase is $O(\kappa n^2)$ bits.

In Proactivize, communication only takes place within the new committee. Firstly, each party sends n $O(\kappa n)$-sized RESHARE messages to the others. Then, n RBC instances are invoked, each consuming $O(n|m| + \kappa n^2)$ bits of communication [16], where $|m| = \kappa n$ is the input size. Next, each party sends out n^2 $O(\kappa)$-sized RECOVER messages.

Finally, using sMVBA [19] of $O(n^2|m|+\kappa n^2)$ communication complexity[4], the MVBA procedure consumes $O(\kappa n^3)$ bits of communication. To sum up, the Proactivize phase consumes $O(\kappa n^3)$ bits of communication.

In ShareDist, 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 $O(\kappa n^3)$ communication complexity.

5. Implementation and Evaluation

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 [14] and BLS threshold signatures [26] as black boxes. The source code is public available[5].

The commitments and signatures are on an elliptic curve 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. Besides, we use SHA256 for hashing.

5.2. Evaluation

We deploy DyCAPS on 128 Amazon EC2 t2.medium instances from 8 regions. Every instance serves as a party. Experiments are conducted between two honest committees of the same size.

Communication cost. We first compare the concrete communication cost of DyCAPS with Yurek-DPSS [27], a concurrent work of ours that also achieves $O(\kappa n^3)$ communication complexity. Figure 8 demonstrates that the concrete cost of DyCAPS is around 3% of Yurek-DPSS. This is due to the heavy encryption and zero-knowledge proofs in Yurek-DPSS (around 10 KB per proof). Remarkably, Yurek-DPSS supports batch proactivization, occurring $O(\kappa n^2)$ amortized overhead, which will be discussed in Section 7.1.

Latency. We focus on the latency of DyCAPS.Handoff here, i.e., the average time for each new party to obtain the refreshed full shares. The handoff between the two smallest committees ($n = 4$) takes around 1.5 seconds, and when the committee is scaled to 64 members, the latency grows to approximately 280 seconds. As shown in Figure 9, the latency is dominated by the Proactivize phase.

To further identify the major bottleneck, we measure the step-by-step latency of Proactivize by executing the procedures sequentially[6]. The results are shown in Figure 10. We omit the statistics of $n = 4$ in this figure because they are too small compared to the others. The growth of latency is mainly caused by the $O(n^2)$ KZG verifications (pairings) in Verify and Vote. These two steps account for around 42% ($n = 4$) to 83% ($n = 64$) of the latency in Proactivize.

Throughput. Observe that DyCAPS.Handoff includes all gadgets of Dumbo2 [10]. Therefore, DyCAPS.Handoff may serve as a dynamic BFT protocol, where the transaction payloads are sent along with the commitments. We evaluate DyCAPS.Handoff and Dumbo2 with different payload sizes. Dumbo2 is implemented with the same building blocks as

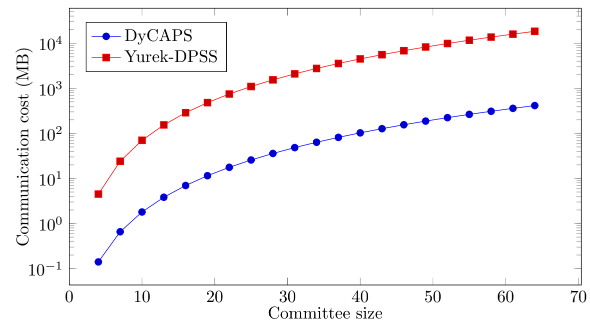

Figure 8: Concrete communication cost of DyCAPS and Yurek-DPSS [27] in log scale.

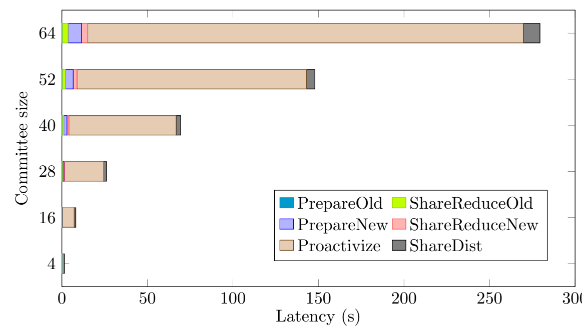

Figure 9: Latency of DyCAPS.Handoff. There is an overlap between the latencies of the old and new committees, which are accumulated here for simplicity.

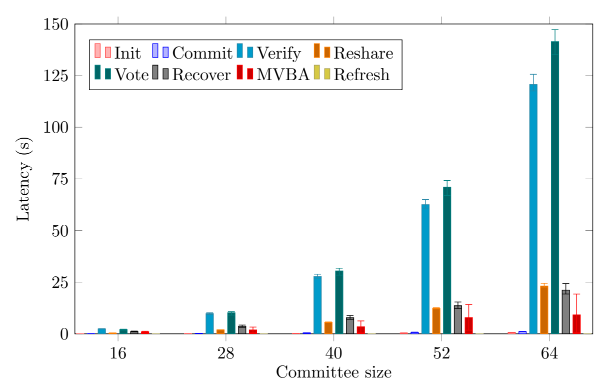

Figure 10: Step-by-step latency of the Proactivize phase.

4. The communication cost can be reduced to $O(n|m| + \kappa n^2)$ at the expense of additional rounds [22], but this will not influence the overall complexity of DyCAPS.

5. https://github.com/DyCAPSTeam/DyCAPS

6. Sequential execution consumes around 20% more seconds than concurrent execution.

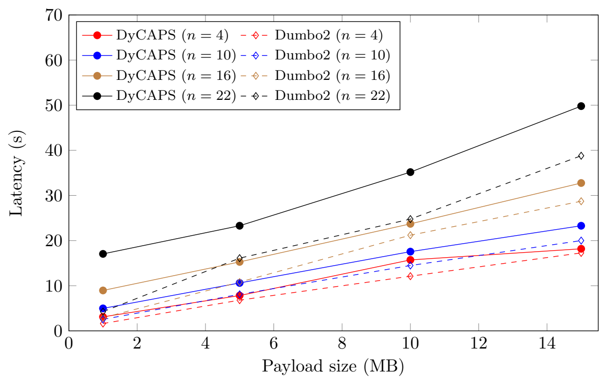

Figure 11: Latency of DyCAPS.Handoff and Dumbo2 with different payload sizes.

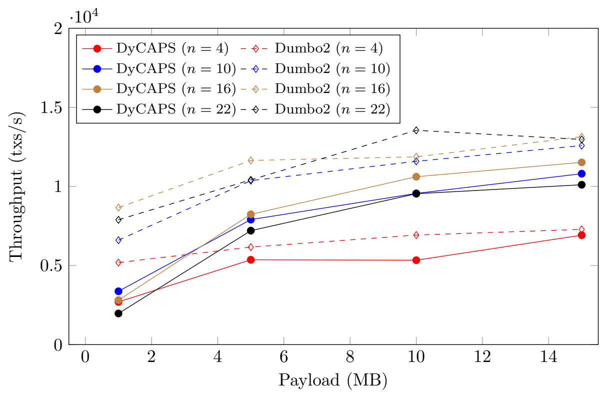

Figure 12: Throughput of DyCAPS.Handoff and Dumbo2.

those in DyCAPS. In both protocols, we set the output size $|\widetilde{V}| = t+1$, which may be configured up to $2t+1$ without influencing the security properties. Besides, we use $2t+1$ as the threshold of the signature in Dumbo2. The results are depicted in Figure 11 and Figure 12, respectively, where the transaction size is set as 250 bytes.

As the payload grows, the latency and throughput of DyCAPS.Handoff become comparable with Dumbo2 (see Figure 11 and Figure 12). Given a committee of 22 parties and a payload of 15 MB per party, the extra latency overhead of DyCAPS.Handoff is about 22% compared to Dumbo2. When n is small, this gap is even smaller, i.e., 12.3% at $n = 16$, 14.1% at $n = 10$, and 5% at $n = 4$. In conclusion, our implementation equips Dumbo2 with the functionality of proactivization, with a considerable latency overhead.

6. Change of Size and Threshold

6.1. Change of Size

Given a fixed threshold, the change of committee size is already taken into consideration 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

Increasing threshold. To increase the threshold from t to t', where $t' > t$, we need to raise the degree of the refreshed polynomial $B'(x,y)$ to $\langle t', 2t' \rangle$. 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 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 obtains 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 a $\langle t', 2t' \rangle$-degree $Q(x,y)$ and refresh $B_{\text{tmp}}(x,y)$ to $B'(x,y)$[7]. In this way, the adversary, who obtains $t + t' < 2t'$ reduced shares $B_{\text{tmp}}(x,*)$, 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. To reduce the threshold from t to $t' = t - d$, where $t > d > 0$, we follow prior schemes [6], [13] and introduce 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.

Specifically, 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) vote for the virtual parties, whose contributions are $Q_v(x,y) = 0$. In this way, the MVBA instance terminates 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. Generating such $Q(x,y)$ requires higher-degree KZG public parameters. We adopt the extended KZG scheme by Maram et al. [6].

7. Discussion

7.1. Related Work

Table 2 concludes the performance and properties of several related DPSS schemes.

Non-asynchronous DPSS. CHURP [6] and COBRA [7] are two state-of-the-art DPSS schemes in synchronous and partially synchronous networks, respectively. CHURP has a communication cost of $O(\kappa n^2)$ bits in the optimistic case. However, if any party misbehaves, CHURP falls into the pessimistic path and consumes $O(\kappa n^3)$ bits of communication[8]. COBRA achieves $O(\kappa n^3)$ bits of communication complexity, but its worst-case complexity grows to $O(\kappa n^4)$ due to continuous view-changes.

Schultz-MPSS [13] realizes DPSS with a communication cost of $O(\kappa n^4)$ bits. Although it is claimed to support asynchrony, its underlying network model has recently been classified as partially synchronous [9].

Asynchronous DPSS. Zhou et al. [8] propose the first asynchronous DPSS. However, their scheme has only theoretical value as it consumes exponential communication.

Research on asynchronous DPSS is revived recently. We have noticed two concurrent works, Shanrang [28] and Yurek-DPSS [27]. Shanrang uses Honeybadger [9] to deal with asynchrony, at a communication cost of $O(\kappa n^3 \log n)$ bits, and it only tolerates $t < n/4$ corrupted parties. Yurek-DPSS achieves the same asymptotic complexity as ours, but they focus on the amortized cost of refreshing multiple secrets. Their scheme uses public-key encryptions (PKE) and zero-knowledge (ZK) proofs, which significantly reduces the performance, as illustrated in Figure 10. The prover and verifier time of ZK proofs is also non-trivial. Therefore, our scheme is more efficient in some scenarios where only several secrets are refreshed, e.g., dynamic BFT [5]. When a large batch is used, Yurek-DPSS becomes more practical, as it is optimized for the batched setting.

Asynchronous complete secret sharing (ACSS). ACSS is first proposed by Patra et al. [30], where all honest parties are guaranteed to receive a valid and consistent share from the sharing protocol. The completeness property of a DPSS scheme is necessary. Otherwise, some honest parties may have no shares to be refreshed in the following handoff protocol. We design our ACSS protocol based on eAVSS-SC [21] (see Appendix A), which avoids expensive PKE and ZK proofs. Besides, Yurek-DPSS [27] uses ACSS [16], [31] during the handoff, which is the main bottleneck of performance, as discussed in Section 5.2.

Asynchronous distributed key generation (ADKG). The core of DPSS schemes is an ADKG protocol [32], [33], where common randomness is jointly generated and added to the original shares. DyCAPS has the same asymptotic communication complexity as several state-of-the-art ADKG schemes [16], [32], [34], but each party obtains a random polynomial instead of a random element.

8. We replace the bulletin board in [6] with RBC [16] to calculate the communication complexity.

7.2. Applications of DyCAPS

Flexible committees for blockchains. Most committee-based blockchains use BFT protocols [9], [10], [35] to order the transactions, where the BFT committee is usually fixed. Using DyCAPS, the committee management becomes more flexible. Adjusting memberships, size, and threshold may strengthen the long-term security of committee-based blockchains against a mobile adversary.

DyCAPS is also promising for proof-of-stake (PoS) blockchains [36], where the committee changes over time according to the stakes. Using DyCAPS, the PoS committee may maintain a consistent key pair 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. However, there are still open problems for DyCAPS to be adopted in PoS protocols, including ensuring the memory erasure of the old parties.

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 exposing the secret key. The secret shares may be kept by personal devices or in the cloud, where the shares are refreshed periodically, and the user may choose to replace some devices or cloud service providers.

Threshold cryptography as a service. As recently pionted out by Benhamouda et al. [12], threshold cryptographic services are attractive in many fields, including private cloud storage [7], [41], document certification, random beacons [32], [42], and cross-chain bridges [43]. Most scenarios above might encounter the demand of dynamic committee and the challenge of asynchrony in practice. DyCAPS takes a step to takle these problems, and may promote these services.

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. 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 into a dynamic setting, especially for blockchains, decentralized autonomous organizations, and threshold cryptographic services. DyCAPS is also attractive for personal use to manage the secret keys of users.

Acknowledgments

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

TABLE 2: Related DPSS schemes. The communication cost is calculated in bits.

Reference	Async.	Adversary	Threshold	Best-case[1] comm. cost	Worst-case comm. cost	Trusted setup	PKE required
Schultz-MPSS [13]	$\times$	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	$t < n/3$	$\exp(n)$	$\exp(n)$	$\times$	$\times$
Shanrang [28]	$\surd$	Mobile	$t < n/4$	$O(\kappa n^3 \log n)$	N/A	$\surd$	$\surd$
Yurek-DPSS [27]	$\surd$	Mobile	$t < n/3$	$O(\kappa n^3)$	$O(\kappa n^3)$	$\times$[2]	$\surd$
DyCAPS (this work)	$\surd$	Mobile	$t < n/3$	$O(\kappa n^3)$	$O(\kappa n^3)$	$\surd$	$\times$

[1] In the best case, all parties behave honestly. In the worst case, there are t corrupted parties behaving maliciously.
[2] Given no trusted setup, zero knowledge proofs in [27] may introduce a large constant factor. Besides, these proofs rely on random oracles [29].

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

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

[13] 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.

[14] 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.

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

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[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] 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

[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] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO 1986*, ser. LNCS, vol. 263. Springer, pp. 186–194.

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

[31] T. Yurek, L. Luo, J. Fairoze, A. Kate, and A. K. Miller, "hbacss: How to robustly share many secrets," in *NDSS 2022*, pp. 1–18.

[32] 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.

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

[34] S. Das, T. Yurek, Z. Xiang, A. K. Miller, L. Kokoris-Kogias, and L. Ren, "Practical asynchronous distributed key generation," in *SP 2022*, pp. 2518–2534.

[35] 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.

[36] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO 2017*, ser. LNCS, vol. 10401. Springer, pp. 357–388.

[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/

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

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

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

Appendix A. The Secret Sharing Protocol of DyCAPS

In this section, we present a leader-based asynchronous complete secret sharing (ACSS) protocol DyCAPS.Share. The existence of a dealer is necessary for many applications. For example, a dealer may delegate its secret key to a group of people to do threshold cryptographic operations. A dealer-free ACSS may be derived from the Proactivize phase in Section 3.5, by replacing the requirement of $F_i(0) = 0$ with $F_i(0) = s_i$, where s_i is randomly generated by P_i.

A.1. Details of Our ACSS

In DyCAPS.Share, a dealer P_d shares a secret s among a committee $\mathcal{C}$, which consists of n parties $\{P_i\}_{i\in[n]}$. As the dealer may simply withhold the messages to block the sharing, we require that if any honest party receives a valid share from DPSS.Share, then each honest party receives a valid share at the end of DyCAPS.Share. This is referred to as the *completeness* property [30] (see Section 3.1).

Before DyCAPS.Share, a trusted setup is required to initialize the public parameters of the KZG commitment scheme [14]. We further assume these parameters are available for all members. We slightly modify eAVSS-SC [21] to support a $\langle t, 2t\rangle$-degree bivariate sharing polynomial. The procedures are shown in Figure 13.

Init. The initialization procedures for the dealer P_d and the committee members $\{P_i\}_{i\in[n]}$ are 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 $\ell \in [2t+1]$. Each committee member P_i only needs to initialize an empty buffer $\mathcal{S}_{\text{full}}$ and a flag $\text{FLG}_{\text{ready}} = 0$.

Commit. To prove the correctness of *Init*, P_d sets $\pi = \{g^s, \langle \ell, C_{B_\ell}, C_{Z_\ell}, w_{Z_\ell(0)}, g^{F(\ell)}\rangle_{\ell\in[2t+1]}\}$, 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 for $Z_\ell(0) = 0$.

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

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 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 $\langle \textsc{Echo}, \pi\rangle$.

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 $\langle \textsc{Ready}, \pi', \textsc{Share}, B^*(i, \ell), w_{B^*(i,\ell)}\rangle$ 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 $\langle \textsc{Ready}, \pi', \textsc{noShare}\rangle$.

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 $\langle \textsc{Distribute}, B(\ell, i), w_{B(\ell,i)}\rangle$.

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.

The procedures above consumes $O(\kappa n^2)$ bits of communication in total.

A.2. Security Analysis

The termination of DyCAPS contains four statements. We prove the first two statements in this section, along with the proof of completeness and secrecy of DyCAPS.Share. The last statement and the proof of correctness is shown in Appendix B, as they both involve DyCAPS.Recon.

Theorem 12 (Termination of DyCAPS.Share)**.** *1) If the dealer is honest, then all honest parties terminate DyCAPS.Share. 2) If an honest party terminates DyCAPS.Share, then all honest parties terminate DyCAPS.Share.*

DyCAPS.Share

1: **Upon** invocation by P_d with input $\langle \text{INITSHARE}, s \rangle$ **do**
2: **Upon** receiving $\langle \text{INITSHARE}, s \rangle$ 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: $C_{B_\ell} \leftarrow \mathsf{KZG.Commit}(B(x, \ell))$
12: $C_{Z_\ell} \leftarrow \mathsf{KZG.Commit}(Z_\ell(x))$
13: $w_{Z_\ell(0)} \leftarrow \mathsf{KZG.CreateWitness}(Z_\ell(x), 0)$
14: $\pi \leftarrow \pi \cup \langle \ell, C_{B_\ell}, C_{Z_\ell}, w_{Z_\ell(0)}, g^{F(\ell)} \rangle$
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: $w_{B(i,\ell)} \leftarrow \mathsf{KZG.CreateWitness}(B(x, \ell), i)$
20: Send $\langle \text{SEND}, \pi, \{B(i, \ell), w_{B(i,\ell)}\}_{\ell \in [2t+1]} \rangle$ 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: $\text{FLG}_{\text{ready}} \leftarrow 0$

25: **Upon** receiving $\langle \text{SEND}, \pi', \{B(i, \ell), w_{B(i,\ell)}\}_{\ell \in [2t+1]} \rangle$ from P_d **do** ▷ Echo
26: Verify π' as line 24-28 in $\mathsf{Proactivize}$ // $\{g^{F(\ell)}\}_{\ell \in [2t+1]}$ *are verified w.r.t.* g^s
27: Verify $\{B(i, \ell), w_{B(i,\ell)}\}_{\ell \in [2t+1]}$ *w.r.t.* π'
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 $\langle \text{ECHO}, \pi \rangle$

32: **Upon** receiving $n - t$ $\langle \text{ECHO}, \pi' \rangle$ or $t + 1$ $\langle \text{READY}, \pi', * \rangle$ **do** ▷ Ready
33: **If** $\text{FLG}_{\text{ready}} = 0$ **then**
34: **If** $\pi' = \pi$ **then**
35: Send $\langle \text{READY}, \pi', \text{SHARE}, B^*(i, \ell), w_{B^*(i,\ell)} \rangle$ 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 $\langle \text{READY}, \pi', \text{NOSHARE} \rangle$
40: $\text{FLG}_{\text{ready}} \leftarrow 1$

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

45: **Upon** reiceiving $\langle \text{DISTRIBUTE}, B(i, j), w_{B(i,j)} \rangle$ from P_j **do** ▷ Recover
46: **Upon** $\text{FLG}_{\text{ready}} = 1$ **do**
47: Interpolate C_{B_j} from π
48: **If** $\mathsf{KZG.VerifyEval}(C_{B_j}, B(i, j), w_{B(i,j)}) = 1$ **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 13: Procedures of DyCAPS.Share.

Proof. 1) If the dealer is honest, all honest parties eventually receive SEND messages from the dealer and send ECHO messages with the same π (line 31, Figure 13). Then, the conditions in line 32 will eventually be satisfied, and every honest party will send a READY message in either line 34 or line 35. The first honest party that sends the READY message must have received $n - t$ ECHO messages (line 32), since there are at most t READY messages from the corrupted parties at that time. Among these ECHO messages, at least $n - 2t = t + 1$ messages come from the honest parties. Namely, at least $t+1$ honest parties have received the SEND messages from the dealer. Therefore, when these parties enter the *Ready* step, they will each send a READY message with the SHARE tag (line 35). These $t+1$ READY messages are enough for all honest parties to send READY messages, with SHARE or NOSHARE tag, enter the *Distribute* step, and interpolate $B(x, *)$ (line 41-43). Finally, all honest parties send DISTRIBUTE messages, and $2t+1$ of them are enough for the honest parties to interpolate the $2t$-degree polynomial $B(*, y)$ and terminate DyCAPS.Share.

2) In our DyCAPS.Share, an honest party terminates iff it has obtained a valid share $B(*, y)$ (line 51). Therefore, we refer to the proof of completeness below as the proof of the second statement of Theorem 12. □

Theorem 13 (Completeness of DyCAPS.Share)**.** *If an honest party obtains a valid share from DyCAPS.Share, then each honest party obtains a share from DyCAPS.Share.*

Proof. If an honest party obtains a valid share from DyCAPS.Share, it has received $2t + 1$ valid DISTRIBUTE messages (line 50), where at least $t + 1$ of them are from honest parties. These honest parties will send DISTRIBUTE messages to all the others (line 44). Therefore, the conditions in line 42 are satisfied. We now prove all honest parties will receive READY messages from $n - t$ parties as required in line 41.

The $t+1$ honest parties sending DISTRIBUTE messages each has received $n-t$ READY messages (line 41). Namely, at least $n - 2t = t + 1$ honest parties have sent READY messages. Hence, all honest parties will receive at least $t+1$ READY messages and send their own READY messages (line 32-39). Therefore, all honest parties will proceed from line 41 to line 51 and obtain the shares $B(*, y)$. □

Theorem 14 (Secrecy of DyCAPS.Share)**.** *An adversary gains no advantage in extracting the secret s than random sampling during DyCAPS.Share.*

Proof. The secrecy is only meaningful when the dealer is honest. Otherwise, the adversary may directly obtain the secret s from the dealer. Given an honest dealer, 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 (line 29, Figure 13). 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, i.e., commitments and witnesses (line 14), so a computationally bounded adversary cannot extract s from these messages. In the subsequent steps, each P_m obtains n READY and DISTRIBUTE messages, respectively. Any $t + 1$ READY messages with SHARE tag result in a reduced share $B(x, m)$, and any $2t+1$ DISTRIBUTE messages lead to a full shares $B(m, y)$. Therefore, the adversary has 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 B. The Secret Reconstruction Protocol of DyCAPS

Compared to DyCAPS.Share and DyCAPS.Handoff, the secret reconstruction protocol DyCAPS.Recon is relatively simple. In the following, we briefly describe DyCAPS.Recon and analyze its security properties.

B.1. Details of Secret Reconstruction

A dealer-based DyCAPS.Recon protocol only involves one round of communication between the dealers and committee members. When a dealer invokes DyCAPS.Recon, the parties 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 $s = B(0, 0)$. The reconstruction can be as simple as interpolate $B(0, 0)$ from $t + 1$ tuples $\langle *, B(*, 0)\rangle$, where $B(*, 0)$ is the evaluation of $B(*, y)$ at $y = 0$.

In case there is no dealer, the committee members may broadcast their shares via RBC, and each of them receives enough shares to recover the secret.

The communication cost of DyCAPS.Recon does not exceed $O(\kappa n^3)$ in either case.

B.2. Security Analysis

We prove the termination of DyCAPS.Recon and the correctness of DyCAPS. The proof of secrecy is omitted, because the secret is exposed to every party in the dealer-free case, whereas in the dealer-based version, there is no interaction among the corrupted parties and the honest ones.

Theorem 15 (Termination of DyCAPS.Recon)**.** *If all honest parties invoke DyCAPS.Recon and all of them have terminated DPSS.Share or DPSS.Handoff, then all honest parties terminate DyCAPS.Recon.*

Proof. If all honest parties have terminated DPSS.Share or DPSS.Handoff, they will each obtain the latest full share $B(*, y)$, where $B(x, y)$ is of degree $\langle t, 2t\rangle$. Therefore, in both dealer-based and dealer-free cases, there are at least $n - t$ valid full shares, so the invoker(s) of DyCAPS.Recon will recover the secret $s = B(0, 0)$ using any $t + 1$ full shares. □

Before proving the correctness of DyCAPS, we first prove by Lemma 16 that all honest parties receive the same commitment set π.

Lemma 16. *If the dealer is honest, then at the end of DyCAPS.Share, all honest parties agree on the same commitment set π as the dealer sends.*

Proof. If the dealer is honest, all honest parties eventually receive SEND messages from the dealer and send ECHO messages with the same π. Every honest party will wait for $n - t$ ECHO messages or $t + 1$ READY messages with the same π'. Among these messages, at least one is from the honest parties, so the commitment set π' is the same as that from the dealer. In conclusion, an honest party will set π as π' either in line 28 or line 37. Namely, all honest parties will agree on the same commitment set π, which is originally sent by the dealer. □

Theorem 17 (Correctness of DyCAPS)**.** *If an honest dealer inputs s to DPSS.Share and v is the output of DPSS.Recon, then $v = s$. An arbitrary number of executions of DPSS.Handoff are allowed before DPSS.Recon.*

Proof. By Theorem 9, DyCAPS.Handoff keeps the secret s invariant. Therefore, we only need to consider the situation where no DyCAPS.Handoff is invoked before DPSS.Recon.

Combining Theorem 12 and Theorem 13, we conclude that all honest parties terminate DyCAPS.Share, and each of them obtains a share $B(*, y)$ in the presence of an honest dealer. We now prove that the shares are consistent with the dealer's input s, i.e., $B(0, 0) = s$.

By Lemma 16, all honest parties receive the same commitment set π from the dealer. The polynomial evaluations within SEND, READY, and DISTRIBUTE messages are verified against this π (line 28, 42, and 48, respectively, Figure 13). Due to the binding property of commitments, these evaluations correspond to the same polynomial as sent by the dealer. Therefore, all honest parties receive consistent shares $B(*, y)$ as the honest dealer sends.

In DPSS.Recon, the shares $B(*, y)$ are transferred to the dealer (or broadcast via RBC) along with the polynomial commitments $\{C_{B(\ell,y)}\}_{P_\ell\in\mathcal{C}}$, where $\mathcal{C}$ is the latest committee. The receiver waits for $t + 1$ valid shares with the same commitment set. The correctness of commitments holds because at least one honest parties send this set. Afterward, the dealer verifies the shares against the commitments and then interpolates the secret $v = B(0, 0)$. The binding property of commitments guarantees the consistency of v and s. □