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, chenhan123, 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 refreshed 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 $\kappa$ 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...
parties are connected by authenticated and private channels. But the messages will be delivered eventually. Besides, the messages will be delivered eventually. Moreover, the adversary is computationally bounded.

Adversary model. We assume a mobile adversary who adaptively corrupts at most \( t_e \) parties in committee \( C_e \), 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**. 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., \( P_{MVBA}(v) = 1 \).

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.

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

- \( \langle \phi(i), \omega(i) \rangle \leftarrow \text{KZG.CreateWitness}(\phi(x), i, pp) \): 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 \( \omega(i) \).

- \( 0/1 \leftarrow \text{KZG.VerifyEval}(C_\phi, i, v, \omega(i), pp) \): this algorithm verifies a polynomial evaluation. It takes as inputs a commitment \( C_\phi \), an index \( i \), an evaluation \( v \), a witness \( \omega(i) \), and public parameters \( pp \). It outputs 1 if \( v = \phi(i) \).

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, \( \omega_1(i) \) and \( \omega_2(i) \), that both pass KZG.VerifyEval.
- **Hiding.** Given a \( t \)-degree polynomial \( \phi(x) \), a commitment \( C_\phi \), and \( t \) evaluation-witness tuples \( \langle i, \phi(i), \omega(i) \rangle \), an adversary cannot determine \( \phi(i') \) with a non-negligible advantage for any queried \( i' \).
- **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, \( \omega(i) = \omega_1(i)\omega_2(i) \) holds for \( \phi(i) = \phi_1(i) + \phi_2(i) \).

### Threshold signature

A quorum of parties to construct a full signature jointly. It consists of five algorithms: \( \text{TS.KeyGen}, \text{TS.SigShare}, \text{TS.VerifySh}, \text{TS.Combine}, \text{and TS.Verify} \):

- \( \{\langle tpk, tvk_i, tsk_i \rangle \in [n] \} \leftarrow \text{TS.KeyGen}(t, n, 1^k) \): this algorithm generates the threshold key pairs. It takes as inputs a threshold \( t \), a committee size \( n \), and a security parameter \( \kappa \) in unary form. The output is a threshold public key \( tpk \), a set of threshold verifier keys \( \{tvk_i\} \in [n] \), and a set of threshold secret keys \( \{tsk_i\} \in [n] \). Each party \( P_i \) is assigned with \( \langle tpk, \{tvk_i\} \in [n], tsk_i \rangle \). We sometimes omit \( tpk \) and \( tvk \) for simplicity.

- \( \sigma_{s,i} \leftarrow \text{TS.SigShare}(m, tsk_i) \): 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 \( \sigma_{s,i} \).

- \( 1/0 \leftarrow \text{TS.VerifySh}(m, tvk_i, \sigma_{s,i}) \): this algorithm verifies a signature share. It takes as inputs a message \( m \), a threshold verifier key \( tvk_i \), and a signature share \( \sigma_{s,i} \). It outputs 1 if \( \sigma_{s,i} \) is correctly generated via \( \text{TS.SigShare}(m, tsk_i) \).

- \( \sigma_m \leftarrow \text{TS.Combine}(m, \{\sigma_{s,i}\}) \): this algorithm generates a full signature from signature shares. The input is a message \( m \) and a share set \( \{\sigma_{s,i}\} \in [I] \). The output is a full signature \( \sigma_m \).

- \( 0/1 \leftarrow \text{TS.Verify}(m, tpk, \sigma_m) \): this algorithm verifies a signature. It takes as inputs a message \( m \), a threshold public key \( tpk \), and a full signature \( \sigma_m \). It outputs 1 if 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 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: \( \text{DPSS.Share}, \text{DPSS.Handoff}, \text{and DPSS.Recon} \):

- \{\langle s_i, \pi_i \rangle_{P \in C} \} \leftarrow \text{DPSS.Share}(t, n, s, 1^k) \): 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 \( \kappa \) in unary form. The output is a set of share-proof tuples \( \{\langle s_i, \pi_i \rangle_{P \in C} \} \).

- \{\langle s'_i, \pi'_i \rangle_{P \neq +1 \in C} \} \leftarrow \text{DPSS.Recon}(\{\langle s_i, \pi_i \rangle_{P \in C} \}) \): this protocol allows the new committee \( C' \) to obtain refreshed secret shares from the old committee \( C \). The input is old share-proof tuples \( \{\langle s_i, \pi_i \rangle_{P \in C} \} \), and the output is refreshed tuples \( \{\langle s'_i, \pi'_i \rangle_{P \neq +1 \in C} \} \).

- \( v \leftarrow \text{DPSS.Recon}(\{\langle s_i, \pi_i \rangle_{i \in I} \}) \): 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 \( C_e \) and \( C_{e+1} \) 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.
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 \((t, 2t)\)-degree bivariate polynomials and adopt the dimension-switching technique 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, \(B(\ast, y)\), are needed to deal with the inquiries, e.g., generating a signature or decrypting a ciphertext. In turn, the reduced shares, \(B(x, \ast)\), 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 bivariate polynomial, and 3) switch back the threshold to \(t\) and restore refreshed full shares. These phases are referred to as ShareReduce, Proactvize, 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 \(C^e\) and \(C^{e+1}\) as the old and new committees, respectively. Similarly, \(P^e_i\) and \(P^{e+1}_i\) are called old and new parties, respectively. Note that “old” and “new” refers to the epoch, so \(P^e_i\) and \(P^{e+1}_i\) might be the same entity.

**Prepare.** In this phase, a new committee \(C^{e+1}\) is selected, and P2P channels are established among all members in \(C^e\) and \(C^{e+1}\). The public parameters are also delivered to \(C^{e+1}\) 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.

**Proactvize.** 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 \(Q\). The randomness from the members in \(Q\) 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 T5.KeyGen\((t, n, 1^k)\) 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 \(B(x, \ast)\) as its reduced share. The specific procedures are depicted in Figure 3.

Each party \(P^e_i\) has at least \(2t + 1\) commitments \(C_{B(x, \ast)}\) and witnesses \(w_{B(i, \ast)}\) from either DyCAPS.Share or the prior DyCAPS.Handoff. With these elements, \(P^e_i\) 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 \(C_{B(x, j)} = \sum_{\ell \in [2t + 1]} \lambda_{\ell, j} C_{B(x, \ell)}\).

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 \(B(x, \ast)\).
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 \((t, 2t)\)-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 commit \( C_{e+1} \), 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 \( Q \). The randomness proposed by the members in \( Q \) is used to compute \( Q(x, y) \). There are four requirements for the joint generation:

1. The agreement on \( Q \) 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 \( Q \), 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 1.** We start from a primary scheme in the non-asynchronous setting, where a timeout exists. In this case, we may decide the candidate set \( Q \) by verifiable challenges.

Specifically, we let each party \( P_i^{e+1} \) initialize \( Q \) as \( C_{e+1} \) and generate a random \((t, 2t)\)-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 polynomial \( \text{Enc}_j(Q_i(x, j)) \), where \( j \in [n] \), along with \( n \) commitments to these polynomials.

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

3. \( \text{Enc}_j(m) \) encrypts a message \( m \) by \( P_j \)'s encryption public key \( epk_j \).
For each $w_i$, let $C_1$, $C_2$, $C_3$, $C_4$, $C_5$, and $C_6$ be the commitments to $Q_i$, $Q_i + 1$, $Q_i$, $Q_i + 1$, $Q_i + 1$, and $Q_i + 1$, respectively. This strawman scheme satisfies the four requirements mentioned above:

1) All challenges will arrive in time, so the honest parties may not raise challenges in time.

2) The adversary has at most $O(Q)$ instances consuming $O(N)$ bits of communication. CHURP [6] reduces the size of each RBC instance to $O(N)$ by dividing the generation of $Q(x, i)$ 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 $Q$.

Figure 4: Procedures of Proactivize.

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 $Q$.

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 $P_{x+i}^j$ broadcasts $2t + 1$ polynomial commitments $C_{Q_i, i}$ to $KZG$.Commit($Q_i(x, i)$) via RBC, where $i \in [2t + 1]$. These commitments are sufficient to derive the commitments to any other $Q_i(x, j)$, where $j \in [n] \setminus \{2t + 1\}$. $P_{x+i}^j$ also sends a polynomial $Q_i(x, i)$ to each $P_{x+i}^j$.

Upon receiving the polynomials and commitments, we let each party use threshold signatures to vote for the correct $Q_i(x, i)$, which is the Lagrange coefficient $\lambda$.
\(\sigma_{j,i} = \text{TS.SigShare}(j, tsk_i),\) denoting that it has received a valid polynomial from \(P^{t+1}_m\). Upon receiving \(2t+1\) signature shares for the same \(j\), \(P^{t+1}_m\) forms a full signature \(\sigma_j\). \(P^{t+1}_m\) 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 \(\hat{V}\), such that \(|\hat{V}| = t+1\). The candidate set \(Q\) is then denoted as \(\{P^{t+1}_j | (j, \sigma_j) \in \hat{V}\}\).

**Analysis.** This strawman scheme satisfies the first and third requirements, due to the termination of MVBA and the condition \(|\hat{V}| = t+1\).

However, a malicious party \(P^{t+1}_m\) may get included in \(Q\) if it obtains \(2t+1\) votes. In the worst case, only \(t+1\) honest parties obtain \(Q_m(x, *)\) and vote for \(P^{t+1}_m\), whereas the other honest parties receive no information from \(P^{t+1}_m\). 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 \(Q(x, \ast)\). 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 \(Q\).

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

The procedures of Proactivize are described in Figure 4.

**Init.** Firstly, each party \(P^{t+1}_i\) initializes several empty sets, including a commitment set \(\pi_i\), two flag sets \(\text{FLG}_{\text{com}}\) and \(\text{FLG}_{\text{rec}}\), two buffers \(S_{\text{rec}}\) and \(S_{\sigma}\), and an MVBA input set \(V_i\). Then, \(P^{t+1}_i\) generates a \(2t\)-degree random polynomial \(F_i(y)\), where \(F_i(0) = 0\). Finally, \(P^{t+1}_i\) 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 \(t\) degree.

**Commit.** Each party \(P^{t+1}_i\) generates a commitment set \(\pi_i = \{\ell, C_{Q_i, \ell}, C_{Z_{i, \ell}}, w_{Z,i_{i}(0)}, g_{F_i(\ell)}\}_{\ell \in [2t+1]}\), where \(C_{Q_i, \ell}\) and \(C_{Z_{i, \ell}}\) are commitments to \(Q_i(x, \ell)\) and \(Z_{i,\ell}(x)\), respectively, \(w_{Z,i_{i}(0)}\) is the witness for \(Z_{i,\ell}\), \(0\), and \(g_{F_i(\ell)}\) is the commitment to \(F_i(\ell)\). Finally, \(P^{t+1}_i\) broadcasts \(\langle \text{COM}, \pi_i \rangle\) via \(\text{RBC}_{1,i}\).

**Verify.** Upon receiving \(\langle \text{COM}, \pi_j \rangle\) from \(\text{RBC}_{1,j}\), \(P^{t+1}_i\) verifies that the resharing polynomials \(Q_i(x, \ast)\) are formulated correctly. Specifically, \(P^{t+1}_i\) first verifies \(F_i(0) = 0\) by checking \(\prod_{\ell=0}^{2t+1}(g_{F_i(\ell)})^{\chi_{0, \ell}} = 1\), where \(\chi_{0, \ell}\) are Lagrange coefficients. Then, \(P^{t+1}_i\) verifies \(Q_j(0, \ell) = F_i(\ell)\) by \(\text{KZG.VerifyEval}(C_{Z_{i, \ell}}, 0, 0, w_{Z,i_{i}(0)}) = 1\) and \(C_{Q_i, \ell} = C_{Z_{i, \ell}} g_{F_i(\ell)}\), where \(\ell \in [2t+1]\). If any verification fails, the \(\text{COM}\) message is discarded, and the changes related to this message are reverted. Finally, \(P^{t+1}_i\) interpolates \(C_{Q_i, \ell}\) for each \(P^{t+1}_i \in C^{t+1}\), and sets \(\text{FLG}_{\text{com}}[j] = 1\).

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

**Vote.** Upon receiving a \(\text{RESHARE}\) message from \(P^{t+1}_j\), party \(P^{t+1}_i\) first verifies it w.r.t. the commitment set \(\pi_j\), which is delivered from \(\text{RBC}_{1,j}\). Then, it formulates a signature \(\sigma^{t+1}_{i,j} = \text{TS.SigShare}(j, tsk_i)\) as a vote for \(P^{t+1}_j\). Afterward, the contents in the \(\text{RESHARE}\) message are split and relayed to the others. Specifically, \(P^{t+1}_j\) calculates an evaluation-witness tuple \(\langle Q_j(\ell, i), w_{Q_j(\ell, i)} \rangle\) and sends it to each \(P^{t+1}_i \in C^{t+1}\) within a \(\text{RECOVER}\) message. The vote \(\sigma^{t+1}_{i,j}\) is also included in this message.

**Recover.** Upon receiving \(t+1\) valid \(\text{RECOVER}\) messages with the same index \(k\), such that \(\text{TS.VerifySh}(k, \sigma^{t+1}_{i,j}) = 1\) and \(\text{KZG.VerifyEval}(C_{Q_i, \ell}, 0, Q_k(\ast, i), w_{Q_k(\ast, i)}) = 1\), \(P^{t+1}_i\) recovers the \(k\)-th shares by interpolating a \(t\)-degree polynomial \(Q_k(x, i)\). \(P^{t+1}_i\) also waits for \(2t+1\) valid votes and composes a full signature \(\sigma_k = \text{TS.Combine}(k, \{\sigma^{t+1}_{i,j} \}_{j \in \ell})\), 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, \(P^{t+1}_i\) inputs \(V_i\) into MVBA. The external predicate \(P^{\text{MVBA}}\) requires the output size \(|\hat{V}| = t+1\) and the full signatures within \(\hat{V}\) are all valid. The candidate set is then referred to as \(Q = \{P^{t+1}_j | (j, \sigma_j) \in \hat{V}\}\).

**Refresh.** Upon receiving \(\hat{V}\) from MVBA, \(P^{t+1}_i\) calculates its random share \(Q(x, i) = \sum_{P^{t+1}_j \in Q} Q_j(x, i)\). The reduced share is thus refreshed as \(B^j(x, i) = B(x, i) + Q(x, i)\). Finally, party \(P^{t+1}_i\) calculates the commitments \(C_{Q(x, i)} = \prod_{P^{t+1}_j \in Q} C_{Q_j, \ell}\) for all \(P_i \in C^{t+1}\) and enters the next phase.

### 3.6. Share Distribution

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

**Init.** \(P^{t+1}_i\) initializes two empty buffers \(S_{\text{com}}\) and \(S_{B^j}\).

**Commit.** \(P^{t+1}_i\) commits to the new reduced share \(B'(x, i)\) and broadcasts \(\langle \text{NEWCOM}, C_{B'(x, i)} \rangle\) via \(\text{RBC}_{2,i}\).

**Distribute.** \(P^{t+1}_i\) sends \(\langle \text{SHAREDIST}, B'(j, i), w_{B'(j, i)} \rangle\) to each \(P^{t+1}_j\), where \(w_{B'(j, i)}\) is the witness for \(B'(j, i)\). This step is executed concurrently with the \(\text{Commit}\) step above.

**Verify.** Upon receiving the \(\text{NEWCOM}\) message from \(\text{RBC}_{2,j}\), \(P^{t+1}_j\) verifies that the sender \(P^{t+1}_j\) uses the common random polynomial \(Q(x, y)\) to fresh its share. Specifically, \(P^{t+1}_j\) 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 \(\text{NEWCOM}\) message will be ignored.

**Interpolate.** \(P^{t+1}_i\) waits for \(2t+1\) valid \(\text{SHAREDIST}\) messages to interpolate the full share \(B'(i, j)\). Next, \(P^{t+1}_i\) multicasts a \(\text{SUCCESS}\) message to notify the other parties.

**Success.** Upon having sent the \(\text{SUCCESS}\) message, \(P^{t+1}_i\) waits for another \(2t\) \(\text{SUCCESS}\) messages and then enters the normal state.
Figure 5: Message flow of Proactivate within epoch $e + 1$. In the Vote stage, the emphasized RECOVER messages received by $P^*_m$ refer to the responses to $P^*_m$’s RESHARE messages. The witnesses are omitted for clarity of expression.

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 $\{P^*_h \mid h \in [n] \setminus [t]\}$, 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 $C^e$ and $C^{e+1}$ invoke DyCAPS.Handoff, respectively, then all honest parties in $C^e$ terminate DyCAPS.Handoff.

**Proof.** The old committee $C^e$ 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 $C^e$ has enough information to generate the REDUCE messages (line 3-4, Figure 5).

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 $P^*_e$ must have delivered a commitment set $\pi$ (line 46, Figure 13) and at least $2t+1$ DISTIBUTE messages (line 50). The required commitments $C^e_B(x, \pi)$ and witnesses $w^e(B(x, \pi))$ are in $\pi$ (line 14) and DISTIBUTE 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 $C^2$ will terminate DyCAPS.Handoff when $e = 1$. Hence, the honest parties in $C^2$ have enough information to generate the REDUCE messages from the prior DyCAPS.Handoff.

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

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

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

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

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

The worst situation for $P_i^{t+1}$ is that the corrupted parties do not send any messages to it. In this case, $P_i^{t+1}$ only receives $n$ COM from RBC (line 22, Figure 4) and $n - t$ RESHAPE messages from the honest parties (line 37). Then the honest parties will send RECOVER messages to each other (line 45). Hence, $P_i^{t+1}$ obtains $n - t$ polynomials $Q_h(x, i)$ and full signatures (line 51-55), where $\{P_h^{t+1}\}_{h \in [n]\setminus [i]}$ are honest parties. Therefore, $P_i^{t+1}$ 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 $V$ and candidate set $Q$ (line 61).

After the termination of MVBA, every honest party can calculate the random share $Q(x, \ast) = \sum_{P_j^{t+1} \in \mathcal{Q}} Q_j(x, \ast)$. We prove this statement in two cases.

**Case 1:** If the members in $Q$ are all honest, as mentioned above, $P_i^{t+1}$ has obtained $Q_h(x, i)$ for all $P_h^{t+1}$, $h \in [n]\setminus [i]$, to compute $Q(x, i) = \sum_{P_j^{t+1} \in \mathcal{Q}} Q_h(x, i)$.

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

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

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

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

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

**Proof.** By Lemma 1 the honest parties in $C^t$ terminate DyCAPS.Handoff. By Lemma 2, Lemma 3 and Lemma 4 the honest parties in $C^{t+1}$ also terminate. Combining these lemmas, we conclude that all honest parties in $C^t$ and $C^{t+1}$ 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 $C^t$ and $C^{t+1}$, respectively.

**Proof.** In ShareReduce, the new committee members wait for enough REDUCE messages from the old committee and interpolate the reduced shares. An honest $P_i^{t+1}$ accepts $(C_B(x, i), B(\ast, i), w_B(\ast, i))$ if it has received at least $t + 1$ messages containing the same commitment $C_B(x, i)$ and the evaluations $B(\ast, i)$ all pass the KZG verifications (line 9, Figure 4). Hence, $t$ corrupted parties cannot convinces an honest party with a different commitment.

By Lemma 2 each honest $P_i^{t+1}$ 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 $C^t$ and $C^{t+1}$, respectively.

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

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

Suppose the malicious party $P_{m+1}^{t+1}$ proposes an illegal polynomial $Q_m^{t+1}(x, y)$. Due to the strong correctness of KZG commitments, the $x$-dimension degree of $Q_m^{t+1}(x, y)$ is bounded by $t$. Hence, only the $y$-dimension degree of
$Q_m(x, y)$ can be manipulated to exceed $2t$. However, $P_{m+1}^e$ is only allowed to broadcast $2t+1$ commitments via RBC$_{C,m}$ (line 15-21), and the other commitments are interpolated by the receivers (line 30). These $2t + 1$ commitments fix a $(t, 2t)$-degree shadow polynomial $Q_m(x, y)$ in the view of honest parties. If $P_{m+1}^e$ sends a point on $Q_m(x, y)$ which is invalid w.r.t. the commitments to $Q_m(x, y)$, the receivers will not accept it (line 39). Therefore, the guaranteed outputs in Lemma 3 are actually the shares of $Q_m(x, y)$. Hence, the honest parties will obtain consistent random shares from $P_{m+1}^e \in Q$, and the common random polynomial $Q(x, y)$ is guaranteed to be $(t, 2t)$-degree.

Besides, within each COM message, the $2t + 1$ commitments $\{g F_i(t)\}_{i \in [2t+1]}$ ensure $Q_i(0, 0) = F_i(0) = 0$ (line 24), so we have $Q(0, 0) = \sum_{\ell \in \mathbb{Q}} Q_\ell(0, 0) = 0$.

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. \hfill \Box

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

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

**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 $C_e$ and $C^{e+1}$, 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. \hfill \Box

**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 $P_{i+1}^e$ is not corrupted.

**Lemma 10.** If $P_{i+1}^e$ 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 $n × t$ RESHAPE messages, each containing 2$t + 1$ 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 $(t, 2t)$ (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 $\sum_{\ell \in \mathbb{Q}} Q_\ell(x, i)$, it needs to obtain $Q_\ell(x, i)$ for all $P_{e+1}^e \in Q$. As $|Q| = |\hat{V}| = t + 1$ (line 59), at least one

honest party is included in $Q$. However, for any honest party $P_{h+1}^e \in Q$, 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 $P_{i+1}^e$, it has no advantage on extracting the polynomial $Q(x, i)$ than random sampling. \hfill \Box

**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 $P_i^e$ and $P_{j+1}^e$ in Figure 7. Specifically, at the end of epoch $e + 1$, each $P_i^e$ holds $B(x, i)$ and $B(i, y)$, and each $P_{j+1}^e$ 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 bivariate 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 $B(x, *)$. These polynomials correspond to $2t^2 + 2t$ 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 $s$. \hfill \Box

### 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 $O(\kappa n^3)$-sized KZG parameters, which lead to $O(\kappa n^5)$ 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 $n$ $O(\kappa n^3)$-sized RESHAPE messages to the others. Then, $n$ RBC instances are invoked, each consuming $O(n^2 m + \kappa n^2)$ bits of communication, 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, 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 $\mathbb{F}_p$, 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 $O(\kappa n^3)$ bits of communication.

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

omitting 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.

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. The results are shown in Figure 10. We
those in DyCAPS. In both protocols, we set the output size $|V| = t + 1$, 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 5.2.

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 $(t', 2t')$. An intuitive solution is directly generating a $(t', 2t')$-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 $C_e$ is refreshed to $B_{\text{imp}}(x, y)$, which is of $(t', 2t')$-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 $P_i^t$ 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 3.3 and Section 3.6. By Lemma 3 and Lemma 4, each $P_i^t$ obtains a $t$-degree reduced share $B_{\text{imp}}(x, i)$ and a $2t'$-degree full share $B_{\text{imp}}(i, y)$. Afterward, the old committee starts the regular DyCAPS.Handoff and hands over the reduced shares to the new committee, which subsequently generates a $(t', 2t')$-degree $Q(x, y)$ and refresh $B_{\text{imp}}(x, y)$ to $B'(x, y)$. In this way, the adversary, who obtains $t + t' < 2t'$ reduced shares $B_{\text{imp}}(x, *)$, 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 $(t, 2t)$, 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 $Q_i(x, y) = 0$. 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 $O(kn^2)$ bits in the optimistic case. However, if any party misbehaves, CHURP falls into the pessimistic path and consumes $O(kn^3)$ bits of communication. COBRA achieves $O(kn^3)$ bits of communication complexity, but its worst-case complexity grows to $O(kn^4)$ due to continuous view-changes.

Schultz-MPSS [13] realizes DPSS with a communication cost of $O(kn^4)$ 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 $O(kn^3 \log n)$ 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 [56], 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 [33], [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 pointed 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 tackle these problems, and may promote these services.

8. Conclusion

In this paper, we propose DyCAPS, an efficient asynchronous DPSS scheme with $O(kn^3)$ 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.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Async.</th>
<th>Adversary</th>
<th>Threshold</th>
<th>Best-case (^1) comm. cost</th>
<th>Worst-case (^1) comm. cost</th>
<th>Trusted setup</th>
<th>PKE required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schultz-MPSS [12]</td>
<td>×</td>
<td>Mobile</td>
<td>(t &lt; n/3)</td>
<td>(O(\kappa n))</td>
<td>(O(\kappa n))</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>Opt-CHURP [6]</td>
<td>×</td>
<td>Mobile &amp; semi-honest</td>
<td>(t &lt; n/2)</td>
<td>(O(\kappa n))</td>
<td>N/A</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Exp-CHURP-A</td>
<td>×</td>
<td>Mobile</td>
<td>(t &lt; n/2)</td>
<td>N/A</td>
<td>(O(\kappa n))</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>COBRA [7]</td>
<td>×</td>
<td>Mobile</td>
<td>(t &lt; n/3)</td>
<td>(O(\kappa n))</td>
<td>(O(\kappa n^3))</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>√</td>
<td>Mobile</td>
<td>(t &lt; n/3)</td>
<td>(\exp(n))</td>
<td>(\exp(n))</td>
<td>√</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Shannara [28]</td>
<td>√</td>
<td>Mobile</td>
<td>(t &lt; n/4)</td>
<td>(O(\kappa n \log n))</td>
<td>N/A</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>Yurek-DPSS [27]</td>
<td>√</td>
<td>Mobile</td>
<td>(t &lt; n/3)</td>
<td>(O(\kappa n))</td>
<td>(O(\kappa n^3))</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>√</td>
<td>Mobile</td>
<td>(t &lt; n/3)</td>
<td>(O(\kappa n))</td>
<td>(O(\kappa n^3))</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) 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**


to support a $(t, 2t)$-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 $\pi$, which originally contains only $y^0$.

Then, $P_d$ generates a $2t$-degree random polynomial $F(y)$, where $F(0) = s$, $F(\ell) = r_\ell$ for each $\ell \in [2t+1]$. Each committee member $P_i$ only needs to initialize an empty buffer $S_{\text{full}}$ and a flag $\text{FLG}_{\text{ready}} = 0$.

**Commit.** To prove the correctness of $\text{Init}$, $P_d$ sets $\pi = \left\{ (y^0, \ell, \bar{C}_{B_\ell}, \bar{C}_{Z_\ell}, \bar{z}_d(0), \bar{g}^F(\ell)) \right\}_{\ell \in [2t+1]}$, where $\bar{C}_{B_\ell}$ and $\bar{C}_{Z_\ell}$ are the commitments to $B(x, \ell)$ and $Z_d(x) = F(x) - F(\ell)$, respectively, and $\bar{z}_d(0)$ is the witness for $Z_d(0) = 0$.

**Send.** $P_d$ sends $\left\{ (\text{SEND}, \pi, \{B(i, \ell), w_{B(i, \ell)}\}) \right\}_{\ell \in [2t+1]}$ to each $P_i \in C$. 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 $\text{SEND}$ 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 $\text{Proactiveize}$. $P_i$ also verifies the evaluation-witness pairs $w_{B_i}(\ell, y)$ for each verification return. $P_i$ sets $\pi' = \pi$. Then, $P_i$ interpolates a $2t$-degree polynomial $B^*(i, y)$. The witnesses $\{w_{B^*(i, j)}\}_{j \in C}$ are also interpolated from $\pi$. Finally, $P_i$ multicasts $\text{ECHO}, \pi'$.

**Recovery.** Upon receiving $n-t$ $\text{ECHO}$ messages or $t+1$ $\text{READY}$ messages with the same $\pi'$, $P_i$ checks whether $\pi = \pi'$ holds. If so, $P_i$ sends $\left\{ \text{READY}, \pi', \text{SHARE}, B^*(i, j), w_{B^*(i, j)} \right\}$ to each $P_j \in C$. Otherwise, $P_i$ resets $\pi$ as $\pi'$ and discards the interpolated $\{w_{B^*(i, j)}\}_{j \in C}$ and $B^*(i, j)$. In the latter case, $P_i$ multicasts $\left\{ \text{READY}, \pi', \text{NO SHARE} \right\}$.

**Distribute.** $P_i$ collects $n-t$ $\text{READY}$ 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 $P_t \in C$ via $\left\{ \text{DISTRIBUTE}, B(\ell, i), w_{B(\ell, i)} \right\}$.

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

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

### 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 Proactiveize 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 $C$, 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].

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

8: Upon receiving Committ from $P_d$ do \(\triangleright\) Commit
9: For each $t \in [2t+1]$ do
10: $Z_i(x) \leftarrow B(x, t) - F(t)$
11: $C_{B_i} \leftarrow \text{KZG.Commit}(B(x, t))$
12: $C_{Z_i} \leftarrow \text{KZG.Commit}(Z_i(x))$
13: $\pi \leftarrow \pi \cup (t, C_{B_i}, C_{Z_i}, w_{Z_i}(0), g^{F(t)})$
14: Send Committ to $P_d$

16: Upon receiving SEND from $P_d$ do \(\triangleright\) Send
17: For each $P_i \in C$ do
18: For each $t \in [2t+1]$ do
19: $w_{B(i,t)} \leftarrow \text{KZG.CreateWitness}(Z_i(x), 0)$
20: Send ($\text{SEND}, \pi, \{B(i,t), w_{B(i,t)}\}_{t \in [2t+1]}$) privately to $P_i$

21: Upon invocation by $P_i \in C$ with input $\text{InitShare}$ do
22: Upon receiving $\text{InitShare}$ from $P_d$ do \(\triangleright\) Init
23: $\text{Send} \leftarrow \emptyset$
24: $\text{FLG}_{\text{ready}} \leftarrow 0$

25: Upon receiving ($\text{SEND}, \pi', \{B(i,t), w_{B(i,t)}\}_{t \in [2t+1]}$) from $P_d$ do \(\triangleright\) Echo
26: Verify $\pi'$ as line 24-28 in Proactiveize / $g^{F'(i)} \in [2t+1]$ are verified w.r.t. $g'$
27: Verify ($B(i,t), w_{B(i,t)}\}_{t \in [2t+1]}$ w.r.t. $\pi'$
28: $\pi \leftarrow \pi'$
29: Interpolate a 2t-degree polynomial $B^*(i, y)$ from $\{[t, B(i,t)]\}_{t \in [2t+1]}$
30: Interpolate $\{w_{B^*(i,j)}\}_{j \in C}$ from $\{w_{B(i,t)}\}_{t \in [2t+1]}$
31: Multicast ($\text{Echo}, x$)
32: Upon receiving $n-t$ ($\text{Echo}, \pi'$) or $t+1$ ($\text{Ready}, \pi', \ast$) do \(\triangleright\) Ready
33: If $\text{FLG}_{\text{ready}} = 0$ then
34: If $\pi' = \pi$ then
35: Send ($\text{Ready}, \pi', \text{Share}, \{B(i,t), w_{B^*(i,t)}\}$) to each $P_i \in C$
36: Else
37: $\pi \leftarrow \pi'$
38: Discard $\{w_{B^*(i,t)}\}_{j \in C}$ and $B^*(i, y)$
39: Multicast ($\text{Ready}, \pi', \text{NOShare}$)
40: $\text{FLG}_{\text{ready}} \leftarrow 1$

41: Upon receiving $n-t$ ($\text{Ready}, \pi', \ast$) do \(\triangleright\) Distribute
42: Upon there are $t+1$ valid READY messages with SHARE tag do
43: Interpolate $B(x, t)$
44: Send ($\text{Distribute}, B(i,t), w_{B(i,t)}$) to each $P_i \in C$
45: Upon receiving $\{\text{Distribute}, B(i,j), w_{B(i,j)}\}$ from $P_d$ do \(\triangleright\) Recover
46: Upon $\text{FLG}_{\text{ready}} = 1$ do
47: Interpolate $C_{B_i}$ from $\pi$
48: If $\text{KZG.VerifyEval}(C_{B_i}, B(i,j), w_{B(i,j)}) = 1$ then
49: $\text{Send} \leftarrow \text{Send} \cup \{j, B(i,j)\}$
50: If $|\text{Send}| \geq 2t+1$ then
51: Interpolate a 2t-degree polynomial $B(i,y)$ from $\text{Send}$

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 $\pi$ (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 $B(x, \ast)$ (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 $B(\ast, y)$ and terminate DyCAPS.Share.

2) In our DyCAPS.Share, an honest party terminates iff it has obtained a valid share $B(\ast, y)$ (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 $B(\ast, y)$.

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 $B(\ast, y)$ is the same as $B^*(\ast, y)$, which is interpolated from the SEND message (line 29, Figure 13). We refer to both $B(\ast, y)$ and $B^*(\ast, y)$ 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 share \( B(m, y) \). Therefore, the adversary has \( t \) reduced shares and \( t \) full shares. As \( B(x, y) = \text{degree} (t, 2t) \), 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 dealer 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 \( \{ C_{B(t, y)} \}_{y \in C} \), where \( C \) 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 \( (*, B(*, 0)) \), 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 \( (t, 2t) \). 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.Recon, we first prove by Lemma 16 that all honest parties receive the same commitment set \( \pi \).

Lemma 16. If the dealer is honest, then at the end of DyCAPS.Share, all honest parties agree on the same commitment set \( \pi \) 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 \( \pi \). Every honest party will wait for \( n - t \) Echo messages or \( t + 1 \) Ready messages with the same \( \pi' \). Among these messages, at least one is from the honest parties, so the commitment set \( \pi' \) is the same as that from the dealer. In conclusion, an honest party will set \( \pi = \pi' \) either in line 28 or line 37. Namely, all honest parties will agree on the same commitment set \( \pi \), 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 \( \pi \) from the dealer. The polynomial evaluations within Send, Ready, and Distribute messages are verified against this \( \pi \) (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 \( \{ C_{B(t, y)} \}_{y \in C} \), where \( C \) 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 \( s \).