GoSSamer: Lightweight and Linear-Communication Asynchronous (Dynamic Proactive) Secret Sharing and the Applications

Xinxin Xing*, Yizhong Liu*✉, Boyang Liao*, Jianwei Liu*, Bin Hu*, Xun Lin*, Yuan Lu♭, Tianwei Zhang†
*Beihang University, Email:{xxxshixiannva, liuyizhong, lby, liujianwei, hubin0205, linxun}@buaa.edu.cn
♭A*STAR Institute for Infocomm Research, Email: lu_yuan@a-star.edu.sg
†Nanyang Technological University, Email: tianwei.zhang@ntu.edu.sg

***Abstract*—Asynchronous complete secret sharing (ACSS) and asynchronous dynamic proactive secret sharing (ADPSS) are fundamental primitives for secret sharing and resharing in modern threshold systems, such as multi-party computation, distributed key management, and blockchain. However, existing ACSS constructions that employ homomorphic commitments incur notable computational overhead, while the lightweight-computation constructions require quadratic per-secret communication, limiting scalability as the number of parties grows. ADPSS constructions inevitably inherit these inefficiencies due to their tight coupling to the commitment-based ACSS and requiring at least quadratic cross-committee communication.**

To break these bottlenecks, we design GoSSamer, a concretely and asymptotically efficient protocol suite, where both our ACSS and ADPSS achieve (1) lightweight computation with only hash function and symmetric encryption; (2) asymptotically optimal, linear per-secret communication; (3) optimal resilience in asynchronous networks; and (4) post-quantum security. In GoSSamer-ACSS, we propose an original-evaluation propagation paradigm for linear communication without commitment-requiring interpolation, which further unlocks our lightweight bivariate-polynomial-based degree checking for share verification. Building on this foundation, GoSSamer-ADPSS contributes two further techniques: a consistency verification technique that decouples the ADPSS framework from the commitment-based ACSS, and a dual-committee reconstruction technique that yields linear per-secret communication. When deployed in distributed AWS instances, GoSSamer-ACSS reduces the runtime by 95.6% compared to the linear-communication scheme hbACSS (NDSS'22) and 45.8% compared to lightweight SS24 (JoC'24). GoSSamer-ADPSS reduces the runtime by at least 67.7% compared to LongLive (Usenix Security'23). Moreover, when applied to practical distributed key management systems, the GoSSamer-ACSS-based distributed key generation is 7-11× faster than DXK+23 (Usenix Security'23), and GoSSamer-ADPSS-based key resharing reaches a throughput of 1994 keys/s across 10-node committees, compared to 85 keys/s in LongLive.

Corresponding author: Yizhong Liu, who is also affiliated with the Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing.

1. Introduction

Threshold secret sharing [1] distributes a secret among n parties and supports quorum-based reconstruction. It has driven two robust extensions with malicious security in the asynchronous networks: asynchronous verifiable/complete secret sharing (AVSS/ACSS) and asynchronous dynamic proactive secret sharing (ADPSS) [2]. The asynchronous model operates without timing assumptions and is immune to the potential failures under synchrony loss. In this setting, ACSS strengthens AVSS with *completeness*, where all honest participants eventually output their shares. Building upon ACSS, ADPSS *proactively* refreshes and migrates the secret shares across *dynamic* committees in the long-lived systems, without revealing or altering the secret.

Owing to their robustness and direct role in deployed multi-party threshold services, ACSS and ADPSS are fundamental building blocks for secure multi-party computation (MPC) [3], Byzantine fault tolerance systems [4], distributed key management (DKM) [5], and more. In practice, these services often batched process many secrets: ACSS enables reliable share distribution, while ADPSS supports proactive share migration. This makes lightweight computation and linear amortized communication crucial for scalability, directly impacting the performance of modern threshold systems.

To maximize execution utility, recent *batched* ACSS schemes share multiple secrets in each instance. As summarized in Table 1, solutions [6], [7], [8] optimize the per-secret communication to the information-theoretic lower bound $\mathcal{O}(\kappa n)$, where κ is the security parameter. Despite being communicationally efficient, their reliance on homomorphic commitments [10], [11] results in a computation bottleneck. Recently, SS24 [9] proposes a "*lightweight*" ACSS with efficient computation, $\mathcal{O}(\kappa n)$ per-secret communication in the happy path, and $\mathcal{O}(\kappa n^2)$ per-secret communication in recovery. It utilizes lightweight cryptographic tools such as hash function and symmetric encryption, yielding post-quantum security and 2 orders of magnitude faster than commitment-based solutions. However, such lightweight tools do not support homomorphic operations, making it challenging to build protocols with the optimal per-secret communication in both happy path and the recovery.

Building upon ACSS, existing ADPSS schemes [2],

TABLE 1: Comparison of relevant batched ACSS schemes under threshold $t = \lfloor \frac{n}{3} \rfloor$

Scheme	Lightweight	Communication†			Setup		Security		
		in total	*per-secret*	*batch size*	*no CRS*	*no PKI*	*assumption*	*post-quantum*	*resilience*
hbACSS [6]	✗	$\mathcal{O}(\kappa bn + \kappa n^2 \log n)$	$\mathcal{O}(\kappa n)$	$\mathcal{O}(n \log n)$	✗	✗	q-SDH	✗	1/3
Bingo [7]	✗	$\mathcal{O}(\kappa bn + \kappa n^2)$	$\mathcal{O}(\kappa n)$	$\mathcal{O}(n)$	✗	✓	q-SDH	✗	
Haven++ [8]	✗	$\mathcal{O}(\kappa bn + \kappa n^2 \log n)$	$\mathcal{O}(\kappa n)$	$\mathcal{O}(n \log n)$	✓	✓	DL + RO	✗	
SS24 [9]	✓	$\mathcal{O}(\kappa bn^2 + \kappa n^3 \log n)$	$\mathcal{O}(\kappa n^2)$	$\mathcal{O}(n \log n)$	✓	✓	RO	✓	
Our Solution	✓	$\mathcal{O}(\kappa bn + \kappa n^3 \log n)$	$\mathcal{O}(\kappa n)$	$\mathcal{O}(n^2 \log n)$	✓	✓	RO	✓	1/3

† **Comm. Complexity** - Communication cost for all messages exchanged among honest nodes in the worst-case bound. Presented as: (1) *total complexity*: overall communication cost for all secrets; (2) *per-secret complexity*: amortized cost per secret; and (3) *batch size*: the minimum batch size required to achieve the stated per-secret communication complexity. We use n as the committee size, b the batch size, and κ the security parameter. SS24 [9] achieves $O(\kappa bn + \kappa n^2 \log n)$ communication in the happy path.

TABLE 2: ADPSS comparison under threshold $t = \lfloor \frac{n}{3} \rfloor$

Scheme♭	Lightweight	Communication†			Setup
		inner-com	*cross-com*	*round*	*no CRS*
CKL+02 [2]	✗	$\mathcal{O}(\kappa n^4)$	$\mathcal{O}(\kappa n^4)$		✗
ZSR05 [12]	✗	$\exp(n)$	$\exp(n)$	$\mathcal{O}(1)$	✓
LongLive^1 [13]	✗	$\mathcal{O}(\kappa n^2)$	$\mathcal{O}(\kappa n^2)$		✓
LongLive^2 [13]	✗	$\mathcal{O}(\kappa n)$	$\mathcal{O}(\kappa n^2)$		✗
Our Solution	✓	$\mathcal{O}(\kappa n)$	$\mathcal{O}(\kappa n)$	$\mathcal{O}(1)$	✓

♭ **Scheme** - LongLive^1 [13] builds upon Pedersen commitment, while LongLive^2 [13] builds upon KZG commitment [10]. Appendix B presents more related works.
† **Communication** - The per-secret communication cost for all messages exchanged among all honest nodes in the worst-case bound. Presented as (1) inner-committee complexity; (2) cross-committee complexity; and (3) round complexity.

[12], [13] derive verification proofs directly from those commitment-based ACSS solutions [6], a tight coupling that precludes a lightweight instantiation and makes computation the primary bottleneck. Moreover, the best-known result [13] achieves the optimal $\mathcal{O}(\kappa n)$ per-secret communication within each committee, while the cross-committee communication remains quadratic, as shown in Table 2.

Although current ACSS and ADPSS made significant steps towards building efficient and robust threshold systems, expensive commitments and quadratic communication still prevent them from scaling to practical batch sizes and committee sizes. This tension motivates a central question:

> Can we design efficient ADPSS and ACSS protocols that simultaneously achieve (i) "*lightweight*" computation using only hash function and symmetric encryption, and (ii) optimal per-secret communication of $\mathcal{O}(\kappa n)$?

1.1. Our Contributions

- **GoSSamer-ACSS: a lightweight batched ACSS scheme with linear communication.** Our design introduces two core techniques: a novel *batched original-evaluation propagation* (B-OEP) paradigm that strategically organizes share distribution, and a *bivariate-polynomial-based degree checking* (BP-DC) for lightweight share verification. Our B-OEP mechanism reduces the transmitted message size in lightweight ACSS scheme [9] by a factor of $\mathcal{O}(n)$, without applying any interpolation to the original evaluations sent from dealer. This design eliminates the structural reliance on the additively homomorphic commitments in prior linear-communication ACSS constructions [6], [7], [8], and enables lightweight share verification through our BP-DC technique. Consequently, GoSSamer-ACSS simultaneously offers lightweight computation and linear per-secret communication with high batch-processing efficiency, which is particularly suitable for practical deployments in MPC, DKM, and random beacon.
- **GoSSamer-ADPSS: a lightweight batched ADPSS protocol with linear communication.** Building upon GoSSamer-ACSS, our ADPSS introduces two key mechanisms: (1) A lightweight *dual-committee sharing consistency verification* component, which decouples commitment-based ACSS schemes from ADPSS, and directly unlocks a lightweight instantiation that builds upon our lightweight GoSSamer-ACSS, yielding a boost in throughput. In this component, we also eliminate any reliance on the distributed random beacon with non-interactive hash derivation. (2) A *dual-committee reconstruction* technique, which reduces the cross-committee communication complexity to the optimal $\mathcal{O}(\kappa n)$ per secret through bivariate polynomials.
- **Diverse applications for ACSS and ADPSS.** We demonstrate various applications of our efficient design. First, we theoretically evaluate the performance gain of GoSSamer-ACSS when applied in random beacon, distributed key generation, and multi-party computation. Moreover, we deploy the GoSSamer suite in a widely-used distributed key management system [5], [14]. In this deployment, GoSSamer-ACSS-based distributed key generation [15] outperforms DXK+23 [16] by 7-11× in runtime, while GoSSamer-ADPSS-based key resharing reaches a throughput of 1994 keys/s at $n = 10$, compared to 85 keys/s in LongLive [13].
- **Implementation and distributed deployment.** We implement GoSSamer-ACSS, GoSSamer-ADPSS, SS24 [9], and reproduce hbACSS [6], Haven++ [8], LongLive [13]. When deployed in wide-area networks of AWS servers with 1 node per server, GoSSamer-ACSS shares 30000 secrets in 48.9s within a 16-node committee, compared with 1128.2s for the commitment-based solution hbACSS [6] and 55.5s for the lightweight scheme SS24 [9]. GoSSamer-ACSS consistently outperforms SS24 [9] and reduces runtime by 45.8% when $n \geq 16$. Moreover, GoSSamer-ADPSS reshares 30000 secrets in 71.8s across committees with 31 nodes each, compared with 1417.9s for LongLive [13], and reduces the runtime by at least 67.7% across the evaluated settings.

2. Technical Contribution Overview

2.1. GoSSamer-ACSS

Motivation. Asynchronous complete secret sharing, as the robustness backbone of modern threshold systems, makes

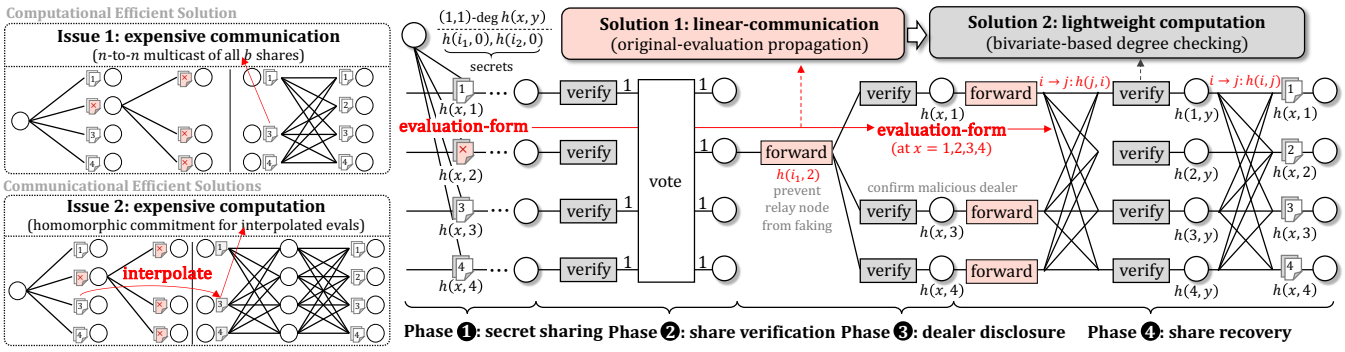

Figure 1: Technical overview of GoSSamer-ACSS with batch size $b = 2$ and committee size $n = 4$ for simplicity.

communicational and computational efficiency an essential design goal. As a warm-up, consider a batch of b secrets, let the dealer encode each secret as $f(0)$ of a t-degree polynomial $f(x)$, and give $f(i)$ to node i as the secret share. In asynchronous networks, completeness requires that every honest node obtains a valid share in a successful instance, even against a corrupted dealer. Current solutions fall into two paradigms. (1) *Computational efficient*: SS24 [9] achieves high throughput using only lightweight primitives, but in its recovery phase, once the dealer is implicated, nodes must relay their received shares to all others so that every honest node can recover its missing share. Under pessimistic conditions, this becomes an all-to-all dissemination of all shares, costing $\mathcal{O}(\kappa n^2)$ per secret, as shown in Figure 1, Issue 1. (2) *Communicational efficient*: hbACSS [6], Bingo [7], and Haven++ [8] achieve $\mathcal{O}(\kappa n)$ per-secret communication by forwarding interpolated evaluations for completeness, as shown in Figure 1, Issue 2. However, verifying them needs additively homomorphic commitments, e.g., KZG [10] or Pedersen commitment, which are computationally expensive.

Above situations become increasingly problematic when scaling: lightweight schemes cannot scale to large networks, while communication-optimal schemes cannot scale to large batches. GoSSamer-ACSS resolves this by establishing a new evaluation propagation paradigm B-OEP and a new share verification mechanism BP-DC, as in Figure 1, right part.

B-OEP: linear-communication original-evaluation forwarding. In contrast to SS24-style recovery, B-OEP avoids relaying full shares by leveraging a bivariate-polynomial row-column transformation to bound each per-party forwarded payload to $\mathcal{O}(\kappa b/n)$ bits, yielding the optimal linear per-secret communication. The challenge, however, is that a direct realization would require forwarding interpolated evaluations, which reintroduces the reliance on homomorphic commitments. B-OEP further overcomes this by circulating only *original evaluations* received from the dealer, thereby preserving lightweight verification.

Specifically, for a batch of $b = t + 1$ secrets, the dealer samples a (t, t)-degree bivariate polynomial $h(x, y)$, where $h(x, 0)$ is interpolated from the $t + 1$ secrets, and thus each row polynomial $h(x, i)$ defines the share for node i. Instead of distributing $h(x, i)$ in the coefficient form to node i directly, the dealer sends it in the *evaluation form*: $h(1, i), \ldots, h(n, i)$. If the dealer misbehaves, a node i detects an invalidity and discloses it by *forwarding an invalid original evaluation* $h(e, i)$ to all nodes. This suffices to implicate the dealer and triggers recovery to ensure completeness. In recovery, each node i with valid evaluations $h(1, i), \ldots, h(n, i)$ *forwards each original evaluation* $h(j, i)$ to its designated recipient j. Upon receiving sufficient consistent evaluations, every node j can verify and interpolate $h(j, y)$, then further distribute the evaluation $h(j, k)$ to each recipient k. Every node k thereby robustly reconstructs its row polynomial $h(x, k)$ that encodes all its shares. In this process, we cryptographically bind every forwarded evaluation to the dealer via secure message distribution (SMD) [9] in the disperse–retrieve paradigm [17], as is standard practice. This ensures relay nodes can only echo what the dealer disseminated and any forged evaluation fails retrieval-time verifications. Consequently, the B-OEP paradigm yields a communication complexity of $\mathcal{O}(\kappa n)$ per secret, but avoids creating any interpolated values that would demand additively homomorphic commitments. Moreover, operating purely on original evaluations enables the lightweight verification technique presented next.

BP-DC: lightweight verification for bivariate polynomial evaluations. We present BP-DC, a field-only, lightweight degree-checking mechanism for bivariate shares. In B-OEP, every share $h(j, i)$ must lie on the (t, t)-degree secret sharing polynomial $h(x, y)$. BP-DC certifies this with a single public *verification script* $v(x, y) = r \cdot h(x, y) + g(x, y)$, where r is a randomly sampled challenge and $g(x, y)$ is a random bivariate polynomial. Inspired by the classical MPC techniques [9], [18], [19], the verification script detects any inconsistent point via the Schwartz-Zippel lemma: if the evaluation $v(j, i)$ lies on the script $v(x, y)$, then $h(j, i)$ lies on $h(x, y)$, except with negligible probability (proved in Appendix C). Thus, after the dealer broadcasts the script $v(x, y)$, the correctness of each evaluation $h(j, i)$ can be verified by letting each recipient locally compute $v(j, i) = r \cdot h(j, i) + g(j, i)$ and check if it lies on the script $v(x, y)$. Moreover, we derive the challenge r via Fiat-Shamir by hashing a public transcript, i.e., a hash commitment to all dealer-disseminated points, thereby removing any reliance on random beacon and preserving a non-interactive, strictly lightweight design.

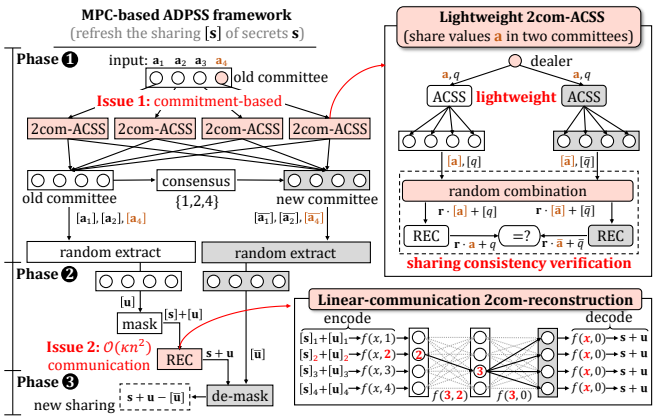

Figure 2: Technical overview of GoSSamer-ADPSS. "2com" abbreviates "dual-committee."

2.2. GoSSamer-ADPSS

Motivation. Existing ADPSS protocols [12], [13], [20], [21], [22], [23] are built on *commitment-based* ACSS. The polynomial commitments published in these ACSS schemes are later reused to certify resharing correctness, which tightly couples ADPSS to heavyweight cryptographic primitives and, in effect, precludes a truly lightweight instantiation. Moreover, existing ADPSS designs [12], [20], [21], [22], [23] incur at least quadratic cross-committee communication. Even the best-known result, LongLive [13], while achieving per-secret communication $\mathcal{O}(\kappa n)$ *within each committee*, still performs an n-to-n multicast *across committees* for each secret, yielding $\mathcal{O}(\kappa n^2)$ overall. This is further evidenced by our bandwidth evaluation in Section 7.

With this state of affairs, we propose GoSSamer-ADPSS, as shown in Figure 2. GoSSamer-ADPSS retains the MPC-based ADPSS framework of [13], [22], [23], but decisively decouples it from polynomial commitments via a lightweight dual-committee ACSS paired with a new sharing-consistency verification mechanism. In parallel, we introduce a dual-committee reconstruction paradigm that achieves optimal linear per-secret communication.

Background: MPC-based framework. We begin by recalling the MPC-based framework. First, the old committee jointly samples a random sharing $[\mathbf{u}]$, and the new committee jointly samples $[\bar{\mathbf{u}}]$ of the same underlying value, i.e., $\mathbf{u} = \bar{\mathbf{u}}$. Within a single committee, random-sharing generation follows the share-consensus-extract paradigm [13], [16], [22], [23]. Specifically, each node ACSS-shares local randomness $\mathbf{a}$, runs consensus to agree on the accepted inputs, and performs standard extraction that yields the random sharing $[\mathbf{u}]$. Next, possessing the sharing $[\mathbf{s}]$ of the secrets to be reshared, the old committee reconstructs $\mathbf{s}+\mathbf{u}$, and multicasts it to the new committee. The new committee then computes the new sharing as $[\bar{\mathbf{s}}] := (\mathbf{s} + \mathbf{u}) - [\bar{\mathbf{u}}]$.

Lightweight dual-committee sharing consistency verification. In the MPC-based framework, prior designs certify $\mathbf{u} = \bar{\mathbf{u}}$ via polynomial-commitment proofs inherited from commitment-based ACSS. Decoupling this guarantee from commitments is nontrivial, one must certify equality of two secrets held by distinct committees in an asynchronous, Byzantine setting without sacrificing secrecy. We resolve this with a lightweight dual-committee ACSS construction. The dealer ACSS-shares the same vector $\mathbf{a}$ to the old and the new committees using GoSSamer-ACSS, yielding $[\mathbf{a}]$ in the old committee and $[\bar{\mathbf{a}}]$ in the new one. Next, all nodes invoke a new sharing consistency verification to confirm $\mathbf{a} = \bar{\mathbf{a}}$, inspired by a classical MPC technique [18]. It operates by having the two committees check the equality of a randomized linear combination. Specifically, the dealer additionally ACSS-shares a randomly sampled scalar q to both committees. Given a public random challenge vector $\mathbf{r}$, if the underlying secrets differ, then $\mathbf{r} \cdot \mathbf{a} + q$ and $\mathbf{r} \cdot \bar{\mathbf{a}} + \bar{q}$ coincide with probability of at most $1/|\mathbb{Z}_p|$. Accordingly, each committee locally forms $\mathbf{r} \cdot [\mathbf{a}] + [q]$ and $\mathbf{r} \cdot [\bar{\mathbf{a}}] + [\bar{q}]$, performs robust interpolation, and accepts only if the two openings match. Thus, by comparing robust interpolation outputs, all parties certify cross-committee sharing consistency for the ACSS instance, except with negligible probability. Immediately after consensus and random extraction, both committees hold consistent random sharings $[\mathbf{u}]$ and $[\bar{\mathbf{u}}]$ with $\mathbf{u} = \bar{\mathbf{u}}$.

Linear-communication dual-committee reconstruction. The MPC-based path [13], [22], [23] has each old-committee node multicast all b masked secrets $\mathbf{s}+\mathbf{u}$ to every node in the new committee, yielding a per-secret communication $\mathcal{O}(\kappa n^2)$. We propose the following dual-committee reconstruction technique with only linear-communication and lightweight tools. The key idea is to compress the cross-committee message size by a factor of $\mathcal{O}(n)$ via bivariate polynomials. Specifically, possessing a vector of b shares $[\mathbf{s} + \mathbf{u}]_i$, each old committee node i interpolates each subset of $(t + 1)$ shares in $[\mathbf{s} + \mathbf{u}]_i$ into a t-degree polynomial $f(x, i)$. Next, each old committee node i performs intra-committee exchange by distributing each evaluation $f(j, i)$ to its designated recipient j, where each old committee node j can collect enough evaluations and run the robust interpolation to get the column $f(j, y)$. Since the values $\mathbf{s} + \mathbf{u}$ are encoded at the row $f(x, 0)$, node j now multicasts the single evaluation $f(j, 0)$ to the new committee. Each new committee node can then run robust interpolation to get $f(x, 0)$ and decode it for $\mathbf{s} + \mathbf{u}$. Consequently, the cross-committee payload is one field element for resharing $(t+1)$ secrets, thus the per-secret communication in the n-to-n multicast is reduced to $\mathcal{O}(\kappa n)$.

3. Preliminaries and Problem Formulation

3.1. Notation

Let $\mathbb{Z}_p$ denote a finite field of prime order p. We use κ to represent the security parameter; in GoSSamer-ACSS and GoSSamer-ADPSS, κ specifically refers to the bit length of an element in $\mathbb{Z}_p$. Let b denote the output batch size. We use com to represent a committee, where n is the total number of participating nodes, and t is the threshold, which also corresponds to the maximum number of corrupted nodes tolerated. We also use $:=$ for deterministic assignment, $\leftarrow$ for probabilistic assignment, and $\|$ for vector concatenation.

Bold symbols such as $\mathbf{a}$ denote vectors or matrices. The element-wise (n,t)-secret sharing of a vector $\mathbf{a}$ is written as $[\mathbf{a}]$, where $[\mathbf{a}]_i$ denotes the share held by the i-th node. For two m-dimensional vectors $\mathbf{a}, \mathbf{b}$, we define their inner product as $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{m} a_i b_i$ and their entry-wise sum as $\mathbf{a} + \mathbf{b} = (a_1 + b_1, \ldots, a_m + b_m)$. Let $\mathbf{f}(x)$ denote a vector of polynomials over $\mathbb{Z}_p[x]$. For any integer i, $\mathbf{f}(i)$ consists of the evaluations at $x = i$ of all component polynomials in $\mathbf{f}(x)$. We say that points $(i, \mathbf{a})$ lie on $\mathbf{f}(x)$ only if $\mathbf{a} = \mathbf{f}(i)$.

3.2. System Model

We consider a distributed system designed for the long-term secure management of secrets. Intuitively, the system's operation is structured into discrete protocol execution periods termed *epochs*. We adopt the rigorous definition of asynchronous epoch from Schultz-MPSS [24] and LongLive [13] without timing assumptions, which is further elaborated in Appendix A. For each epoch, a committee is defined, consisting of a set of nodes responsible for managing the shares of the secrets. Each node is assigned a unique identity for its role during that epoch, e.g., $\mathcal{P}_i$ for the i-th node. The composition of the committee may change across epochs, with the relationship between successive committees ranging from overlap to complete disjointness. If the same node participates in the committees of two different epochs, it will operate under two distinct identities.

Adversary model. Our adversary model follows that of LongLive [13] by considering a static, probabilistic polynomial-time (P.P.T.) adversary $\mathcal{A}$[1]. In each epoch e, a committee com_e of n nodes is responsible for executing the protocol, and the adversary may statically corrupt up to t of them before the epoch begins, where $t < n/3$. Once corrupted, a node is fully controlled by the adversary, who can read all its inputs intended for the current epoch and determine the messages it sends. During committee transitions from epoch e to $e+1$, the active node set is the union $\mathsf{com} = \mathsf{com}_e \cup \mathsf{com}_{e+1}$, and the corruption of such a node is counted in the adversary's budget in the union. In a static committee, this adversary model is equivalent to a mobile adversary, which may eventually corrupt all nodes over the system lifetime but is constrained to corrupting at most t nodes in any given epoch.

Network model. Communication occurs over an *asynchronous* network of authenticated and secure pairwise p2p channels. The adversary can delay and reorder the messages on any channel. However, messages transmitted between honest nodes are guaranteed to be eventually delivered.

3.3. Threshold Secret Sharing Protocols

Definition 1 - (Batched) Asynchronous Complete Secret Sharing. An (n,t,b)-ACSS protocol allows a dealer $\mathcal{P}$ to share a batch of b secrets $\mathbf{a} \in \mathbb{Z}_p^b$ among a committee of nodes $\mathcal{P}_1, \ldots, \mathcal{P}_n$, such that each node $\mathcal{P}_i$ receives a share $[\mathbf{a}]_i$. The secret vector $\mathbf{a}$ can be reconstructed from the valid shares of $t+1$ nodes using Lagrange interpolation. Any subset of at most t shares reveals no information about $\mathbf{a}$. The protocol is batched if $b > 1$. In such settings, we assume $b = \mathsf{poly}(n)$ as in prior work [9], [13]. An ACSS protocol satisfies the following properties:

- *Correctness*: If the dealer $\mathcal{P}$ is honest and inputs the secret vector $\mathbf{a}$, then there exists a batch of b polynomials $\mathbf{f}(x) \in \mathbb{Z}_p[x]^b_{\leq t}$, each of degree at most t, such that $\mathbf{f}(0) = \mathbf{a}$ and every honest node $\mathcal{P}_i$ outputs the share $[\mathbf{a}]_i = \mathbf{f}(i)$.
- *Completeness*: If any honest node outputs, then there exists a vector of polynomials $\tilde{\mathbf{f}}(x) \in \mathbb{Z}_p[x]^b_{\leq t}$, such that every honest node $\mathcal{P}_i$ outputs $\tilde{\mathbf{f}}(i)$ eventually.
- *Secrecy*: If the dealer is honest, the protocol reveals nothing about $\mathbf{a}$ to the adversary. Formally, there exists a P.P.T simulator $\mathcal{S}_{\mathsf{acss}}$ which, given the shares of corrupted nodes and a vector of independently sampled secrets, can produce a simulated view in the ideal world, such that the ideal and real worlds are indistinguishable from the view of $\mathcal{A}$.[2]

Applications such as MPC, ADKG, and ADPSS critically rely on *completeness*, to prevent a malicious dealer from sending invalid shares to some honest nodes in a successful sharing instance and thus affecting subsequent computations.

Definition 2 - (Batched) Asynchronous Dynamic Proactive Secret Sharing. An (n,t,n',t',b)-ADPSS protocol facilitates the *proactive* resharing of b secrets $\mathbf{s} \in \mathbb{Z}_p^b$ from an old committee com in epoch $(e-1)$ to a *dynamically* reconfigured new committee com' in epoch e. Specifically, the old committee consists of n nodes $\mathcal{P}_1, \ldots, \mathcal{P}_n$, where each node $\mathcal{P}_i$ holds an (n,t)-threshold share $[\mathbf{s}]_i$ of $\mathbf{s}$. The protocol enables the nodes in the old committee to collaboratively reshare the secrets to the new committee of n' nodes, such that each node $\mathcal{P}'_i$ obtains an (n',t')-threshold share $[\bar{\mathbf{s}}]_i$ of the same secrets $\mathbf{s}$. An ADPSS protocol satisfies the following properties:

- *Correctness*: Suppose there exists a vector of t-degree polynomials $\mathbf{f}(x)$ with $\mathbf{f}(0) = \mathbf{s}$, and each node $\mathcal{P}_i$ in the old committee holds the share $[\mathbf{s}]_i = \mathbf{f}(i)$. If an honest node $\mathcal{P}'_i$ in the new committee com' outputs its share $[\bar{\mathbf{s}}]_i$, then all honest nodes in com' output their shares eventually, and there exists a vector of t'-degree polynomials $\bar{\mathbf{f}}(x)$ such that $\bar{\mathbf{f}}(0) = \mathbf{s}$ and the share $[\bar{\mathbf{s}}]_i = \bar{\mathbf{f}}(i)$ for each honest node $\mathcal{P}'_i$.
- *Termination*: If all honest nodes in com initiate the protocol, all honest nodes in com' output eventually.
- *Secrecy*: No information about $\mathbf{s}$ is revealed to the adversary $\mathcal{A}$. Formally, there exists a P.P.T simulator $\mathcal{S}_{\mathsf{adpss}}$ which, given random secrets and the shares of corrupted nodes in the old committee, can produce a simulated view in the ideal world such that the adversary's view in the ideal and real world is indistinguishable.

3.4. Building Blocks

3.4.1. Secure Message Distribution. SMD is a lightweight message distribution protocol proposed in SS24 [9]. SMD [9]

1. When modeling the hash function as a quantum random oracle [25], the adversary can also access a quantum computer.

without secrecy is wrapped around the well-known asynchronous verifiable information disperse (AVID) [17], as repeatedly exploited in asynchronous protocols [6]. For secrecy, conventional approaches layer public-key cryptography atop the disperse-retrieve paradigm of AVID. SMD [9] instead employs symmetric encryption and introduces a symmetric key distribution phase, thereby keeping the computational path strictly lightweight while avoiding any secure-channel assumption. It provides two primitives:

- $\mathsf{distribute}(m_1, \ldots, m_n) \rightarrow \{\mathcal{P}_i : (r, m_i)\}_{i\in[n]}$: distribute enables a dealer to distribute message $\mathbf{m} = (m_1, \ldots, m_n)$ to a committee with n nodes. Every node $\mathcal{P}_i$ receives its designated message m_i, along with a Merkle root r that serves as a lightweight commitment of $\mathbf{m}$.
- $\mathsf{forward}(m_u, \mathcal{P}_i, \mathcal{P}_j) \rightarrow \{\mathcal{P}_j : m_u\}$: It enables node $\mathcal{P}_i$ to forward a message m_u, received in distribute or forward, to another node $\mathcal{P}_j$. The forwarded data includes a proof tied to r, so recipient $\mathcal{P}_j$ can authenticate that m_u is exactly the dealer's message to the node $\mathcal{P}_u$ in distribute, without faking by relay node $\mathcal{P}_i$.

An (n, t)-SMD [9] satisfies the following properties if up to t nodes are corrupted:

- *Correctness*: (1) If any honest node $\mathcal{P}_i$ outputs a message in distribute or forward, the output must be consistent with a vector $\mathbf{m} = (m_1, \ldots, m_n)$ in which each message m_j is output by node $\mathcal{P}_j$ in distribute, and the Merkle root r is calculated from $\mathbf{m}$. If the dealer is honest, then $\mathbf{m}$ must equal the input of the dealer. (2) In forward, any message output by an honest node that is claimed to be the dealer's message to $\mathcal{P}_u$ in distribute, must be m_u.
- *Secrecy*: If both the dealer and node $\mathcal{P}_i$ are honest, and no honest node forwards m_i to any corrupted party, then the adversary learns nothing about m_i.
- *Completeness*: In distribute, if the dealer is honest or any honest node outputs a message, all honest nodes output eventually. If an honest node forward a message m_u to another honest node $\mathcal{P}_i$, then $\mathcal{P}_i$ outputs m_u eventually.

Overview. At a high level, the dealer chooses for each message a t-degree polynomial, derives symmetric keys from its evaluations, encrypts the message under the key at index zero, then encodes the ciphertext with an erasure code. For every party the dealer forms a chunk with the hash of a designated key evaluation and a ciphertext fragment, and commits to all chunks using a Merkle tree. Each participant receives its own chunk together with a valid Merkle authentication path and, after local verification, multicasts the validated material to all nodes. A Bracha-style agreement [4] then ensures that all nodes output the same Merkle root, after which each node can decode from the received chunks to recover its designated message. In the forward phase, the corresponding Merkle path authenticates the origin of each message, and the relay nodes are unable to fake the message without breaking the collision resistance of hash functions. The communication complexity for distribute is $\mathcal{O}(|\mathbf{m}| + \kappa n^2 \log n)$ [9], while forward requires $\mathcal{O}(|m_i| + \kappa n \log n)$ for a message m_i.

3.4.2. Robust polynomial interpolation. We consider $n = 3t{+}1$ evaluations of an unknown univariate t-degree polynomial, of which up to t may be adversarially withheld or corrupted. In asynchronous settings where evaluations arrive sequentially, an online error-correction variant [26] of robust polynomial interpolation incrementally updates the candidate polynomial as each new evaluation arrives, and uniquely reconstructs the polynomial under this error bound.

Overview. The routine proceeds in up to $t{+}1$ iterations. Given an accumulating set $\mathcal{Q}$ that contains code-words of the Reed-Solomon code, in the r-th iteration, the algorithm waits for $|\mathcal{Q}| = 2t{+}r$ and decodes the fragments in set $\mathcal{Q}$ to get $\mathcal{L}$. Once obtaining message $\mathcal{L}$, we encode it using Reed-Solomon code for fragments set $\hat{\mathcal{Q}}$. If there are more than $2t+1$ same fragments in $\mathcal{Q}$ and $\hat{\mathcal{Q}}$, online error-correction successfully outputs the decoded message $\mathcal{L}$. In Reed-Solomon codes, the message vector is in one-to-one correspondence with a t-degree polynomial via the fixed encoding, so decoding $\mathcal{L}$ is exactly the same as reconstructing the designated polynomial.

3.4.3. Asynchronous Agreement Primitives. Next, we present three widely-used asynchronous components.

Reliable broadcast. A reliable broadcast protocol ensures that a message m distributed by a dealer $\mathcal{P}$ is consistently received by all participants. We adopt the construction from DXR21 [27], which achieves a communication complexity of $\mathcal{O}(n|m| + \kappa n^2)$ when executed among n nodes. A reliable broadcast protocol satisfies the following properties:

- *Correctness*: All honest nodes that output a message output the same one. If the dealer $\mathcal{P}$ is honest, the output message is the original input of $\mathcal{P}$.
- *Completeness*: If any honest node outputs a message, or if an honest dealer inputs a message, all honest nodes output a message eventually.

Multi-valued validated asynchronous byzantine agreement (MVBA). MVBA is an asynchronous Byzantine fault-tolerant consensus protocol, initially proposed in CKPS01 [28]. In MVBA, each participating node proposes an input value, the protocol ensures agreement on one of these values that satisfies a predefined common predicate. An MVBA protocol satisfies the following properties:

- *Agreement*: All honest nodes output the same value.
- *Termination*: If all honest nodes propose inputs that satisfy the predicate, then all honest nodes eventually output.
- *External Validity*: If an honest node outputs a value, then the output satisfies the predicate.

One-sided voting (OSV)**.** One-sided voting is a weak agreement primitive in which each node independently decides whether to endorse a given event, and the participation of the protocol is counted as a vote. A successful decision is reached once at least $n - t$ nodes honestly participate. OSV is instantiated using the Bracha broadcast [4], omitting the proposal phase. This protocol terminates in $\mathcal{O}(1)$ rounds and incurs a communication complexity of $\mathcal{O}(\kappa n^2)$. An (n, t)-OSV protocol satisfies the following properties if whenever at most t-out-of-n nodes are corrupted:

- *Correctness*: If an honest node outputs done, then at least $n - 2t$ honest nodes initiated the protocol.

- *Completeness*: If any honest node outputs or all honest nodes initiate it, then all honest nodes output eventually.

4. GoSSamer-ACSS

GoSSamer-ACSS is a batched ACSS (defined in Section 3.3) scheme that simultaneously shares a batch of secrets $\mathbf{a}$ within a committee. Its public input includes the field $\mathbb{Z}_p$ of prime order p, the number of participating nodes n, threshold t, batch size b, and evaluation coordinates $\mathbf{c} = (c_1, \ldots, c_{t+1})$. We use $b \equiv 0 \mod (t+1)$ for convenience and to avoid extensive discussion on padding. The dealer privately inputs secrets $\mathbf{a}$. If the protocol outputs, then it outputs the (n,t)-threshold share $[\mathbf{a}]_i$ to each node $\mathcal{P}_i$, for all $i \in [n]$. The performance and security analysis is given in Appendix C.

Initially, secret shares are distributed by the dealer to all participating nodes, along with a lightweight verification script (Phase ❶). Upon successfully verifying its share, each node casts a vote to endorse the sharing instance, which is considered successful if at least $n-2t$ honest nodes obtain its valid share (Phase ❷). Nodes that fail to obtain valid shares disclose evidence of the dealer's misbehavior. A confirmed malicious dealer implies secret leakage (Phase ❸). Finally, to ensure completeness, nodes forward their valid shares to assist in recovering shares for nodes without them (Phase ❹).

4.1. Concrete Scheme

Phase ❶ : lightweight secret sharing.

Algorithm 1 illustrates the procedure of lightweight secret sharing in Phase 1. The dealer begins by encoding the input secrets into the rows of bivariate polynomials. Specifically, it partitions the b input secrets into groups of size $t+1$. For each group, it interpolates the secrets at coordinates $(c_1, \ldots, c_{t+1})$ into a t-degree univariate polynomial. Each such polynomial is then embedded into a randomly sampled (t,t)-degree bivariate polynomial, as the row at $y = 0$. This process results in a vector of bivariate polynomials $\mathbf{A}(x,y)$. By construction, the row $\mathbf{A}(x,0)$ encodes the b secrets and $\mathbf{A}(x,i)$ encodes the share for node $\mathcal{P}_i$ (Lines 1-8).

Next, the dealer distributes each share-encoding polynomial $\mathbf{A}(x,i)$ to its designated recipient $\mathcal{P}_i$ through the following steps. The dealer first samples a random (t,t)-degree blinding polynomial $E(x,y)$ (Line 9), which is used for blinding the subsequent verification script. It then invokes n parallel distribute instances of secure message distribution [9], as detailed in Section 3.4.1. In each instance k, the dealer distributes to each node $\mathcal{P}_i$ the evaluations $\mathbf{A}(k,i)$ and $E(k,i)$ (Lines 10-12). In the above process, the dealer provides the entire column of *original evaluations* $(\mathbf{A}(1,i), \ldots, \mathbf{A}(n,i))$ together with $(E(1,i), \ldots, E(n,i))$ for a fixed node $\mathcal{P}_i$. Note that holding all n evaluations rather than merely $t+1$ is critical for recovery in Phase 4, where each honest $\mathcal{P}_i$ forwards the authenticated evaluation $(\mathbf{A}(j,i), E(j,i))$ via SMD to each of the n designated recipients $\mathcal{P}_j$.

Algorithm 1 GoSSamer-ACSS

public input: field $\mathbb{Z}_p$ of prime order p, committee size n, threshold t, batch size b, evaluation coordinates $\mathbf{c} = (c_1, \ldots, c_{t+1})$
private input: secret vector $\mathbf{a}$ of length b (as dealer $\mathcal{P}$)
output: share $[\mathbf{a}]_i$ for each participating node $\mathcal{P}_i$

Phase ❶ : lightweight secret sharing (as dealer $\mathcal{P}$)
1: • we use $b \equiv 0 \mod (t+1)$ for description convenience
2: • divide $\mathbf{a}$ into $m = b/(t+1)$ groups $(\mathbf{a}_1, \ldots, \mathbf{a}_m)$ of size $t+1$
3: **for each** group $\mathbf{a}_j$ **do** ▷ *following steps encode* $\mathbf{a}$ *in* $\mathbf{A}(x,0)$
4: ∘ $\mathbf{a}_j = (a_{1,j}, \ldots, a_{t+1,j})$
5: ∘ interpolate $\{(c_k, a_{k,j})\}_{k\in[1,t+1]}$ into a t-degree polynomial $f(x)$
6: ∘ sample a (t,t)-degree polynomial $A_j(x,y)$ with $A_j(x,0) := f(x)$
7: • let $\mathbf{A}(x,y) := (A_1(x,y), \ldots, A_m(x,y))$
8: ▷ *the obtained* $\mathbf{A}(x,y)$ *is the secret sharing polynomial of* $\mathbf{a}$
9: • sample a (t,t)-degree blinding polynomial $E(x,y)$
10: **for** $k \in [n]$ **do** ▷ *distribute evaluation-form* $\mathbf{A}(x,j)$ *to* $\mathcal{P}_j$
11: ∘ $m_{k,j} = (\mathbf{A}(k,j), E(k,j))$
12: ∘ SMD.distribute$(m_{k,1}, m_{k,2}, \ldots, m_{k,n}) \to \{\mathcal{P}_j : (r_k, m_{k,j})\}_{j\in[n]}$
13: • derive an m-dimension challenge $\mathbf{r}$ by hashing $r_1, \ldots, r_n$
14: • reliable broadcast $v(x,y) := \mathbf{r} \cdot \mathbf{A}(x,y) + E(x,y)$ to all nodes

Phase ❷ : lightweight share verification (as node $\mathcal{P}_i$)
15: **upon** receiving $(\mathbf{A}(k,i), E(k,i), r_k)$ from SMD for all $k \in [n]$ **do**
16: • calculate the challenge vector $\mathbf{r}$ as in Line 13
17: **upon** outputting $v(x,y)$ from reliable broadcast **do**
18: **if** $v(k,i) = \mathbf{r} \cdot \mathbf{A}(k,i) + E(k,i)$ for all $k \in [n]$ **do**
19: ∘ state.set(happy) and init OSV ▷ *valid share, vote*
20: **else** locally store a faulty pair $(\mathbf{A}(e,i), E(e,i))$
21: **upon** OSV output **do** ▷ *at least* $n-2t$ *honest nodes received valid shares and set happy state, ACSS succeeds*
22: **if** state.get = happy **do** ▷ *current node* $\mathcal{P}_i$ *received valid shares*
23: • interpolate $\{(k, \mathbf{A}(k,i))\}_{k\in[t+1]}$ into the t-degree $\mathbf{A}(x,i)$
24: **for each** $A_j(x,i)$ in vector $\mathbf{A}(x,i)$ **do**
25: ∘ append $A_j(c_1,i), \ldots, A_j(c_{t+1},i)$ to $[\mathbf{a}]_i$
26: • **output** share $[\mathbf{a}]_i$
27: **else** state.set(complain) ▷ *nodes without valid shares complain*

During each distribute instance k, the dealer also computes a Merkle root r_k that commits to all inputs. After preparing $(r_1, \ldots, r_n)$, the dealer hashes these roots to derive a challenge vector $\mathbf{r}$ whose length matches $\mathbf{A}(x,y)$ (Line 13). The dealer then computes the verification script $v(x,y) := \mathbf{r} \cdot \mathbf{A}(x,y) + E(x,y)$ and reliably broadcasts it (Line 14). Here, the blinding polynomial $E(x,y)$ serves to conceal the distribution of $\mathbf{A}(x,y)$.

Challenge vector derivation. Recall the BP-DC technique in Section 2.1. Each recipient $\mathcal{P}_i$ verifies that each received evaluation $\mathbf{A}(j,i)$ equals $\mathbf{A}(x,y)$ evaluated at (j,i) by locally computing $v(j,i) := \mathbf{r} \cdot \mathbf{A}(j,i) + E(j,i)$. The evaluation is accepted if $v(j,i)$ equals the verification script $v(x,y)$ evaluated at (j,i). This check is sound only if all parties use the same, unpredictable $\mathbf{r}$. One could obtain such a challenge via interactive challenge-response [9], [15] through random beacon [29]. Instead, we apply the Fiat-Shamir transform in the random-oracle model to avoid this extra cost. Specifically, both the dealer and all recipients derive a common seed by hashing a domain-separated concatenation of the protocol session identifier and the Merkle roots $(r_1, \ldots, r_n)$ returned by the n SMD instances. From this seed, they deterministically derive the challenge vector $\mathbf{r}$. Since $(r_1, \ldots, r_n)$ commits to all input evaluations, fixing the

roots irrevocably fixes those inputs. Any attempt to forecast or bias $\mathbf{r}$ beforehand would require finding a hash collision. Hence, all honest parties derive the same, unpredictable $\mathbf{r}$.

Phase ❷: lightweight share verification.

Algorithm 1 also illustrates the procedure of lightweight share verification in Phase 2. In this phase, each node verifies the received evaluations. First, all participating nodes derive the same challenge vector $\mathbf{r}$ as the dealer by hashing the Merkle roots received in distribute. The Bracha-style agreement embedded in SMD guarantees that these roots are consistent across honest parties (Lines 15-16). Next, each node $\mathcal{P}_i$ waits for the verification script $v(x, y)$ to arrive from reliable broadcast (Line 17). Upon delivery, node $\mathcal{P}_i$ verifies $\{\mathbf{A}(k, i)\}_{k\in[n]}$ by computing $\{v(k, i)\}_{k\in[n]}$ and checking that they are precisely the evaluations of $v(x, y)$ at the corresponding coordinates (Line 18). If the check passes, the node accepts its valid shares, sets its local state to happy, and initiates the one-sided voting protocol (Line 19). Otherwise, it records a faulty evaluation pair $(\mathbf{A}(e, i), E(e, i))$ that fails the verification (Line 20). When the one-sided voting protocol outputs a signal, at least $n-2t$ honest nodes hold valid shares. The ACSS instance is then deemed successful. Each happy node $\mathcal{P}_i$ interpolates $\mathbf{A}(1, i), \ldots, \mathbf{A}(t + 1, i)$ to reconstruct $\mathbf{A}(x, i)$ (Lines 21-23). It then decodes $\mathbf{A}(x, i)$ into its secret share $[\mathbf{a}]_i$ by evaluating $\mathbf{A}(x, i)$ at coordinates $(c_1, \ldots, c_{t+1})$ (Lines 24-26). Any node that is not in state happy when OSV outputs switches its state to complain (Line 27). All nodes proceed to the following dealer disclosure phase.

Phase ❸: lightweight dealer disclosure.

Algorithm 2 illustrates the procedure of lightweight dealer disclosure in Phase 3. In this phase, each node in the complain state discloses the evidence of dealer misbehavior to all nodes. Specifically, the complaining node $\mathcal{P}_i$ locally retrieves the recorded faulty evaluation pair $(\mathbf{A}(e, i), E(e, i))$ (Lines 1-2) and forwards it to all nodes as a complain message using SMD, to inform this invalidity. In this process, SMD guarantees that the forwarded evaluation is exactly the dealer's message to $\mathcal{P}_i$, preventing a malicious $\mathcal{P}_i$ from falsely accusing the dealer of misbehavior (Lines 3-4). Upon receiving a complain message from a node $\mathcal{P}_k$, each node $\mathcal{P}_i$ locally verifies the correctness of the evaluation pair by calculating $\tilde{v}(e, k) := \mathbf{r}\cdot\mathbf{A}(e, k)+E(e, k)$. If $\tilde{v}(e, k)$ does not equal the evaluation at (e, k) on the lightweight verification script $v(x, y)$, the evaluation pair sent from $\mathcal{P}_k$ is indeed invalid, indicating the dealer's misbehavior. As a result, node $\mathcal{P}_i$ enters the following recovery phase (Lines 5-7).

Phase ❹: lightweight share recovery.

Algorithm 2 also illustrates the procedure of lightweight share recovery in Phase 4. In this phase, a malicious dealer is confirmed, and secrecy is no longer required. The protocol then focuses on achieving *completeness*, which ensures that any node $\mathcal{P}_i$ missing its valid share $\mathbf{A}(x, i)$ can successfully recover it. The recovery process is structured into two rounds to deliver amortized linear communication. In the first round, each happy node $\mathcal{P}_i$ with valid evaluations $(\mathbf{A}(1, i), E(1, i)), \ldots, (\mathbf{A}(n, i), E(n, i))$ uses SMD to forward each pair $(\mathbf{A}(l, i), E(l, i))$ to its designated recipient $\mathcal{P}_l$ (Lines 8-9). Upon receiving $(\mathbf{A}(i, j), E(i, j))$ from node $\mathcal{P}_j$ as node $\mathcal{P}_i$, it verifies the evaluation correctness through the script $v(x, y)$ (Lines 10-11). Since each forwarded point is individually authenticated by SMD and checked against $v(x, y)$, $\mathcal{P}_i$ can reconstruct $\mathbf{A}(i, y)$ from any $t + 1$ verified evaluations using standard interpolation (Lines 12-14).

Algorithm 2 GoSSamer-ACSS (handle dealer misbehavior)

Phase ❸ : lightweight dealer disclosure (as node $\mathcal{P}_i$)

1: **upon** state.get(complain) **do**
2: • locally retrieve the faulty pair $(\mathbf{A}(e, i), E(e, i))$
3: • forward message (complain, $\mathbf{A}(e, i), E(e, i)$) to all nodes with SMD
4: ▷ SMD *ensures the message is initially distributed by the dealer*
5: **upon** receiving (complain, $\mathbf{A}(e, k), E(e, k)$) from $\mathcal{P}_k$ in forward **do**
6: **if** $v(e, k) \neq \mathbf{r} \cdot \mathbf{A}(e, k) + E(e, k)$ **do**
7: • enter the recovery phase ▷ *the malicious dealer is confirmed*

Phase ❹ : lightweight share recovery (as node $\mathcal{P}_i$)

8: **if** state.get(happy) and hasn't sent rec message **do**
9: **for** $l \in [n]$ **do** forward (rec, $\mathbf{A}(l, i), E(l, i)$) to $\mathcal{P}_l$ through SMD
10: **upon** outputting (rec, $\mathbf{A}(i, j), E(i, j)$) from $\mathcal{P}_j$ in SMD **do**
11: **if** $v(i, j) = \mathbf{r} \cdot \mathbf{A}(i, j) + E(i, j)$ **do**
12: • $\mathcal{Q}_{\text{val}} := \mathcal{Q}_{\text{val}} \cup (j, \mathbf{A}(i, j))$
13: **if** $\mathcal{Q}_{\text{val}}$ reaches the size $(t + 1)$ **do**
14: ◦ interpolate $\mathcal{Q}_{\text{val}}$ for $\mathbf{A}(i, y)$ ▷ *every honest $\mathcal{P}_i$ get $\mathbf{A}(i, y)$*
15: ◦ send $\mathbf{A}(i, k)$ to $\mathcal{P}_k$ for all $k \in [n]$
16: **upon** receiving $\mathbf{A}(j, i)$ from node $\mathcal{P}_j$ **do**
17: • add $(j, \mathbf{A}(j, i))$ into an accumulating set $\mathcal{Q}_{\text{oec}}$
18: **upon** $|\mathcal{Q}_{\text{oec}}| = n - t$ **do**
19: ◦ run robust interpolation on the accumulating set $\mathcal{Q}_{\text{oec}}$ ▷ *it continuously updates the candidates and stabilizes the t-degree correct polynomial when $n - t$ correct evaluations eventually arrive*
20: **upon** outputting $\mathbf{A}(x, i)$ from robust interpolation **do**
21: • decode $\mathbf{A}(x, i)$ into $[\mathbf{a}]_i$ and output, as in Lines 24-26, Algorithm 1

The second round differs from the first in the number of honest senders. In the first round, only a subset of honest nodes may initially hold valid shares. By contrast, once the first round completes, every honest node holds its reconstructed row polynomial $\mathbf{A}(i, y)$. Thus, every honest node $\mathcal{P}_i$ sends to each $\mathcal{P}_k$ the evaluation $\mathbf{A}(i, k)$ on $\mathbf{A}(i, y)$ (Line 15), enabling each receiver $\mathcal{P}_k$ to collect these evaluations in set $\mathcal{Q}_{\text{oec}}$ and run robust interpolation of $\mathbf{A}(x, k)$ over the set (Lines 16-20). The robust interpolation continuously feeds $\mathcal{Q}_{\text{oec}}$ into a Reed-Solomon decoder and updates the candidate t-degree polynomial. By the time $n - t$ correct evaluations eventually arrive, the decoder stabilizes on the true $\mathbf{A}(x, i)$. Once produced, recovery concludes by decoding $\mathbf{A}(x, i)$ into share $[\mathbf{a}]_i$ (Lines 21-22).

Verifying generated sharings without homomorphic commitments. Higher-level computations over the generated sharings remain verifiable without homomorphic commitments. After applying MPC operations to the underlying sharings, the resulting sharings can be verified using a standard randomized linear-combination check [18]: parties locally form a random linear combination of their shares and jointly reconstruct a polynomial as the verification script. By the Schwartz-Zippel lemma, any inconsistent shares are detected except with negligible probability $1/|\mathbb{Z}_p|$.

4.2. Performance Analysis

Theorem 1. Consider a committee of n nodes, where up to $t < n/3$ are corrupted. Let the batch size be $b = \mathsf{poly}(n)$. GoSSamer-ACSS implements batched ACSS with a total communication complexity of $\mathcal{O}(\kappa bn + \kappa n^3 \log n)$ and an amortized communication complexity of $\mathcal{O}(\kappa n)$ per secret.

Proof. Let $\mathcal{O}_{\mathsf{dist}}(x)$ denote the communication complexity of using SMD to distribute $(m_1, \ldots, m_n)$ of length x to n participating nodes, such that each node $\mathcal{P}_i$ receives m_i. Let $\mathcal{O}_{\mathsf{forw}}(x)$ denote the complexity of using SMD to forward a message of length x to a node. Let $\mathcal{O}_{\mathsf{osv}}$ denote the communication of one-sided voting, and let $\mathcal{O}_{\mathsf{rbc}}(x)$ denote the communication of reliably broadcasting a message of length x. Let $m = b/(t+1)$, the communication complexity of GoSSamer-ACSS in the worst-case bound is given by:

$$n\mathcal{O}_{\mathsf{dist}}(\kappa nm) + \mathcal{O}_{\mathsf{rbc}}(\kappa n^2) + \mathcal{O}_{\mathsf{osv}} + n^2\mathcal{O}_{\mathsf{forw}}(\kappa m) + n^2\mathcal{O}_{\mathsf{forw}}(\kappa m) + \mathcal{O}(\kappa mn^2)$$

Given $\mathcal{O}_{\mathsf{dist}}(x) = \mathcal{O}(x + \kappa n^2 \log n)$, $\mathcal{O}_{\mathsf{osv}} = \mathcal{O}(\kappa n^2)$, $\mathcal{O}_{\mathsf{rbc}}(x) = \mathcal{O}(nx + \kappa n^2)$, and $\mathcal{O}_{\mathsf{forw}}(x) = \mathcal{O}(x + \kappa n \log n)$, the total asymptotic communication complexity of GoSSamer-ACSS is $\mathcal{O}(\kappa bn + \kappa n^3 \log n)$. The additive term $\mathcal{O}(\kappa n^3 \log n)$ is a per-instance fixed overhead independent of the batch size. When $b \geq \mathcal{O}(n^2 \log n)$, the amortized communication is $\mathcal{O}(\kappa n)$ per secret. □

Performance analysis compared to SS24. The lightweight ACSS solution SS24 offers a faster *happy path* with $\mathcal{O}(\kappa bn + \kappa n^2 \log n)$ communication when the dealer behaves correctly. SS24 only triggers recovery and satisfies ACSS completeness in the *pessimistic* path, at the cost of $\mathcal{O}(\kappa bn^2 + \kappa n^3 \log n)$ communication. GoSSamer avoids this quadratic per-secrect communication through the B-OEP paradigm, with $\mathcal{O}(\kappa bn + \kappa n^3 \log n)$ complexity in both happy and pessimistic paths.

5. GoSSamer-ADPSS

GoSSamer-ADPSS is a batched ADPSS protocol (defined in Section 3.3) that enables the resharing of Shamir shares [1] for a batch of secrets from an old committee to a new committee. Its public input includes the field $\mathbb{Z}_p$ and the batch size b. In the old committee com, the number of nodes is n and the threshold is t, while in the new committee com′, these parameters are n' and t'. Each node $\mathcal{P}_i$ in the old committee privately inputs the (n,t)-threshold share $[\mathbf{s}]_i$ of b secrets $\mathbf{s}$. The protocol outputs the (n',t')-threshold share $[\bar{\mathbf{s}}]_i$ of the secrets $\mathbf{s}$ to each new committee node $\mathcal{P}'_i$. Appendix E gives the performance and security analysis.

In Phase ❶, each node in the old committee initiates two lightweight batched ACSS instances, and respectively shares a small batch of random values among the old and new committees. To ensure the shared values are the same across the two committees, a lightweight dual-committee sharing consistency verification protocol is executed for each ACSS instance. The old committee then runs an MVBA protocol to agree on $n-t$ valid ACSS instances, and informs their indices to the new committee. Using the same public, deterministic extractor, the old committee derives the sharing $[\mathbf{u}]$ of the random mask vector $\mathbf{u}$, from the agreed ACSS outputs, while the new committee derives a sharing $[\bar{\mathbf{u}}]$ of the same $\mathbf{u}$. In Phase ❷, each node in the old committee masks its secret share $[\mathbf{s}]$ by locally computing $[\mathbf{s}] + [\mathbf{u}]$. The masked values are then jointly reconstructed through our linear-communication dual-committee reconstruction, thereby producing the masked secrets $\mathbf{s} + \mathbf{u}$ in the new committee. In Phase ❸, each new committee node subtracts its own share $[\bar{\mathbf{u}}]$ of the random mask from the reconstructed $\mathbf{s} + \mathbf{u}$ to obtain its final share $[\bar{\mathbf{s}}]$ of the secrets $\mathbf{s}$.

Algorithm 3 dual-committee sharing consistency verification

public input: field $\mathbb{Z}_p$, the number of nodes and threshold (n,t) in committee com and (n',t') in committee com′, batch size b
private input:
• as node $\mathcal{P}_i$ in committee com: Shamir share $[\mathbf{a}]_i$ of b secrets $\mathbf{a}$, and $[q]_i$ for blinding
• as node $\mathcal{P}'_i$ in committee com′: Shamir share $[\bar{\mathbf{a}}]_i, [\bar{q}]_i$

1: **upon** initialization **do** ▷ *as node $\mathcal{P}_i$ in* com *node $\mathcal{P}'_i$ in* com′
2: • seed seed′ ← beacon.open ▷ **we replace random beacon with non-interactive hash derivation in Remark (i)**
3: • multicast seed seed′ to the other committee
4: **upon** receiving $t'+1$ same seed′ $t+1$ same seed from the other committee **and** outputting seed seed′ from beacon.open **do**
5: • extract the b-dimension challenge $\mathbf{r}$ through $\mathsf{H}(\mathsf{seed}, \mathsf{seed}')$
6: • calculate $[w]_i := \mathbf{r} \cdot [\mathbf{a}]_i + [q]_i$ $[\bar{w}]_i := \mathbf{r} \cdot [\bar{\mathbf{a}}]_i + [\bar{q}]_i$
7: • multicast $[w]_i$ $[\bar{w}]_i$ among its own committee
8: **upon** receiving $[w]_j$ $[\bar{w}]_j$ from node $\mathcal{P}_j$ node $\mathcal{P}'_j$ **do**
9: • add $(j, [w]_j)$ $(j, [\bar{w}]_j)$ to the accumulating set $\mathcal{Q}_{\mathsf{oec}}$
10: **upon** $|\mathcal{Q}_{\mathsf{oec}}| = n - t$ $|\mathcal{Q}_{\mathsf{oec}}| = n' - t'$ **do**
11: • run robust interpolation on $\mathcal{Q}_{\mathsf{oec}}$ ▷ *it continuously updates the candidates and stabilizes the t-degree t'-degree correct polynomial when $n-t$ $n'-t'$ correct evaluations eventually arrive*
12: **upon** outputting $w(x)$ $\bar{w}(x)$ from robust interpolation **do**
13: • multicast $w(0)$ $\bar{w}(0)$ to the other committee
14: **upon** receiving $t'+1$ same $\bar{w}(0)$ from the other committee com′ receiving $t+1$ same $w(0)$ from the other committee com **do**
15: **if** $\bar{w}(0) = w(0)$ **do** output $[\mathbf{a}]_i$ $[\bar{\mathbf{a}}]_i$ ▷ $w(0) = w, \bar{w}(0) = \bar{w}$

5.1. Sub-Protocol: Sharing Consistency Verification

To eliminate ADPSS's dependence on the commitment-based ACSS schemes and further enable a fully lightweight construction, we introduce a new component: lightweight dual-committee sharing consistency verification, as presented in Algorithm 3. This sub-protocol verifies the secret equality of the two sharings produced during GoSSamer-ADPSS. We consider two committees, denoted as com and com′ respectively, which execute symmetric operations. The Algorithm 3 is described from the point of view of a node in com; the symmetric steps of com′ are colored red. This sub-protocol assumes that both committees have already obtained two candidate sharings from an external sharing procedure. In GoSSamer-ADPSS, they are generated by the dual-committee ACSS of a dealer, which shares the same secrets $\mathbf{a}$ and a blinding value q with both committees: the old committee receives $([\mathbf{a}], [q])$, and the new committee receives $([\bar{\mathbf{a}}], [\bar{q}])$.

Problem formulation. Each of the two committees, com and com′, holds the Shamir sharing of b secrets $\mathbf{a}$. The objective

is to ensure two *consistent* sharings; namely, reconstruction from either committee yields the same underlying secrets. Meanwhile, no information about the secrets is disclosed.

Definition 3 - Sharing consistency. Let $\mathbf{a}$ be a vector of b secrets. Suppose committee com of size n holds an (n,t)-sharing of $\mathbf{a}$, i.e., each node $\mathcal{P}_i$ holds a share $[\mathbf{a}]_i = \mathbf{f}(i)$, where $\mathbf{f}(x)$ is t-degree with $\mathbf{f}(0) = \mathbf{a}$. Similarly, committee com$'$ holds an (n',t')-sharing of $\bar{\mathbf{a}}$. We define the two sharings to be *consistent* if $\mathbf{a} = \bar{\mathbf{a}}$.

Protocol description. Intuitively, our solution is to compress the entire batch of sharings into a single random linear combination, and then compare the openings of the resulting sharings. Assume that each node $\mathcal{P}_i$ in committee com holds a secret share $[\mathbf{a}]_i$ of b secrets $\mathbf{a}$, along with a blinding share $[q]_i$. Similarly, each $\mathcal{P}'_i$ in com$'$ holds $[\bar{\mathbf{a}}]_i$ and $[\bar{q}]_i$. The protocol begins by letting each committee run a random beacon instance to distributively open a random seed, while we show how to eliminate random beacon in Remark (i). The seed is then multicast to the other committee (Lines 1-3). As a node $\mathcal{P}_i$ in com, after receiving $t'+1$ same seeds from the other committee com$'$, and outputting the seed of its own committee, it derives a challenge vector $\mathbf{r}$ of length b by hashing the two seeds (Lines 4-5). Then, $\mathcal{P}_i$ calculates a random linear combination $[w]_i$ from its secret share $[\mathbf{a}]_i$ and blinding share $[q]_i$ through equation $[w]_i := \mathbf{r}\cdot[\mathbf{a}]_i + [q]_i$, and multicasts it within its own committee. All nodes in com$'$ perform the symmetric operations (Lines 6-7).

By collecting $(j,[w]_j)$ sent from different nodes $\mathcal{P}_j$ into $\mathcal{Q}_{\text{oec}}$, each node in the committee com runs robust interpolation over the accumulating set $\mathcal{Q}_{\text{oec}}$, which continuously updates the candidate polynomial at each new evaluation arrives (Lines 8-9). Eventually, when $n-t$ correct evaluations arrive, the algorithm outputs polynomial $w(x)$. Each node then multicasts its evaluation at $x=0$ to the other committee com$'$. Similarly, nodes in committee com$'$ reconstruct $\bar{w}(x)$ from the robust polynomial interpolation and multicast $\bar{w}(0)$ to com (Lines 10-13). A node in com waits for at least $t'+1$ identical values $\bar{w}(0)$ from com$'$, while a node in com$'$ waits for at least $t+1$ identical values $w(0)$ from com. If $w(0) = \bar{w}(0)$, there is $w = \bar{w}$ and thus $\mathbf{a} = \bar{\mathbf{a}}$ except with negligible probability. The two committees therefore hold a consistent sharing. Consequently, each node in com outputs share $[\mathbf{a}]_i$, and each node in com$'$ outputs $[\bar{\mathbf{a}}]_i$. (Lines 14-15).

Remark (i): eliminate the expensive random beacon. By default, we sample seeds from a random beacon to keep this component plug-and-play with arbitrary Shamir sharings. In the instantiation for ADPSS, however, the beacon is unnecessary. ADPSS obtains its Shamir sharings via ACSS, whose completeness already mandates a commitment that binds all input secrets and is finalized by agreement. Once this commitment is fixed, seeds and the challenge vector can be derived non-interactively via Fiat-Shamir transformation in the random-oracle model. Specifically, in lightweight ACSS (e.g., GoSSamer-ACSS and SS24 [9]), the commitment is the Merkle root output by SMD. Hashing this root under domain-separated labels directly yields the challenge.

5.2. Concrete Scheme of GoSSamer-ADPSS

Algorithm 4 specifies the procedure of our GoSSamer-ADPSS, and a detailed explanation is provided below. Note that we use the batch size $b \equiv 0 \mod (t+1)$ for convenience and to avoid extensive description on padding.

Phase ❶ : dual-committee random sharing.

The objective of this phase is to generate consistent sharings of b random values for both the old and new committees, without revealing the underlying random values. Intuitively, these random values serve as a one-time pad for the subsequent resharing step.

Step 1: share. Each node $\mathcal{P}_i$ in the old committee independently samples a batch of $\hat{b} = b/(t+1)$ random values $\mathbf{a}_i$, and invokes the lightweight dual-committee ACSS (Lines 1-3); the definition of this primitive is provided in Appendix D. In the dual-committee ACSS, node $\mathcal{P}_i$ further samples a blinding value q_i, and initiates two independent, batched ACSS instances to share the secrets $(\mathbf{a}_i, q_i)$ to the old and new committees, respectively (Lines 4-6). To ensure that the sharings held by the two committees correspond to the same underlying secrets, both committees invoke the lightweight dual-committee sharing consistency verification, where each q_i is consumed as a blinding value (Lines 7-8).

Step 2: consensus. In asynchronous networks, ADPSS must proceed once $n-t$ dual-committee ACSS instances output, because the t instances that corrupted nodes never initiate are indistinguishable from those initiated by honest nodes but maliciously delayed. To reach consensus on these instances, each node waits until it outputs from $n-t$ ACSS instances that are verified by sharing consistency verification, and then submits the corresponding indices into MVBA (Lines 9-11). All nodes in the old committee wait for the MVBA protocol to produce a common index set $\mathcal{Q}_{\text{ss}}$ agreed upon by all participating nodes. The old committee then multicasts $\mathcal{Q}_{\text{ss}}$ to the new committee (Lines 12-14). MVBA employs an external predicate [13], [16] to guarantee that $|\mathcal{Q}_{\text{ss}}| = n-t$ and that every instance in $\mathcal{Q}_{\text{ss}}$ succeeds.

Step 3: extract. As $\mathcal{Q}_{\text{ss}}$ may include up to t ACSS instances from corrupted dealers, each node locally performs a random extraction step to eliminate their influence while retaining as many useful secrets as possible. Following the well-established technique in [15], [16], [30], every node $\mathcal{P}_i$ deterministically calculates $n-2t$ share vectors from the $n-t$ share vectors $\{[\mathbf{a}_j]_i\}_{j\in\mathcal{Q}_{\text{ss}}}$, using the hyper-invertible matrix. For ease of description, we assume $\hat{b}=1$, so that each share $[\mathbf{a}_j]_i$ is a scalar. After collecting the $n-t$ shares $\{[\bar{\mathbf{a}}_j]_i\}_{j\in\mathcal{Q}_{\text{ss}}}$, node $\mathcal{P}_i$ assembles them into a vector and multiplies it by a hyper-invertible matrix of size $(n-t)\times(n-2t)$, to obtain the vector $[\mathbf{u}]_i$ of $n-2t$ shares. As a result, when $\hat{b} = b/(t+1)$, the resulting vector is b-dimension (Lines 15-16)

Phase ❷: lightweight secret masking.

The objective of this phase is to conceal the sharing $[\mathbf{s}]$ of the secrets by adding the sharing $[\mathbf{u}]$ of the random masks, and then to reconstruct the masked secrets $\mathbf{s}+\mathbf{u}$.

Each node $\mathcal{P}_i$ in the old committee calculates $[\mathbf{s}]_i + [\mathbf{u}]_i$ by adding the random-mask share $[\mathbf{u}]_i$ to its secret share $[\mathbf{s}]_i$.

Algorithm 4 GoSSamer-ADPSS

public input: field $\mathbb{Z}_p$, batch size $b \equiv 0 \mod (t+1)$ for convenience, the committee size n and threshold t in the old committee com, (n', t') in the new committee com′, evaluation coordinates $\mathbf{c} = (c_1, \ldots, c_{t+1})$
private input: share $[\mathbf{s}]_i$ of b secrets $\mathbf{s}$ as an old committee node $\mathcal{P}_i$
output: share $[\bar{\mathbf{s}}]_i$ of secrets $\mathbf{s}$ for each new committee node $\mathcal{P}'_i$

Phase ❶: 2com random sharing (as old committee node $\mathcal{P}_i$)
1: **upon** invocation by $\mathcal{P}_i$ in old committee com **do** ▷ **Step 1: share**
2: • sample a vector $\mathbf{a}_i$ of $\hat{b} = b/(t+1)$ random values
3: • invoke dual-committee ACSS by:
4: ◦ sample a random blinding value q_i
5: ◦ share $(\mathbf{a}_i, q_i)$ to the old committee through ACSS
6: ◦ share $(\mathbf{a}_i, q_i)$ to the new committee through ACSS
7: ◦ invoke Algorithm 3 for sharing consistency verification
8: ▷ *verify if the dealer shares the same secrets in both committees*
9: **upon** outputting $[\mathbf{a}_j]_i$ from dual-committee ACSS **do**
10: • add index j into an accumulating set $\mathcal{Q}^i_{\text{ss}}$ ▷ **Step 2: consensus**
11: **upon** $|\mathcal{Q}^i_{\text{ss}}| = n - t$ **do** invoke MVBA with input $\mathcal{Q}^i_{\text{ss}}$
12: **upon** outputting the common set $\mathcal{Q}_{\text{ss}}$ in MVBA **do**
13: ▷ MVBA *utilizes external predicate to ensure* $|\mathcal{Q}_{ss}| = n - t$ *and all dual-committee ACSS instances in* $\mathcal{Q}_{ss}$ *succeeds at all honest nodes*
14: • multicast $\mathcal{Q}_{\text{ss}}$ to all nodes in the new committee com′
15: ▷ **Step 3: extract**
16: • deterministically extract $[\mathbf{u}]_i$ from $[\mathbf{a}_j]_i, j \in \mathcal{Q}_{\text{ss}}$ through hyper-invertible matrix ▷ *extract shares of* $\hat{b}(t+1)$ *random secrets from* $\hat{b}(n-t)$ *secrets, where at most* $\hat{b}t$ *secrets come from corrupted nodes*

Phase ❷: lightweight secret masking (as old committee node $\mathcal{P}_i$)
17: **upon** outputting $[\mathbf{u}]_i$ from last phase **do**
18: • divide $[\mathbf{s}]_i + [\mathbf{u}]_i$ into $\hat{b}$ groups of size $t+1$
19: **for each** group k **do**
20: ◦ interpolate the $(t+1)$ elements into t-degree $F_k(x, i)$ at coordinates $(c_1, \ldots, c_{t+1})$, $\mathbf{F}(x, i) := \mathbf{F}(x, i) \parallel F_k(x, i)$
21: • send $\mathbf{F}(j, i)$ to each node $\mathcal{P}_j$ for all $j \in [n]$
22: **upon** receiving $\mathbf{F}(i, j)$ from $\mathcal{P}_j$ **do**
23: • add $(j, \mathbf{F}(i, j))$ into an accumulating set $\mathcal{Q}_{\text{oec}}$
24: **upon** $|\mathcal{Q}_{\text{oec}}| = n - t$ **do** initiate robust interpolation on $\mathcal{Q}_{\text{oec}}$
25: **upon** outputting $\mathbf{F}(i, y)$ from robust interpolation **do**
26: • multicast $\mathbf{F}(i, 0)$ to all nodes in new committee com′

Phase ❸: secret share derivation (as new committee node $\mathcal{P}'_i$)
27: **upon** receiving $\mathcal{Q}_{\text{ss}}$ from the old committee for $t+1$ times **do**
28: **upon** outputting $\{[\bar{\mathbf{a}}_j]_i\}_{j \in \mathcal{Q}_{\text{ss}}}$ from dual-committee ACSS **do**
29: • extract $[\bar{\mathbf{u}}]_i$ from $[\bar{\mathbf{a}}_j]_i, j \in \mathcal{Q}_{\text{ss}}$ ▷ *same as Lines 15-16*
30: **upon** receiving $\mathbf{F}(j, 0)$ from $\mathcal{P}_j$ in the old committee **do**
31: • add $(j, \mathbf{F}(j, 0))$ into an accumulating set $\mathcal{Q}_{\text{oec}}$
32: **upon** $|\mathcal{Q}_{\text{oec}}| = n - t$ **do** initiate robust interpolation on $\mathcal{Q}_{\text{oec}}$
33: **upon** outputting $\mathbf{F}(x, 0)$ from robust interpolation **do**
34: • evaluating $\mathbf{F}(x, 0)$ at $(c_1, \ldots, c_{t+1})$ for $\mathbf{s} + \mathbf{u}$
35: • output $[\bar{\mathbf{s}}]_i := \mathbf{s} + \mathbf{u} - [\bar{\mathbf{u}}]_i$

The vector $[\mathbf{s}]_i + [\mathbf{u}]_i$ is then partitioned into $\hat{b}$ groups, each of size $t+1$, and each group is interpolated into a t-degree polynomial at coordinates $(c_1, \ldots, c_{t+1})$. The resulting $\hat{b}$ polynomials are denoted as $\mathbf{F}(x, i)$, with $\mathbf{F}(x, 0)$ encoding the masked secrets $\mathbf{s} + \mathbf{u}$ (Lines 17-20). Subsequently, each node $\mathcal{P}_i$ sends the evaluation $\mathbf{F}(j, i)$ to each node $\mathcal{P}_j$ within its old committee (Line 21). Each node then collects the received evaluations into set $\mathcal{Q}_{\text{oec}}$, and initiates robust interpolation over the accumulating set to reconstruct $\mathbf{F}(i, y)$ as $\mathcal{P}_i$ (Lines 22-25). Finally, to enable the recovery of $\mathbf{F}(x, 0)$ in the new committee and hence derive the masked secrets $\mathbf{s} + \mathbf{u}$, each node $\mathcal{P}_i$ in the old committee multicasts $\mathbf{F}(i, 0)$ to all nodes in the new committee (Line 26).

Phase ❸: secret share derivation.

This phase derives the new secret sharing $[\bar{\mathbf{s}}]$ for the new committee. Specifically, each new committee node $\mathcal{P}'_i$ first extracts its share $[\bar{\mathbf{u}}]_i$ of b random masks. The extraction is identical to that in Lines 15-16 (Lines 27-29). Subsequently, each new committee node $\mathcal{P}'_i$ collects the evaluation $\mathbf{F}(j, 0)$ sent from distinct old committee nodes $\mathcal{P}_j$ into set $\mathcal{Q}_{\text{oec}}$, and run robust interpolation over this accumulating set to reconstruct $\mathbf{F}(x, 0)$ (Lines 30-33). The masked secrets $\mathbf{s} + \mathbf{u}$ are decoded by evaluating $\mathbf{F}(x, 0)$ at coordinates $(c_1, \ldots, c_{t+1})$, and each $\mathcal{P}'_i$ computes its target share $[\bar{\mathbf{s}}]_i$ by subtracting the share of random masks from the masked secrets, i.e., $[\bar{\mathbf{s}}]_i = \mathbf{s} + \mathbf{u} - [\bar{\mathbf{u}}]_i$ (Lines 34-35). It is straightforward to verify that $[\bar{\mathbf{u}}]_i$ is the i-th share of $\mathbf{u}$, thus the new sharing $[\bar{\mathbf{s}}]$ is consistent with the original $[\mathbf{s}]$.

Lightweight MVBA and common coin. As GoSSamer-ADPSS operates solely with lightweight cryptographic primitives, the underlying MVBA shall be a lightweight design. DDL+24 [31] is one of such candidates without trusted setup, which has a communication complexity of $\mathcal{O}(\kappa n^3)$. Nonetheless, MVBA protocols with a common-coin setup typically are concretely more efficient, as exemplified by the lightweight protocol of Fin [32]. In GoSSamer-ADPSS, establishing such a common-coin setup (for MVBA) among the old committee is also feasible, since they have been active since an epoch before the initiation of ADPSS, thereby providing sufficient time in practice for coin pre-processing.

5.3. Discussion of High-Threshold Extension

The ADPSS protocol LongLive [13] has a high-threshold version when instantiated with the Pedersen commitment, achieving a reconstruction threshold up to $l < 2n/3$ at the cost of higher communication $\mathcal{O}(\kappa n^2)$. This naturally raises the question of whether our lightweight resharing technique extends to this high-threshold setting. While a full treatment lies beyond our scope, we sketch a concrete upgrade and evaluate its performance in Section 7.4.

Since $t = n/3$ is the information-theoretic upper bound for robust interpolation, any reconstruction exceeding this bound must equip each secret share with an explicit correctness proof. Our approach begins with the shares produced by GoSSamer-ADPSS at the low reconstruction threshold t. Any node that holds $l+1$ such shares distributively combines them into an evaluation of an l-degree polynomial, following the polynomial-sampling method of DXK+23 [16]. The resulting evaluation yields a high-threshold share $[s]$. To equip this share with a correctness proof, we adopt the approach of GS24 [15] and distributively derive the proof $g^{[s]}$, where g is a group generator. While we leave a fully lightweight realization to future work, this upgrade already closes a practical gap: existing high-threshold ADPSS constructions [13] devote roughly 90% of their running time to the underlying commitment-based ACSS; in contrast,

TABLE 3: Applications of GoSSamer-ACSS

Scheme	Lightweight	Communication	Assumption	Reference
ARB	✗	$\mathcal{O}(\kappa n^2)$	q-SDH	[7]
	✓	$\mathcal{O}(\kappa n^3 \log n)$	RO	[29]
	✓	$\mathcal{O}(\kappa n^2)$	RO	**GoSSamer-based**
ADKG*	✗	$\mathcal{O}(\kappa n^2 \log n)$	DL+RO	[34]
	✓	$\mathcal{O}(\kappa n^3)$	DL+RO	[15]
	✓	$\mathcal{O}(\kappa n^2)$	DL+RO	**GoSSamer-based**
AMPC	✓	$\mathcal{O}(\kappa n)$	RO	[35]
	✓	$\mathcal{O}(\kappa n)$	RO	**GoSSamer-based**

$*$ We mark the ADKG protocols that can derive public keys from a lightweight ACSS with ✓ and those that rely on a commitment-based ACSS with ✗.

our solution obtains high-threshold shares and proofs via lightweight ACSS, yielding concrete efficiency gains.

6. Applications

6.1. Distributed Key Management Systems

Real-world distributed key management platforms, such as Web3Auth [5] in MetaMask and Lit Protocol [14], rely on ACSS and ADPSS to deliver decentralized, non-custodial control over users' key pairs. In such systems, mutually distrustful operators collaboratively generate threshold key pairs via ACSS-based asynchronous distributed key generation [16] (ADKG), and refresh them across epochs through ACSS-based ADPSS [2]. These threshold keys power broad applications: authentication platforms [5], wallet custody services [14], and cryptographic operations in Byzantine agreement and blockchain, such as threshold signature [30] and threshold decryption [33]. Consequently, the security and efficiency of ACSS and ADPSS are pivotal to their overall performance. We evaluate the performance of GoSSamer on a distributed key management system in Section 7.5.

6.2. An Upgrade To Previous Applications

For GoSSamer-ACSS, we analytically characterize its performance in three applications in Table 3, including ADKG, asynchronous random beacon (ARB), and asynchronous multi-party computation (AMPC).

For GoSSamer-ADPSS, it serves as an efficient, readily deployable primitive for the following systems. (1) *Dynamic MPC service.* MPC enables a committee of servers to evaluate user-defined functions over private inputs [36]. In asynchronous dynamic MPC services [37], ADPSS is utilized to facilitate the secure migration of MPC state under dynamic committee reconfiguration, ensuring it persists without restarting or secret leakage. Furthermore, proactive resharing allows costly preprocessing to be performed opportunistically during periods of low network activity and securely retained for future use, particularly in settings such as the BMR Escape Hatch [13]. (2) *General share migration service.* In long-lived threshold systems under dynamic committee membership or mobile adversaries, proactive secret resharing is essential for maintaining consistent system status across epochs. For example, extractable witness encryption on blockchains [38] enables the distributed storage and retrieval of secrets among a committee, where secrets are reshared upon epoch transitions. Dynamic Byzantine fault tolerance protocols [39], [40] and dynamic threshold signatures [41] utilize share resharing for committee reconfiguration in blockchain. Some decentralized identity protocols [42] distribute identity as Shamir shares for privacy-preserving, requiring resharing under membership updates. In all such cases, GoSSamer-ADPSS provides the requisite robust share-migration with linear communication and lightweight computation, breaking the bottlenecks that make prior solutions costly in practice.

7. Evaluation

This section evaluates GoSSamer and its deployment in distributed key management. We implement GoSSamer-ACSS, GoSSamer-ADPSS, SS24 [9], and GoSSamer-ACSS-based ADKG that builds upon GS24 [15]. We also reproduce LongLive [13], hbACSS [6], Haven++ [8], and DXK+23 [16]. Our code is available at https://github.com/xxxsh1x1annva/GoSSamer, which is built on top of implementations including [6], [8], [13], [16], [43].

Implementation details. For a fair comparison, we ensure all implementations and the baselines are developed in Python 3.7.13, with the core cryptographic operations delegated to a Rust-based library from Zcash [44]. The `BLS12-381` curve is used to support downstream pairing-based primitives. We also equip them with the same components: the reliable broadcast from [27], the MVBA from [43], and the robust interpolation from [45]. Moreover, we implement SMD [9] with modifications to support batch-processing.

Evaluation environment. Our evaluations include both wide-area network (WAN) and local evaluations. In the WAN setting, each node runs on a t3.large AWS EC2 instance with 2 virtual CPUs and 8GB of RAM. All instances are evenly distributed across 4 AWS regions, including California, Singapore, Seoul, and Paris. As the primary performance bottleneck in batched protocols is computational rather than communicational, we also perform local evaluations on the machine with a 16-core Intel i9-12900KF CPU and 64GB of RAM, where a Bash script simulates the distributed environment, following the approaches in LongLive [13] and hbACSS [6]. In both settings, each node is instantiated as an independent process bound to a unique network port, enabling inter-process message exchanges.

7.1. Evaluation of GoSSamer-ACSS

Baselines. We compare GoSSamer-ACSS with three batched ACSS protocols, including SS24 [9], hbACSS [6], and Haven++ [8]. We first implement the theoretical solution SS24 [9] to facilitate direct comparison with the lightweight scheme. For protocols that achieve linear per-secret communication, we choose two representatives hbACSS [6] and Haven++ [8]. Among them, the univariate-polynomial-based scheme hbACSS [6] requires a trusted setup, whereas the bivariate-polynomial-based scheme Haven++ [8] operates without one. Unless otherwise stated, our evaluation focuses

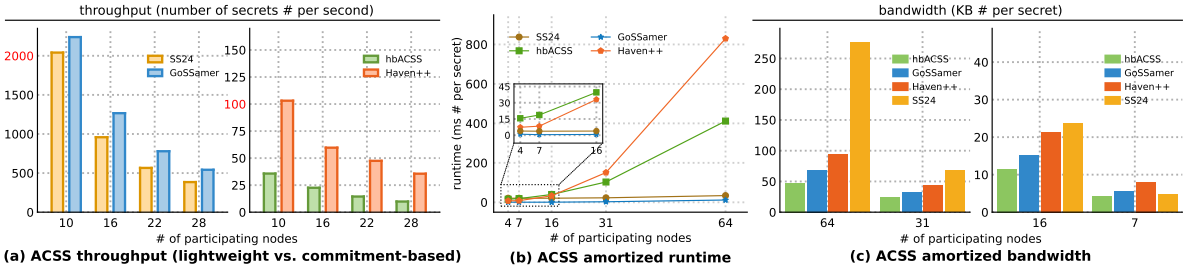

Figure 4: The evaluation results of batched ACSS under threshold $t = \lfloor \frac{n}{3} \rfloor$

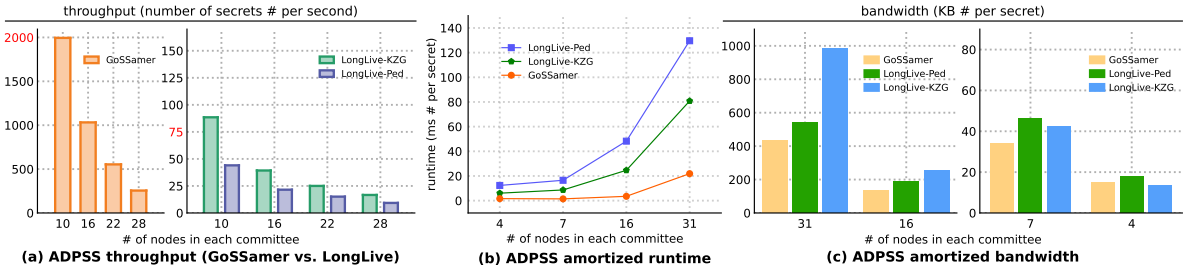

Figure 5: The evaluation results of batched ADPSS under threshold $t = \lfloor \frac{n}{3} \rfloor$

on adversarial scenarios where the dealer may distribute faulty shares to t nodes, which is critical to applications like ADPSS, where t nodes may act as malicious ACSS dealers.

Throughput. Figure 4(a) reports the *throughput* comparison, defined as the number of secrets shared per second. When varying the number of the participating nodes at $n = 10, 16, 22, 28$, lightweight schemes (Figure 4(a), left) share approximately 10^3 secrets/s, whereas the commitment-based protocols (Figure 4(a), right) share 10-10^2 secrets/s. Within the lightweight category, GoSSamer-ACSS consistently outperforms SS24 [9], sharing 111–432 additional secrets/s across the evaluated configurations.

Runtime. Figure 4(b) presents the *amortized runtime*, defined as the per-secret runtime. To capture the asymptotic performance of each scheme, we set the number of nodes to $n = 4, 7, 16, 31, 64$ and set the batch size to $n^2 \log n$. The results illustrate that the runtime in GoSSamer-ACSS is only 1.3-11.5% of that in commitment-based solutions hbACSS [6] and Haven++ [8]. Moreover, GoSSamer-ACSS outperforms the lightweight solution SS24 [9] by 2.6-27.3%.

Bandwidth. Figure 4(c) reports the *amortized bandwidth usage*, defined as the total volume of data sent by all nodes, divided by the batch size. Both the number of nodes and the batch sizes are configured identically to Figure 4(b). The results indicate that Haven++ [8], hbACSS [6], and GoSSamer-ACSS achieve linear per-secret communication with respect to n, whereas SS24 [9] incurs a quadratic cost. Specifically, at $n = 64$, SS24 incurs 276.4 KB/secret over the full protocol execution, whereas GoSSamer requires only 68.6 KB/secret, reducing bandwidth by 75.2%. For completeness, we also implement and evaluate the fault-free happy-path communication cost of SS24. In this setting, GoSSamer requires 43.9 KB/secret at $n = 64$, while SS24 requires only 12.1 KB/secret. Relative to hbACSS [6], GoSSamer-ACSS requires slightly more bandwidth (15.1 KB vs. 11.5 KB at $n = 16$). However, our solution is computationally lightweight and removes the trusted setup of [6]. Concretely, compared with GoSSamer, hbACSS [6] reduces bandwidth by 30.8% but prolongs runtime by 369.7% at $n = 31$.

7.2. Evaluation of GoSSamer-ADPSS

Baseline. We adopt LongLive [13] as our primary baseline. The original LongLive implementation uses the KZG commitment [10] with a trusted setup; we denote this version as LongLive-KZG. We also re-implement the LongLive protocol [13] with the Pedersen commitment to eliminate the trusted setup; we refer to this variant as LongLive-Ped. Across all configurations, we set the threshold $t = \lfloor 1/3n \rfloor$.

Throughput. Figure 5(a) reports the *throughput* comparison. When varying the number of nodes at $n = 10, 16, 22, 28$ in each committee, GoSSamer (Figure 5(a), left) achieves 14-26× more throughput compared to LongLive-KZG, and 27-48× compared to LongLive-Ped (Figure 5(a), right).

Runtime. Figure 5(b) presents the *amortized runtime*, measured with the number of nodes in each committee set to $n = 4, 7, 16, 31$ and the batch size set to $n^2 \log n$. The results demonstrate that GoSSamer-ADPSS achieves significantly lower runtime than other implementations, with its overhead ranging from 14.2-27.1% of LongLive-KZG, and 7.3-13.3% of LongLive-Ped.

Bandwidth. Figure 5(c) reports the *amortized bandwidth usage*. The batch sizes are configured identically to Figure 5(b). The results illustrate that GoSSamer incurs relatively low communication, followed by LongLive-Ped and LongLive-KZG. The bandwidth in GoSSamer is 44.2% of LongLive-KZG and 80.0% of LongLive-Ped at $n = 31$.

7.3. Geo-Distributed End-to-End Evaluation

To assess performance under real-world WAN conditions and provide deployment guidance, we further conduct a geo-distributed end-to-end evaluation on Amazon AWS.

TABLE 4: High-threshold ADPSS evaluation with $t = \lfloor \frac{2n}{3} \rfloor$

Metric	Scheme	$n = 4$	$n = 7$	$n = 16$	$n = 31$
Runtime (s)	LongLive	150.90	319.37	1271.39	5024.01
	GoSSamer	**5.37**	**6.06**	**12.61**	**341.47**
Bandwidth (MB)	LongLive	61.96	234.017	1459.20	6078.05
	GoSSamer	**4.03**	**20.78**	**227.90**	**2034.52**

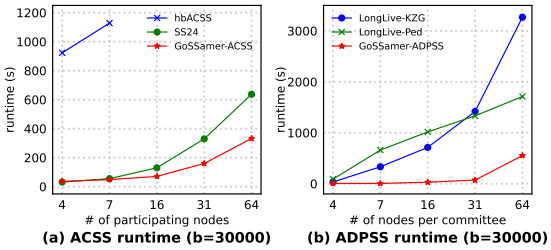

Figure 6: Geo-distributed evaluation of ACSS and ADPSS.

TABLE 5: Runtime (s) in key management lifecycle

(n, t)	Key Generation		Key Resharing		Retrieval
	DXK+23	*GoSSamer*	*LongLive*	*GoSSamer*	*robust*
(10,3)	650.62	**88.76**	115.18	**8.77**	3.35
(16,5)	1682.57	**151.21**	259.05	**20.98**	5.85
(22,7)	3238.00	**316.64**	397.68	**45.37**	7.74

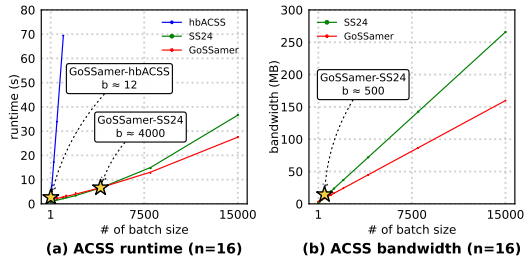

Figure 7: ACSS evaluation with varying batch sizes.

Figure 6(a) presents the runtime comparison of ACSS schemes in the WAN setting. GoSSamer-ACSS shows a clear performance advantage over the other two protocols. The runtime of our protocol is reduced by more than 45.8% compared to SS24 [9] ($n \geq 16$) and 95.6% compared to hbACSS [6] ($n \geq 4$). Relative to the local evaluation in Figure 4(b), the WAN setting amplifies the penalty of quadratic communication in SS24 [9].

Figure 6(b) presents the runtime comparison of ADPSS schemes with $t = \lfloor n/3 \rfloor$. GoSSamer reduces the runtime by 67.7%-99.2% compared to both variants LongLive [13]. Although the batch size $b = 30000$ is insufficient to reach peak throughput at $n = 64$, GoSSamer is still 3.1× faster than LongLive-Ped and 5.9× faster than LongLive-KZG, with the speedup increasing as the batch size grows.

ACSS evaluation with varying batch sizes and target applications discussion. Figure 7 reports both runtime and total communication across different batch sizes. Although GoSSamer achieves asymptotic linear amortized communication when $b = \mathcal{O}(n^2 \log n)$, the practical crossover occurs much earlier. GoSSamer already outperforms hbACSS [6] in runtime at $b \approx 12$. Compared with SS24 [9], GoSSamer avoids quadratic communication growth as n increases. Even at a modest system size of $n = 16$, where SS24's quadratic communication does not yet dominate, GoSSamer already requires less bandwidth at around $b \approx 500$, and becomes faster at around $b \approx 4000$. This regime matches target applications such as MPC and ADPSS, which process large batches of sharings and require high throughput.

7.4. Evaluation of High-Threshold ADPSS

To quantify the performance gain of our high-threshold extension (Section 5.3), we compare it against the state-of-the-art high-threshold baseline LongLive [13].

The results in Table 4 demonstrate that high-threshold protocols perform worse than their low-threshold version. When setting the batch size to merely $b = 300$, LongLive [13] requires more than 20 minutes to refresh at $n = 16$, compared to 12.61 seconds in our extension. Although this extension is not strictly lightweight, it demonstrates concrete runtime reduction by more than 93.2% compared to LongLive [13], and reduces the bandwidth by more than 66.5%.

7.5. Key Management Evaluation in Web3Auth

Web3Auth [5] is deployed atop the Torus Network, whose mainnet currently consists of around 10 nodes, and will be expanded to roughly 22 nodes. We therefore conduct experiments with $n = 10, 16, 22$.

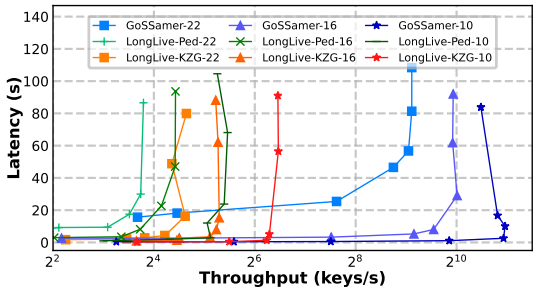

Figure 8: The throughput-latency trade-off in key resharing.

We evaluate the runtime of key generation, resharing, and retrieval phases in DKM. Table 5 reports our results with batch size $b = 10^4$. For key generation, we implement a GoSSamer-ACSS-based ADKG scheme that builds upon GS24 [15], the results demonstrate that it achieves 7-11× faster than the state-of-the-art ADKG implementation DXK+23 [16] at threshold $t = \lfloor n/3 \rfloor$, which we also modified to support batch processing. For key retrieval, we use the online error-correction [26] for robust reconstruction, which is relatively cheap. For key resharing, GoSSamer-ADPSS demonstrates 9-13× faster than LongLive [13], markedly shortening the service downtime required for reconfiguration.

We further evaluate the batch-processing efficiency in key resharing with GoSSamer-ADPSS, LongLive-Ped [13] and LongLive-KZG [13] at $t = \lfloor n/3 \rfloor$. Figure 8 reports our results in *throughput-latency trade-off*, where throughput is defined as the number of keys reshared per second and latency is the end-to-end runtime. The results demonstrate that lightweight protocol achieves strong batch-processing

performance. Specifically, GoSSamer-ADPSS achieves a peak throughput of 1994 keys/s with a low latency of 10.03s at $n = 10$. Compared with both LongLive variants, our solution reshares at least 1900 additional keys per second at $n = 10$, and 500 additional keys per second at $n = 22$.

Acknowledgment

This work is supported by the National Key R&D Program of China (2024YFB3108901); the National Natural Science Foundation of China (62472015, U22B2008, U21A20467, U2241213, 62172025); Beijing Natural Science Foundation (L251003); Zhejiang Provincial Natural Science Foundation of China (LMS25F020014); Open Research Fund of the State Key Laboratory of Blockchain and Data Security, Zhejiang University; and the Fundamental Research Funds for the Central Universities, Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing. This work is also supported by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme and CyberSG R&D Cyber Research Programme Office. Any opinions, findings and conclusions or recommendations expressed in these materials are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, Cyber Security Agency of Singapore as well as CyberSG R&D Programme Office, Singapore. Yuan Lu's contributions to this work were conducted while he was affiliated with the Institute of Software, Chinese Academy of Sciences.

References

[1] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[2] C. Cachin *et al.*, "Asynchronous verifiable secret sharing and proactive cryptosystems," in *ACM CCS*, 2002, pp. 88–97.

[3] K. Srinathan and C. Pandu Rangan, "Efficient asynchronous secure multiparty distributed computation," in *INDOCRYPT*. Springer, 2000, pp. 117–129.

[4] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[5] MetaMask, "Web3auth," https://web3auth.io/docs/infrastructure/nodes-and-dkg.

[6] T. Yurek *et al.*, "hbacss: How to robustly share many secrets," in *NDSS*, 2022.

[7] I. Abraham *et al.*, "Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation," in *CRYPTO*. Springer, 2023, pp. 39–70.

[8] N. Alhaddad *et al.*, "Haven++: Batched and packed dual-threshold asynchronous complete secret sharing with applications," *IACR Communications in Cryptology*, vol. 1, no. 4, 2025.

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

[10] A. Kate *et al.*, "Constant-size commitments to polynomials and their applications," in *ASIACRYPT*. Springer, 2010, pp. 177–194.

[11] B. Bünz *et al.*, "Bulletproofs: Short proofs for confidential transactions and more," in *S&P*. IEEE, 2018, pp. 315–334.

[12] L. Zhou *et al.*, "Apss: Proactive secret sharing in asynchronous systems," *ACM TISSEC*, vol. 8, no. 3, pp. 259–286, 2005.

[13] T. Yurek *et al.*, "Long live the honey badger: Robust asynchronous {DPSS} and its applications," in *USENIX Security*, 2023, pp. 5413–5430.

[14] L. Association, "Lit protocol," https://www.litprotocol.com.

[15] J. Groth and V. Shoup, "Fast batched asynchronous distributed key generation," in *EUROCRYPT*. Springer, 2024, pp. 370–400.

[16] S. Das *et al.*, "Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling," in *USENIX Security*, 2023, pp. 5359–5376.

[17] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *SRDS*. IEEE, 2005, pp. 191–201.

[18] M. Hirt and U. Maurer, "Robustness for free in unconditional multi-party computation," in *CRYPTO*. Springer, 2001, pp. 101–118.

[19] S. Atapoor *et al.*, "Vss from distributed zk proofs and applications," in *AISACRYPT*. Springer, 2023, pp. 405–440.

[20] Y. Yan *et al.*, "Shanrang: Fully asynchronous proactive secret sharing with dynamic committees," *Cryptology ePrint Archive*, 2022.

[21] Z. Zongyang *et al.*, "DyCAPS: Asynchronous dynamic-committee proactive secret sharing," *Chinese Journal of Electronics*, vol. 35, no. 1, pp. 307–321, 2026.

[22] B. Hu *et al.*, "Optimistic asynchronous dynamic-committee proactive secret sharing," *Cryptology ePrint Archive*, 2025.

[23] H. Feng *et al.*, "Practical asynchronous distributed key reconfiguration and its applications," *Cryptology ePrint Archive*, 2025.

[24] D. A. Schultz *et al.*, "Mobile proactive secret sharing," in *ACM PODC*, 2008, pp. 458–458.

[25] D. Boneh *et al.*, "Random oracles in a quantum world," in *ASIACRYPT*. Springer, 2011, pp. 41–69.

[26] M. Ben-Or *et al.*, "Asynchronous secure computation," in *ACM STOC*, 1993, pp. 52–61.

[27] S. Das *et al.*, "Asynchronous data dissemination and its applications," in *ACM CCS*, 2021, pp. 2705–2721.

[28] C. Cachin *et al.*, "Secure and efficient asynchronous broadcast protocols," in *CRYPTO*. Springer, 2001, pp. 524–541.

[29] A. Bandarupalli *et al.*, "Random beacons in monte carlo: Efficient asynchronous random beacon without threshold cryptography," in *ACM CCS*, 2024, pp. 2621–2635.

[30] F. Benhamouda *et al.*, "Sprint: High-throughput robust distributed schnorr signatures," in *EUROCRYPT*. Springer, 2024, pp. 62–91.

[31] S. Das *et al.*, "Asynchronous consensus without trusted setup or public-key cryptography," in *ACM CCS*, 2024, pp. 3242–3256.

[32] S. Duan *et al.*, "Fin: Practical signature-free asynchronous common subset in constant time," in *ACM CCS*, 2023, pp. 815–829.

[33] Y. G. Desmedt, "Threshold cryptography," *European Transactions on Telecommunications*, vol. 5, no. 4, pp. 449–458, 1994.

[34] R. Bacho *et al.*, "Harts: High-threshold, adaptively secure, and robust threshold schnorr signatures," in *ASIACRYPT*. Springer, 2024, pp. 104–140.

[35] A. Bandarupalli *et al.*, "Computationally and communication efficient batched asynchronous dpss from lightweight cryptography," *Cryptology ePrint Archive*, 2025.

[36] D. Lu *et al.*, "Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication," in *ACM CCS*, 2019, pp. 887–903.

[37] W. Yu *et al.*, "Ad-mpc: Fully asynchronous dynamic mpc with guaranteed output delivery," in *ACM PODC*, 2025.

[38] V. Goyal *et al.*, "Storing and retrieving secrets on a blockchain," in *PKC*. Springer, 2022, pp. 252–282.

[39] Y. Gao *et al.*, "Turritopsis: Practical dynamic asynchronous bft," *TIFS*, 2025.

[40] A. Kate *et al.*, "Dyna-hints: Silent threshold signatures for dynamic committees," *Cryptology ePrint Archive*, 2025.

[41] A. Cimatti *et al.*, "Dynamic-frost: Schnorr threshold signatures with a flexible committee," *Journal of Mathematical Cryptology*, vol. 19, no. 1, p. 20240045, 2025.

[42] Y. Liu *et al.*, "Fully anonymous decentralized identity supporting threshold traceability with practical blockchain," in *ACM WWW*, 2025, pp. 3628–3638.

[43] S. Das *et al.*, "Practical asynchronous distributed key generation," in *S&P*. IEEE, 2022, pp. 2518–2534.

[44] J. Grigg and S. Bowe, "zkcrypto/pairing," https://github.com/zkcrypto.

[45] S. Gao, "A new algorithm for decoding reed-solomon codes," in *Communications, information and network security*. Springer, 2003, pp. 55–68.

[46] A. B. Alexandru *et al.*, "State machine replication under changing network conditions," in *ASIACRYPT*. Springer, 2022, pp. 681–710.

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

[48] A. Choudhury *et al.*, "Asynchronous multiparty computation with linear communication complexity," in *International Symposium on Distributed Computing*. Springer, 2013, pp. 388–402.

[49] A. Patra *et al.*, "Efficient asynchronous verifiable secret sharing and multiparty computation," *JoC*, vol. 28, pp. 49–109, 2015.

[50] A. Choudhury and A. Patra, "An efficient framework for unconditionally secure multiparty computation," *TIT*, vol. 63, no. 1, pp. 428–468, 2016.

[51] A. Patra *et al.*, "Communication efficient perfectly secure vss and mpc in asynchronous networks with optimal resilience," in *AFRICACRYPT*. Springer, 2010, pp. 184–202.

[52] A. Choudhury and A. Patra, "On the communication efficiency of statistically secure asynchronous mpc with optimal resilience," *JoC*, vol. 36, no. 2, p. 13, 2023.

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

[54] A. Patra *et al.*, "Efficient statistical asynchronous verifiable secret sharing with optimal resilience," in *International Conference on Information Theoretic Security*. Springer, 2009, pp. 74–92.

[55] X. Ji *et al.*, "Linear-communication asynchronous complete secret sharing with optimal resilience," in *CRYPTO*. Springer, 2024, pp. 418–453.

[56] M. Backes *et al.*, "Asynchronous computational vss with reduced communication complexity," in *CT-RSA*. Springer, 2013, pp. 259–276.

[57] S. Das *et al.*, "Verifiable secret sharing simplified," in *S&P*. IEEE, 2025, pp. 633–651.

[58] N. Alhaddad *et al.*, "High-threshold avss with optimal communication complexity," in *International Conference on Financial Cryptography and Data Security*. Springer, 2021, pp. 479–498.

[59] E. Kokoris Kogias *et al.*, "Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures." in *ACM CCS*, 2020, pp. 1751–1767.

Appendix A. Epoch Definition

ABK+22 [46] proved that if a mobile adversary can eventually compromise an unbounded number of shareholders, no threshold protocol can protect the secret indefinitely. To avoid this, we adopt the epoch-based model of MPSS [24], refined by LongLive [13], which restricts the adversary to at most t corruptions per epoch. In an asynchronous setting, epochs are event-driven rather than time-based.

Local epoch. A node is in epoch e while it holds any epoch-e secret material, including original shares, keys, or reshares. It enters $e+1$ only after irreversibly erasing all epoch-e material. If it initiates resharing while still holding epoch-e material, it is considered in e and $e+1$ until it has received all honest resharing materials and securely deleted both the prior shares and resharing materials.

Global epoch. Let G_e denote the nodes responsible for epoch e. The system is in epoch e from when the first honest node in G_e enters e until the last honest node in G_e leaves e.

Appendix B. Supplementary for Related Work

First, we review the ACSS works in relevant settings.

Information-theoretic ACSS. Prior information-theoretic ACSS targets perfect or statistical security. Instantiations [3], [47], [48], [49], [50], [51] with perfect security offer the strongest guarantees with a resilience of $1/4$. Under statistical security, schemes [52], [53], [54] have a per-secret communication of at least $\mathcal{O}(\kappa n^3)$. JLS24 [55] reduces it to $\mathcal{O}(\kappa n)$ with a total communication of $\mathcal{O}(\kappa bn + \kappa^2 n^{12})$.

Computational secure ACSS. As an early representative work, CKLS02 [2] achieves communication complexity $O(\kappa n^3)$ under $1/4$ resilience. Later, BDK13 [56] and DXR21 [27] presented ACSS constructions with quadratic communication complexity. DXK+25 [57] attains the same complexity while additionally providing termination and public verifiability. Schemes [27], [34], [43], [58], [59] present high-threshold ACSS protocols that incur at least $\mathcal{O}(\kappa n^2)$ communication under various assumptions. hbACSS [6] provides a series of batched ACSS protocols, where the variant that employs KZG commitments [10] achieves a per-secret communication complexity of $\mathcal{O}(\kappa n)$. Bingo [7] and Haven++ [8] attain $\mathcal{O}(\kappa n)$ per-secret cost for high-threshold ACSS through bivariate polynomials. Bingo [7] depends on the KZG [10], whereas Haven++ [8] eliminates this trusted setup. All the above schemes rely on homomorphic commitments for share verification. SS24 [9] presents the first lightweight batched ACSS using only lightweight cryptographic primitives, reaching $O(\kappa n)$ amortized communication in the happy path, and $\mathcal{O}(\kappa n^2)$ in recovery.

Next, we review the ADPSS work.

Non-batched asynchronous protocols. Asynchronous proactive secret sharing protocol (APSS) was initially proposed in CKL+02 [2] with a communication complexity of $\mathcal{O}(\kappa n^4)$. ZSR05 [12] introduces ADPSS, the first dynamic-committee APSS protocol with exponential communication. Shanrang [20] reduces the cost to $\mathcal{O}(\kappa n^4)$ with a resilience of $1/4$. DyCAPS [21] achieves $\mathcal{O}(\kappa n^3)$ communication through zero-secret sharing. Opt-DPSS [22] further reduces this cost to $\mathcal{O}(\kappa n^2)$ communication optimistically, and FGL+25 [23] attains the same cost in the worst-case using a fixed-size committee and public verifiable secret sharing.

Batched asynchronous protocols. LongLive [13] introduces the first batched ADPSS protocol under the MPC-based

framework. In the high-threshold setting, it attains a per-secret communication of $\mathcal{O}(\kappa n^2)$ by employing Pedersen commitment. In the low-threshold setting, the per-secret cost drops to the optimal $\mathcal{O}(\kappa n)$ inner committee, provided a trusted setup for KZG commitment [10]. FGL+25 [23] and Opt-DPSS [22] extend the same MPC-based approach to their own batched variants. Specifically, the batched Opt-DPSS [22] incurs a communication of $\mathcal{O}(\kappa n^2)$ per secret in both high-threshold and low-threshold settings. A recent concurrent work on lightweight ADPSS [35] achieves $\mathcal{O}(\kappa bn + n^3 \log^2(n))$ communication in $\mathcal{O}(n)$ rounds.

Appendix C. Security Analysis of GoSSamer-ACSS

Lemma 1. GoSSamer-ACSS satisfies ACSS correctness.

Proof. In GoSSamer-ACSS, an honest dealer encodes secrets $\mathbf{a}$ following Lines 1-8 of Algorithm 1. Thus, for any secret $a_{\nu,u}$ (the ν-th secret in group $\mathbf{a}_u$), there is $A_u(c_\nu, 0) = a_{\nu,u}$ This implies that $A_u(c_\nu, y)$ is a t-degree univariate secret sharing polynomial for the secret value $a_{\nu,u}$. The honest dealer adheres to Lines 9-14 of Algorithm 1. Thus, each node $\mathcal{P}_i$ verifies the correctness of each received evaluation $\mathbf{A}(k,i)$ through $v(k,i) = \mathbf{r} \cdot \mathbf{A}(k,i) + E(k,i)$, which always holds for an honest dealer. All honest nodes then proceed to one-sided voting. Guaranteed by its completeness, every honest node eventually outputs in voting. Each $\mathcal{P}_i$ interpolates the t-degree polynomial $A_j(x,i)$ from the consistent points $\{(k, A_j(k,i))\}_{k\in[1,t+1]}$, then outputs $[\mathbf{a}]_i$ by evaluating $\{A_j(x,i)\}_{j\in[m]}$ at coordinates $c_1, \ldots, c_{t+1}$. For each secret $s_\tau \in \mathbf{a}$ that corresponds to some $a_{\tilde{\nu},\tilde{u}}$, define $h_\tau(y) = A_{\tilde{u}}(c_{\tilde{\nu}}, y)$. Then each h_τ is of t-degree and $h_\tau(0) = a_{\tilde{\nu},\tilde{u}}$. After gathering all $h_\tau(x)$ into $\mathbf{h}(x)$, there is $\mathbf{h}(0) = \mathbf{a}$. Moreover, by construction, every honest node outputs $[\mathbf{a}]_i = \mathbf{h}(i)$, establishing correctness. □

Lemma 2 (Schwartz-Zippel Lemma). Suppose there is a multivariate polynomial $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ with degree at most t. Let $r_1, \ldots, r_n$ be chosen independently and uniformly from $\mathbb{Z}_p$. Then, $\Pr[f(r_1, \ldots, r_n) = 0] \le t/|\mathbb{Z}_p|$.

Lemma 3. Assuming $|\mathbb{Z}_p| = 2^{\mathsf{poly}(n)} > 2^{2n}$ and 2^{-n} is negligible, GoSSamer-ACSS satisfies ACSS completeness.

Proof. If an honest node outputs, by OSV correctness, at least $t+1$ honest nodes initiated OSV; for each such $\mathcal{P}_i$ the received shares satisfy $v(x,i) = \mathbf{r}\cdot\mathbf{A}(x,i)+E(x,i)$; and by OSV completeness, all honest nodes output done eventually. Any honest node detecting an invalid share forwards $(\mathbf{A}(e,i), E(e,i))$ to all others with $v(e,i) \neq \mathbf{r}\cdot\mathbf{A}(e,i)+E(e,i)$, so the malicious dealer is confirmed and recovery starts. The output of OSV indicates that at least $t+1$ honest nodes hold valid shares and perform Lines 8-9, Algorithm 2. For each u, node $\mathcal{P}_u$ collects from $\ge t+1$ honest senders the pairs $(\mathbf{A}(u,i), E(u,i))$ that verify $v(u,i) = \mathbf{r}\cdot\mathbf{A}(u,i)+E(u,i)$, interpolates the t-degree column $\mathbf{A}(u,y)$. and forwards $\mathbf{A}(u,\nu)$ to $\mathcal{P}_\nu$. As all honest nodes can output $\mathbf{A}(u,y)$, each $\mathcal{P}_\nu$ further gathers $\ge n-t$ valid evaluations, robustly interpolates $(\mathbf{A}(x,\nu), E(x,\nu))$, decodes and outputs $\mathbf{A}(x,\nu)$ as its shares.

Reconstructed any $t+1$ honest columns into $\mathbf{A}(x,y)$. For any honest $\mathcal{P}_u$ that outputs $(\tilde{\mathbf{A}}(x,u), \tilde{E}(x,u))$, $v(x,u) = \mathbf{r}\cdot\mathbf{A}(x,u) + E(x,u) = \mathbf{r}\cdot\tilde{\mathbf{A}}(x,u) + \tilde{E}(x,u)$. If $\mathbf{A}(x,u) \neq \tilde{\mathbf{A}}(x,u)$, then $\mathbf{r}\cdot(\mathbf{A}(x,u)-\tilde{\mathbf{A}}(x,u))+(E(x,u)-\tilde{E}(x,u)) = 0$, a nontrivial polynomial constraint on $\mathbf{r}$. By Lemma 2 its probability is $\le 1/|\mathbb{Z}_p|$ per u; a union bound over $\le 2^n$ events yields error $\le 2^n/|\mathbb{Z}_p|$, negligible when $|\mathbb{Z}_p| > 2^{2n}$. Hence, with overwhelming probability, all honest nodes receive the correct rows on $\mathbf{A}(x,y)$. With decoding process in Lemma 1, each honest node outputs $\tilde{\mathbf{f}}(i)$ on $\tilde{\mathbf{f}}(x) \in \mathbb{Z}_p[x]^b_{\le t}$. Thus, GoSSamer-ACSS satisfies completeness. □

Simulator $\mathcal{S}_{\mathsf{acss}}$

input: (n,t), batch size b, field $\mathbb{Z}_p$, corrupted node set $\mathcal{C} \subseteq [n]$ and $|\mathcal{C}| \le t$, secret shares $[\mathbf{a}]_i$ of b secrets $\mathbf{a}$ of corrupted nodes $\mathcal{P}_i \in \mathcal{C}$
notation: $b \equiv 0 \mod t+1$, the number of groups $m = b/(t+1)$

1: partition $[\mathbf{a}]_i$ into m groups of size $t+1$, interpolate each group, and obtain a vector of m polynomials $\mathbf{A}(x,i)$ of t-degree for $i \in \mathcal{C}$
2: uniformly sample a vector of m polynomials $\tilde{\mathbf{A}}(x,y)$ with (t,t)-degree, such that $\tilde{\mathbf{A}}(x,i) := \mathbf{A}(x,i)$ for $i \in \mathcal{C}$
3: uniformly sample m-dimension challenge vector $\tilde{\mathbf{r}} \leftarrow \mathbb{Z}_p^m$
4: uniformly sample (t,t)-degree $\tilde{v}(x,y)$ as the verification script
5: calculate blinding polynomial as $\tilde{E}(x,y) := \tilde{v}(x,y) - \tilde{\mathbf{r}}\cdot\tilde{\mathbf{A}}(x,y)$
6: calculate Merkle roots $(r_1, \ldots, r_n)$ from secure message distribution, and program the random oracle to output $\tilde{\mathbf{r}}$ on input the roots
7: distribute evaluations $(\tilde{\mathbf{A}}(1,j), \tilde{E}(1,j)), \ldots, (\tilde{\mathbf{A}}(n,j), \tilde{E}(n,j))$ to each node $\mathcal{P}_j$ for $j \in [n]$ through secure message distribution
8: simulate the rest of the protocol on behalf of all honest nodes

Figure 9: The simulator $\mathcal{S}_{\mathsf{acss}}$ of GoSSamer-ACSS

Lemma 4. GoSSamer-ACSS satisfies ACSS secrecy.

Proof. We employ a hybrid argument to prove that the adversary $\mathcal{A}$'s view in the ideal protocol are indistinguishable from its view in the real protocol. In the ideal world, we construct a simulator $\mathcal{S}_{\mathsf{acss}}$ that, given only the secret shares held by corrupted nodes, simulates the protocol with uniformly sampled secrets, as summarized in Figure 9.

Hybrid 0. This is a real-world execution of GoSSamer-ACSS.

Hybrid 1. Same as Hybrid 0, except that all honest nodes are simulated and honestly follow the protocol. In simulation, the vector of secrets $\mathbf{a} \leftarrow \mathbb{Z}_p^b$ and the blinding secret $q \leftarrow \mathbb{Z}_p$ are randomly sampled. The tuple $(\mathbf{a}, q)$ is then independently shared with committees com and com′ using two separate instances of the ACSS protocol. The indistinguishability between Hybrid 0 and Hybrid 1 follows from the fact that the distributions of $(\mathbf{a}, q)$ in both hybrids are identical.

Hybrid 2. Same as Hybrid 1, except that all honest nodes are simulated by the simulator $\mathcal{S}_{\mathsf{acss}}$ described in Figure 9. Specifically, the simulator replaces the real secret sharing polynomials $\mathbf{A}(x,y)$ with independently sampled polynomials $\tilde{\mathbf{A}}(x,y)$, which encode uniformly sampled secrets $\tilde{\mathbf{A}}(x,0)$ in place of real secrets $\mathbf{A}(x,0)$. For any corrupted node $\mathcal{P}_i$, the simulator ensures that its share-encoding polynomials $\tilde{\mathbf{A}}(x,i)$ remain the same as $\mathbf{A}(x,i)$. Subsequently, the

simulator randomly samples a verification script $\tilde{v}(x, y)$. $\tilde{E}(x, y)$ is thus computed as $\tilde{v}(x, y) - \mathbf{r} \cdot \tilde{\mathbf{A}}(x, y)$

For any corrupted i, in Hybrid 1, $v(x, i) = \mathbf{r} \cdot \mathbf{A}(x, i) + E(x, i)$ with independent, uniform $E(x, i)$; in Hybrid 2, $\tilde{v}(x, i) = \tilde{\mathbf{r}} \cdot \tilde{\mathbf{A}}(x, i) + \tilde{E}(x, i)$ with independent, uniform $\tilde{v}(x, i)$. Since $\tilde{\mathbf{r}}$ is uniform in the random-oracle model and $\tilde{\mathbf{A}}(x, i) = \mathbf{A}(x, i)$ for corrupted i, the joint distributions $(\mathbf{A}(x, i), v(x, y), E(x, i))$ and $(\tilde{\mathbf{A}}(x, i), \tilde{v}(x, y), \tilde{E}(x, i))$ are identical. SMD secrecy ensures no further leakage from honest nodes. Hence Hybrid 2 is indistinguishable from Hybrid 1, which is identical to Hybrid 0. Therefore, the simulator produces a view indistinguishable from the real execution, and secrecy established. □

Theorem 2. GoSSamer-ACSS is an ACSS protocol that satisfies correctness, completeness and secrecy against any P.P.T Byzantine adversary corrupting up to $t < n/3$ nodes.

Proof. The proof follows Lemma 1, 3, and 4. □

Appendix D. Analysis of Dual-Committee ACSS

Definition 4 - (Batched) Dual-Committee ACSS. Consider two committees, com and com′, whose parameters are (n, t) and (n', t'), respectively, where n (resp., n') is the committee size and t (resp., t') the corresponding threshold. Dual-committee ACSS executes an (n, t, b)-ACSS among com and an (n', t', b)-ACSS among com′. Both ACSS instances have the same dealer $\mathcal{P}$ and share an identical vector of b input secrets, denoted as $\mathbf{a} \leftarrow \mathbb{Z}_p^b$. A dual-committee ACSS protocol satisfies:

- *Correctness*: If the dealer $\mathcal{P}$ is honest and inputs the vector of secrets $\mathbf{a}$, then there exist two batches of b polynomials $\mathbf{f}(x)$ and $\mathbf{f}(x)'$ in $\mathbb{Z}_p[x]_{\leq t}^b$, each of degree at most t, such that $\mathbf{f}(0) = \mathbf{f}(0)' = \mathbf{a}$. Every honest node $\mathcal{P}_i$ in com outputs the share $[\mathbf{a}]_i = \mathbf{f}(i)$, and every honest node $\mathcal{P}_i'$ in com′ outputs the share $[\bar{\mathbf{a}}]_i = \mathbf{f}(i)'$.
- *Completeness*: If any honest node outputs, there exists polynomials $\tilde{\mathbf{f}}(x), \tilde{\mathbf{f}}(x)' \in \mathbb{Z}_p[x]_{\leq t}^b$ with $\tilde{\mathbf{f}}(0) = \tilde{\mathbf{f}}(0)'$, such that every honest node $\mathcal{P}_i$ in com outputs $\tilde{\mathbf{f}}(i)$ and every honest $\mathcal{P}_i'$ in com′ outputs $\tilde{\mathbf{f}}(i)'$ eventually.
- *Secrecy*: Dual-committee ACSS reveals no information about the secrets $\mathbf{a}$ given an honest dealer. Formally, there exists a P.P.T simulator $\mathcal{S}_{\text{dc-acss}}$ such that, given the shares of corrupted nodes and independently sampled secrets, it can produce a simulated view in the ideal world, such that the ideal and real worlds are indistinguishable from the view of $\mathcal{A}$.

Theorem 3. The lightweight dual-committee ACSS protocol, as specified in Lines 1-8 of Algorithm 4, implements a dual-committee ACSS protocol that satisfies correctness, completeness, and secrecy in Definition 4 against any P.P.T adversaries that corrupt up to $1/3$ nodes in each committee.

Proof. Correctness follows from ACSS guarantees directly. We sketch the proofs of completeness and secrecy next.

Simulator $\mathcal{S}_{\text{dc-acss}}$

input: field $\mathbb{Z}_p$, batch size b
input for committee com**:** (n, t), $\mathcal{C}$, secret shares $[\mathbf{a}^*]_i$ for $i \in \mathcal{C}$
input for committee com′**:** (n', t'), $\mathcal{C}'$, secret shares $[\bar{\mathbf{a}}^*]_i$ for $i \in \mathcal{C}'$

1: sample challenge vector $\mathbf{r} \leftarrow \mathbb{Z}_p^b$, program the random oracle $\mathcal{H}$ to output $\mathbf{r}$ on input next seed and seed′
2: sample polynomials $w(x), \bar{w}(x) \leftarrow \mathbb{Z}_p[x]_{\leq t}$ such that $w(0) = \bar{w}(0)$
3: uniformly sample secrets $\mathbf{a} \leftarrow \mathbb{Z}_p^b$
▷ *for committee* com
4-a: calculate the secret sharing polynomials $\mathbf{a}(x)$ of t-degree, such that $\mathbf{a}(0) = \mathbf{a}$ and $\mathbf{a}(i) = [\mathbf{a}^*]_i$ for all $i \in \mathcal{C}$
5-a: calculate $q(x) = w(x) - \mathbf{r} \cdot \mathbf{a}(x)$, denote $q = q(0)$
6-a: share $\mathbf{a}$ and q to com through ACSS, such that the secret sharing polynomials are $\mathbf{a}(x)$ and $q(x)$, and each node $\mathcal{P}_i$ outputs $(\mathbf{a}(i), q(i))$
▷ *for committee* com′
4-b: calculate the secret sharing polynomials $\bar{\mathbf{a}}(x)$ of t'-degree, such that $\bar{\mathbf{a}}(0) = \mathbf{a}$ and $\bar{\mathbf{a}}(i) = [\bar{\mathbf{a}}^*]_i$ for all $i \in \mathcal{C}'$
5-b: calculate $\bar{q}(x) = \bar{w}(x) - \mathbf{r} \cdot \bar{\mathbf{a}}(x)$, denote $\bar{q} = \bar{q}(0)$
6-b: share $\bar{\mathbf{a}}$ and $\bar{q}$ to com′ through ACSS, such that the secret sharing polynomials are $\bar{\mathbf{a}}(x)$ and $\bar{q}(x)$, and each node $\mathcal{P}_i'$ outputs $(\bar{\mathbf{a}}(i), \bar{q}(i))$
▷ *for committee* com *and committee* com′
7: execute the dual-committee sharing consistency verification protocol

Figure 10: The simulator $\mathcal{S}_{\text{dc-acss}}$ of dual-committee ACSS

Completeness. If an honest node outputs, then it (i) reconstructed $w(x)$ (or $\bar{w}(x)$) via robust interpolation and (ii) received $t+1$ identical $\bar{w}(0)$ (or $w(0)$) from the other committee, with $w(0) = \bar{w}(0)$. Intuitively, by ACSS completeness, all honest nodes in both committees eventually output.

Let $w(x), \bar{w}(x)$ be the polynomials reconstructed in com, com′. For at least $n - t$ (resp. $n' - t'$) indices i, $w(i) = \mathbf{r} \cdot [\mathbf{a}]_i + [q]_i$ and $\bar{w}(i) = \mathbf{r} \cdot [\bar{\mathbf{a}}]_i + [\bar{q}]_i$, and cross-checking yields $w(0) = \bar{w}(0)$, i.e., $\mathbf{r} \cdot \mathbf{a} + q = \mathbf{r} \cdot \bar{\mathbf{a}} + \bar{q}$. With $\mathbf{r}$ uniform, if $\mathbf{a} \neq \bar{\mathbf{a}}$ this imposes a nontrivial linear constraint on $\mathbf{r}$, which holds with probability $\leq 1/|\mathbb{Z}_p|$ under Lemma 2, negligible for $|\mathbb{Z}_p| = 2^{\text{poly}(n)}$. Thus, except with negligible probability, $\mathbf{a} = \bar{\mathbf{a}}$. By ACSS completeness, there exist t-degree polynomials $\mathbf{f}(x)$ and $\bar{\mathbf{f}}(x)$ with $\mathbf{f}(0) = \bar{\mathbf{f}}(0) = \mathbf{a}$, such that each honest $\mathcal{P}_i$ in com outputs $\mathbf{f}(i) = [\mathbf{a}]_i$ (resp. $\mathcal{P}_i'$ in com′ outputs $\bar{\mathbf{f}}(i) = [\bar{\mathbf{a}}]_i$). Hence, dual-committee ACSS satisfies completeness, as in Definition 4.

Secrecy. The proof strategy follows Lemma 4. Hybrid 0 is the real-world protocol execution. Hybrid 1 is the same as Hybrid 0, except that all honest nodes are simulated and honestly follow the protocol with randomly sampled secrets. Hybrid 0 and Hybrid 1 are indistinguishable due to the identical distributions of the secrets.

Hybrid 2 is the same as Hybrid 1, except that all honest nodes are simulated as specified in Figure 10. Specifically, the simulator replaces the real secrets $\mathbf{a}^*$ with randomly sampled $\mathbf{a}$, while ensuring that the secret shares for corrupted nodes remain unchanged. The simulator then randomly samples t-degree polynomials $w(x)$ and t'-degree $\bar{w}(x)$ with $w(0) = \bar{w}(0)$. It calculates the polynomials $q(x)$ and $\bar{q}(x)$ through $q(x) = w(x) - \mathbf{r} \cdot \mathbf{a}(x)$ and $\bar{q}(x) = \bar{w}(x) - \mathbf{r} \cdot \bar{\mathbf{a}}(x)$, where $\mathbf{r}$ is a randomly sampled challenge vector.

The difference between Hybrid 2 and Hybrid 1 arises from $\{q(i)\}_{i \in \mathcal{C}}$ and $\{\bar{q}(i)\}_{i \in \mathcal{C}'}$. By ACSS secrecy, addi-

tional information $\mathcal{A}$'s view is given by $w(x)$ and $\bar{w}(x)$. Because these values are randomly sampled, $q(x), \bar{q}(x)$ are indistinguishable from random, and $\{q(i)\}_{i\in\mathcal{C}}, \{\bar{q}(i)\}_{i\in\mathcal{C}'}$ are indistinguishable from those in Hybrid 1. Thus, Hybrid 2 is indistinguishable from Hybrid 1. The sequence implies that the adversary cannot distinguish between the real and the ideal worlds. Hence, the protocol satisfies secrecy. □

Appendix E. Analysis of GoSSamer-ADPSS

Theorem 4. Consider two (n,t)-committees and batch size $b = \mathsf{poly}(n)$. GoSSamer-ADPSS implements batched ADPSS with a total communication complexity of $\mathcal{O}(\kappa bn + \kappa n^4 \log n)$ and an amortized $\mathcal{O}(\kappa n)$ per secret, under $\mathcal{O}(1)$ rounds.

Proof. Let $\mathcal{O}_{\mathsf{acss}}(x)$ denote the communication complexity of using GoSSamer-ACSS to share x secrets, and let $\mathcal{O}_{\mathsf{mvba}}$ denote the complexity of MVBA. Let $m = b/(t+1)$, the communication in GoSSamer-ADPSS is:

$$n\mathcal{O}_{\mathsf{acss}}(m) + n\mathcal{O}(\kappa n^2) + \mathcal{O}_{\mathsf{mvba}} + \mathcal{O}(\kappa n^3) + \mathcal{O}(\kappa n^2 m)$$

Given $\mathcal{O}_{\mathsf{mvba}} = \mathcal{O}(\kappa n^3)$, $\mathcal{O}_{\mathsf{acss}}(x) = \mathcal{O}(\kappa x n + \kappa n^3 \log n)$, the total communication of GoSSamer-ADPSS is $\mathcal{O}(\kappa bn + \kappa n^4 \log n)$. When $b \geq \mathcal{O}(n^3 \log n)$, it is $\mathcal{O}(\kappa n)$ per secret. Moreover, the $\mathcal{O}(1)$ rounds include n parallel ACSS instances, an MVBA, and $\mathcal{O}(1)$ multicast instances. □

Theorem 5. GoSSamer-ADPSS is an ADPSS protocol that satisfies correctness, termination, and secrecy, as defined in Definition 2, against the adversaries as defined in Section 3.3.

Proof. Due to page limits, we present a proof sketch below.

Termination. If all honest nodes in the old committee com initiate, by the correctness of dual-committee ACSS, every honest node in both committees outputs in each honestly initiated instance, giving at least $n-t$ completed instances. The honest nodes in com then run MVBA and decide a common set $\mathcal{Q}_{\text{ss}}$ of dual-committee ACSS instances that will output; $\mathcal{Q}_{\text{ss}}$ is of size $n-t$ and is multicast to com′. Thus, all honest nodes can deterministically extract their mask shares from the dual-committee ACSS outputs indexed by $\mathcal{Q}_{\text{ss}}$. Next, each honest $\mathcal{P}_i \in$ com encodes for $\mathbf{F}(x,i)$ and sends evaluations $\mathbf{F}(u,i)$ to $\mathcal{P}_u$. As all honest nodes send this evaluation, robust interpolation yields $\mathbf{F}(u,y)$ to $\mathcal{P}_u$. It then multicasts $\mathbf{F}(u,0)$ to com′. As all honest nodes in com multicast this evaluation, all honest nodes in com′ can robustly interpolate $\mathbf{F}(x,0)$, decode $\mathbf{s}+\mathbf{u}$, and, calculate their new secret shares. Hence, termination holds.

Correctness. An honest output of $\mathbf{F}(x,0)$ in com′ implies robust interpolation succeeded, hence at least $t+1$ honest nodes in com multicast $\mathbf{F}(u,0)$; for any such u, obtaining $\mathbf{F}(u,y)$ by robust interpolation requires at least $t+1$ honest nodes $\mathcal{P}_i$ to have multicast the evaluations of $\mathbf{F}(x,i)$, which encode masked shares derived from the MVBA-decided set $\mathcal{Q}_{\text{ss}}$. Thus at least $t+1$ honest nodes obtained $\mathcal{Q}_{\text{ss}}$ and completed all dual-committee ACSS instances in $\mathcal{Q}_{\text{ss}}$. By MVBA agreement and termination, all honest nodes in com output the same $\mathcal{Q}_{\text{ss}}$. By dual-committee ACSS completeness, for each $j \in \mathcal{Q}_{\text{ss}}$ there exist vectors of t-degree polynomials $\mathbf{a}_j(x)$ and t'-degree polynomials $\bar{\mathbf{a}}_j(x)$ with $\mathbf{a}_j(0) = \bar{\mathbf{a}}_j(0)$ such that every honest $\mathcal{P}_i \in$ com outputs $[\mathbf{a}_j]_i = \mathbf{a}_j(i)$ and honest $\mathcal{P}'_i \in$ com′ outputs $[\bar{\mathbf{a}}_j]_i = \bar{\mathbf{a}}_j(i)$. Moreover, honest nodes extract mask shares $[\mathbf{u}]_i = \mathbf{u}(i)$ and $[\bar{\mathbf{u}}]_i = \bar{\mathbf{u}}(i)$ via a Vandermonde matrix, where $\deg \mathbf{u} \leq t$, $\deg \bar{\mathbf{u}} \leq t'$, and $\mathbf{u}(0) = \bar{\mathbf{u}}(0) = \mathbf{u}$. Each honest $\mathcal{P}_i \in$ com forms $\mathbf{F}(x,i)$ by interpolating $[\mathbf{s}]_i + [\mathbf{u}]_i = \mathbf{f}(i) + \mathbf{u}(i)$ and sends $\mathbf{F}(u,i)$, enabling each $\mathcal{P}_u$ to robustly interpolate $\mathbf{F}(u,y)$ and multicast $\mathbf{F}(u,0)$ to com′. All honest nodes in com′ robustly interpolate $\mathbf{F}(x,0)$ and decode $\mathbf{f}(0)+\mathbf{u}(0) = \mathbf{s}+\mathbf{u}$, then compute secret shares $[\bar{\mathbf{s}}]_i = \mathbf{s} + \mathbf{u} - \bar{\mathbf{u}}(i)$, i.e., $\bar{\mathbf{f}}(i)$ of t'-degree polynomials $\bar{\mathbf{f}}(x) = \mathbf{f}(0) + \mathbf{u}(0) - \bar{\mathbf{u}}(x)$ with $\bar{\mathbf{f}}(0) = \mathbf{f}(0) = \mathbf{s}$. Hence, correctness established.

Secrecy. The proof strategy follows Lemma 4. Hybrid 0 is the real-world execution of GoSSamer-ADPSS. Hybrid 1 is the same as Hybrid 0, except that the honest nodes are simulated and honestly follow the protocol, and the randomly sampled $(\mathbf{a}_i, q_i)$ constitutes the input to the dual-committee ACSS protocol for node $\mathcal{P}_i$. Hybrid 1 is indistinguishable from Hybrid 0 due to the identical distributions of input.

Hybrid 2 is the same as Hybrid 1, except that all honest nodes are simulated as specified in Figure 11. In hybrid 2, the original secrets $\mathbf{s}^*$ are replaced with the randomly sampled $\mathbf{s}$. The differences in the adversary's view are: (i) evaluations $\mathbf{F}(j,i)$ sent from honest $\mathcal{P}_i$ to corrupted $\mathcal{P}_j$, and (ii) evaluations $\mathbf{F}(j,0)$ multicast from honest $\mathcal{P}_j$. These values reveal nothing about $\mathbf{s}$ beyond $[\mathbf{s}]_i + [\mathbf{u}]_i, i \in [n]$. By the secrecy of dual-committee ACSS, corrupted mask shares leak no information about $\mathbf{u}$, so the view distribution depends only on $\mathbf{s}+\mathbf{u}$, which is identical in Hybrids 1 and 2. Hence Hybrid 2 is indistinguishable from Hybrid 1. Thus, the real and ideal executions are indistinguishable due to the sequence, and GoSSamer-ADPSS satisfies secrecy. □

Simulator $\mathcal{S}_{\mathsf{adpss}}$

public input: batch size b, field $\mathbb{Z}_p$, corrupted node set $\mathcal{C} \subseteq [n]$ and $|\mathcal{C}| \leq t$ in the old committee, $\mathcal{C}' \subseteq [n']$ and $|\mathcal{C}'| \leq t'$ in the new committee
private input: share $[\mathbf{s}^*]_i$ of each corrupted node $\mathcal{P}_i$, where $i \in \mathcal{C}$
notation: $m = b/(t+1)$

1: for each honest node $\mathcal{P}_i$ in the old committee, the simulator randomly samples a vector of m values $\mathbf{a}_i \leftarrow \mathbb{Z}_p^m$ and a blinding value $q_i \leftarrow \mathbb{Z}_p$
2: simulate Phase 1 of GoSSamer-ADPSS on behalf of all honest nodes and denote $\mathcal{Q}_{\text{ss}}$ as the output of MVBA
3: for each $i \in \mathcal{Q}_{\text{ss}}$, the dual-committee ACSS instance initiated by $\mathcal{P}_i$ outputs shares for all honest nodes, thus the simulator interpolates these shares into t-degree polynomials $(\mathbf{a}_i(x), q_i(x))$ and $(\bar{\mathbf{a}}_i(x), \bar{q}_i(x))$, which are the secret sharing polynomials in the old and new committees, respectively
4: for each honest node $\mathcal{P}_i$ in the old committee, the simulator extracts the share $[\mathbf{u}]_i$; for each honest node $\mathcal{P}'_i$ in the new committee, the simulator extracts the share $[\bar{\mathbf{u}}]_i$
5: sample a vector of random secrets $\mathbf{s}$, and sample the polynomials $\mathbf{s}(x) \leftarrow \mathbb{Z}_p[x]^b_{\leq t}$, such that $\mathbf{s}(0) = \mathbf{s}$ and $\mathbf{s}(i) = [\mathbf{s}^*]_i$ for all $i \in \mathcal{C}$
6: calculate $[\mathbf{s}]_i + [\mathbf{u}]_i = \mathbf{s}(i) + [\mathbf{u}]_i$ for each honest node $\mathcal{P}_i$, and simulate the rest of the protocol on behalf of all honest nodes

Figure 11: The simulator $\mathcal{S}_{\mathsf{adpss}}$ of GoSSamer-ADPSS

Appendix F.
Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

F.1. Summary

This paper presents GoSSamer, a protocol suite for Asynchronous Complete Secret Sharing (ACSS) and Asynchronous Dynamic Proactive Secret Sharing (ADPSS) that simultaneously achieves lightweight computation - using only hash functions and symmetric-key encryption and optimal linear per-secret communication. The key contributions are two new techniques - Batched Original-Evaluation Propagation (B-OEP) and Bivariate-Polynomial-based Degree Checking (BP-DC) – that eliminate the need for homomorphic commitments while preserving linear communication. The authors implement GoSSamer and evaluate it on geo-distributed AWS nodes, demonstrating runtime improvements compared with prior work.

F.2. Scientific Contributions

- 6. Provides a Valuable Step Forward in an Established Field.

F.3. Reasons for Acceptance

1) The paper provides a valuable step forward in an established field. GoSSamer is the first ACSS construction that only relies on symmetric key primitives while achieving amortized $O(\kappa n)$ communication cost per secret, resolving an important question. For sufficiently large batch sizes, the protocol demonstrates practical performance.
2) The paper introduces novel techniques with practical impact. B-OEP and BP-DC are novel and conceptually clean. The extension to ADPSS via dual-committee sharing consistency verification – which decouples proactive resharing from commitment-based ACSS – is a meaningful conceptual and technical advance. Experimental results on a WAN deployment confirm concrete efficiency gains.

F.4. Noteworthy Concerns

1) Minimum batch size dependency: The protocol has a larger additive term compared to the prior work SS24 and requires a batch size of $b \geq O(n^2 \log n)$ to achieve amortized linear per-secret communication.