Communication-Efficient (Proactive) Secure Computation for Dynamic General Adversary Structures and Dynamic Groups

Karim Eldefrawy[1], Seoyeon Hwang[2], Rafail Ostrovsky[3], and Moti Yung[4]

[1] SRI International, USA
[2] University of California Irvine, USA
[3] University of California Los Angeles, USA
[4] Columbia University and Google

Abstract. In modern distributed systems, an adversary's limitations when corrupting subsets of servers may not necessarily be based on threshold constraints, but rather based on other technical or organizational characteristics in the systems. For example, it can be based on the operating systems they run, the cost of corrupting insiders in a sub-organization, et cetera. This means that the corruption patterns (and thus protection guarantees) are not based on the adversary being limited by a threshold, but on the adversary being limited by other constraints, in particular by what is known as a *General Adversary Structure (GAS)*. GAS settings may come up in situations like large enterprises, computing and networking infrastructure of Internet Service Providers, data centers and cloud infrastructure, IT infrastructure of government agencies, computerized military systems, and critical infrastructure. We consider efficient secure multiparty computation (MPC) under such dynamically-changing GAS settings. During these changes, one desires to protect against and during corruption profile change, which renders some (secret sharing-based) encoding schemes underlying the MPC protocol more efficient than others when operating with the (currently) considered GAS.
One of our contributions is a set of novel protocols to efficiently and securely convert back and forth between different MPC schemes for GAS; this process is often called *share conversion*. Specifically, we consider two MPC schemes, one based on additive secret sharing and the other based on Monotone Span Programs (MSP). The ability to efficiently convert between the secret sharing representations of these MPC schemes enables us to construct *the first communication-efficient structure-adaptive proactive MPC protocol for dynamic GAS settings*. By structure-adaptive, we mean that the choice of the MPC protocol to execute in future rounds after the GAS is changed (as specified by an administrative entity) is chosen to ensure communication-efficiency (the typical bottleneck in MPC). Furthermore, since such secure collaborative computing may be long-lived, we consider the *mobile adversary setting*, often called the *proactive security setting*. As our second contribution, we construct communication-efficient MPC protocols that can adapt to the proactive security setting. Proactive security assumes that at each (well defined) period of time the adversary corrupts different parties and over time may visit the entire system and corrupt all parties, provided that in each period it controls groups obeying the GAS constraints. In our protocol, the shares can be refreshed, meaning that parties receive new shares reconstructing the same secret, and some parties who lost their shares because of the reboot/resetting can recover their shares. As our third contribution, we consider another aspect of global long-term computations, namely, that of the dynamic groups. It is worth pointing out that such setting with dynamic groups and GAS was not dealt with in existing literature on (proactive) MPC. In dynamic group settings, parties can be added and eliminated from the computation, under different GAS restrictions. We extend our protocols to this additional dynamic group settings defined by different GAS.

Keywords: secure multiparty computation, secret sharing, share conversion, dynamic general adversary structures, monotone span programs, proactive security

1 Introduction

Secure Multiparty Computation (MPC) is a general primitive consisting of several protocols executed among a set of parties, and has motivated the study of different adversary models and various new settings in cryptography [24, 14, 34, 33, 17, 7, 16, 9, 26, 3]. For groups with more than two parties, i.e., the multiparty setting, secret sharing (SS) is often an underlying primitive used in constructing MPC; SS also has other secure distributed systems and protocols applications [12, 25, 23, 13, 2, 18, 20, 21].

In typical arithmetic MPC, the underlying SS [36, 10] used is of the threshold scheme type, i.e., a dealer shares a secret s among n parties such that an adversary that corrupts no more than a threshold t of the parties

(called corruption threshold) does not learn s, while any $t+1$ parties can efficiently recover it. MPC protocols built on top of SS allow a set of distrusting parties $P_1, \ldots, P_n$, with private inputs $x_1, \ldots, x_n$, to jointly compute (in a secure distributed manner) a function $f(x_1, x_2, \ldots, x_n)$ while guaranteeing correctness of its evaluation and privacy of inputs for honest parties. The study of secure computation was initiated by [39] for two parties and [24] for three or more parties. Constructing efficient MPC protocols withstanding stronger adversaries has been an important program in cryptography, and witnessed significant progress since its inception, e.g., [8, 14, 34, 28, 17, 16, 9, 33, 3, 4].

Enforcing a bound on adversary's corruption limit renders the problem efficiently solvable, but such a bound may be (and is often) criticized as arbitrary for protocols with long execution times, especially when considering the so-called "reactive" functionalities that may continuously run a control loop. Such reactive functionalities become increasingly important as MPC is adopted to resiliently implement privacy-sensitive control functions in critical infrastructures such as power-grids or command-and-control in distributed network monitoring and defense infrastructure. In those two cases one should expect resourceful adversaries to continuously attack parties/servers involved in such an MPC, and given enough time, vulnerabilities in underlying software will eventually be found.

An approach to deal with the ability of adversaries to eventually corrupt all parties is the *proactive security model* [33]. This model introduces the notion of a mobile adversary motivated by the persistent corruption of parties in an MPC protocol. A mobile adversary is one that can corrupt all parties in a distributed protocol during the execution but with the following limitations: (i) only a constant fraction (in the threshold setting) of parties can be corrupted during any round of the protocol; (ii) parties periodically get rebooted to a clean initial state, guaranteeing small fraction of corrupted parties, assuming that the corruption rate is not more than the reboot rate[5]. The [33] model also assumes that an adversary does not have the ability to predict or reconstruct the randomness used by parties in any uncorrupted period of time, as demarcated by rebooting.

In most of the (standard and proactive) MPC literature, the adversary's corruption capability is characterized by a threshold t. More generally, however, the adversary's corruption capability could be specified by a so-called *general adversary structure (GAS)*, i.e., a set of potentially corruptible subsets of parties. Even more generally, the corruption ability of the adversary can be specified by a set of corruption scenarios, one of which the adversary can choose (secretly). For instance, each scenario can specify a set of parties that can be passively corrupted and a subset of them that can even be actively corrupted. Furthermore, such scenarios may change over time, thus effectively rendering the GAS describing them to itself be dynamic evolve over time. There are currently no proactive MPC protocols that can efficiently handle such dynamic general specifications of adversaries, especially when the group of parties performing the MPC is also dynamic.

Our main objective is to address a setting that is as close as possible to the complex dynamic reality of today's distributed systems. We accomplish this by answering the following question: *Can we design a communication-efficient proactively secure MPC (PMPC) protocol for dynamic groups with security against dynamic general adversary structures?*

Contributions: We answer the above question affirmatively. One of our main new contributions is a set of protocols to efficiently convert back and forth between different MPC schemes for GAS; this process is often called *share conversion*. Specifically, we consider an MPC scheme based on additive secret sharing and one based on Monotone Span Program (MSP). The ability to efficiently and securely convert between these MPCs enables us to construct *the first communication-efficient structure-adaptive proactive MPC (PMPC) protocol for dynamic GAS settings.* We stress that *all* existing proactive secret sharing and PMPC protocols (details in Appendix A.1 and Table 4 therein) can only handle (threshold) adversary structures that describe sets of parties with cardinality less than a fraction of the total number of parties.

Given the large number of "moving parts" and complexity of PMPC protocols and the additional complexity for specifying them for GAS, we start from a (standard, i..e, non-proactive) MPC protocol for static groups with GAS and extend it to the proactive setting and dynamic GAS. Note that MPC protocols typically extend secret sharing and perform computations on secret shared inputs, we thus focus the discussion in this paper on MPC with the understanding that results also apply to secret sharing.

As part of the proactive protocols, we support recovering shares of parties that lost them. This implies that we can also deal with dynamic sets of parties, where parties can be eliminated (not recovered in a refresh phase)

[5] In our model rebooting to a clean initial state includes global information, e.g., circuits to be computed via MPC, identities of parties in the computation, and access to secure point-to-point channels and a broadcast channel.

and a party can be added (i.e., start with a recovery of its shares in a refresh phase). We also deal with settings where the entire set of parties changes and the existing shared data has to be moved to a new set of parties with a possibly new specification of the GAS they should protect against. This original set of parties then redistributes the shared secrets to the latter new set (which may, or may not, have some overlap with the original set).

Paper Outline: We first motivate the need for secure computation for dynamic general adversary structures and dynamic groups in section 2. Section 3 overviews the typical blueprint of PMPC, briefly discusses related work (with more discussion in Appendix A.1), and discusses roadblocks and challenges facing constructing communication-efficient structure-adaptive PMPC protocols for dynamic groups and dynamic GAS settings. Section 4 contains necessary preliminaries and specifications of underlying network and communication models, the adversary model, and some other basic building blocks required in the rest of the paper. Section 5 contains the details of the new protocols developed in this paper (with security and correctness proofs in Appendix.

2 The Need for Secure Computation for Dynamic Groups with Changing Specifications of the (General) Adversary Structures

Large networked systems, such as public clouds, global clouds, the infrastructure of companies, and so on, are managed for their security and reliability by specialists who employ tools, measurements, and reporting systems (including AI tools nowadays). These specialists maintain such large systems while facing changes and failures. This methodology of managing large systems is known as *DevOps* which is a set of practices that combines software development (Dev) and information-technology operations (Ops) that aims to shorten the systems development life cycle and provide continuous delivery with high software quality and system reliability [30]. In particular (starting with Google) the profession of such people, performing these tasks, is called *Site Reliability Engineering (SRE).* Some of the responsibilities of SRE include: (1) Reduce organizational silos (separate sections of engineers to create joint coherent responsibilities in large systems with various elements cooperating); (2) Accept failure as normal (and react to failures such as security breaches, overloading of subsystems, etc. managing system configuration with responsiveness and agility); (3) Implement gradual changes (long term maintenance based on past issues and future needs as they come or envisioned); (4) Leverage tooling and automation (as the large system need to be controlled remotely and effectively it cannot be done manually and control tools are needed); and (5) Measure everything (constantly monitoring the needs and act according to the data, while keeping statistics of systems performance).

A modern Information Security concept in managing large systems against threats is Moving Target Defense (MTD), which is the method of controlling change across multiple system dimensions in order to increase uncertainty and apparent complexity for attackers, reduce their window of opportunity and increase the costs of their probing and attack efforts. MTD assumes that perfect security is unattainable and adds system changes as increased challenges to the potential attacker).

One can view an SRE team getting information about and reacting to a system's suspicious behavior (at some parts of the network) and employing analysis which dictates configuration change. One can also view the team as occasionally and proactively, for the sake of implementing MTD strategy, calling the network of servers to rearrange itself in a different fashion than the current setting, in the (general) scenarios we consider, this would correspond to changing the specification of the general adversary structure being protected against. When the team manages the configuration, they employ a secure and authenticated control and command system over servers, they can notify servers to reconfigure and organize their distributed data according to some protocol and dictated parameters. In our treatment we assume that such a system is available in our underlying secure computation system and we augment existing configuration tools with the ability to manage and dynamically change the underlying "secret sharing" settings among the network's server.

3 Overview of Proactive MPC and Design Roadblocks

This section reviews the typical blueprint of Proactive MPC (PMPC) protocols. Due to space constrains, we discuss related work in more detail in Appendix A.1. We conclude this section by a discussion of roadblocks facing designing communication-efficient structure-adaptive PMPC protocols for GAS.

3.1 Blueprint of Proactive Secret Sharing (PSS) and Proactive MPC (PMPC)

PMPC protocols [33, 3] are usually constructed on top of (linear) secret sharing schemes, and involve alternating compute and refresh (and reboot/reset) phases. The refresh phases involve distributed rerandomization of the secret shares, and deleting old ones to ensure that a mobile adversary does not obtain enough shares (from the same phase) that can allow them to violate secrecy of the shared inputs and intermediate compute results. A PMPC protocol usually consist of the following six sub-protocols:

1. `Share`: allows a dealer (typically one of the parties) to share a secret s among the n parties.
2. `Reconstruct`: allows parties to reconstruct a shared secret s using the shares they collectively hold.
3. `Refresh`: is executed between two consecutive phases, w and $w+1$, and generates new shares for phase $w+1$ that encode the same secret as shares in phase w, but are independent of shares of the previous phases.
4. `Recover`: allows parties that lost their shares (due to rebooting/resetting or other reasons) to obtain new shares encoding the same secret s, with the help of other online parties.
5. `Add`: allows parties holding shares of two secrets s and t to obtain shares encoding the sum $s+t$.
6. `Multiply`: allows parties holding shares of two secrets s and t to obtain shares encoding the product $s \cdot t$.

The overall operation of a standard PMPC protocol is as follows: First each party uses the `Share` sub-protocol to securely distribute its private inputs among the n parties (including itself). The function to be computed on the inputs of parties is transformed into an arithmetic circuit that is public. The circuit to be computed is composed of multiple layers (the depth of the circuit) each consisting of a set of `Add` and `Multiply` gates which are computed via the corresponding sub-protocols one layer at a time. At the end of each circuit layer[6] shares of all nodes can be refreshed via the `Refresh` protocol and old shares are deleted; refreshing and deleting old shares ensures that different shares collected by the adversary at different phases can not be used together to reconstruct the secret shared inputs and intermediate and final results of the computation. In addition, during refresh phases, some nodes are randomly reset/rebooted, these then use the `Recover` protocol to obtain new shares encoding the same shared secrets corresponding to the current state of the PMPC computation, i.e., the output of the current circuit layer and any shard values that will be needed in future layers. When the (secret shared) output of the final layer of the computation is produced, parties use the `Reconstruct` protocol to compute the final output in the clear (or towards whichever nodes are supposed to obtain it).
To deal with dynamic groups, where parties can leave, or new parties can join the group, the following additional sub-protocol `Redistribute` is required:

7. `Redistribute`: is executed between two consecutive protocol phases, w and $w+1$, and allows parties in a new group (in phase $w+1$) to obtain new shares that encode the same secret as the shares in phase w.

In addition, we observe that the specifics of the secret sharing-based encoding underlying the PMPC protocol largely dictates the communication-efficiency. This is an issue that is often overlooked and that does not appear when one only considers the threshold adversary structure as opposed to GAS. For example, if one considers an additive secret sharing scheme similar to the one used in the MPC protocol in [32], and if the adversary structure one should protect against is the threshold one, then there is an exponential blowup in the share size compared to a monotone span program (MSP) based scheme. Thus any protocols that requires transmitting such shares encoded additively, e.g., multiplication, recovery, or redistribution of shares, is going to be inefficient compared to an MSP-based one. A communication-efficient protocol should thus be structure-adaptive when considering evolving GAS, this means that if the set of parties performing the MPC receive (from an administrator) a request to adapt to a new GAS, for which it is known that another (secret sharing) encoding scheme is more efficient, they need to convert. We stress that this is different than the `Redistribute` protocol, which re-shares a shared secret, but *with the same secret sharing scheme.* We require a non-trivial additional protocol to perform such conversion:

8. `Convert`: is executed between two consecutive protocol phases, w and $w+1$, and allows parties in a new group defined by a new GAS (in phase $w+1$) to obtain new shares under a different secret sharing scheme but that encode the same secret as the shares in phase w (under the old secret sharing scheme and the old GAS).

[6] Or after several layers, or at the end of one execution of a circuit of reactive functionalities executing in a loop. In this paper we do not specify when parties should refresh shares, we just develop the protocol to accomplish this.

3.2 Roadblocks Facing PMPC for Dynamic General Adversary Structures and Dynamic Groups

Starting with an appropriate SS scheme and an MPC protocol that can handle GAS, one has to address the following to design a communication-efficient PMPC protocol for dynamic groups and dynamic GAS:

1. *Design a convert protocol to be structure-adaptive:* Given that we are considering settings with changing GASs, and given that some (secret sharing) encoding schemes underlying MPC result in different communication complexity, we design new efficient protocols (secure against GASs) to convert between different secret sharing schemes. We consider converting from an additive sharing to an MSP-based sharing, and the opposite direction. Such conversion protocols may be of independent interest.
2. *Design refresh and recover protocols to proactivize the underlying SS scheme:* This enables the parties to able to securely rerandomize the shares in a secure distributed manner. To enable rebooted/reset parties to recover their shares, and not to loose shared inputs or intermediate results of the computation over time, rebooted parties have to be able to recover shares with the help of the rest of the parties.
3. *Design a redistribute protocol for dynamic groups settings:* Dealing with dynamic groups, which means parties can leave and/or newly join in the group of computation, the GAS as well as the number of parties in the group might be changed. Therefore, one has to redistribute new shares to parties in the new group which encoding the same secret as the shares in the previous group, but also needs to prevent leaving parties from using their shares to obtain any information about the secret.
4. *Efficient communication in all protocols:* All the involved protocols should be efficient, e.g., ideally linear dependence on the specification of adversary structures and the number of parties, or at least polynomial. We note that in this work we do not attempt to minimize the descriptions of the adversary structures, i.e., the size of specifications of some structures may be exponential in the number of parties n.

4 Preliminaries

This section provides preliminaries required for the rest of the paper. Table 1 summarizes the notion used in this paper. We then discuss the underlying communication model, security guarantees, and then introduce the terminology and specifications of GAS. The section concludes by reviewing the information checking (IC) used in MPC protocols [27, 29], on which we build our protocol as a building block.

Notation	Explanation
$\mathcal{P} = \{P_1, ..., P_n\}$	the set of all participating parties in a protocol
(Γ, Σ, Δ)	access/secrecy/adversary structures in a GAS
$\mathbb{S}$	a sharing specification
$w, w+1$	a phase (number)
$[s]^w$	a sharing of s in phase w
I_j	a set of indices of shares that P_j has
R	a set of all parties who lost its share (need recovery)
$\mathbb{F}$	a finite field
M	a matrix from a monotone span program (MSP) $\widehat{M} = (\mathbb{F}, M, \rho, \mathbf{r})^*$
$\mathbf{a}$	a target vector of a MSP
ρ	an indexing function of a MSP
M_i	a matrix of rows of M assigned to P_i according to ρ
M_A	a matrix of rows of M assigned to all $P_i \in A$ according to ρ
$\langle \, , \, \rangle$	the inner product
$a \xleftarrow{\$} \mathbb{F}$	randomly chosen element a from the finite field $\mathbb{F}$

Table 1. Notations used in this paper. GAS denotes general adversary structure.

4.1 Adversary Model, Communication Model, and Security Guarantees

In this work, we consider protocols with **unconditional security** for both passive and active adversaries. In terms of the communication model, we consider **a synchronous network** of n parties connected by an authenticated **broadcast channel**. The different security guarantees and communication models in the MPC literature are discussed in Appendix A in more detail. We consider the adversarial capabilities in terms of the **general adversary structure (GAS)**, which is more general and flexible notion to reason about adversaries (compared to only the threshold limitation on corruptions) and applicable to various case, e.g. when special combination of parties is needed, when some parties are authorized, etc. the terminology and formalization of GAS are summarized below.

Let $2^{\mathcal{P}}$ denote the set of all the subsets of $\mathcal{P}$. A subset of $2^{\mathcal{P}}$ is called **qualified** if parties in the subset can reconstruct/access the secret, while a subset of $2^{\mathcal{P}}$ that parties in the set obtain no information about the secret is called **ignorant**. Every subset of $\mathcal{P}$ is either qualified or ignorant. The secrecy condition is stronger: even if any ignorant set of parties holds any kind of partial information about the shared value, they must not obtain any additional information about the shared value.

Definition 4.1. *(Access Structure and Secrecy Structure)*
*The **access structure** Γ is the set of all qualified subsets of $\mathcal{P}$ and the **secrecy structure** Σ is the set of all ignorant subsets of $\mathcal{P}$. Naturally, Γ includes all supersets of each element in it (so often called monotone access structure), while Σ includes all subsets of each element in it. We call such a minimized set as **basis structure**, and denote with $\widetilde{\cdot}$. i.e., the basis access structure $\widetilde{\Gamma}$ is the set of all minimal subsets in Γ, and the basis secrecy structure $\widetilde{\Sigma}$ is the set of all maximal subsets in Σ.*

As a generalization of specifying the adversary's capabilities by a corruption type (passive or active) and a threshold t, an adversary can be described by a corruption type and an adversary structure Δ. The **adversary structure** Δ is a set of subsets of parties that can be potentially corrupted. The adversary can choose a set in Δ and corrupt all the parties in the set. Note that the adversary structure in t-threshold SS is the set of all subsets of $\mathcal{P}$ of at most t parties and GAS extends this to non-threshold models. In GAS, the types of adversaries can be classified as below.

Definition 4.2. *(Passive/Active Adversary in GAS)*
*In GAS, an adversary is specified by the secrecy structure Σ and the adversary structure $\Delta \subseteq \Sigma$. A **passive adversary** can only perform passive corruptions on Σ, eavesdropping on all the inputs and outputs of corrupted party in Σ. i.e., $\Delta = \Sigma$. On the other hand, an **active adversary**, which is also called (Σ, Δ)-adversary, can passively corrupt some parties in a set A and actively corrupt some parties in a set B, where $A \in \Delta$ and $(A \cup B) \in \Sigma$.*

In real-world scenarios, it is natural to deal with **dynamic groups**, which means participating parties can leave and/or newly join in the group of computation. Then, the GAS (Γ, Σ, Δ) as well as the number of parties $|\mathcal{P}|$ in the group might be changed. Therefore, to deal with dynamic groups, we need one more protocol, which redistributes new shares to the new group which encodes the same secret as the ones in the previous group. It also needs to prevent leaving parties from using their shares to obtain any information about the secret.

4.2 Information Checking (IC) and Dispute Control

In MPC protocols that we are considering, a technique, called information checking (IC), is used to prevent active adversaries from announcing wrong values through corrupted parties. It is a three party protocol among a sender P_s, a receiver P_r, and a verifier P_v. When P_s sends a message m to P_r, P_s also encloses an authentication tag to P_r, while giving a verification tag to P_v. Whenever any disagreement about what P_s sent to P_r happens, P_k acts as an objective third party and verifies the authenticity of m to P_r. The MPC protocols in this paper use different variants of information checking but the common idea is to check if all the points that P_r and P_v have lie on the polynomial of degree 1. Note that this can be expanded to the polynomial of degree l, the number of secrets in a batch of sharing, as in [29]. The protocols for information checking are `Authenticate` and `Verify` in each scheme, which is presented in Appendix B.1 and D.1.

MPC protocols in this paper also use the dispute control to deal with detected cheaters. That means, each party P_i locally maintains a list $\mathcal{D}_i$ of parties that P_i distrusts, and the list $\mathcal{D}$ of pairs of parties who are in dispute with each other. These lists are empty when the MPC protocol begins, and whenever any dispute arises between two parties P_i and P_j (for example, P_i insists that P_j is lying), the pair $\{P_i, P_j\}$ is added to the dispute list $\mathcal{D}$. Since all disputes are broadcasted, each party P_i has the same list $\mathcal{D}$, while maintaining its own list $\mathcal{D}_i$. After P_j is added to $\mathcal{D}_i$, P_i behaves in all future invocations of the protocol for authentication and verification with P_j as if it fails whether this is the case or not. Some MPC schemes also maintain a list $\mathcal{C}$ of parties that everyone agrees are corrupted.

5 Proactive MPC Protocols for Dynamic GAS and Dynamic Groups

Our communication-efficient PMPC protocols build on two MPC protocols with different underlying (secret sharing based) encoding schemes. One is an MPC protocol [27] based on additive secret sharing and the other [29]

is based on a monotone span program (MSP) with multiplication. For convenience, we call the former as *additive MPC* and the latter as *MSP-based MPC*. Both guarantees unconditional security against active $Q2$ adversaries, which means no two sets in Δ cover the entire party set; i.e., for $\forall A, B \in \Delta$, $\mathcal{P} \not\subseteq A \cup B$.

In Section 5.1 and Section 5.2, we show the additive PMPC and MSP-based PMPC schemes, respectively. Due to the space limit, we formalize the based protocols of [27] and [29] in Appendix B and D, and focus on our new protocols. For "proactivizing" a MPC scheme, we build two (or three, for dynamic groups) new main protocols, called `Refresh` and `Recover` (and `Redistribute`, for dynamic groups). The resulting PMPC scheme is composed of 6 protocols, `Share`, `Reconstruct`, `Add`, `Multiply`, `Refresh`, and `Recover` (or 7 protocols for dynamic groups including `Redistribute`). We denote protocols with superscripts, `A` or `M`, for additive PMPC and MSP-based PMPC, respectively. All the proofs for our protocols are presented in Appendix C and E. Note that the complexity of additive PMPC protocols depends on $|\widetilde{\Sigma}|$ and n, while the one of MSP-based PMPC protocols depends on d and n, where n is the number of participating parties, $|\widetilde{\Sigma}|$ is the size of the set of all maximal subsets in the secrecy structure, and d is the number of rows of the MSP matrix.

In Section 5.3, we develop conversion protocols between these two PMPC schemes to provide users options to adapt the utilized protocol according to the dynamic GAS. As we mentioned in Section 2, this is necessary and important because one can become more communication-efficient than the other depending on the circumstances. Considering the upper bound on d is about $|\widetilde{\Sigma}|^{2.7}$ [29], the MSP-based MPC is more expensive than the additive MPC, but d can be also low as $n = |\mathcal{P}|$ in some cases, which makes the MSP-based MPC more communication-efficient. The proofs for these protocols are presented in Appendix F.

5.1 Additive PMPC

We build our additive PMPC protocol on top of Hirt and Tschudi's unconditional MPC protocol [27] based on additive secret sharing. Due to space limitations, we briefly review their protocol in this section with formal specifications presented in Appendix B. Also, all proofs for our new protocols are provided in Appendix C.

Assuming n participating parties $\mathcal{P} = \{P_1, ..., P_n\}$ per notation in Table 1, we denote the sharing of a value s by $[s]$ and the sharing specification by $\mathbb{S} = (S_1, ..., S_k)$. Any sharing specification which is Δ-*private*, i.e. for every $Z \in \Delta$, $\exists S \in \mathbb{S}$ such that $S \cup Z = \emptyset$, can be used to securely share a secret. We adopt one from [32], which is $\mathbb{S} = (S_1, ..., S_k)$, where $S_i = \mathcal{P} \setminus T_i$ for $\widetilde{\Sigma} = \{T_1, ..., T_k\}$, the set of all maximal subsets in Σ. In [27], they use an IC scheme for dealing with active adversaries, which consists of $\texttt{Authenticate}^{\texttt{A}}$ and $\texttt{Verify}^{\texttt{A}}$. $\texttt{Authenticate}^{\texttt{A}}(P_s, P_r, P_v, m)$ is for P_s to distribute the authentication tag of m to P_r and the verification tag of m to P_v, and $\texttt{Verify}^{\texttt{A}}(P_s, P_r, P_v, m', tags)$ is for P_j to request P_v to verify the value m' with an authentication tag and a verification tag. Each protocol is presented in Appendix B.1 in detail.

Definition 5.1. *A value s is* ***shared*** *with respect to a sharing specification $\mathbb{S} = (S_1, ..., S_k)$ by additive secret sharing, if the following holds:*

a) There exists shares $s_1, ..., s_k$ such that $s = \sum_{i=1}^{k} s_i$.
b) Each s_i is known to every party in S_i, for all i.
c) $\forall P_s, P_r \in S_i$, $\forall P_v \in \mathcal{P}$, P_v can verify the value s_i using the IC, for each i.

A secret value s is shared among $\mathcal{P}$ through the protocol $\texttt{Share}^{\texttt{A}}$ and any qualified subgroup B of $\mathcal{P}$ can reconstruct the secret value through $\texttt{Reconstruct}^{\texttt{A}}$. In $\texttt{Share}^{\texttt{A}}$ protocol, a dealing party randomly chooses $k-1$ values in $\mathbb{F}$, sets the k-th value as $s - \sum_{i=1}^{k-1} s_i$, and sends each i-th value to every player in S_i. Then multiple $\texttt{Authenticate}^{\texttt{A}}$ are invoked to generate the IC tags. In $\texttt{Reconstruct}^{\texttt{A}}$, parties in B verify the forwarded values of each share from the others and reconstruct the secret value by locally adding all the verified share values. The formal protocols are presented in Appendix B.2. Note that the sharing of s is linear and does not leak any information about s without the whole set of sharing. Assuming the shares for the values s and t are already shared among $\mathcal{P}$, addition of s and t can be done naturally even without any interaction among n parties by the linearity. However, multiplication is quite tricky and requires a lot of communications to securely form the share of $(s * t)$ among n parties, as $s \cdot t = \sum_{i=1}^{k} \sum_{j=1}^{k} (s_i \cdot t_i)$. All the protocols for $\texttt{Add}^{\texttt{A}}$ and $\texttt{Multiply}^{\texttt{A}}$ are presented in the Appendix B.3.

To make this additive MPC scheme to be a PMPC that can also handle dynamic groups, we build three protocols, called $\texttt{Refresh}^{\texttt{A}}$, $\texttt{Recover}^{\texttt{A}}$, and $\texttt{Redistribute}^{\texttt{A}}$. Note that the protocol $\texttt{Redistribute}^{\texttt{A}}$ is only required for the dynamic group setting. The protocol $\texttt{Refresh}^{\texttt{A}}$ periodically refreshes or rerandomizes the shares in a distributed manner. This can be done naturally by every party's sharing zero and locally adding all the received

shares to the current holding share. The execution of this protocol does not reveal any additional information about the secret as only the shares of zeros are communicated. The security proof for this protocol is provided in Appendix C.1.

Protocol $\texttt{Refresh}^{\texttt{A}}(w, [s]) \longrightarrow [s]^{w+1}$

Input: a phase w and a sharing of s
Output: new sharing of s in phase $w+1$, $[s]^{w+1}$

1. Every party P_i in $\mathcal{P}$ invokes $\texttt{Share}^{\texttt{A}}(w, 0, \mathcal{P})$. (in parallel)
2. Each party adds all shares received in Step 1 to shares of s, and set result as the new share of s in phase $w+1$.
3. parties in $\mathcal{P}$ collectively output $[s]^{w+1}$.

Theorem 5.1. *(Correctness and Secrecy of* $\texttt{Refresh}^{\texttt{A}}$*) When the protocol* $\texttt{Refresh}^{\texttt{A}}$ *terminates, all parties receive new shares encoding the same secret as old shares they had before with error probability* $n^4|\mathbb{S}|/|\mathbb{F}|$*, and they cannot get any information about the secret by the execution of the protocol. It communicates* $|\mathbb{S}|(7n^4+n^2)\log|\mathbb{F}|$ *bits and broadcasts* $|\mathbb{S}|((3n^4+n)\log|\mathbb{F}|+n^3)$ *bits.*

For the protocol $\texttt{Recover}^{\texttt{A}}$, we construct the following two sub-protocols, $\texttt{ShareRandom}^{\texttt{A}}$ and $\texttt{RobustReshare}^{\texttt{A}}$. $\texttt{ShareRandom}^{\texttt{A}}$ generates a sharing of a random element r in $\mathbb{F}$ and parties in the same S_i receive the i-th share of r for each i, but the value of r is not revealed to anyone. It broadcasts at most $O(|\mathbb{S}|n\log|\mathbb{F}|)$ bits and no communications is required.

Protocol $\texttt{ShareRandom}^{\texttt{A}}(w, \mathcal{P}) \longrightarrow [r]^w$

Input: a phase w and a set of participating parties $\mathcal{P}$
Output: a sharing $[r]$ of a random number r, shared among $\mathcal{P}$

1. For each $S_q \in \mathbb{S} = \{S_1, ..., S_k\}$:
2. Each party $P_i \in S_q$ generates a random number r_{qi} and broadcast it among all parties in S_q.
3. Each $P_i \in S_q$ locally adds up all values received in Step 2, and set it as r_q.
4. The parties in $\mathcal{P}$ collectively output $[r]$, where $r = \sum_{q=1}^{k} r_q$.

The protocol $\texttt{RobustReshare}^{\texttt{A}}$ allows parties in $\mathcal{P}_R \in \Gamma$ to receive a sharing of r (with the value of r) from the parties in $\mathcal{P}_S$ where everyone in $\mathcal{P}_S$ knows the value of r. Distributing one sharing of r is non-trivial in the active adversary model because we cannot trust one party who might be corrupted to share a sharing of r. Let $Honest := \{\mathcal{P} \setminus A \mid A \in \overline{\Delta}\}$, where $\overline{\Delta}$ is the set of all maximal subsets in Δ, the adversary structure. Since the adversary can corrupt one set of parties in Δ in each phase, there exists at least one set of parties in $Honest$ that includes only honest parties in that phase. The main idea of the protocol $\texttt{RobustReshare}^{\texttt{A}}$ below is to find such set by repeating to share and reconstruct for each party's holding value for r. At the end of the protocol, parties in $\mathcal{P}_R$ can set a sharing of r and also know the value of the random number r.

Protocol $\texttt{RobustReshare}^{\texttt{A}}(w, r, \mathcal{P}_S, \mathcal{P}_R) \longrightarrow [r]^w$

Input: a phase w, random number r, a set $\mathcal{P}_S$ of parties sending r, and a set $\mathcal{P}_R \in \Gamma$ of receiving parties
Output: a sharing of r in phase w, $[r]^w$

1. Every party in $\mathcal{P}_S$ executes $\texttt{Share}^{\texttt{A}}(w, r, \mathcal{P}_R)$ according to the sharing specification $\mathbb{S}_R$ on $\mathcal{P}_R$. Let $[r]^{(i)}$ be the sharing of r that $P_{k_i} \in \mathcal{P}_S$ shares.
2. Parties in $\mathcal{P}_R$ invoke $\texttt{Reconstruct}^{\texttt{A}}([r]^{(i)}, \mathcal{P}_R)$, for each $i = 1, 2, ..., |\mathcal{P}_S|$. Let $r^{(i)}$ be the output of each invocation.
3. Each party chooses a set $H \in Honest$ such that $\exists v, v = r^{(i)}$ for all $P_{k_i} \in H$. If there are multiple such sets, choose the minimal set including P_i with lower id, i.

4. Output the sharing of r from the party P_i in H with the minimum id, i.
i.e. Output $[r] \leftarrow [r]^{(min)}$, where $min := \min_{P_i \in H}\{i\}$.

The security of $\texttt{RobustReshare}^{\texttt{A}}$ relies on the security of $\texttt{Share}^{\texttt{A}}$ and $\texttt{Reconstruct}^{\texttt{A}}$, as the rest is executed locally. For complexities, as both protocols $\texttt{Share}^{\texttt{A}}$ and $\texttt{Reconstruct}^{\texttt{A}}$ are invoked for each party in $\mathcal{P}_S$ and $|Honest| = |\overline{\Delta}| \leq |\overline{\Sigma}| = |\mathbb{S}|$, the total communication and broadcast complexities of $\texttt{RobustReshare}^{\texttt{A}}$ is $O(|\mathbb{S}|n^3 + |\mathcal{P}_S||\mathbb{S}|n^3 + |Honest|) = O(|\mathcal{P}_S||\mathbb{S}|n^3)$. The total analysis of all additive PMPC protocols is shown in Table 2.

The protocol $\texttt{Recover}^{\texttt{A}}$ allows rebooted/reset parties to obtain new shares for the same secret with the assistance of other parties. Let $R \subset \mathcal{P}$ be a set of parties who need to recover their shares. Note that $\mathcal{P} \setminus R$ must be still in Γ for the protocol $\texttt{Recover}^{\texttt{A}}$ to output the new sharing because otherwise, it contradicts to the definition of the access structure. It needs the condition $\mathcal{Q}^1(S_q, \mathcal{Z})$, which is already a necessary condition for the protocol $\texttt{Reconstruct}^{\texttt{A}}$. The main idea is as follows: a sharing of unknown random value r is generated among entire parties in $\mathcal{P}$ by $\texttt{ShareRandom}^{\texttt{A}}$ and the parties in $\mathcal{P} \setminus R$ who hold the shares of s reshare the value $r' = r + s$ and a sharing of r' to entire parties. Then, all parties including R can compute the new shares of s by $[r'] - [r]$. The proof for Theorem 2 is presented in Appendix C.2.

Protocol $\texttt{Recover}^{\texttt{A}}(w, [s], R) \longrightarrow [s]^{w+1}$ or $\perp$

Input: a phase w, a sharing of s, and a set of rebooted parties R
Output: new sharing of s in phase $w+1$, $[s]^{w+1}$, or aborted

1. Parties in $\mathcal{P}$ invoke $\texttt{ShareRandom}^{\texttt{A}}(w, \mathcal{P})$ to generate a sharing $[r]$ of r, where r is a random number in $\mathbb{F}$.
2. Each party in $\mathcal{P} \setminus R$ invokes $\texttt{Add}^{\texttt{A}}(w, [r], [s])$ to share the sharing of $r + s$.
3. $\texttt{Reconstruct}^{\texttt{A}}(w, [r+s], \mathcal{P} \setminus R)$ is invoked and every party in $\mathcal{P} \setminus R$ gets $r' := r + s$.
4. $\texttt{RobustReshare}^{\texttt{A}}(w, r', \mathcal{P} \setminus R, \mathcal{P})$ is invoked, and each party in $\mathcal{P}$ gets $[r']$.
5. Each party computes $[r'] - [r]$ by executing $\texttt{Add}^{\texttt{A}}(w, [r'], -[r])$, where $-[r]$ is the additive inverses of the shares in $\mathbb{F}$.

Theorem 5.2. *(Correctness and Secrecy of $\texttt{Recover}^{\texttt{A}}$) If $\mathbb{S}$ and $\mathcal{Z}$ satisfy $\mathcal{Q}^1(\mathbb{S}, \mathcal{Z})$, the protocol $\texttt{Recover}^{\texttt{A}}$ allows a set $\forall R \in \Delta$ of rebooted parties to recover their shares encoding the same secret with error probability $O((n-|R|)|\mathbb{S}|n^3/|\mathbb{F}| + (n-|R|)|\mathbb{S}|n^2/(|\mathbb{F}|-2))$, and does not reveal any additional information about the secret. It communicates $O((n-|R|)|\mathbb{S}|n^3 \log |\mathbb{F}|)$ bits and broadcasts $O((n-|R|)|\mathbb{S}|n^3 \log |\mathbb{F}|)$ bits.*

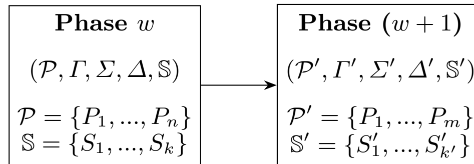

Fig. 1. Dynamic groups and GAS in two consecutive phases, w and $w+1$

To handle dynamic groups and dynamic GASs, assume that the participating parties and structures are given as in Figure 1. As mentioned in Section 2, these phase information is specified by a trusted third party, e.g. by a *Site Reliability Engineering (SRE)* organization. The protocol $\texttt{Redistribute}^{\texttt{A}}$ allows the new participating parties to obtain a sharing of the same secret as the previous phase according to the new structures. The idea is to double-share the sharing of a secret from the previous participating group to the new group. The security proof is presented in Appendix C.3.

Protocol $\texttt{Redistribute}^{\texttt{A}}(w, s) \longrightarrow [s]^{w+1}$

Input: phase w and a secret s
Output: shares of s in phase $w+1$
Precondition: parties in P share $[s]^w$ for a secret s
Postcondition: parties in P' share $[s]^{w+1}$ encoding the same secret s

1. For each $S_i \in \mathbb{S}$:
2. Each party P_y in S_i forwards its holding value $[s_i]_y$ for s_i to every party in S_i who is supposed to hold the same share (over the secure channel).
3. $\texttt{Verify}^{\texttt{A}}(P_S, P_R, P_V, w, [s_i]_y, A_{S,R,V}(s_i))$ is invoked for all $P_R, P_V \in S_i$, $\forall P_S \in S_i$. If P_V outputs $[s_i]_y$ in each invocation, P_V accepts it as value for s_i. Denote v_i as the accepted value for s_i, for each i.
4. Each party $P_y \in S_i$ runs $\texttt{Share}^{\texttt{A}}(w+1, v_i, \mathcal{P}')$ according to $\mathbb{S}'$. i.e., The j-th share of v_i, v_{ij}, is sent from P_y in S_i to all parties in $S'_j \in \mathbb{S}'$ and $\texttt{Authenticate}^{\texttt{A}}$ is invoked for each share of v_i.
5. For each $S'_j \in \mathbb{S}'$:
6. Each party in S'_j holds $\{v_{ij}\}_{i=1}^{k}$. For each v_{ij}, all $P_R, P_V \in S'_j$ invoke $\texttt{Verify}^{\texttt{A}}(P_S, P_R, P_V, w, v_{ij}, A_{S,R,V}(v_{ij}))$ for $\forall P_S \in S'_j$ and accept the output value as v_{ij}.
7. Each party in S'_j sums up all k values accpeted in step 6, and set it as new j-th share of s. i.e., $s'_j := \sum_{i=1}^{k} v_{ij}$.

Theorem 5.3. *(Correctness and Secrecy of* $\texttt{Redistribute}^{\texttt{A}}$*) By executing the protocol* $\texttt{Redistribute}^{\texttt{A}}$*, new participating parties receive a sharing of the same secret as the old shares with error probability* $((|\mathbb{S}|+|\mathbb{S}'|)n^3/(|\mathbb{F}|-2)+n^4|\mathbb{S}|/|\mathbb{F}|)$ *and it does not reveal any additional information about the secret. It communicates* $O(|\mathbb{S}|^2 n^4 \log|\mathbb{F}| + |\mathbb{S}'|n^3 \log|\mathbb{F}|)$ *bits and broadcasts* $O(|\mathbb{S}|^2 n^4 \log|\mathbb{F}|)$ *bits, where* $\mathbb{S}$ *and* $\mathbb{S}'$ *denote the sets for sharing specification in two consecutive phases.*

Note that the function of $\texttt{Recover}^{\texttt{A}}$ can be naturally substituted with $\texttt{Redistribute}^{\texttt{A}}$ with the same participating groups and the same sharing specification, but using our $\texttt{Recover}^{\texttt{A}}$ protocol is more efficient as it has linear complexity in $|\mathbb{S}|$, while $\texttt{Redistribute}^{\texttt{A}}$ has quadratic complexities in $|\mathbb{S}|$. Table 2 shows the total analysis of communication and broadcast complexities with error probability for each protocol in our additive PMPC scheme.

	Additive PMPC based on [27]			
	Protocol	Comm. (bits)	Broad. (bits)	Error Pr.
IC	`Authenticate` [27]	$7\log\|\mathbb{F}\|$	$3\log\|\mathbb{F}\|+1$	$1/\|\mathbb{F}\|$
	`Verify` [27]	$\log\|\mathbb{F}\|$	-	$1/(\|\mathbb{F}\|-2)$
SS/MPC	`Share` [27]	$O(\|\mathbb{S}\|n^3\log\|\mathbb{F}\|)$	$O(\|\mathbb{S}\|n^3\log\|\mathbb{F}\|)$	$n^3\|\mathbb{S}\|/\|\mathbb{F}\|$
	`Reconstruct` [27]	$O(\|\mathbb{S}\|n^3\log\|\mathbb{F}\|)$	-	$n^2\|\mathbb{S}\|/(\|\mathbb{F}\|-2)$
	`Add` [27]	-	-	-
	`BasicMultiply` [27]	$O(\|\mathbb{S}\|n^4\log\|\mathbb{F}\|)$	$O(\|\mathbb{S}\|n^4\log\|\mathbb{F}\|)$	$O(n^4\|\mathbb{S}\|/\|\mathbb{F}\|)$
	`RandomTriple` [27]	$O(\|\mathbb{S}\|n^4\log\|\mathbb{F}\|)$	$O(\|\mathbb{S}\|n^4\log\|\mathbb{F}\|)$	$O(n^4\|\mathbb{S}\|/\|\mathbb{F}\|)$
	`Multiply` [27]	$O(\|\mathbb{S}\|n^5\log\|\mathbb{F}\|)$	$O(\|\mathbb{S}\|n^5\log\|\mathbb{F}\|)$	$O(n^5\|\mathbb{S}\|/\|\mathbb{F}\|)$
Our Additional Protocols	`Refresh`	$O(\|\mathbb{S}\|n^4\log\|\mathbb{F}\|)$	$O(\|\mathbb{S}\|n^4\log\|\mathbb{F}\|)$	$n^4\|\mathbb{S}\|/\|\mathbb{F}\|$
	`ShareRandom`	$\|\mathbb{S}\|n\log\|\mathbb{F}\|$	$\|\mathbb{S}\|n\log\|\mathbb{F}\|$	-
	`RobustReshare`	$O(\|\mathcal{P}_S\|\|\mathbb{S}\|n^3\log\|\mathbb{F}\|)$	$O(\|\mathcal{P}_S\|\|\mathbb{S}\|n^3\log\|\mathbb{F}\|)$	$O(\|\mathbb{S}\|(\|\mathcal{P}_S\|n^3/\|\mathbb{F}\|+\|\mathcal{P}_S\|n^2/(\|\mathbb{F}\|-2)))$
	`Recover`	$O((n-\|R\|)\|\mathbb{S}\|n^3\log\|\mathbb{F}\|)$	$O((n-\|R\|)\|\mathbb{S}\|n^3\log\|\mathbb{F}\|)$	$O((n-\|R\|)n^3\|\mathbb{S}\|/\|\mathbb{F}\|+(n-\|R\|)n^2\|\mathbb{S}\|/(\|\mathbb{F}\|-2))$
	`Redistribute`	$O(\|\mathbb{S}\|^2n^4\log\|\mathbb{F}\|+\|\mathbb{S}'\|n^3\log\|\mathbb{F}\|)$	$O(\|\mathbb{S}\|^2n^4\log\|\mathbb{F}\|)$	$((\|\mathbb{S}\|+\|\mathbb{S}'\|)n^3/(\|\mathbb{F}\|-2)+n^4\|\mathbb{S}\|/\|\mathbb{F}\|)$
Total	`Additive PMPC` for Static groups	$O(\|\mathbb{S}\|n^5\log\|\mathbb{F}\|)$	$O(\|\mathbb{S}\|n^5\log\|\mathbb{F}\|)$	
	`Additive PMPC` for Dynamic groups	$O(\|\mathbb{S}\|^2n^4\log\|\mathbb{F}\|)$	$O(\|\mathbb{S}\|^2n^4\log\|\mathbb{F}\|)$	

Table 2. Total Analysis of Protocols in Additive PMPC based on [27]. Each column denotes communication complexity (bits), broadcast complexity (bits), and error probability for the protocol failure (abbreviated Comm., Braod., and Error Pr., respectively). For rows, IC denotes the information checking scheme, SS denotes the secret sharing scheme, and MPC denotes the multi-party computation scheme. Assuming $|\mathcal{P}_S| \leq n$ and $|\mathbb{S}| = |\mathbb{S}'|$, the total complexities for static group (without `Redistribute`) are less than the complexities for `Multiply` and remain linear in $|\mathbb{S}|$, but for dynamic groups (with `Redistribute`), communication and broadcast complexit ies are quadratic in $|\mathbb{S}|$.

5.2 MSP-based PMPC

Lampkins and Ostrovsky [29] presented an unconditionally secure MPC protocol based on Monotone Span Program (MSP) secret sharing against any $Q2$-adversary, which has linear communication complexity in the size of multiplicative MSP. We build our MSP-based PMPC protocol on top of their MPC protocol, ***without increasing the complexity in terms of the size of MSP,*** d.

Definition 5.2. *$(\mathbb{F}, M, \rho, \boldsymbol{a})$ is called a* ***monotone span program (MSP)****, if $\mathbb{F}$ is a finite field, M is a $d \times e$ matrix over $\mathbb{F}$, $\rho : \{1, 2, ..., d\} \to \{1, 2, ..., n\}$ is a surjective indexing function for each row of M, and $\boldsymbol{a} \in \mathbb{F}^e \backslash \boldsymbol{0}$ is a (fixed) target vector, where $\boldsymbol{0} = (0, ..., 0) \in \mathbb{F}^e$. $(\mathbb{F}, M, \rho, \boldsymbol{a}, \boldsymbol{r})$ is called a* ***multiplicative MSP****, if $(\mathbb{F}, M, \rho, \boldsymbol{a})$ is a MSP and $\boldsymbol{r}$ is a recombination vector, which means the vector $\boldsymbol{r}$ satisfies the property that $\langle \boldsymbol{r}, M\boldsymbol{b} * M\boldsymbol{b}' \rangle = \langle \boldsymbol{a}, \boldsymbol{b} \rangle \cdot \langle \boldsymbol{a}, \boldsymbol{b}' \rangle$, for all $\boldsymbol{b}, \boldsymbol{b}'$, where $*$ is the Hadamard product and $\cdot$ is the inner product.*

The target vector $\mathbf{a}$ can be any vector in $\mathbb{F}^e \backslash \mathbf{0}$; we use $\mathbf{a} = (1, 0, ..., 0)^t \in \mathbb{F}^e$ for convenience, as in [29]. Let $f : \{0, 1\}^n \to \{0, 1\}$ be a monotone function. A MSP $(\mathbb{F}, M, \rho, \mathbf{a})$ is said to **compute** f if for all nonempty set $A \subset \{1, ..., n\}$, $f(A) = 1 \Leftrightarrow \mathbf{a} \in \mathrm{Im} M_A^t$, i.e., $\exists \lambda_A$ such that $M_A^t \lambda_A = \mathbf{a}$. Also, a MSP $(\mathbb{F}, M, \rho, \mathbf{a})$ computing f is said to **accept** Γ if $B \in \Gamma \Leftrightarrow f(B) = 1$. Note that any given MSP computes a monotone Boolean function f, defined $f(x_1, ..., x_n) = 1 \Leftrightarrow \mathbf{a} \in \mathrm{Im} M_A^t$ where $A = \{1 \le i \le n | x_i = 1\}$, and it is well known that any monotone Boolean function can be computes by a MSP.

The secret sharing (SS) scheme based on the MSP accepting Γ [15, 29] also consists of `Share` and `Reconstruct` protocols. The protocol $\texttt{BasicShare}^{\texttt{M}}$ generates a sharing of s by sending each assigned row of $\mathbf{s} = M\mathbf{b}$ by the indexing function ρ, where $M \in \mathcal{M}(d \times e)$ is the matrix of the MSP corresponding to Δ and $\mathbf{b} := (s, r_2, ..., r_e) \in \mathbb{F}^e$ is a vector containing the secret value s and $(e-1)$ random values r_i's. For reconstruction, since the MSP accepts Γ, $B \in \Gamma \Leftrightarrow f(B) = 1 \Leftrightarrow \mathbf{a} \in \mathrm{Im} M_B^t$, which means there is some vector λ_B such that $M_B^t \lambda_B = \mathbf{a}$. Therefore, parties can reconstruct the secret value with shares that parties in B hold by computing $\langle \lambda_B, [s]_B \rangle = \langle \lambda_B, M_B \mathbf{b} \rangle = \langle M_B^t \lambda_B, \mathbf{b} \rangle = \langle \mathbf{a}, \mathbf{b} \rangle = s$, because $\mathbf{a} = (1, 0, ..., 0)$ and $\mathbf{b} = (s, r_2, ..., r_e)$.

For dealing with active adversaries, they use a (different) IC scheme, which accepts the Shamir's secret sharing techniques [37], and the dispute control. We describe the IC scheme in Appendix D.1. For dispute control, one more list $\mathcal{C}$ is also used, where $\mathcal{C}$ is a set of parties known by all parties to be corrupted. That means, the list $\mathcal{D}$ maintains the parties in each dispute list $\mathcal{D}_i$, for all i, and some of them move to the list $\mathcal{C}$ when all parties agree that they are corrupted. Note that their SS scheme for active adversaries allows parties to share and reconstruct multiple secret values in one execution of the protocol, but we only consider the case with one secret value per execution to fairly compare the complexities with additive PMPC protocols. So we call the share and reconstruct protocols in [29] as $\texttt{ShareMultiple}^{\texttt{M}}$ and $\texttt{ReconstructMultiple}^{\texttt{M}}$ and our considering special case protocols as $\texttt{Share}^{\texttt{M}}$ and $\texttt{Reconstruct}^{\texttt{M}}$. On the other hand, the protocol $\texttt{LC-Reconstruct}^{\texttt{M}}$ [29] allows parties to reconstruct linear combinations of multiple secrets that have been shared using $\texttt{Share}^{\texttt{M}}$ protocol and to detect corrupted parties while reconstructing. As we adopt their protocol with small variants, we present the formal descriptions in Appendix D.2. For computations, addition can be done with no communication thanks to the linearity of the shares, while multiplication needs a little trick. As we just adopt the protocols from [29], we present the protocols $\texttt{Add}^{\texttt{M}}$ and $\texttt{Multiply}^{\texttt{M}}$ in Appendix D.3.

To make this MPC scheme to PMPC handling dynamic groups, we build three new main protocols called $\texttt{Refresh}^{\texttt{M}}$, $\texttt{Recover}^{\texttt{M}}$, and $\texttt{Redistribute}^{\texttt{M}}$. Because of the space limit, we present all the proofs of our protocols in Appendix E. Recall that the protocol $\texttt{Refresh}^{\texttt{M}}$ re-randomizes the shares that each party has regularly so that the adversary cannot reconstruct the secret until he corrupts any set in the access structure in the period. By the linearity of the shares, the main idea is same as before.

Protocol $\texttt{Refresh}^{\texttt{M}}(w, [s]) \longrightarrow [s]^{w+1}$

Input: a phase w and a sharing of s
Output: new sharing of s in phase $w+1$, $[s]^{w+1}$

1. Every party P_i in $\mathcal{P}$ invokes $\texttt{Share}^{\texttt{M}}(w, 0, P_i)$. (in parallel)
2. Each party locally does component-wise addition with all the shares received in Step 1 and the shares of s, and set it as the new share of s in phase $w+1$.
3. parties in $\mathcal{P}$ collectively output $[s]^{w+1}$.

Theorem 5.4. *(Correctness and Secrecy of $\texttt{Refresh}^{\texttt{M}}$) When the protocol $\texttt{Refresh}^{\texttt{M}}$ terminates, all parties receive new shares encoding the same secret as old shares they had before, and they cannot get any information about the secret by the execution of the protocol. It communicates $O((n^2 d + n^3\kappa) \log |\mathbb{F}| + n^3 \kappa \log d)$ bits and broadcasts $O(n^3 \log d + (n^3 + nd) \log |\mathbb{F}|)$ bits.*

For $\texttt{Recover}^{\texttt{M}}$ and $\texttt{Redistribute}^{\texttt{M}}$, we construct two sub-protocols, $\texttt{ShareRandom}^{\texttt{M}}$ and $\texttt{RobustReshare}^{\texttt{M}}$. The goals of the protocols are similar to the ones in Section 5.1, but due to the fact that each party holds the unique

share of a secret, $\texttt{ShareRandom}^{\texttt{M}}$ can be generalized for multiple groups of parties, which enables to build the efficient $\texttt{Redistribute}^{\texttt{M}}$ protocol. The protocol $\texttt{ShareRandom}^{\texttt{M}}$ allows participating parties to generate multiple sharings of a random value $r \in \mathbb{F}$ for each group without reconstructing the value r. Note that $W = \{w\}$ for $\texttt{Recover}^{\texttt{M}}$, while $W = \{w, w+1\}$ for $\texttt{Redistribute}^{\texttt{M}}$. The protocol $\texttt{ShareRandom}^{\texttt{M}}$ outputs $|W|$ sharings of the same r, where r is the summation of all random elements from each party in each phase. For instance, when $W = \{w\}$, the output is one sharing of r, say $[r] = \{\mathbf{r}_1, ..., \mathbf{r}_n\}$, where $\texttt{LC-Reconstruct}^{\texttt{M}}(w, [r])$ reconstructs $r = \sum_{P_i \notin \mathcal{C}} r^{(i)}$. We denote $\texttt{ShareRandom}^{\texttt{M}}(w)$ in this case. On the other hand, when $W = \{w, w+1\}$, it outputs two sharings of r, $[r]^w := \{\mathbf{r}_1^w, ..., \mathbf{r}_n^w\}$ and $[r]^{w+1} := \{\mathbf{r}_1^{w+1}, ..., \mathbf{r}_m^{w+1}\}$, where both sharings reconstruct the same r. i.e., $\texttt{LC-Reconstruct}^{\texttt{M}}(w, [r]^w) = \texttt{LC-Reconstruct}^{\texttt{M}}(w+1, [r]^{w+1}) = r$, where $r = \sum_{P_i \notin \mathcal{C}^w} r^{(w,i)} + \sum_{P_j \notin \mathcal{C}^{w+1}} r^{(w+1,j)}$.

Protocol $\texttt{ShareRandom}^{\texttt{M}}(W) \longrightarrow \{[r]^w\}_{w \in W}$

Input: a list W of phases where parties in phase $\forall w \in W$ participate in generating sharing(s) of a random value
Output: $|W|$ sharing(s) of a random value r, for each $\mathcal{P}^w$ in $w \in W$

1. For each $w \in W$:
2. Every party $P_i \notin \mathcal{C}^w$ chooses a random value $r^{(w,i)}$ and invokes $\texttt{Share}^{\texttt{M}}(w', r^{(w,i)}, \mathcal{P}^{w'})$ $|W|$ times in parallel with respect to $\mathbb{S}^w$, for each $w' \in W$.
3. For each $w \in W$:
4. Each party $P_i \in \mathcal{P}^w$ locally computes $\mathbf{r}_i^w := \sum_{w' \in W} \sum_{P_j \notin \mathcal{C}^{w'}} [r^{(w',j)}]_i^w$, where $[r^{(w',j)}]_i^w$ is P_i's holding share of $r^{(w',j)}$ received in Step 2 from $P_j \notin \mathcal{C}^{w'}$.
5. $|W|$ sharings of r, $\{[r]^w\}_{w \in W}$, are collectively output, where $r := \sum_{P_j \notin \mathcal{C}^w, w \in W} r^{(w,j)}$ and $[r]^w := \{\mathbf{r}_1^w, ..., \mathbf{r}_{|\mathcal{P}^w|}^w\}$.

Note that all summand vectors $\{[r^{(w',j)}]_i^w\}$ have the same lengths for each party. For $N := \sum_{w \in W} |\mathcal{P}^w|$, the protocol communicates $O(N|W|((nd + n^2\kappa)\log|\mathbb{F}| + n^2\kappa \log d))$ bits and broadcasts $O(N|W|(n^2 \log d + (n^2 + d)\log|\mathbb{F}|))$ bits.

Recall that $Honest := \{\mathcal{P} \setminus A \mid A \in \overline{\Delta}\}$ is a set of potential honest parties sets.The protocol $\texttt{RobustReshare}^{\texttt{M}}$ similarly works as the one in Section 5.1. Every party in $\mathcal{P}_S \subseteq \mathcal{P}^{w_S}$ in phase w_S knows the value r and wants to send a right sharing of r to the parties in $\mathcal{P}_R \subseteq \mathcal{P}^{w_R}$ in phase w_R. As the adversary picks one subset of parties in Δ in each phase, there exists at least one set in $Honest$ consisting of only honest parties in that phase.

Protocol $\texttt{RobustReshare}^{\texttt{M}}(r, w_S, \mathcal{P}_S, w_R, \mathcal{P}_R) \longrightarrow [r]^{w_R}$

Input: a random element $r \in \mathbb{F}$, a phase w_S, a set of parties $\mathcal{P}_S$ in phase w_S, a phase w_R, and a set of parties $\mathcal{P}_R \in \Gamma$ in phase w_R.
Output: a sharing of r in phase w_R, $[r]^{w_R}$

Precondition: All parties in $\mathcal{P}_S$ know the value of r.
Postcondition: Each party in $\mathcal{P}_R$ receives the share of new sharing of r.

1. Every party in $\mathcal{P}_S$ executes $\texttt{Share}^{\texttt{M}}(w_R, r, \mathcal{P}_R)$ according to $\mathbb{S}_R$. Let $[r]^{(i)}$ be the sharing of r that $P_{k_i} \in \mathcal{P}_S$ shares.
2. For each $i = 1, 2, ..., |\mathcal{P}_S|$, $\texttt{Reconstruct}^{\texttt{M}}(w_R, [r]^{(i)}, \mathcal{P}_R)$ is invoked. Let $r^{(i)}$ be the result of each reconstruction.
3. There exists a value v such that $v = r^{(i)}$ for all $P_{k_i} \in H$, for some $H \in Honest$. Each party chooses such set $H \in Honest$. If there exists multiple such sets, the minimal set including P_i with lower id, i, is chosen.
4. Parties in $\mathcal{P}_R$ collectively outputs the sharing of r from the party P_i in H with the minimum id, i.
i.e. Output $[r] \leftarrow [r]^{(min)}$, where $min := \min_{P_i \in H}\{i\}$.

The security of $\texttt{RobustReshare}^{\texttt{M}}$ relies on the security of $\texttt{Share}^{\texttt{M}}$ and $\texttt{Reconstruct}^{\texttt{M}}$ and communicates $O(|\mathcal{P}_S|(|\mathcal{P}_R|^2 \kappa \log d_R + (|\mathcal{P}_R|^3 + |\mathcal{P}_R|^2\kappa + |\mathcal{P}_R|d_R)\log|\mathbb{F}|))$ and broadcasts $O(|\mathcal{P}_S|(|\mathcal{P}_R|^2 \log d_R + (|\mathcal{P}_R|^2 + d_R)\log|\mathbb{F}|))$ bits.

Using these sub-protocols, $\texttt{Recover}^{\texttt{M}}$ allows the rebooted/reset parties to recover their shares, by generating new sharing of the same secret in $\mathcal{P}$ with the assistance of other parties. A sharing of a random element r is generated using $\texttt{ShareRandom}^{\texttt{M}}$, $\texttt{LC-Reconstruct}^{\texttt{M}}$ allows every party to reconstruct a publicly known random

value $r' := r + s$, and $\texttt{RobustReshare}^{\texttt{M}}$ helps parties to set one same sharing of r'.

Protocol $\texttt{Recover}^{\texttt{M}}(w, [s], R) \longrightarrow [s]^{w+1}$ or $\perp$

Input: a phase w, a sharing of s, and a set of rebooted parties R
Output: new sharing of s in phase $w+1$, $[s]^{w+1}$, or aborted

1. Invoke $\texttt{ShareRandom}^{\texttt{M}}(w)$ and generate a sharing $[r] := \{\mathbf{r}_1, ..., \mathbf{r}_n\}$ of a random r in $\mathbb{F}$.
2. Each party P_i in $\mathcal{P} \setminus R$ locally computes $\mathbf{r}_i + \mathbf{s}_i$, the share of $r' := r + s$.
3. $\texttt{LC-Reconstruct}^{\texttt{M}}(w, [r'])$ is invoked in $\mathcal{P} \setminus R$ and every party in $\mathcal{P} \setminus R$ gets r'.
4. $\texttt{RobustReshare}^{\texttt{M}}(r', w, \mathcal{P} \setminus R, w, \mathcal{P})$ is invoked, and each party in $\mathcal{P}$ gets $[r']^w := \{\mathbf{r}'_1, ..., \mathbf{r}'_n\}$.
5. Each party locally computes $\mathbf{r}'_i - \mathbf{r}_i$ and sets it as new share of s.

Theorem 5.5. *(Correctness and Secrecy of* $\texttt{Recover}^{\texttt{M}}$*) The protocol* $\texttt{Recover}^{\texttt{M}}$ *allows a set R of parties who were rebooted to recover their shares encoding the same secret, for any $R \in \Delta$, and does not reveal any additional information about the secret except the shares each party had before the execution of the protocol. It communicates* $O(n^3\kappa \log d + (n^4 + n^3\kappa + n^2 d) \log |\mathbb{F}|)$ *bits and broadcasts* $O(n^3 \log d + (n^3 + nd) \log |\mathbb{F}|)$ *bits.*

Assuming the dynamic settings in Figure 1, recall that the protocol $\texttt{Redistribute}^{\texttt{M}}$ allows parties in the new group $\mathcal{P}'$ to receive the shares encoding the same secret. The main idea is similar to the one in $\texttt{Recover}^{\texttt{M}}$, but as parties might be different in two phases, it needs to be considered very carefully. To send a right sharing of s from $\mathcal{P}$ to $\mathcal{P}'$ without revealing the secret value s to the parties, both parties in two phases generate a sharing of random value r without reconstructing the value r using $\texttt{ShareRandom}^{\texttt{M}}$. Then, parties holding the share of s locally compute the share of r to the share of s and reconstruct $s + r$ using them. Now, all parties in $\mathcal{P}$ knows the value $s + r$, but not s or r, so that they invoke $\texttt{RobustReshare}^{\texttt{M}}$ to send a right sharing of $s + r$ to the parties in $\mathcal{P}'$. As parties in $\mathcal{P}'$ are also holding the share of r, now each party can locally computes the share of s.

Protocol $\texttt{Redistribute}^{\texttt{M}}(w, [s]^w) \longrightarrow [s]^{w+1}$

Input: a phase w and the sharing of s in phase w, $[s]^w = \{\mathbf{s}_1^w, ..., \mathbf{s}_n^w\}$
Output: new sharing of s for phase $w+1$, $[s]^{w+1} = \{\mathbf{s}_1^{w+1}, ..., \mathbf{s}_m^{w+1}\}$

1. Parties in $\mathcal{P}$ and $\mathcal{P}'$ invoke $\texttt{ShareRandom}^{\texttt{M}}(W)$, where $W = \{w, w+1\}$, to generate two sharings of a random value r, unknown to every party. That is, parties in $\mathcal{P}$ separately receive a sharing $[r]^w := \{\mathbf{r}_1^w, ..., \mathbf{r}_n^w\}$, while parties in $\mathcal{P}'$ receive a sharing $[r]^{w+1} := \{\mathbf{r}_1^{w+1}, ..., \mathbf{r}_m^{w+1}\}$, and no one knows the value of r.
2. Each party P_i in $\mathcal{P}$ locally computes $\mathbf{x}_i := \mathbf{r}_i^w + \mathbf{s}_i^w$, where $\mathbf{s}_i^w$ is the share of s.
3. Parties in $\mathcal{P}$ invoke $\texttt{LC-Reconstruct}^{\texttt{M}}(w, [x])$ with $[x] := \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ and the result is denoted by x. Note that $x = s + r$, where r is random and unknown to everyone.
4. Parties invoke $\texttt{RobustReshare}^{\texttt{M}}(x, w, \mathcal{P}, w+1, \mathcal{P}')$ so that parties in $\mathcal{P}'$ receive a sharing of x, say $[x] := \{\mathbf{z}_1, ..., \mathbf{z}_m\}$, for $\mathbf{z}_i := M'_i \mathbf{X}$, where the vector $\mathbf{X} = (x, \$, ..., \$) \in \mathbb{F}^{e'}$ with random \$'s.
5. Each party P'_i in $\mathcal{P}'$ locally computes $\mathbf{s}_i^{w+1} := \mathbf{z}_i - \mathbf{r}_i^{w+1}$, for $i = 1, ..., m$.
6. Parties in $\mathcal{P}'$ collectively output $\{\mathbf{s}_1^{w+1}, ..., \mathbf{s}_m^{w+1}\}$ as a sharing of s in new phase.

Theorem 5.6. *(Correctness and Secrecy of* $\texttt{Redistribute}^{\texttt{M}}$*) When the protocol terminates, all parties in new participating group have the shares of the same secret as the old shares, and the protocol does not reveal any information about the secret. It communicates* $O(n^2\kappa \log d + nm^2\kappa \log d' + ((n^2+mn)d + (m^2+mn)d' + (n^3+m^3)\kappa + (m+n)mn\kappa + nm^3) \log |\mathbb{F}|)$ *bits and broadcasts* $O((n^3+mn^2) \log d + nm^2 \log d' + (n^3 + (n+m)(mn+d) + nd') \log |\mathbb{F}|)$ *bits, where* $|\mathcal{P}| = n$, $|\mathcal{P}'| = m$, *size(M) = d, and size(M') = d'.*

Table 3 shows the total analysis of MSP-based PMPC protocols based on the protocols in [29]. Even after adding our new protocols, for both static groups and dynamic groups, the total communication/broadcast complexities remain linear in the size of MSP, d, the number of rows of the corresponding matrix M.

	MSP-based PMPC based on [29]		
	Protocol	Communication Comp.	Broadcast Comp.
IC	`Authenticate`*	$O(\kappa(\log d + \log \lvert\mathbb{F}\rvert))$	$O(\log d + \log \lvert\mathbb{F}\rvert)$
	`Verify`*	$O(\kappa \log d + (l+\kappa) \log \lvert\mathbb{F}\rvert)$	1
SS / MPC	`BasicShare`	$O(d \log \lvert\mathbb{F}\rvert)$	-
	`Share`	$O((nd + n^2\kappa) \log \lvert\mathbb{F}\rvert + n^2\kappa \log d)$	$O(n^2 \log d + (n^2 + d) \log \lvert\mathbb{F}\rvert)$
	`Reconstruct`	$O(n^2\kappa \log d + (n^3 + n^2\kappa) \log \lvert\mathbb{F}\rvert)$	$O(d \log \lvert\mathbb{F}\rvert)$
	`LC-Reconstruct`*	$O(n^2\kappa \log d + (n^3 + n^2\kappa) \log \lvert\mathbb{F}\rvert)$	$O(n(\log_2 L + 1) d \log \lvert\mathbb{F}\rvert)$
	`Add`	-	-
	`Gen-Rand`	$O((n^2 d + n^3\kappa) \log \lvert\mathbb{F}\rvert + n^3\kappa \log d)$	$O(n^3 \log d + (n^3 + nd) \log \lvert\mathbb{F}\rvert)$
	`Gen-Mult-Triples`	$O((n^4 + n^3\kappa + n^2 d) \log \lvert\mathbb{F}\rvert + n^3\kappa \log d)$	$O(n^3 \log d + (n^3 + n^2 d) \log \lvert\mathbb{F}\rvert)$
Our Additional Protocols	`Refresh`	$O((n^2 d + n^3\kappa) \log \lvert\mathbb{F}\rvert + n^3\kappa \log d)$	$O(n^3 \log d + (n^3 + nd) \log \lvert\mathbb{F}\rvert)$
	`ShareRandom`	$O(N\lvert W\rvert((nd + n^2\kappa) \log \lvert\mathbb{F}\rvert + n^2\kappa \log d))$	$O(N\lvert W\rvert(n^2 \log d + (n^2 + d) \log \lvert\mathbb{F}\rvert))$
	`RobustReshare`	$O(\lvert\mathcal{P}_S\rvert(\lvert\mathcal{P}_R\rvert^2\kappa \log d_R + (\lvert\mathcal{P}_R\rvert^3 + \lvert\mathcal{P}_R\rvert^2\kappa + \lvert\mathcal{P}_R\rvert d_R) \log \lvert\mathbb{F}\rvert))$	$O(\lvert\mathcal{P}_S\rvert(\lvert\mathcal{P}_R\rvert^2 \log d_R + (\lvert\mathcal{P}_R\rvert^2 + d_R) \log \lvert\mathbb{F}\rvert))$
	`Recover`	$O(n^3\kappa \log d + (n^4 + n^3\kappa + n^2 d) \log \lvert\mathbb{F}\rvert)$	$O(n^3 \log d + (n^3 + nd) \log \lvert\mathbb{F}\rvert)$
	`Redistribute`	$O(n^3\kappa \log d + (n^2 d + n^4 + n^3\kappa) \log \lvert\mathbb{F}\rvert)$	$O(n^3 \log d + (n^3 + nd) \log \lvert\mathbb{F}\rvert)$
Total	`MSP-based PMPC`	$O(n^3\kappa \log d + (n^2 d + n^4 + n^3\kappa) \log \lvert\mathbb{F}\rvert)$	$O(n^3 \log d + (n^3 + n^2 d) \log \lvert\mathbb{F}\rvert)$

Table 3. Total analysis of protocols in MSP-based PMPC scheme based on [29]. Each column denotes communication complexity (bits) and broadcast complexity (bits). For rows, IC denotes the information checking scheme, SS denotes the secret sharing scheme, and MPC denotes the multi-party computation scheme. κ denotes the security parameter. Only IC scheme and `LC-Reconstruct` are based on multiple secret values, and the others are based on one secret value. In IC, l is the number of secret values and in `LC-Reconstruct`, $L := max_j(l_j)$, where l_j is the number of secrets from P_j. In `ShareRandom`, $N = \sum_{w \in W} |\mathcal{P}^w|$ and W is a set of phases which parties participate in the protocol. In `Redistribute`, it is assumed that $|\mathcal{P}| = |\mathcal{P}'| = n$ and $size(MSP) = d = d'$. Note that all protocols still have linear complexities in the size of MSP, d, even after adding out new protocol, for both static and dynamic groups.

5.3 Conversions between Additive and MSP-based MPC

Now, we present the way to convert the additive PMPC scheme into the MSP-based PMPC, and the way in the opposite direction. Recall that the complexity of an additive PMPC scheme depends on the size of the sharing specification $|\mathbb{S}|$ (we use the basic secrecy structure $|\widetilde{\Sigma}|$), while the one of a MSP-based PMPC scheme depends on the number of rows of the MSP matrix, d. Since d can be varied from n to $|\widetilde{\Sigma}|^{2.7}$ depending on the adversary structures [29], there are some cases worth to convert the schemes even though the conversion itself needs some resource. The better PMPC scheme can be chosen, only when participating groups or any structures are changed. That is, when dynamic groups and structures of two consecutive phases are given, participating parties can continue the current PMPC scheme by executing `Redistribute` protocol or they can convert the scheme from one to the other by calling the protocols, called `ConvertAdditiveIntoMSP` and `ConvertMSPIntoAdditive`. We present the security proofs in Appendix F.

Let dynamic groups and structures in consecutive phases are given as $\mathcal{S}^w := (\mathcal{P}, \Gamma, \Sigma, \Delta, \mathbb{S})$ and $\mathcal{S}^{w+1} := (\mathcal{P}', \Gamma', \Sigma', \Delta', \mathbb{S}')$ and let the additive PMPC scheme has been using in phase w with sharing specification $\mathbb{S} = \{S_1, S_2, ..., S_k\}$. The protocol `ConvertAdditiveIntoMSP` converts current additive sharing of s into a MSP-based sharing of s. By definition, if no qualified subset of parties in the access structure Γ remains in $\mathcal{P}$, then the secret value s cannot be reconstructed even though the protocol is executed. That is, there exists at least one honest party in each $S_i \in \mathbb{S}$. For dealing with active adversaries, all parties in each S_i needs to share their holding share s_i to the parties in $\mathcal{P}'$ using the $\texttt{Share}^{\texttt{M}}$ protocol. Then parties in $\mathcal{P}'$ hide their shares with the shares of a random number and open (reconstruct) the hided values to decide one sharing of s_i from the honest party in S_i. By linearity of shares, each party in $\mathcal{P}'$ can locally compute the MSP-based share of s by component-wise adding all its receiving shares. The formal protocol is as follows.

Protocol `ConvertAdditiveIntoMSP`$([s]^w, w, \mathcal{S}^w, w+1, \mathcal{S}^{w+1}) \longrightarrow [s]^{w+1}$

Input: the structure $\mathcal{S} := (\mathcal{P}, \Gamma, \Sigma, \Delta, \mathbb{S})$ in phase w and the sharing $\{s_1, ..., s_{|\mathbb{S}|}\}$ of s such that $\sum_{i=1}^{|\mathbb{S}|} s_i = s$
Output: the sharing of s for the structure $\mathcal{S}' := (\mathcal{P}', \Gamma', \Sigma', \Delta', \widehat{M})$ in phase w, where $M \in \mathcal{M}(d \times e)$ is corresponding matrix of the MSP $\widehat{M}$

1. For each $i \in \{1, ..., |\mathbb{S}|\}$ (in parallel):
2. Every party in S_i invokes $\texttt{Share}^{\texttt{M}}(s_i, w+1, \mathcal{P}')$. Denote $|S_i|$ sharings of s_i by $[s_i]^{(1)}, ..., [s_i]^{(|S_i|)}$.
3. Parties in $\mathcal{P}'$ invoke $\texttt{ShareRandom}^{\texttt{M}}(w+1)$ to generate a sharing of a random number, say $r^{(i)}$.

4. Parties in $\mathcal{P}'$ locally compute $[x_i^{(j)}] := [s_i]^{(j)} + [r^{(i)}]$, for $j = 1, ..., |S_i|$.
5. Parties in $\mathcal{P}'$ execute $\texttt{LC-Reconstruct}^{\texttt{M}}(w+1, [x_i^{(j)}])$ (in parallel) $|S_i|$ times for each sharing and choose a set $H \in Honest := \{\mathcal{P} \setminus A | A \in \overline{\Delta}\}$ that $x_i^{(j)} = v$ for all $P_{k_j} \in (S_i \cap H)$. If there exists multiple such sets, they choose the minimal set including P_{id} with lower id.
6. The sharing $[s_i^{(min)}]$ of s_i from $P_{min} \in H$ is chosen as a sharing of s_i, say $[s_i]$.
7. At this point, parties in $\mathcal{P}'$ hold $|\mathbb{S}|$ sharings for each s_i and each party $P_j \in \mathcal{P}'$ holds $|\mathbb{S}|$ vectors of length d_j, for each sharing. Each party locally computes component-wise addition with these vectors and set it as its share of s. i.e., P_j computes $\mathbf{s}_j := \sum_{i=1}^{|\mathbb{S}|}[s_i]_j \in \mathbb{F}^{d_j}$, where each share is the vector of length d_j.
8. Parties in $\mathcal{P}'$ collectively output a sharing of s, $[s]^{w+1} := \{\mathbf{s}_1, ..., \mathbf{s}_m\}$, where $m = |\mathcal{P}'|$.

Theorem 5.7. *(Correctness and Secrecy of* `ConvertAdditiveIntoMSP`*) When the protocol terminates, all parties in new participating group have the shares of the same secret as the old shares, and the protocol does not reveal any information about the secret. It communicates $O(k((m^2 + mn)d + (m^3 + m^2n)\kappa + nm^3)\log|\mathbb{F}| + k(m^3 + m^2n)\kappa \log d)$ bits and broadcasts $O(k(mnd + m^3 + m^2n)\log|\mathbb{F}| + k(m^3 + m^2n)\log d)$ bits, where $|\mathcal{P}| = n$, $|\mathcal{P}'| = m$, $|\mathbb{S}| = k$, and size(M) = d.*

On the other hand, when participating parties currently use the MSP-based PMPC scheme and want to convert it to the additive PMPC in the next phase, they can execute the protocol `ConvertMSPIntoAdditive`. It converts a MSP-based sharing of s in phase w into an additive sharing of s in phase $w+1$. Note that each party P_i has different shares of s in MSP-based PMPC and P_i's share of s is the vector of length d_i. For these reasons, each party needs to invoke multiple $\texttt{Share}^{\texttt{A}}$ protocols to share each component of the vector according to the sharing specification $\mathbb{S}'$ in phase $w+1$. Each party in each $S_j \in \mathbb{S}'$ collects all the shares received from the same party P_i and form a vector of length d_i. Then, all the parties in S_j hold the same n vectors of different lengths. When recomposing these n vectors according to the indexing function ρ of phase w, each party can compute its share of s by inner product with the vector λ such that $M^t\lambda = \mathbf{a}$.

Protocol $\texttt{ConvertMSPIntoAdditive}([s]^w, w, \mathcal{S}^w, w+1, \mathcal{S}^{w+1}) \longrightarrow [s]^{w+1}$

Input: the structure $\mathcal{S} := (\mathcal{P}, \Gamma, \Sigma, \Delta, \widehat{M})$ in phase w and the sharing $\{\mathbf{s}_i\}_{i=1}^n$ of s such that $\mathbf{s}_i = M_i\mathbf{b}$ for each i, where $M \in \mathcal{M}(d \times e)$ of the MSP $\widehat{M}$ computes f and accepts Γ and $\mathbf{b} = (s, r_2, ..., r_e)$ for $r_i \xleftarrow{\$} \mathbb{F}$
Output: a sharing of s for $\mathcal{S}' := (\mathcal{P}', \Gamma', \Sigma', \Delta', \mathbb{S}')$ in phase w, where $\mathbb{S}' := \{S_1, S_2, ..., S_k\}$ is the sharing specification in phase $w+1$

1. Each party $P_i \in \mathcal{P}$ invokes $\texttt{Share}^{\texttt{A}}(w+1, s_j^{(i)}, \mathcal{P}')$, for each $s_j^{(i)}$ of $\mathbf{s}_i := (s_1^{(i)}, ..., s_{d_i}^{(i)})$.
2. Every party in $S_j \in \mathbb{S}'$ forms a vector of shares received in Step 1 from the same party P_i, as $\mathbf{s}_{ij} := ((s_1^{(i)})_j, ..., (s_{d_i}^{(i)})_j)$. i.e., Parties in S_j hold n different vectors $\{\mathbf{s}_{1j}, ..., \mathbf{s}_{nj}\}$ from every party in $\mathcal{P}$ and each vector $\mathbf{s}_{ij}$ has length d_i.
3. Every parties in S_j forms the recomposition vector $\mathbf{Q}_j$ with n vectors $\{\mathbf{s}_{1j}, ..., \mathbf{s}_{nj}\}$ with respect to the indexing function ρ of $\widehat{M}$. Note that the length of $\mathbf{Q}_j$ is d for all $j = 1, ..., k$.
4. Each party in S_j sets $s_j := \langle \lambda, \mathbf{Q}_j \rangle$, where λ is the vector such that $M^t\lambda = \mathbf{a}$, for each $j = 1, ..., |\mathbb{S}|$.
5. Players in $\mathcal{P}'$ collectively output $[s]^{w+1} := \{s_1, ..., s_k\}$.

Theorem 5.8. *(Correctness and Secrecy of* `ConvertMSPIntoAdditive`*) When the protocol terminates, all parties in new participating group have the shares of the same secret as the old shares, and the protocol does not reveal any information about the secret. It communicates $O(dkn^3 \log|\mathbb{F}|)$ bits and broadcasts $O(dkn^3 \log|\mathbb{F}|)$ bits, where $|\mathcal{P}| = n$, $|\mathcal{P}'| = m$, $|\mathbb{S}'| = k$, and size(M) = d.*

From the Theorem 5.7 and Theorem 5.8, we can derive the following corollary.

Corollary. *A proactive MPC scheme based on additive secret sharing and a proactive MPC scheme based on MSP-based secret sharing are convertible. That is, one can transform an additive sharing of a secret to a MSP-based sharing of the same secret and also transform a MSP-based sharing of a secret to an additive sharing of the same secret, without revealing any information about the secret among participating parties.*

References

1. Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold rsa with adaptive and proactive security. In *Proceedings of the 24th annual international conference on The Theory and Applications of Cryptographic Techniques*, EUROCRYPT'06, pages 593–611, 2006.
2. Michael Backes, Christian Cachin, and Reto Strobl. Proactive secure message transmission in asynchronous networks. In *Proceedings of the Twenty-Second ACM Symposium on Principles of Distributed Computing, PODC 2003, Boston, Massachusetts, USA, July 13-16, 2003*, pages 223–232, 2003.
3. Joshua Baron, Karim ElDefrawy, Joshua Lampkins, and Rafail Ostrovsky. How to withstand mobile virus attacks, revisited. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 293–302, 2014.
4. Joshua Baron, Karim ElDefrawy, Joshua Lampkins, and Rafail Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In *Proceedings of the 2015 International Conference on Applied Cryptography and Network Security*, ACNS '15, 2015.
5. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
6. Zuzana Beerliová-Trubíniová and Martin Hirt. Efficient multi-party computation with dispute control. In *TCC*, pages 305–328, 2006.
7. Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure mpc with linear communication complexity. In *TCC*, pages 213–230, 2008.
8. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.
9. Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *CRYPTO*, pages 663–680, 2012.
10. G. R. Blakley. Safeguarding cryptographic keys. *Proc. of AFIPS National Computer Conference*, 48:313–317, 1979.
11. Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *ACM Conference on Computer and Communications Security*, pages 88–97, 2002.
12. Ran Canetti and Amir Herzberg. Maintaining security in the presence of transient faults. In *CRYPTO*, pages 425–438, 1994.
13. Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.
14. David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 11–19, 1988.
15. Ronald Cramer, Ivan Damgrd, and Ueli Maurer. Span programs and general secure multi-party computation. *BRICS Report Series*, 4(28), Jan. 1997.
16. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT*, pages 445–465, 2010.
17. Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, pages 241–261, 2008.
18. S. Dolev, J. Garay, N. Gilboa, and V. Kolesnikov. Swarming secrets. In *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*, pages 1438–1445, Sept 2009.
19. Shlomi Dolev, Karim Eldefrawy, Joshua Lampkins, Rafail Ostrovsky, and Moti Yung. Proactive secret sharing with a dishonest majority. In *Proceedings of the 10th International Conference on Security and Cryptography for Networks - Volume 9841*, pages 529–548, 2016.
20. Shlomi Dolev, Juan A. Garay, Niv Gilboa, and Vladimir Kolesnikov. Secret sharing krohn-rhodes: Private and perennial distributed computation. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 32–44, 2011.
21. Shlomi Dolev, Juan A. Garay, Niv Gilboa, Vladimir Kolesnikov, and Yelena Yuditsky. Towards efficient private distributed computation on unbounded input streams. *J. Mathematical Cryptology*, 9(2):79–94, 2015.
22. Karim Eldefrawy, Rafail Ostrovsky, Sunoo Park, and Moti Yung. Proactive secure multiparty computation with a dishonest majority. In *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, pages 200–215, 2018.
23. Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Proactive rsa. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '97, pages 440–454, London, UK, UK, 1997. Springer-Verlag.
24. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, 1987.
25. Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO*, pages 339–352, 1995.

26. Martin Hirt, Ueli Maurer, and Christoph Lucas. A Dynamic Tradeoff between Active and Passive Corruptions in Secure Multi-Party Computation. In Ran Canetti and Juan A. Garay, editors, *Advances in cryptology - CRYPTO 2013 : 33rd Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013 : proceedings*, volume 8043 of *Lecture notes in computer science*, pages 203–219, Heidelberg, 2013. Springer.
27. Martin Hirt and Daniel Tschudi. Efficient general-adversary multi-party computation. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, pages 181–200, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
28. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *STOC*, pages 433–442, 2008.
29. Joshua Lampkins and Rafail Ostrovsky. Communication-efficient mpc for general adversary structures. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks*, pages 155–174, Cham, 2014. Springer International Publishing.
30. Mike Loukides. What is devops? OReilly Media, 2012. 7 June.
31. Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Xiaodong Song. Churp: Dynamic-committee proactive secret sharing. *IACR Cryptology ePrint Archive*, 2019:17, 2019.
32. Ueli Maurer. Secure multi-party computation made simple. In *Proceedings of the 3rd International Conference on Security in Communication Networks*, SCN'02, pages 14–28, Berlin, Heidelberg, 2003. Springer-Verlag.
33. Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In *PODC*, pages 51–59, 1991.
34. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, STOC '89, pages 73–85, 1989.
35. David Schultz. *Mobile Proactive Secret Sharing*. PhD thesis, Massachusetts Institute of Technology, 2007.
36. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
37. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612613, November 1979.
38. Theodore M. Wong, Chenxi Wang, and Jeannette M. Wing. Verifiable secret redistribution for archive system. In *IEEE Security in Storage Workshop*, pages 94–106, 2002.
39. Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, FOCS '82, pages 160–164, 1982.
40. Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. Apss: proactive secret sharing in asynchronous systems. *ACM Trans. Inf. Syst. Secur.*, 8(3):259–286, 2005.

Appendix A

A.1 Related Work in Proactive Secret Sharing and Proactive MPC

Proactive Secret Sharing (PSS): SS is typically utilized as a building block for MPC protocols. Table 4 summarizes existing PSS schemes which all consider only the threshold adversary structures and are typically insecure when majority of the parties are compromised. PSS schemes [33, 25, 38, 40] [35, 3, 4] typically store the secret as the free term in a polynomial of degree $t < n/2$, thus once an adversary compromises $t + 1$ parties (even if only passively), it can reconstruct the polynomial and recover the secret. PSS schemes with optimal-communication [3, 4] also use a similar technique but instead of storing the secret in the free term, they store a batch of b secrets at different points in the polynomial; similar to the single secret case, even when secrets are stored as multiple points on a polynomial, once the adversary compromises $t + 1$ parties, it can reconstruct the polynomial and recover the stored secrets. Different techniques are required to construct PSS secure against GAS. Recently [19] developed the first PSS scheme for a dishonest majority but also only for a threshold adversary structure, the scheme cannot be generalized for other structures. Also, the work in [19] only describes a PSS scheme and does not specify how to perform PMPC for the same thresholds. While it may be possible to extend that PSS scheme to PMPC, it will remain limited to the threshold adversary structure.

Scheme	Threshold Passive (Active)	Security	Network Type	Dynamic Groups	Adversary Structure	Comm. Complexity
[38]	$t < n/2$ $(n/2)$	Crypt.	Sync.	Static	Threshold	exp(n)
[40]	$t < n/3$ $(n/3)$	Crypt.	Async.	Static	Threshold	exp(n)
[11]	$t < n/3$ $(n/3)$	Crypt.	Async.	Static	Threshold	$O(n^4)$
[35]	$t < n/3$ $(n/3)$	Crypt.	Async.	Static	Threshold	$O(n^4)$
[25]	$t < n/2$ $(n/2)$	Crypt.	Sync.	Static	Threshold	$O(n^2)$
[3]	$t < n/3 - \epsilon$ $(n/3 - \epsilon)$	Perfect	Sync.	Static	Threshold	$O(1)^*$
[3]	$t < n/2 - \epsilon$ $(n/2 - \epsilon)$	Uncond.	Sync.	Static	Threshold	$O(1)^*$
[4]	$t < n/2 - \epsilon$ $(n/2 - \epsilon)$	Uncond.	Sync.	Dynamic	Threshold	$O(1)^*$
[19]	$t < n - 1$ $(n/2 - 1)$	Crypt.	Sync.	Static	Threshold	$O(n^4)$
[31]	$(t < n/2)$	Crypt.	Sync.	Dynamic	Threshold	$O(n^3)$
This work						
Additive	N/A	Uncond.	Sync.	Static	General	$O(\lvert\mathbb{S}\rvert * poly(n))$
Additive	N/A	Uncond.	Sync.	Dynamic	General	$O(\lvert\mathbb{S}\rvert^2 * poly(n))$
MSP-based	N/A	Uncond.	Sync.	Static	General	$O(d * poly(n))$
MSP-based	N/A	Uncond.	Sync.	Dynamic	General	$O(d * poly(n))$

Table 4. Comparing existing proactive secret sharing (PSS) schemes; n is the number of parties, threshold t is for each refresh phase. 'Crypt.', 'Perfect', and 'Uncond.' denote cryptographic, perfect, and unconditional security, respectively; 'Sync.' and 'Async.' denote the synchronous and asynchronous networks, respectively. Note that [19] also handles mixed adversaries which are characterized by two thresholds, one for passive corruptions and one for active corruptions. (*) Communication complexities in [3, 4] are amortized. $|\mathbb{S}|$ denotes the size of sharing specification and the maximal secrecy structure $\widetilde{\Sigma}$ can be used [32] as $\mathbb{S}$. d is the size of a monotone span program, which is the number of rows of the matrix M. Notation details are further clarified in Section 4 and detail communication complexity analysis for our work is provided in Table 2 in Section 5.1 and Table 3 in Section 5.2.

Proactive Secure Multiparty Computation (PMPC): To the best of our knowledge, there are currently only a few PMPC protocols, e.g., [33] requiring $O(Cn^3)$ communication, where C is the size of the circuit to be computed via MPC, and [3] requiring $O(Clog^2(C)polylog(n) + Dpoly(n)log^2(C))$. Existing PMPC protocols are only specified for threshold adversary structures and cannot be easily[7] augmented to handle GAS; this is because these protocols all rely on secret sharing via polynomials. In addition, all current PMPC can only tolerate dishonest minorities, except one protocol [22], but even that one is only limited to the threshold adversary structure. The reason is that the underlying SS stores secrets as points on polynomials so once the adversary compromises enough parties (even if only passively), it can reconstruct the polynomial and recover the secret. The only structure that can be described is one in terms of a fraction of the degree of the polynomial (typically also a fraction of number of parties) and once the adversary compromises enough parties (even if only passively), it can reconstruct the polynomial and recover the secret.

[7] Or at least it is not obvious to us how to easily augment them to accommodate GAS.

A.2 Types of Security and Communication Models

MPC literature distinguishes between two types of security, perfect (or information-theoretic) security and cryptographic security. Protocols with information-theoretic security can withstand an adversary with unrestricted computing power. On the other hand, protocols with cryptographic security restrict an adversary's computing power and assume certain assumptions about the hardness of some computational problems, e.g., factoring large integers or computing discrete logarithms. In this paper, we consider protocol with perfect security for general adversary structures (GAS) consisting of passive adversaries, and cryptographic security for GAS consisting of active adversaries or mixed (both passive and active) adversaries.

For communication models, the literature consider the following two models: synchronous and asynchronous. In asynchronous communication models, there is no guarantees about data transmission between sender and receiver. In contrast, synchronous communication models (which we consider in this paper) guarantee that any pair of parties can communicate over a bilateral secure channel. That is, when a sender sends data to a receiver, the receiver is guaranteed to get data in certain times. The synchronous communication models sometimes include a broadcast channel (as we consider in this paper). The broadcast channel guarantees consistency of received values, when a sender sends a value to several parties.

A.3 Adversary Models

An adversary's capability can be described by a corruption type and an adversary structure. The adversary structure Δ is a set of subsets of parties that are potentially corruptible. The adversary can choose a set of parties in Δ and corrupt all the parties listed in the set. The corruption types are classified as passive corruptions, active corruptions, and both.

Passive Adversary Models The adversary can eavesdrop all the view of corrupted parties, but cannot forge the process that parties should follow. i.e., The corrupted parties should follow the protocol honestly. This type of adversaries are also called honest-but-curious (HBC). In HBC models, the adversary structure Δ is equal to the secrecy structure Σ.

Active Adversary Models The adversary can take full control of the corrupted parties and can make them behave arbitrarily from the protocol. i.e., The adversary can forge the messages of corrupted parties as well as eavesdrop all of their views.

Threshold Adversary Models In the classic t-threshold SS and MPC, adversaries are assumed to be able to either passively corrupt or actively corrupt up to t parties. The adversary structure in this t-threshold models is the set of all subsets of P which size is at most t.

General Adversary Models General adversary structures (GAS) extend this threshold settings to non-threshold models. The adversary can actively corrupt a subset of parties and passively corrupt another subset of parties. Sometimes it is classified as in general adversary models and mixed general adversary models as follows: The former has adversaries who can either passively corrupt or actively corrupt the parties, while the latter has adversaries who can do both passive corruptions and active corruptions. We collectively describe both models as general adversary models in this paper. The adversary is specified by the secrecy structure Σ and the adversary structure $\Delta \subseteq \Sigma$. The (Σ, Δ)-adversary denotes the adversary that can passively corrupt some parties in a set A and actively corrupt some parties in a set B, where $A \in \Delta$ and $(A \cup B) \in \Sigma$.

A.4 Phases and Stages of a Proactively Secure MPC Protocol

We adopt terminology in previous formalization of the proactive security model such as [1, 3].

Phases The rounds of a proactive protocol are grouped into *phases* $\phi_1, \phi_2, \ldots$: a phase ϕ consists of a sequence of consecutive rounds, and every round belongs to exactly one phase. There are two types of phases: *refresh* phases and *operation* phases. The phases alternate between refresh and operation phases; the first and last phase of the protocol are both operation phases. Each refresh phase is furthermore subdivided into a *closing period* consisting of the first k rounds of the phase, followed by an *opening period* consisting of the final $\ell - k$ rounds of the phase, where ℓ is the total number of rounds in the phase.

In non-reactive MPC, the number of operation phases can be thought to correspond to the depth of the circuit to be computed in the MPC. Intuitively, each operation phase serves to compute a layer of the circuit to be computed, and each refresh phase serves to re-randomize the data held by parties such that combining the data of corrupt parties across different phases will not be helpful to an adversary.

Stages A *stage* σ of the protocol consists of an *opening period* of a refresh phase, followed by the subsequent *operation phase*, followed by the *closing period* of the subsequent refresh phase. In the case of the first and last stages of a protocol, there is an exception to the alternating "refresh-operation-refresh" format: the first stage starts with the first operation phase, and the last stage ends with the last operation phase. Thus, a stage spans (but does not cover) three consecutive phases, and the number of stages in a protocol is equal to its number of operation phases. ($\because$ For a protocol Π, if the number of phases in Π is $\#(\phi) = m$, then the number of operation phases in Π is $\#(\text{op}) = \lceil \frac{m}{2} \rceil$ and the number of stages in Π is $\#(\sigma) = \frac{m-3}{2} + 2 = \frac{m+1}{2}$. Since $m = 2m' + 1$ for some $m' \in \mathbb{N}$, $\#(\text{op}) = \#(\sigma) = m' + 1$.)

Stage Changes The adversary $\mathcal{A}$ can trigger a new stage at any point during an operation phase, by sending a special message newstage to all parties. Upon receiving the newstage message, the parties initiate a refresh phase.

Corruptions If a party P_i is corrupted by the adversary $\mathcal{A}$ during an operation phase of a stage σ_j, then $\mathcal{A}$ learns the view of P_i starting from his state at the beginning of stage σ_j. If the corruption is made during a refresh phase between consecutive stages σ_j and σ_{j+1}, then $\mathcal{A}$ learns P_i's view starting from the beginning of stage σ_j. Moreover, in the case of a corruption during a refresh phase, P_i is considered to be corrupt in both stages σ_j and σ_{j+1}.

Finally, if P_i is corrupted during the closing period of a refresh phase in stage σ_j, $\mathcal{A}$ may decide to *decorrupt* him. In this case, P_i is considered to be no longer corrupted in stage σ_{j+1} (unless $\mathcal{A}$ corrupts him again before the end of the next closing period). A decorrupted party P_i immediately rejoins the protocol as an honest party: if P_i was passively corrupted, then it rejoins with the correct state according to the protocol up to this point; or if P_i was actively corrupted, then it is restored to a clean default state (which may be a function of the current round). Note that in restoring a party to the default state, its randomness tapes are overwritten with fresh randomness: this is important since otherwise, any once-corrupted party would be deterministic to the adversary.

Erasing State In our model, parties erase their internal state (i.e., the content of their tapes) between phases. The capability of erasing state is necessary in the proactive model: if an adversary could learn all previous states of a party upon corruption, then achieving security would be impossible, since over the course of a protocol execution a proactive adversary would be able learn the state of *all* parties in certain rounds.

Appendix B Protocols for the Additive MPC Scheme [27]

B.1 Information Checking (IC)

The information checking scheme consists of two protocols, called `Authenticate` and `Verify`. The protocol `Authenticate` generates valid tags for participating parties with respect to the input value, and the protocol `Verify` verifies the input value with respect to the input tags from participating parties. There are three participating parties, a sender P_s, a receiver P_r, and a verifier P_v. The sender P_s sends a message s along with the authentication tag tag_{auth} to P_r, while sending the verification tag tag_{ver} to P_v. Also, we assume that any pair of two parties (P_s, P_v) knows a fixed secret value, denoted by $\alpha_{s,v} \in \mathbb{F} \setminus \{0, 1\}$.

Definition B.1. *A vector (s, y, z, α) is **1-consistent** if there exists a polynomial f of degree 1 over $\mathbb{F}$ such that $f(0) = s, f(1) = y, f(\alpha) = z$.*

Definition B.2. *A value s is called (P_s, P_r, P_v)**-authenticated** if P_r knows s and some authentication tag y and P_v knows a verification tag z such that $(s, y, z, \alpha_{s,v})$ is 1-consistent. The vector $(y, z, \alpha_{s,v})$ is denoted by $A_{s,r,v}(s)$*

Protocol $\texttt{Authenticate}^{\texttt{A}}(P_i, P_j, P_k, w, s) \longrightarrow (y, z)$ or $\perp$

Input: P_i and P_j holding s, $P_k \in \mathcal{P}$, a phase w, and a value s
Output: a pair of authentication tag y and verification tag z, or aborted

1. P_i chooses random values $(y, z) \in \mathbb{F}^2$ such that $(s, y, z, \alpha_{i,k})$ is 1-consistent, and random values $(s', y', z') \in \mathbb{F}^3$ such that $(s', y', z', \alpha_{i,k})$ is 1-consistent, and sends (s', y, y') to P_j and (z, z') to P_k.
2. P_k broadcasts random value $r \in \mathbb{F}$.
3. P_i broadcasts $s'' := rs + s'$ and $y'' := ry + y'$.
4. P_j checks if the forwarded values s'', y'' are correct by comparing $s'' \stackrel{?}{=} rs + s'$ and $y'' \stackrel{?}{=} ry + y'$ and broadcasts OK/NOK. If NOK is broadcasted, then P_j adds P_i to the list $\mathcal{D}_j$ and the protocol is aborted, which outputs $\perp$.
5. P_k checks if $(s'', y'', rz + z', \alpha_{i,k})$ is 1-consistent. If it is, then P_k sends OK to P_j. Otherwise, P_k sends $(\alpha_{i,k}, z)$ to P_j and adds P_i to the list $\mathcal{D}_k$. When P_j receives $(\alpha_{i,k}, z)$, P_j adjusts y such that $(s, y, z, \alpha_{i,k})$ is 1-consistent.
6. P_j outputs y as the authentication tag and P_k outputs z as the verification tag.

The protocol $\texttt{Authenticate}^{\texttt{A}}$ allows P_i to securely (P_i, P_j, P_k)-authenticate the value s. If P_k is honest and s is known to the honest parties $\{P_i, P_j\}$, then $\texttt{Authenticate}^{\texttt{A}}(P_i, P_j, P_k, s)$ either securely (P_i, P_j, P_k)-authenticate s or aborts with error probability at most $1/|\mathbb{F}|$.

Assume that P_k knows a candidate s' or a (P_i, P_j, P_k)-authenticates value s and P_j wants to prove the authenticity of s'. The protocol $\texttt{Verify}^{\texttt{A}}$ allows parties to authenticate s' with their tags. If P_k and P_j are honest parties knowing $s' = s$, P_k will output s in $\texttt{Verify}^{\texttt{A}}$, or output $\perp$ otherwise, except with error probability at most $1/(\mathbb{F} - 2)$.

Protocol $\texttt{Verify}^{\texttt{A}}(P_i, P_j, P_k, w, s', A_{i,j,k}(s)) \longrightarrow s$ or $\perp$

Input: a candidate value s' known to P_j and P_k for a (P_i, P_j, P_k)-authenticated value s, a phase w, and the authentication for s, $A_{i,j,k}(s) = (y, z, \alpha_{i,k})$ where P_j has the authentication tag y, and P_k has the verification tag z
Output: s or $\perp$

1. P_j sends y to P_k.
2. P_k checks if $(s', y, z, \alpha_{i,k})$ is 1-consistent and outputs s' if it is. Otherwise, P_k adds P_j to the list $\mathcal{D}_k$ and outputs $\perp$.

As the parties use local dispute control, even though the adversary has at most n^2 attempts to cheat, the total error probability of arbitrary many instances of each protocols is at most $O(n^2/|\mathbb{F}|)$, which is independent of secrecy structure. The security proofs are provided in [27].

B.2 Secret Sharing

For any adversary structure Δ, the following protocol allows a dealer P_D to securely share a secret value s among n parties in $\mathcal{P}$ with respect to the sharing specification $\mathbb{S} = (S_1, ..., S_k)$.

---

[27] Protocol $\texttt{Share}^{\texttt{A}}(w, s, \mathcal{P}) \longrightarrow [s]^w$

Input: a phase w, a secret value s, and a set $\mathcal{P}$ of parties who receives the shares
Output: k shares of s in phase w

1. A dealing party P_D chooses $k-1$ random integers, $s_1, ..., s_{k-1} \stackrel{\$}{\leftarrow} \mathbb{F}$, and set up the k-th share as $s_k := s - \sum_{i=1}^{k-1} s_i$.
2. For all $i \in \{1, ..., k\}$, do the following:
3. P_D sends s_i to every party in S_i.
4. $\forall P_a, P_b \in S_i$ and $\forall P_c \in \mathcal{P}$ invoke $\texttt{Authenticate}^{\texttt{A}}(P_a, P_b, P_c, s_i)$. If any result was aborted, P_D broadcasts s_i, the parties in S_i replace their share, and $\forall P_a, P_b \in S_i$ and $\forall P_c \in \mathcal{P}$ set the forwarded value s_i as the tags, authentication tag and the verification tag.
5. The parties in $\mathcal{P}$ collectively output $[s]$.

For any qualified subset B in Γ, the protocol $\texttt{Reconstruct}^{\texttt{A}}$ enables each parties in B to reconstruct (or access) the secret as below. The value for s_q forwarded from P_j to P_k is denoted by $s_q^{(j,k)}$. It is shown in [27] that the following protocol securely reconstruct s to the parties in B, if $Q^2(\mathbb{S}, \Delta)$ is met. i.e., For $\forall Z_1, Z_2 \in \Delta$, $\forall S \in \mathbb{S}$, $S \not\subseteq Z_1 \cup Z_2$.

---

[27] Protocol $\texttt{Reconstruct}^{\texttt{A}}(w, [s], B) \longrightarrow s$ or $\perp$

Input: a phase w, the sharing of s (collectively), and a set B of parties participating in reconstruction
Output: s or aborted

1. For all $q \in \{1, ..., k\}$, do the following:
2. Every party in S_q sends s_q to each party in B.
3. For all $P_j \in S_q$ and $P_k \in B$,
4. $\texttt{Verify}^{\texttt{A}}(P_i, P_j, P_k, w, s_q^{(j,k)}, A_{i,j,k}(s_q))$ is invoked for $\forall P_i \in S_q$. If P_k outputs $s_q^{(j,k)}$ in each invocation, P_k accepts it as value for s_q.
5. Each $P_k \in B$ outputs $\perp$ if he never accepted in Step 4.
6. Each party in B locally adds up the accepted shares, and output the sum.

B.3 Addition and Multiplication

Assuming the shares for the values s and t are already shared among $\mathcal{P}$, addition of s and t can be done naturally even without any interaction among n parties Because of the linearity, each party can locally add two shares and set it as the new share for $s+t$. i.e., $[s+t] = \{(s+t)_1, ..., (s+t)_k\}$ where $(s+t)_i; = s_i + t_i$. It can be formally specified as follows:

---

[27] Protocol $\texttt{Add}^{\texttt{A}}(w, [s], [t]) \longrightarrow [s+t]$

Input: phase w, shares of s, and shares of t
Output: new shares of $s+t$

Precondition: Two values $s = \sum_{i=1}^{k} s_i$ and $t = \sum_{i=1}^{k} t_i$ are shared
Postcondition: $s+t$ is shared independently

1. Each party P_h locally adds each share of s to the share of t and keep the result as a share of $s+t$. i.e., $(s+t)_i; = s_i + t_i$ for each $i \in \{i \in \{1, ..., k\} | P_h \in S_i\}$.

On the other hand, it is quite tricky and requires a lot of communications to securely form the share of $(s * t)$ among n parties, as $s \cdot t = \sum_{i=1}^{k} \sum_{j=1}^{k} (s_i \cdot t_i)$. To securely form the share of $(s * t)$ among n parties, where s and t are pre-shared through $\texttt{Share}^{\texttt{A}}$ protocol, participating parties need to perform the protocol $\texttt{Multiply}^{\texttt{A}}$. The basic idea is adopted from [32]. Each party computes the local product $(s_p * s_q)$ for all s_p and s_q that the party holds and shares it. In addition, a probabilistic check is performed in each loop to identify corrupted parties. For privacy, the multiplication of random values is used instead of actual multiplying values. The protocol $\texttt{Multiply}^{\texttt{A}}$ is as follows:

---

[27] Protocol $\texttt{Multiply}^{\texttt{A}}(w, [x], [y]) \longrightarrow [xy]$

Input: a phase w, a sharing of x, and a sharing of y, collectively
Output: a sharing of xy

1. Set $M = \emptyset$ and invoke $\texttt{RandomTriple}^{\texttt{A}}(w, M)$.
2. If the protocol outputs M', then repeat Step 1 with M'. Otherwise, use the output as random multiplication triple $([a], [b], [c])$ such that $c = ab$.
3. Each party locally computes $[d_x] := [x] - [a]$ and $[d_y] := [y] - [b]$.

4. parties invoke $\texttt{Reconstruct}^{\texttt{A}}(w, [d_x], \mathcal{P})$ and $\texttt{Reconstruct}^{\texttt{A}}(w, [d_y], \mathcal{P})$ to get d_x and d_y, and locally compute $d_x d_y + d_x[b] + d_y[a] + [c]$ and set it as the share of xy.

The protocol $\texttt{Multiply}^{\texttt{A}}$ uses the protocol $\texttt{RandomTriple}^{\texttt{A}}$ as a subprotocol to obtain a random multiplication triple $([a], [b], [c])$ such that $c = ab$, and compute the sharing of xy by computing $xy = ((x-a)+a)((y-b)+b) = (d_x + a)(d_y + b) = d_x d_y + d_x b + d_y a + ab = d_x d_y + d_x b + d_y a + c$. In the protocol $\texttt{RandomTriple}^{\texttt{A}}$, $I_Z(i)$ denotes the set of pairs of shares assigned to P_i, i.e., $I_Z(i) := \{(p, q) | P_i = min_{\mathcal{P}}(P \in (S_p \cap S_q) \setminus Z)\}$, for some $Z \in \Delta$.

---

[27] Protocol $\texttt{RandomTriple}^{\texttt{A}}(w, M) \longrightarrow ([a], [b], [c])$ or M'

Input: a phase w, and a set of (identified) malicious parties M
Output: a random multiplication triple $([a], [b], [c])$ or a set M' such that $M \subsetneq M'$

1. Parties generate random shared values $[a], [b], [b'], [r]$ by summing up shared random values (one from each party) for each value.
2. $\texttt{BasicMultiply}^{\texttt{A}}([a], [b], M)$ is invoked to compute $(([c_1], ..., [c_n]), [c])$ and $\texttt{BasicMultiply}^{\texttt{A}}([a], [b'], M)$ is invoked to compute $(([c'_1], ..., [c'_n]), [c'])$.
3. $\texttt{Reconstruct}^{\texttt{A}}(w, [r], \mathcal{P})$ is invoked and each party gets the value r.
4. Each party locally computes $[e] := r[b] + [b']$.
5. $\texttt{Reconstruct}^{\texttt{A}}(w, [e], \mathcal{P})$ is invoked and each party gets the value e.
6. Each party locally computes $[d] := e[a] - r[c] - [c']$.
7. $\texttt{Reconstruct}^{\texttt{A}}(w, [d], \mathcal{P})$ is invoked and each party gets the value d.
8. If $d = 0$, each party collectively outputs $([a], [b], [c])$. Otherwise, reconstruct the sharings $[a], [b], [b'], [c_1], ..., [c_n], [c'_1], ..., [c'_n]$ and output $M' := M \cup \{P_i : rc_i + c'_i \neq \sum_{(p,q) \in I(i)} r(a_p b_q) + (a_p b'_q)\}$.

The protocol $\texttt{RandomTriple}^{\texttt{A}}$ uses the subprotocol $\texttt{BasicMultiply}^{\texttt{A}}$, which inputs the sharings of two values, $[a]$ and $[b]$, and a set of malicious parties M, and outputs the sharing of $c = ab$ and the sharing of the shares c_i's of c such that $[c] = \sum_{i=1}^{n} [c_i]$, if no more actively corrupted parties exist in $\mathcal{P} \setminus M$ as below.

---

[27] Protocol $\texttt{BasicMultiply}^{\texttt{A}}(w, [a], [b], M) \longrightarrow ([c_1], ..., [c_n]), [c])$ or $\perp$

Input: a phase w, the sharings of a and b, collectively, and a set of (identified) malicious parties M
Output: $([c_1], ..., [c_n])$ and $[c] = \sum_{i=1}^{n} [c_i]$, if no party in $\mathcal{P} \setminus M$ actively cheats

1. For all S_q such that $S_q \cap M \neq \emptyset$,
2. Every party in S_q sends their holding values for a_q and b_q to each other.
3. For all $P_j, P_k \in S_q$, $\texttt{Verify}^{\texttt{A}}(P_i, P_j, P_k, w, a_q^{(j,k)}, A_{i,j,k}(a_q))$ and $\texttt{Verify}^{\texttt{A}}(P_i, P_j, P_k, w, b_q^{(j,k)}, A_{i,j,k}(b_q))$ are invoked
for $\forall P_i \in S_q$. If P_k outputs $a_q^{(j,k)}$(or $b_q^{(j,k)}$, respectively) in each invocation, P_k accepts it as value for a_q (or b_q).
If all output $\perp$, the protocol is aborted.
4. a) Each party $P_i \in \mathcal{P} \backslash M$ locally computes and shares $c_i = \sum_{(p,q) \in I(i)} a_p b_q$, where $I(i) := \{p, q) | P_i = min_P(P \in S_p \cap S_q)\}$.
b) Each party $P_i \in M$ sets the sharing of c_i as $(c_i, 0, ..., 0)$ where $c_i = \sum_{(p,q) \in I(i)} a_p b_q$, and $\forall P_j, P_k$ set corresponding tags as $y_j = [c_i]_j, z_j = [c_i]_j$, for $j = 1, ..., k$. i.e., The tags are (c_i, c_i) only for $[c_i]_1$ and the rest is $(0, 0)$ for all $[c_i]_j$.
5. parties in $\mathcal{P}$ collectively output $([c_1], ..., [c_n])$ and $[c] = \sum_{i=1}^{n} [c_i]$.

Appendix C Our Protocols for the Additive PMPC and Their Proofs

C.1 The Protocol $\texttt{Refresh}^{\texttt{A}}$

Protocol $\texttt{Refresh}^{\texttt{A}}(w, [s]) \longrightarrow [s]^{w+1}$

Input: a phase w and a sharing of s
Output: new sharing of s in phase $w+1$, $[s]^{w+1}$

1. Every party P_i in $\mathcal{P}$ invokes $\texttt{Share}^{\texttt{A}}(w, 0, \mathcal{P})$. (in parallel)
2. Each party adds all shares received in Step 1 to shares of s, and set result as the new share of s in phase $w+1$.
3. parties in $\mathcal{P}$ collectively output $[s]^{w+1}$.

Theorem C.3. *(Correctness and Secrecy of $\texttt{Refresh}^{\texttt{A}}$) When the protocol $\texttt{Refresh}^{\texttt{A}}$ terminates, all parties receive new shares encoding the same secret as old shares they had before with error probability $n^4|\mathbb{S}|/|\mathbb{F}|$, and they cannot get any information about the secret by the execution of the protocol. It communicates $|\mathbb{S}|(7n^4+n^2)\log|\mathbb{F}|$ bits and broadcasts $|\mathbb{S}|((3n^4+n)\log|\mathbb{F}|+n^3)$ bits.*

Proof. Because of the linearity of the authentication, each party can locally set up corresponding authentication tag y and verification tag z. Let $r_{i,q}$ be the q-th share of zero from P_i. Then, for $[s]_q^w := (s_q, A_{i,j,k}(s_q))$, the new q-th share of s in phase $w+1$ is $[s]_q^{w+1} = (s'_q, A_{i,j,k}(s'_q))$, where $s'_q = s_q + \sum_{i=1}^{n} r_{i,q}$ and $A_{i,j,k}(s'_q) = (y_{s_q} + \sum_{i=1}^{n} y_{r_{i,q}}, z_{s_q} + \sum_{i=1}^{n} z_{r_{i,q}}, \alpha_{i,k})$. Also, since every party shares the sharing of zero, new sharing of s also reconstruct the same value s as follows:

$$\sum_{q=1}^{h} s'_q = \sum_{q=1}^{h}(s_q + \sum_{i=1}^{n} r_{i,q}) = \sum_{q=1}^{h} s_q + \sum_{q=1}^{h}\sum_{i=1}^{n} r_{i,q} = \sum_{q=1}^{h} s_q + \sum_{i=1}^{n}\sum_{q=1}^{h} r_{i,q} = s + \sum_{i=1}^{n} 0 = s$$

In addition, each q-th share is verified with $A_{i,j,k}(s'_q) = (y', z', \alpha_{i,k})$ because $(s'_q, y', z', \alpha_{i,k})$ is 1-consistent for $\forall P_i, P_j \in S_q$ and $\forall P_k \in B$ for $\forall B \in \Gamma$. i.e.,

$$(f+F)(0) = f(0) + F(0) = s_q + \sum_{q=1}^{h} r_{i,q} = s'_q,$$

$$(f+F)(1) = f(1) + F(1) = y_{s_q} + \sum_{i=1}^{n} y_{r_{i,q}} = y',$$

and

$$(f+F)(\alpha_{i,k}) = f(\alpha_{i,k}) + F(\alpha_{i,k}) = z_{s_q} + \sum_{i=1}^{n} z_{r_{i,q}} = z'$$

As every party in $\mathcal{P}$ invokes $\texttt{Share}^{\texttt{A}}$, they communicate $n * Cost(\texttt{Share}^{\texttt{A}})$ bits. □

C.2 The Protocol $\texttt{Recover}^{\texttt{A}}$

Protocol $\texttt{Recover}^{\texttt{A}}(w, [s], R) \longrightarrow [s]^{w+1}$ or $\perp$

Input: a phase w, a sharing of s, and a set of rebooted parties R
Output: new sharing of s in phase $w+1$, $[s]^{w+1}$, or aborted

1. Parties in $\mathcal{P}$ invoke $\texttt{ShareRandom}^{\texttt{A}}(w, \mathcal{P})$ to generate a sharing $[r]$ of r, where r is a random number in $\mathbb{F}$.
2. Each party in $\mathcal{P} \setminus R$ invokes $\texttt{Add}^{\texttt{A}}(w, [r], [s])$ to share the sharing of $r+s$.
3. $\texttt{Reconstruct}^{\texttt{A}}(w, [r+s], \mathcal{P} \setminus R)$ is invoked and every party in $\mathcal{P} \setminus R$ gets $r' := r+s$.
4. $\texttt{RobustReshare}^{\texttt{A}}(w, r', \mathcal{P} \setminus R, \mathcal{P})$ is invoked, and each party in $\mathcal{P}$ gets $[r']$.
5. Each party computes $[r'] - [r]$ by executing $\texttt{Add}^{\texttt{A}}(w, [r'], -[r])$, where $-[r]$ is the additive inverses of the shares in $\mathbb{F}$.

Theorem C.4. *(Correctness and Secrecy of* $\texttt{Recover}^{\texttt{A}}$*) If* $\mathbb{S}$ *and* $\mathcal{Z}$ *satisfy* $\mathcal{Q}^1(\mathbb{S},\mathcal{Z})$*, the protocol* $\texttt{Recover}^{\texttt{A}}$ *allows a set* $\forall R \in \Delta$ *of rebooted parties to recover their shares encoding the same secret with error probability* $O((n-|R|)|\mathbb{S}|n^3/|\mathbb{F}| + (n-|R|)|\mathbb{S}|n^2/(|\mathbb{F}|-2))$*, and does not reveal any additional information about the secret. It communicates* $O((n-|R|)|\mathbb{S}|n^3\log|\mathbb{F}|)$ *bits and broadcasts* $O((n-|R|)|\mathbb{S}|n^3\log|\mathbb{F}|)$ *bits.*

Proof. **Correctness:** Since all parties in $\mathcal{P}$ holds both sharings of r and r' and by linearity of additive sharing, each party's locally computing value is $[r']-[r]=[r+s]+[-r]=[(r+s)-r]=[s]$. As $\texttt{Reconstruct}^{\texttt{A}}$ terminates with error probability $n^2|\mathbb{S}|/(|\mathbb{F}|-2)$ and $\texttt{RobustReshare}^{\texttt{A}}$ has error probability $O(|\mathcal{P}_S||\mathbb{S}|n^3/|\mathbb{F}|+|\mathcal{P}_S||\mathbb{S}|n^2/(|\mathbb{F}|-2))$, the protocol $\texttt{Recover}^{\texttt{A}}$ successfully ends with error probability $O((n-|R|)|\mathbb{S}|n^3/|\mathbb{F}|+(n-|R|)|\mathbb{S}|n^2/(|\mathbb{F}|-2))$. **Secrecy:** As each party locally adds its holding share of r and the share of s without reconstructing r or s, all they can see is each sharing of r' and the reconstructed value r'. Since r is a random shared element in $\mathbb{F}$, $r'=r+s$ is also random in $\mathbb{F}$ and it does not reveal any information about s without reconstructing r. Each party can sync the sharing of r' by $\texttt{RobustReshare}^{\texttt{A}}$. **Communication:** Recall that the protocol $\texttt{ShareRandom}^{\texttt{A}}$ communicates $|\mathbb{S}|n$ values in $\mathbb{F}$ and also broadcasts $|\mathbb{S}|n$ values in $\mathbb{F}$, the protocol $\texttt{RobustReshare}^{\texttt{A}}$ communicates/broadcasts $O(|\mathcal{P}_S||\mathbb{S}|n^3)$ values in $\mathbb{F}$, and the protocol $\texttt{Reconstruct}^{\texttt{A}}$ communicates $|\mathbb{S}|(n^3+n^2)$ values in $\mathbb{F}$ without broadcasting. Therefore, the total communication complexity is $|\mathbb{S}|n^2+|\mathbb{S}|(n^3+n^2)+O((n-|R|)|\mathbb{S}|n^3)=O((n-|R|)|\mathbb{S}|n^3)$, and the total broadcast complexity is $|\mathbb{S}|n^2+O((n-|R|)|\mathbb{S}|n^3)=O((n-|R|)|\mathbb{S}|n^3)$. □

C.3 The Protocol $\texttt{Redistribute}^{\texttt{A}}$

Protocol $\texttt{Redistribute}^{\texttt{A}}(w,s)\longrightarrow[s]^{w+1}$

Input: phase w and a secret s
Output: shares of s in phase $w+1$
Precondition: parties in P share $[s]^w$ for a secret s
Postcondition: parties in P' share $[s]^{w+1}$ encoding the same secret s

1. For each $S_i \in \mathbb{S}$:
2. Each party P_y in S_i forwards its holding value $[s_i]_y$ for s_i to every party in S_i who is supposed to hold the same share (over the secure channel).
3. $\texttt{Verify}^{\texttt{A}}(P_S,P_R,P_V,w,[s_i]_y,A_{S,R,V}(s_i))$ is invoked for all $P_R,P_V\in S_i$, $\forall P_S\in S_i$. If P_V outputs $[s_i]_y$ in each invocation, P_V accepts it as value for s_i. Denote v_i as the accepted value for s_i, for each i.
4. Each party $P_y\in S_i$ runs $\texttt{Share}^{\texttt{A}}(w+1,v_i,\mathcal{P}')$ according to $\mathbb{S}'$. i.e., The j-th share of v_i, v_{ij}, is sent from P_y in S_i to all parties in $S'_j\in\mathbb{S}'$ and $\texttt{Authenticate}^{\texttt{A}}$ is invoked for each share of v_i.
5. For each $S'_j\in\mathbb{S}'$:
6. Each party in S'_j holds $\{v_{ij}\}_{i=1}^k$. For each v_{ij}, all $P_R,P_V\in S'_j$ invoke $\texttt{Verify}^{\texttt{A}}(P_S,P_R,P_V,w,v_{ij},A_{S,R,V}(v_{ij}))$ for $\forall P_S\in S'_j$ and accept the output value as v_{ij}.
7. Each party in S'_j sums up all k values accpeted in step 6, and set it as new j-th share of s. i.e., $s'_j:=\sum_{i=1}^k v_{ij}$.

Theorem C.5. *(Correctness and Secrecy of* $\texttt{Redistribute}^{\texttt{A}}$*) By executing the protocol* $\texttt{Redistribute}^{\texttt{A}}$*, new participating parties receive a sharing of the same secret as the old shares with error probability* $((|\mathbb{S}|+|\mathbb{S}'|)n^3/(|\mathbb{F}|-2)+n^4|\mathbb{S}|/|\mathbb{F}|)$ *and it does not reveal any additional information about the secret. It communicates* $O(|\mathbb{S}|^2n^4\log|\mathbb{F}|+|\mathbb{S}'|n^3\log|\mathbb{F}|)$ *bits and broadcasts* $O(|\mathbb{S}|^2n^4\log|\mathbb{F}|)$ *bits, where* $\mathbb{S}$ *and* $\mathbb{S}'$ *denote the sets for sharing specification in two consecutive phases.*

Proof. **Correctness:** The new sharing reconstructs the same secret s, as

$$\sum_{j=1}^{k'} s'_j=\sum_{j=1}^{k'}\sum_{i=1}^{k} v_{ij}=\sum_{i=1}^{k}\sum_{j=1}^{k'} v_{ij}=\sum_{i=1}^{k} v_i=\sum_{i=1}^{k} s_i=s.$$

For error probability, as the error probability of $\texttt{Verify}^{\texttt{A}}$ is $1/(|\mathbb{F}|-2)$ and the one of $\texttt{Share}^{\texttt{A}}$ is $n^3|\mathbb{S}|/|\mathbb{F}|$, the protocol $\texttt{Redistribute}^{\texttt{A}}$ outputs new sharing of s with error probability $|\mathbb{S}|n^3Err(\texttt{Verify}^{\texttt{A}})+max|S_i|Err(\texttt{Share}^{\texttt{A}})+|\mathbb{S}'|n^3Err(\texttt{Verify}^{\texttt{A}})=(|\mathbb{S}|+|\mathbb{S}'|)n^3/(|\mathbb{F}|-2)+n^4|\mathbb{S}|/|\mathbb{F}|$. **Secrecy:** Each party forwards their share to the parties who are supposed to have the same share, Step 1 does not reveal additional information about the share. Step 3 to 6 reply on the secrecy of the protocols $\texttt{Verify}^{\texttt{A}}$ and $\texttt{Share}^{\texttt{A}}$, and Step 7 is local computation, which does not reveal any. **Communication:** In Step 1, each party in S_i sends their share value to each other, so

they communicate $O(max|S_i|^2 \log|\mathbb{F}|)$ bits for each $i = 1, ..., |\mathbb{S}|$. Thus, the total communication complexity is $O(|\mathbb{S}|(n^2 \log|\mathbb{F}| + n^3 Cost(\texttt{Verify}^{\texttt{A}}) + nCost(\texttt{Share}^{\texttt{A}})) + |\mathbb{S}'|(n^3 Cost(\texttt{Verify}^{\texttt{A}}))) = O(|\mathbb{S}|^2 n^4 \log|\mathbb{F}| + |\mathbb{S}'| n^3 \log|\mathbb{F}|)$ bits and the total broadcast is $|\mathbb{S}| nCost(\texttt{Share}^{\texttt{A}}) = O(|\mathbb{S}|^2 n^4 \log|\mathbb{F}|)$. □

Appendix D Protocols for the MSP-based MPC Scheme [29]

D.1 Information Checking

In [29], they use the variant of [6] for information checking. They use an extension field $\mathbb{G}$ over $\mathbb{F}$ such that the field $\mathbb{G}$ has minimal size satisfying $|\mathbb{G}| \geq d|\mathbb{F}|$, to allow the sender to produce tags for messages of length at most d. Note that κ is a security parameter, P_S is the sender, P_R is the receiver, and P_V is the verifier.

Protocol $\texttt{Authenticate}^{\texttt{M}}(P_S, P_R, P_V, w, \mathbf{s}) \longrightarrow (y, z)$ or $\perp$

Input: the sender P_S and the receiver P_R both knowing $\mathbf{s}$, the verifier P_V, a phase w, and a vector of secret values $\mathbf{s} = (s^{(1)}, ..., s^{(l)}) \in \mathbb{F}^l$ s.t. $l \leq d$

Output: a pair of tags (y, z), where $y = \{y_i\}_{i=1}^{\kappa}$ is a set of authentication tags and $z = \{z_i\}_{i=1}^{\kappa}$ is a set of verification tags, or aborted ($\perp$)

1. P_S picks 2κ random elements $y_1, ..., y_\kappa, u_1, ..., u_\kappa \in \mathbb{G}$.
2. For each $i = 1, ..., \kappa$, P_S determines v_i such that the $(l+2)$ points, $(0, y_i), (1, s^{(1)}), ..., (l, s^{(l)}), (u_i, v_i)$ lie on a polynomial of degree l over $\mathbb{G}$.
3. P_S sends $y_1, ..., y_\kappa$ to P_R and $z_1, ..., z_\kappa$ to P_V, where $z_i = (u_i, v_i)$ for each i.
4. P_V partitions the set $\{1, ..., \kappa\}$ into two sets I and $\overline{I}$ of almost equal size $(||I| - |\overline{I}|| \leq 1)$, and sends $\{z_i\}_{i \in I}$ to P_R.
5. P_R checks if the $(l+2)$ points, $(0, y_i), (1, s^{(1)}), ..., (l, s^{(l)}), (u_i, v_i)$ lie on a polynomial of degree l, for each z_i. P_R broadcasts NOK, if any of these checks fails, or OK, otherwise.
6. Only if P_R broadcasts NOK in Step 5, the followings are executed:
   a) P_R picks one z_i that failed the check and broadcasts (i, z_i).
   b) P_S and P_V broadcast z_i for i received in Step a).
   c) Based on the values broadcasted in Step a) and b), a pair $\{P_i, P_j\}$ of parties is added to the dispute list $\mathcal{D}$, where P_i, P_j are two parties over P_S, P_R, P_V such that their broadcasted values are different. The protocol is aborted.
7. Output $\{y_i\}_{i=1}^{\kappa}$ as authentication tags and $\{z_i\}_{i=1}^{\kappa}$ as verification tags.

If the protocol $\texttt{Authenticate}^{\texttt{M}}$ succeeds, P_R receives the messages and authentication tags and P_V receives verification tags, which obtains no information about the messages. On the other hand, the following protocol $\texttt{Verify}^{\texttt{M}}$ allows P_V to verify the authenticity of the messages P_R requested as below. The security proofs are shown in [6] and [29].

Protocol $\texttt{Verify}^{\texttt{M}}(P_S, P_R, P_V, w, \mathbf{s}', (y, z)) \longrightarrow \mathbf{s}'$ or $\perp$

Input: a phase w, a candidate vector $\mathbf{s}'$ for $\mathbf{s} = (s^{(1)}, ..., s^{(l)}) \in \mathbb{F}^l$, the authentication tag y that P_R has, and the verification tag z that P_V has

Output: s or $\perp$

1. P_R sends $\mathbf{s}' = (s'^{(1)}, ..., s'^{(l)})$ and authentication tags $y = \{y_i\}_{i \in \overline{I}}$ to P_V.
2. P_V broadcasts OK and outputs $\mathbf{s}'$, if the points $(0, y_i), (1, s^{(1)}), ..., (l, s^{(l)}), (u_i, v_i)$ form a polynomial of degree l, for any $i \in \overline{I}$. Otherwise, P_V broadcasts NOK and the protocol is aborted.

As shown in [29], the communication complexity of $\texttt{Authenticate}^{\texttt{M}}$ is $O(\kappa \log d)$, and the one of $\texttt{Verify}^{\texttt{M}}$ is $O(l + \kappa \log d)$, with negligible error probability less than $\kappa/(d(2^\kappa - 1) - 1)$.

D.2 Secret Sharing

In [29], multiple secrets are generated from a dealer and one pair of authentication and verification tags is generated for multiple secrets. To compare with additive SS, which a dealer share one secret value per one execution of the protocol $\texttt{Share}^{\texttt{M}}$, we present naturally reduced version of $\texttt{Share}^{\texttt{M}}$ protocol, where a dealer shares one secret per execution of the protocol. The original protocols for SS are shown in this section. For clarifying, we call the original SS scheme in [29] as $\texttt{ShareMultiple}^{\texttt{M}}$ and $\texttt{ReconstructMultiple}^{\texttt{M}}$, and the reduced version of SS scheme in this paper as $\texttt{Share}^{\texttt{M}}$ and $\texttt{Reconstruct}^{\texttt{M}}$. We only present the protocols $\texttt{Share}^{\texttt{M}}$ and $\texttt{Reconstruct}^{\texttt{M}}$ in this paper.

The protocol $\texttt{Share}^{\texttt{M}}$ uses the protocol $\texttt{BasicShare}^{\texttt{M}}$ as a subprotocol to make it tolerable against active adversaries. The protocol $\texttt{BasicShare}^{\texttt{M}}$ [15] is a basic secret sharing protocol using a MSP with matrix M of size $d \times e$ described as above. After the execution of this protocol, each party not in dispute with dealing party P_D will get the shares of the secret value s, while the parties in dispute with P_D will receive the all-zero vectors as their shares, called *Kudzu share* [6].

[15, 29] Protocol $\texttt{BasicShare}^{\texttt{M}}(w, s) \longrightarrow [s]^w$

Input: a phase w and a secret value $s \in \mathbb{F}$
Output: the sharing of s in phase w

1. A dealing party P_D constructs a vector $\mathbf{b} := (s, r_2, ..., r_e) \in \mathbb{F}^e$, where $r_i \overset{\$}{\leftarrow} \mathbb{F}$ such that all parties in $\mathcal{D}_D$ will receive the all-zero vectors as their shares.
2. P_D computes $\mathbf{s} = M\mathbf{b}$, where M is the MSP corresponding to Δ.
3. P_D sends $\mathbf{s}_j = M_j\mathbf{b}$ to each $P_j \notin \mathcal{D}_D$, where M_j denotes the matrix collecting all the rows assigned to P_j, i.e., all i's such that $\rho(i) = j$.

For active adversaries, an information checking (IC) and dispute control are used. IC for the MSP-based MPC consists of two protocols, $\texttt{Authenticate}^{\texttt{M}}$ and $\texttt{Verify}^{\texttt{M}}$, described in Appendix D.1. For dispute control, one more list $\mathcal{C}$ is also used, where $\mathcal{C}$ is a set of parties known by all parties to be corrupted. That means, the list $\mathcal{D}$ maintains the parties in each dispute list $\mathcal{D}_i$, for all i and some of them move to the list $\mathcal{C}$ when all parties agree their being corrupted. Our considering $\texttt{Share}^{\texttt{M}}$ protocol is as follows.

---

[29] Protocol $\texttt{Share}^{\texttt{M}}(w, s, \mathcal{P}) \longrightarrow [s]^w$

Input: a phase w, a secret value $s \in \mathbb{F}$, and a group $\mathcal{P}$ of parties receiving shares
Output: the sharing of s in phase w

1. A dealing party P_D chooses n extra random values, $u^{(1)}, ..., u^{(n)}$, then invokes $(n+1)$ $\texttt{BasicShare}^{\texttt{M}}$ (in parallel) for each $\{u^{(i)}\}_{i=1}^n$ and s.
2. For each pair $P_R, P_V \notin \mathcal{D}_D$ such that $\{P_R, P_V\} \notin \mathcal{D}$, $\texttt{Authenticate}^{\texttt{M}}(P_D, P_R, P_V, \mathbf{v}_R)$ is invoked, where $\mathbf{v}_R := (\mathbf{s}_R, \mathbf{u}_R^{(1)}, ..., \mathbf{u}_R^{(n)})$. Note that each $\mathbf{s}_R$ and $\mathbf{u}_R^{(i)}$ is a vector of length d_R so that the length of the vector $\mathbf{v}_R$ is $d_R * (n+1)$.
3. For each $P_V \notin \mathcal{D}_D$, the followings are performed (in parallel):
4. P_V chooses a random vector $r \in \mathbb{F}$ and broadcasts it.
5. Each party $P_i \notin \mathcal{D}_D$ sends his share of $r * \mathbf{s}_i + \mathbf{u}_i^{(V)}$ to P_V. Recall that $\mathbf{s}_i := M_i\mathbf{b}$, where $\mathbf{b}$ is a random vector in $\mathbb{F}^e$ with first component s, and $\mathbf{u}_i^{(V)}$ is similarly defined.
6. If the shares received in Step 5 forms a consistent sharing, then P_V broadcasts OK, or NOK otherwise. i.e., P_V accepts if the sharing is a vector in the span of the matrix $M_{\mathcal{G}}$, where $\mathcal{G} = \mathcal{P} - \mathcal{C}$.
7. For the lowest P_V who broadcast NOK in Step 6, the followings are executed:
   a) P_D broadcasts each share of $r * \mathbf{s}_k + \mathbf{u}_k^{(V)}$ for $k = 1, ..., n$.
   b-1) If this sharing is not in Span($M_{\mathcal{G}}$), then each party adds P_D to his list $\mathcal{D}_i$, i.e., P_D is added to $\mathcal{C}$, and the protocol is aborted.
   b-2) Otherwise, there is a share of some party $P_i \notin \mathcal{D}_D$ which is different from the one broadcasted by P_D. P_V broadcasts $(\texttt{accuse}, P_i, P_D, v_i, v_D)$, where v_i is the share sent by P_i and v_D is the value of the share sent by P_D for i-th share.
   c) If P_i disagrees with the value v_i broadcasted by P_V, then P_i broadcasts $(\texttt{dispute}, P_i, P_V)$ so that the pair $\{P_i, P_V\}$ is added to $\mathcal{D}$, and the protocol is aborted.
   d) If P_D disagrees with the value v_D broadcasted by P_V, then P_D broadcasts $(\texttt{dispute}, P_D, P_V)$, the pair $\{P_D, P_V\}$

is added to $\mathcal{D}$, and the protocol is aborted.

e) If neither P_i nor P_D complained in previous steps, then $\{P_i, P_D\}$ is added to $\mathcal{D}$ and the protocol is aborted.

8. Otherwise, the parties in $\mathcal{P}$ collectively output $[s] := \{\mathbf{s}_1, ..., \mathbf{s}_n\}$ with $\{[u^{(i)}]\}_{i=1}^n$.

$\texttt{Reconstruct}^{\texttt{M}}$ allows parties in $\forall B \in \Gamma$ to reconstruct the secret which has been shared using $\texttt{Share}^{\texttt{M}}$.

---

[29] Protocol $\texttt{Reconstruct}^{\texttt{M}}(w, [s]) \longrightarrow s$ or $\perp$

Input: a phase w and a sharing of s (collectively), shared by P_D
Output: s or aborted

1. Each party in $\mathcal{G} := \mathcal{P} \setminus \mathcal{C}$ holding a non-Kudzu share of $[s]$ broadcasts its share.
2. If the shares broadcast in Step 1 and Kudzu shares form a consistent sharing, i.e. they are in Span($M_\mathcal{G}$), then the protocol terminates with the output of $\langle \lambda_\mathcal{G}, [s]_\mathcal{G} \rangle$, where $\lambda_\mathcal{G}$ is a vector satisfying $M_\mathcal{G}^t \lambda_\mathcal{G} = \mathbf{a}$ and $[s]_\mathcal{G}$ is the recomposition vector with respect to the indexing function ρ with all the shares of parties in $\mathcal{G}$.
3. If the shares broadcast in Step 1 are inconsistent, i.e., not in Span($M_\mathcal{G}$), then P_D broadcasts the index i of each party he accuses of sending an incorrect share.
4. If P_D did not broadcast an index in Step 3, or if the remaining shares after removing the shares that P_D accused are still inconsistent, or if the set of parties in dispute with P_D is no longer in Δ, then P_D is added to $\mathcal{C}$.
5. If $P_D \notin \mathcal{C}$, do the following:
   5.a) For each party P_i accused by P_D in Step 3, parties invoke $\texttt{Verify}(P_D, P_i, P_k, w, \mathbf{v}_i, tags)$ for each party $P_k \notin \mathcal{D}_i \cup \mathcal{D}_j$, where $\mathbf{v}_i = (\mathbf{s}_i, \mathbf{u}_i^{(1)}, ..., \mathbf{u}_i^{(n)})$ as defined in Step 2 of $\texttt{Share}^{\texttt{M}}$.
   5.b) For any P_i who sent a share to P_k that was different than the share broadcast in Step 1, P_k broadcasts ($\texttt{accuse}, i$) and $\{P_k, P_i\}$ is added to the list $\mathcal{D}$.
   5.c) If $P_k \notin \mathcal{D}_i$ rejects in Step 5.a, then $\{P_k, P_j\}$ is added to the list $\mathcal{D}$. Otherwise, $\{P_D, P_k\}$ is added to $\mathcal{D}$.
   5.d) If the shares of parties not in $\mathcal{C}$ (after some parties are added to $\mathcal{C}$) and the Kudzu shares form a consistent sharing, then those shares are used to reconstruct s. Otherwise, P_D is added to $\mathcal{C}$ and proceed to Step 6.
6. If $P_D \in \mathcal{C}$, do the following:
   6.a) For all P_j holding non-Kudzu shares and for all $P_k \notin \mathcal{D}_j$, the parties invoke $\texttt{Verify}^{\texttt{M}}(P_D, P_j, P_k, w, \mathbf{v}_j, tags)$ where $\mathbf{v}_j = (\mathbf{s}_j, \mathbf{u}_j^{(1)}, ..., \mathbf{u}_j^{(n)})$.
   6.b) For any P_j who sent a share to P_k that is different than the share broadcast in Step 1, P_k broadcasts ($\texttt{accuse}, j$) and $\{P_k, P_j\}$ is added to the list $\mathcal{D}$.
   6.c) The shares of parties not in $\mathcal{C}$ are used to reconstruct s as in Step 2.

In Step 2, the correctness holds when $B \in \Gamma \Leftrightarrow f(B) = 1$, which means there is some vector λ_B such that $M_B^t \lambda_B = \mathbf{a}$. Therefore, $\langle \lambda_B, [s]_B \rangle = \langle \lambda_B, M_B \mathbf{b} \rangle = \langle M_B^t \lambda_B, \mathbf{b} \rangle = \langle \mathbf{a}, \mathbf{b} \rangle = s$, as $\mathbf{a} = (1, 0, ..., 0)$ and $\mathbf{b} = (s, r_2, ..., r_e)$. Through out the steps in $\texttt{Reconstruct}$, parties can detect all the potentially corrupted parties. Note that whenever the remaining parties in $\mathcal{G}$ is no longer in Γ, parties cannot reconstruct the secret value.

For $\texttt{LC-Reconstruct}^{\texttt{M}}$, we add more explanations about the assumptions and how this protocol works in detail in addition to the one in [29] as it is not trivial. Intuitively, each party first broadcasts its share of q and reconstruct the value q if all broadcast shares are consistent. However, if they are inconsistent, they divide it to small chunks and see if which parties sent the wrong values. Since the IC scheme does not satisfy the linearity, parties holding tags of two shared secrets cannot locally compute the right tag for the linear computation of two secret values. To be specific, authentication tags can be computed locally by adding two existing authentication tags because an authentication tag is defined as the y-intercept of a function. However, verification tags cannot be computed locally as the X coordinate value of each tag is randomly chosen so that the probability of having same X coordinate values for two verification tags is very low. That is, even though one party knows two verification tags (u, v) and (u', v') for function (of same degree) f and f', respectively, $(u + u', v + v')$ is not on $f + f'$ and he cannot locally compute $(u, v + f'(u))$ or $(u', f(u') + v')$ without knowing f or f'. For these reasons, the protocol $\texttt{LC-Reconstruct}$ uses "divide-and-conquer" method to find the parties who sent the wrong shares when the shares of q are inconsistent.

Now, let us see the assumptions and settings of this protocol. Assume that each party P_j shared l_j secrets, $s^{(j,1)}, s^{(j,2)}, ..., s^{(j,l_j)}$, and parties want to compute the total summation of multiple linear combinations of these l_j secrets for each P_j. i.e., parties in $\mathcal{P}$ want to reconstruct the value q, where $q := q^{(1)} + ... + q^{(n)}$ and $q^{(j)} := \sum_{i=1}^{l_j} a_i^{(j)} s^{(j,i)}$ for some $a_i^{(j)} \in \mathbb{F}$, $i = 1, ..., l_j$ for each $j = 1, ..., m$. Note that m can be up to n. For instance, if P_1 shares l_1 secret values, $s^{(1,1)}, ..., s^{(1,l_1)}$, P_2 shares l_2 secret values, $s^{(2,1)}, ..., s^{(2,l_2)}$, and P_3 shares

l_3 secret values, $s^{(3,1)}, ..., s^{(3,l_3)}$ (i.e., $j = 3$), then we are assuming that the parties in $\mathcal{P} = \{P_1, ..., P_n\}$ want to reconstruct $q = q^{(1)} + q^{(2)} + q^{(3)}$, where $q^{(1)} := \sum_{i=1}^{l_1} a_i^{(1)} s^{(1,i)}$, $q^{(2)} := \sum_{i=1}^{l_2} a_i^{(2)} s^{(2,i)}$, and $q^{(3)} := \sum_{i=1}^{l_3} a_i^{(3)} s^{(3,i)}$.

---

[29] Protocol $\texttt{LC-Reconstruct}^{\texttt{M}}(w, [q]) \longrightarrow q$ or $\perp$

Input: a phase w and locally computed sharing of q (collectively), $[q] = \{\mathbf{q}_1, ..., \mathbf{q}_n\}$, where $q = q^{(1)} + ... + q^{(n)}$ and each $q^{(j)}$ is a linear combination of l_j secrets shared by P_j, i.e., $q^{(j)} := \sum_{i=1}^{l_j} a_i^{(j)} s^{(j,i)}$ for some $a_i^{(j)} \in \mathbb{F}$ and $\{s^{(j,i)}\}_{i=1}^{l_j}$: secrets shared by P_j. Note that $[s^{(j,i)}] = \{\mathbf{s}_1^{(\mathbf{j},\mathbf{i})}, ..., \mathbf{s}_n^{(\mathbf{j},\mathbf{i})}\}$ is a sharing of i-th secret that P_j shared.
Output: q or aborted

1. Each $P_i \notin \mathcal{C}$ broadcasts its share $\mathbf{q}_i$ of $[q]$.
2. a) If the sharing broadcast in Step 1 is consistent (i.e., in Span$(M_\mathcal{G})$), then q is reconstructed by $\langle \lambda_\mathcal{G}, [q]_\mathcal{G} \rangle$ and the protocol terminates, where $[q]_\mathcal{G}$ is the recomposition vector with all the shares $\mathbf{q}_i$'s for $P_i \in \mathcal{G}$.
   b) Otherwise, each $P_i \notin \mathcal{C}$ broadcasts its share $\mathbf{q}_i^{(j)}$ of $[q^{(j)}]$, for each P_j.
3. a) If any party P_i broadcasted values such that $\mathbf{q}_i \neq \sum_{j=1}^{n} \mathbf{q}_i^{(j)}$, then all such parties are added to $\mathcal{C}$ and the protocol terminates.
   b) Otherwise, for the lowest j such that the shares of $[q^{(j)}]$ broadcasted in Step 2.b are inconsistent, do one of the followings depending on $P_j \notin \mathcal{C}$ or not.
4. If $P_j \notin \mathcal{C}$, do the followings:
   a) P_j broadcasts $(\texttt{accuse}, i)$ for P_i he thinks to have sent an incorrect share.
   b) Since $[q^{(j)}]$ is a linear combination of sharings generated by P_j, the parties internally know that $[q^{(j)}] = \sum_{k=1}^{l_j} a_k^{(j)} [s^{(j,k)}]$, where each $[s^{(j,k)}]$ was generated with $\texttt{Share}$ and each $a_k^{(j)}$ is non-zero. From l_j sharings $a_1^{(j)}[s^{(j,1)}], a_2^{(j)}[s^{(j,2)}], ..., a_{l_j}^{(j)}[s^{(j,l_j)}]$, P_i accused in Step 4.a broadcasts his shares of $\sum_{k=1}^{\lfloor l_j/2 \rfloor} a_k^{(j)}[s^{(j,k)}]$ and $\sum_{k=\lfloor l_j/2 \rfloor+1}^{l_j} a_k^{(j)}[s^{(j,k)}]$, i.e., $\sum_{k=1}^{\lfloor l_j/2 \rfloor} a_k^{(j)} \mathbf{s}_i^{(\mathbf{j},\mathbf{k})}$ and $\sum_{k=\lfloor l_j/2 \rfloor+1}^{l_j} a_k^{(j)} \mathbf{s}_i^{(\mathbf{j},\mathbf{k})}$.
   c) If P_i's two broadcasted shares in Step 4.b do not match up with the previously sent share of their sum (e.g. $\mathbf{q}_i^{(j)}$ for the first round), then P_i is added to $\mathcal{C}$ and the protocol terminates.
   d) P_j broadcasts which of shares broadcasted in Step 4.b he disagrees with. If this is a single sharing $a_k^{(j)}[s^{(j,k)}]$, then parties proceed Step 4.e. Otherwise, parties return to Step 4.b with the sharings P_j disagreed with. i.e., If P_j disagrees with some sum $a_{k_1}^{(j)}[s^{(j,k_1)}] + ... + a_{k_2}^{(j)}[s^{(j,k_2)}]$, then parties repeat Step 4.b to Step 4.d with $a_{k_1}^{(j)}[s^{(j,k_1)}], ..., a_{k_2}^{(j)}[s^{(j,k_2)}]$ instead of $a_1^{(j)}[s^{(j,1)}], ..., a_{l_j}^{(j)}[s^{(j,l_j)}]$.
   e) At this point, P_i broadcasted its share $a_k^{(j)} \mathbf{s}_i^{(\mathbf{j},\mathbf{k})}$ of $a_k^{(j)}[s^{(j,k)}]$ for some k and P_j broadcasted that he disagrees with this share. For each $P_V \notin \mathcal{D}_j \cup \mathcal{D}_i$, parties invoke $\texttt{Verify}^{\texttt{M}}(P_j, P_i, P_V, w, \mathbf{v}_i, tags)$, where $\mathbf{v}_i = (\mathbf{s}_i^{(j,k)}, \mathbf{u}_i^{(1)}, ..., \mathbf{u}_i^{(n)})$ as in Step 2 in $\texttt{Share}^{\texttt{M}}$ protocol.
   f) If the shares sent from P_i to P_V in $\texttt{Verify}^{\texttt{M}}$ do not match with the share of $a_k^{(j)}[s^{(j,k)}]$, then P_V broadcasts $(\texttt{accuse}, i)$, and $\{P_i, P_V\}$ is added to $\mathcal{D}$.
   g) $\{P_i, P_V\}$ is added to $\mathcal{D}$ for each $P_V \notin \mathcal{D}_i$ who rejected in the invocation of $\texttt{Verify}^{\texttt{M}}$ in Step 4.e, or $\{P_j, P_V\}$ is added to $\mathcal{D}$ for each P_V who accepted it.
   h) At this point, all parties are in dispute with either P_i or P_j and by the Q^2 property of Δ, one of $\mathcal{D}_i$ or $\mathcal{D}_j$ is no longer in Δ. If $\mathcal{D}_i \notin \Delta$, P_i is added to $\mathcal{C}$, and if $\mathcal{D}_j \notin \Delta$, P_j is added to $\mathcal{C}$. Then the protocol terminates.
5. If $P_j \in \mathcal{C}$, do the followings:
   a) Since $[q^{(j)}]$ is a linear combination of sharings generated by P_j, the parties internally know that $[q^{(j)}] = \sum_{k=1}^{l_j} a_k^{(j)} [s^{(j,k)}]$, where each $[s^{(j,k)}]$ was generated with $\texttt{Share}^{\texttt{M}}$ and each $a_k^{(j)}$ is non-zero. From l_j sharings $a_1^{(j)}[s^{(j,1)}], a_2^{(j)}[s^{(j,2)}], ..., a_{l_j}^{(j)}[s^{(j,l_j)}]$, each party $P_i \notin \mathcal{C}$ broadcasts its shares of $\sum_{k=1}^{\lfloor l_j/2 \rfloor} a_k^{(j)}[s^{(j,k)}]$ and $\sum_{k=\lfloor l_j/2 \rfloor+1}^{l_j} a_k^{(j)}[s^{(j,k)}]$, i.e., $\sum_{k=1}^{\lfloor l_j/2 \rfloor} a_k^{(j)} \mathbf{s}_i^{(\mathbf{j},\mathbf{k})}$ and $\sum_{k=\lfloor l_j/2 \rfloor+1}^{l_j} a_k^{(j)} \mathbf{s}_i^{(\mathbf{j},\mathbf{k})}$.
   b) Any party P_i whose sum of two broadcasted shares in Step 5.a does not match up with the previously sent share of their sum (e.g. $\mathbf{q}_i^{(j)}$ for the first round) is added to $\mathcal{C}$ and the protocol terminates.
   c) At this point, one of two shares broadcasted in Step 5.a is inconsistent. If this is a single sharing $a_k^{(j)}[s^{(j,k)}]$, then parties proceed Step 5.d. Otherwise, if this is some sum $a_{k_1}^{(j)}[s^{(j,k_1)}] + ... + a_{k_2}^{(j)}[s^{(j,k_2)}]$, , then parties return to Step 5.a with $a_{k_1}^{(j)}[s^{(j,k_1)}], ..., a_{k_2}^{(j)}[s^{(j,k_2)}]$ instead of $a_1^{(j)}[s^{(j,1)}], ..., a_{l_j}^{(j)}[s^{(j,l_j)}]$.
   d) Parties invoke $\texttt{Reconstruct}^{\texttt{M}}(w, [s^{(j,k)}])$ for the single sharing $a_k^{(j)}[s^{(j,k)}]$ decided in Step 5.c, but skip Step 1,

as they already broadcasted their shares. As a result of $\texttt{Reconstruct}^{\texttt{M}}$, a new party is added to $\mathcal{C}$ and the protocol terminates.

Note that if m parties only shared one secret $s^{(i,1)}$ for each party P_i, then each $\mathbf{q}_i^{(j)}$ of $[q^{(j)}]$ is already a single sharing, i.e., $q = q^{(1)} + ... + q^{(m)}$ where $q^{(j)} = a^{(j)}s^{(j,1)}$. Therefore, they can directly jump to Step 4.e or Step 5.d according to whether $P_j \notin \mathcal{C}$ or not.

D.3 Addition and Multiplication

By the linearity of the shares, addition can be done naturally without any communication or broadcast as below.

---

[29] Protocol $\texttt{Add}^{\texttt{M}}(w, [s], [t]) \longrightarrow [s + t]$

Input: phase w, shares of s, and shares of t
Output: new shares of $s + t$

Precondition: Two values s and t are shared with $\texttt{Share}^{\texttt{M}}$
Postcondition: $s + t$ is shared independently

1. Each party P_i locally adds each share of s to the share of t and keep the result as a share of $s + t$. i.e., $(\mathbf{s{+}t})_i := \mathbf{s}_i + \mathbf{t}_i \in \mathbb{F}^{d_i}$, for each $i = 1, ..., n$.

The protocol $\texttt{Generate-Randomness}^{\texttt{M}}$ generates l random elements, which are publicly known in $\mathcal{P}$. This is used in $\texttt{Generate-Multiplication-Triples}^{\texttt{M}}$ protocol for error detection.

---

[29] Protocol $\texttt{Generate-Randomness}^{\texttt{M}}(w, l) \longrightarrow r^{(1)}, ..., r^{(l)}$

Input: a phase w and the non-negative integer l
Output: publicly known l random elements in $\mathbb{F}$

1. Every party $P_i \notin \mathcal{C}$ chooses l random values $r^{(1,i)}, ..., r^{(l,i)}$.
2. Each P_i invokes $\texttt{ShareMultiple}^{\texttt{M}}(w, \mathbf{r}, P_i)$, where $\mathbf{r} := (r^{(1,i)}, ..., r^{(l,i)})$ to verifiably share these l random values.
3. parties in $\mathcal{P}$ call $\texttt{LC-Reconstruct}^{\texttt{M}}(w, [r^{(j)}])$ l times in parallel, to reconstruct l random values, $r^{(1)}, ..., r^{(l)}$, where $r^{(j)} := \sum_{P_i \notin \mathcal{C}} r^{(j,i)}$.

For multiplication gates, the protocol $\texttt{Generate-Multiplication-Triples}^{\texttt{M}}$ generates random sharings of l multiplication triples (a, b, c) such that $c = ab$, without revealing any values of a, b, or c to parties. These random triple can be used in each multiplication gate, by computing $[st] := [c]+[s](t-b)+[t](s-a)-(s-a)(t-b)$ as in [5]. To generate a sharing of a random triple $(a^{(k)}, b^{(k)}, c^{(k)})$, a random element $a^{(k)}$ is generated and each P_i creates a random triple $a^{(k)}b^{(i,k)} = c^{(i,k)}$. After verifying each triple's correctness using a triple $a^{(k)}\widetilde{b}^{(i,k)} = \widetilde{c}^{(i,k)}$ also created by each P_i, the final triple is defined as $(a^{(k)}, \sum_{i=1}^{n} b^{(i,k)}, \sum_{i=1}^{n} c^{(i,k)})$ for each $k = 1, ..., l$. For simplicity, we present the reduced version, which generates a sharing of only one multiplication triple, say (a, b, c).

---

[29] Protocol $\texttt{Generate-Multiplication-Triples}^{\texttt{M}}(w) \longrightarrow [(a, b, c)]$

Input: phase w
Output: a sharing of random triple (a, b, c) such that $c = ab$

1. Each $P_i \notin \mathcal{C}$ invokes $\texttt{Share}^{\texttt{M}}$ $(2n+3)$ times (in parallel), for each $a^{(i)}, b^{(i)}, \widetilde{b}^{(i)}$, $\{r^{(i,j)}\}_{j=1}^{n}$, and $\{\widetilde{r}^{(i,j)}\}_{j=1}^{n}$, and $\texttt{Share}^{\texttt{M}}$ $2n$ times for generating sharings of 1 (in parallel), denoted by $\{1^{(i,j)}\}_{j=1}^{n}$ and $\{\widetilde{1}^{(i,j)}\}_{j=1}^{n}$. The sharings of parties in $\mathcal{C}$ are defined to be all-zero sharings.
2. Each party defines and locally computes $[a] := \sum_{m=1}^{n}[a^{(m)}]$, $[r^{(i)}] := \sum_{m=1}^{n}[r^{(i,m)}]$, $[1^{(i)}] := \sum_{m=1}^{n}[1^{(m,i)}] + w[1^{(i,i)}]$, and $[\widetilde{1}^{(i)}] := \sum_{m=1}^{n}[\widetilde{1}^{(m,i)}] + \widetilde{w}[\widetilde{1}^{(i,i)}]$, where each w and $\widetilde{w} \in \mathbb{F}$ is the unique element that makes $[1^{(i)}]$ and $[\widetilde{1}^{(i)}]$ a sharing of 1.

3. Each $P_j \notin \mathcal{C}$ sends its share of $[a][b^{(i)}] + [r^{(i)}][1^{(i)}]$ and $[a][\widetilde{b}^{(i)}] + [\widetilde{r}^{(i)}][\widetilde{1}^{(i)}]$ to $P_i \notin \mathcal{C}$.
4. Each $P_i \notin \mathcal{C}$ reconstructs $D^{(i)} := ab^{(i)} + r^{(i)}$ and $\widetilde{D}^{(i)} := a\widetilde{b}^{(i)} + \widetilde{r}^{(i)}$ with the shares received in Step 3, and broadcasts $D^{(i)}$ and $\widetilde{D}^{(i)}$.
5. Each party locally computes $[c^{(i)}] := D^{(i)} - [r^{(i)}]$ and $[\widetilde{c}^{(i)}] := \widetilde{D}^{(i)} - [\widetilde{r}^{(i)}]$.
6. parties invoke $\texttt{Generate-Randomness}^{\texttt{M}}(w, 1)$ to generate a random element s.
7. Each party $\notin \mathcal{D}_i$ broadcasts its share of $[\widehat{b}^{(i)}] := [\widetilde{b}^{(i)}] + s[b^{(i)}]$, for $i = 1, ..., n$.
8. If the sharing of some $[\widehat{b}^{(i)}]$ broadcast in Step 7 is inconsistent, P_i broadcasts $(\texttt{accuse}, P_j)$ for such sharing sent by $P_j \notin \mathcal{D}_i$. $\{P_i, P_j\}$ is added to $\mathcal{D}$ and the protocol terminates.
9. parties invoke $\texttt{LC-Reconstruct}^{\texttt{M}}$ n times (in parallel) to reconstruct $z^{(i)} := [a]\widehat{b}^{(i)} - [\widetilde{c}^{(i)}] - s[c^{(i)}]$, for $i = 1, ..., n$.
10. If all reconstructed values in Step 9 are zero, then the protocol terminates successfully with the triple (a, b, c) with $[b] := \sum_{m=1}^{n} [b^{(m)}]$ and $[c] := \sum_{m=1}^{n} [c^{(m)}]$. Otherwise, if any $z^{(i)}$ is non-zero, then proceed into the Step 11 for the lowest index i such that $z^{(i)} \neq 0$.
11. a) Each P_j broadcasts its share of $[a^{(m)}], [\widetilde{r}^{(m,i)}]$, and $[r^{(m,i)}]$ for each $P_m \notin \mathcal{D}_j$.
    b) If P_i sees that the shares of some $P_j \notin \mathcal{D}_i$ sent in Step 11.a are inconsistent with the share sent in Step 3 or 9, then P_i broadcasts $(\texttt{accuse}, P_j)$ and $\{P_i, P_j\}$ is added to $\mathcal{D}$ and the protocol terminates.
    c) Each P_m examines the shares (broadcast in Step 11.a) of all sharings that P_m generated. If P_m notices that some $P_j \notin \mathcal{D}_m$ broadcast an incorrect share, then P_m broadcasts $(\texttt{accuse}, P_j)$ and $\{P_m, P_j\}$ is added to $\mathcal{D}$ and the protocol terminates.
    d) If no one broadcasts, then P_i is added to $\mathcal{C}$ and the protocol terminates.

Appendix E Our Protocols for the MSP-based PMPC and Their Proofs

E.1 The Protocol $\texttt{Refresh}^{\texttt{M}}$

Protocol $\texttt{Refresh}^{\texttt{M}}(w, [s]) \longrightarrow [s]^{w+1}$

Input: a phase w and a sharing of s
Output: new sharing of s in phase $w+1$, $[s]^{w+1}$

1. Every party P_i in $\mathcal{P}$ invokes $\texttt{Share}^{\texttt{M}}(w, 0, P_i)$. (in parallel)
2. Each party locally does component-wise addition with all the shares received in Step 1 and the shares of s, and set it as the new share of s in phase $w+1$.
3. parties in $\mathcal{P}$ collectively output $[s]^{w+1}$.

Theorem E.1. *(Correctness and Secrecy of* $\texttt{Refresh}^{\texttt{M}}$*) When the protocol* $\texttt{Refresh}^{\texttt{M}}$ *terminates, all parties receive new shares encoding the same secret as old shares they had before, and they cannot get any information about the secret by the execution of the protocol. It communicates* $O((n^2 d + n^3\kappa)\log|\mathbb{F}| + n^3\kappa\log d)$ *bits and broadcasts* $O(n^3\log d + (n^3 + nd)\log|\mathbb{F}|)$ *bits.*

Proof. **Correctness:** Recall that each party P_i has the vector $\mathbf{s}_i = M_i\mathbf{b} \in \mathbb{F}^{d_i}$ as the share of s, where $\mathbf{b} = (s, r_2, ..., r_e)$ for some random values r_j's. After Step 1, every party P_i receives n vectors $\{\mathbf{0}_i^{(j)}\}_{j=1}^{n}$ as shares of 0's, where $\mathbf{0}_i^{(j)} = M_i\mathbf{b}^{(j)}$ is the share of 0 from each party P_j, where $\mathbf{b}^{(j)} = (0, \$, ..., \$) \in \mathbb{F}^{d_i}$ with some random values (denoted by \$). Since these n vectors have the same lengths d_i, each party P_i can locally compute the vector addition $\mathbf{s}'_i := \mathbf{s}_i + \sum_{j=1}^{n}\mathbf{0}_i^{(j)} \in \mathbb{F}^{d_i}$. As all the summands of s and n zeros are shared by $\texttt{Share}^{\texttt{M}}$, for $s' = s + 0 + ... + 0$, the invocation of $\texttt{LC-Reconstruct}^{\texttt{M}}(w, [s'])$ with $[s'] := \{\mathbf{s}'_1, ..., \mathbf{s}'_n\}$ outputs s', which is equal to s. **Secrecy:** parties communicate only the shares of zeros but nothing about the secret s or the shares of it, the protocol does not reveal any information about s. **Communication:** As every party shares zero to each other, they communicates and broadcasts $n * Cost(\texttt{Share}^{\texttt{M}})$ bits. ◻

E.2 The Protocol $\texttt{Recover}^{\texttt{M}}$

Protocol $\texttt{Recover}^{\texttt{M}}(w, [s], R) \longrightarrow [s]^{w+1}$ or $\perp$

Input: a phase w, a sharing of s, and a set of rebooted parties R
Output: new sharing of s in phase $w+1$, $[s]^{w+1}$, or aborted

1. Invoke $\texttt{ShareRandom}^{\texttt{M}}(w)$ and generate a sharing $[r] := \{\mathbf{r}_1, ..., \mathbf{r}_n\}$ of a random r in $\mathbb{F}$.
2. Each party P_i in $\mathcal{P} \setminus R$ locally computes $\mathbf{r}_i + \mathbf{s}_i$, the share of $r' := r + s$.
3. $\texttt{LC-Reconstruct}^{\texttt{M}}(w, [r'])$ is invoked in $\mathcal{P} \setminus R$ and every party in $\mathcal{P} \setminus R$ gets r'.
4. $\texttt{RobustReshare}^{\texttt{M}}(r', w, \mathcal{P} \setminus R, w, \mathcal{P})$ is invoked, and each party in $\mathcal{P}$ gets $[r']^w := \{\mathbf{r}'_1, ..., \mathbf{r}'_n\}$.
5. Each party locally computes $\mathbf{r}'_i - \mathbf{r}_i$ and sets it as new share of s.

Theorem E.2. *(Correctness and Secrecy of* $\texttt{Recover}^{\texttt{M}}$*) The protocol* $\texttt{Recover}^{\texttt{M}}$ *allows a set R of parties who were rebooted to recover their shares encoding the same secret, for any $R \in \Delta$, and does not reveal any additional information about the secret except the shares each party had before the execution of the protocol. It communicates $O(n^3\kappa \log d + (n^4 + n^3\kappa + n^2 d) \log |\mathbb{F}|)$ bits and broadcasts $O(n^3 \log d + (n^3 + nd) \log |\mathbb{F}|)$ bits.*

Proof. **Correctness:** For each party, as the length of its share is same as the number of rows in M mapped to that party, the party can locally compute component-wise addition/subtraction with its shares. By the linearity of the shares, $\mathbf{r}'_i - \mathbf{r}_i$ is the i-th share of $r' - r = (r+s) - r = s$ for each P_i. Thus, all parties in $\mathcal{P}$ including parties in R receive a new sharing of s, the same secret. **Secrecy:** In Step 3, every party receives the reconstruction result of $r' = r + s$, but as r is random in $\mathbb{F}$ and not reconstructed in $\mathcal{P}$, each party has any information about r. Thus, the value r' does not reveal any information about s in Step 3. Step 1 and 4 are to share a sharing of the random elements and Step 2 and 5 are local computations. Therefore, the execution of this protocol does not reveal any information about the secret s. **Communication:** The protocol $\texttt{ShareRandom}^{\texttt{M}}$ is invoked by the group of parties in phase w, where $W = \{w\}$. i.e. $N = n$ and $|W| = 1$. In $\texttt{LC-Reconstruct}^{\texttt{M}}$, each parties only have $l = 1$ secret values to share. In $\texttt{RobustReshare}^{\texttt{M}}$, $\mathcal{P}_S = \mathcal{P} \setminus R$ and $\mathcal{P}_R = \mathcal{P}$ in the same phase w. Thus, the total number of communication bits is $O(n^3\kappa \log d + (n^4 + n^3\kappa + n^2 d) \log |\mathbb{F}|$ and the total number of broadcast bits is $O(n^3 \log d + (n^3 + nd) \log |\mathbb{F}|)$. □

E.3 The Protocol $\texttt{Redistribute}^{\texttt{M}}$

Protocol $\texttt{Redistribute}^{\texttt{M}}(w, [s]^w) \longrightarrow [s]^{w+1}$

Input: a phase w and the sharing of s in phase w, $[s]^w = \{\mathbf{s}_1^w, ..., \mathbf{s}_n^w\}$
Output: new sharing of s for phase $w+1$, $[s]^{w+1} = \{\mathbf{s}_1^{w+1}, ..., \mathbf{s}_m^{w+1}\}$

1. Parties in $\mathcal{P}$ and $\mathcal{P}'$ invoke $\texttt{ShareRandom}^{\texttt{M}}(W)$, where $W = \{w, w+1\}$, to generate two sharings of a random value r, unknown to every party. That is, parties in $\mathcal{P}$ separately receive a sharing $[r]^w := \{\mathbf{r}_1^w, ..., \mathbf{r}_n^w\}$, while parties in $\mathcal{P}'$ receive a sharing $[r]^{w+1} := \{\mathbf{r}_1^{w+1}, ..., \mathbf{r}_m^{w+1}\}$, and no one knows the value of r.
2. Each party P_i in $\mathcal{P}$ locally computes $\mathbf{x}_i := \mathbf{r}_i^w + \mathbf{s}_i^w$, where $\mathbf{s}_i^w$ is the share of s.
3. Parties in $\mathcal{P}$ invoke $\texttt{LC-Reconstruct}^{\texttt{M}}(w, [x])$ with $[x] := \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ and the result is denoted by x. Note that $x = s + r$, where r is random and unknown to everyone.
4. Parties invoke $\texttt{RobustReshare}^{\texttt{M}}(x, w, \mathcal{P}, w+1, \mathcal{P}')$ so that parties in $\mathcal{P}'$ receive a sharing of x, say $[x] := \{\mathbf{z}_1, ..., \mathbf{z}_m\}$, for $\mathbf{z}_i := M'_i \mathbf{X}$, where the vector $\mathbf{X} = (x, \$, ..., \$) \in \mathbb{F}^{e'}$ with random \$'s.
5. Each party P'_i in $\mathcal{P}'$ locally computes $\mathbf{s}_i^{w+1} := \mathbf{z}_i - \mathbf{r}_i^{w+1}$, for $i = 1, ..., m$.
6. Parties in $\mathcal{P}'$ collectively output $\{\mathbf{s}_1^{w+1}, ..., \mathbf{s}_m^{w+1}\}$ as a sharing of s in new phase.

Theorem E.3. *(Correctness and Secrecy of* $\texttt{Redistribute}^{\texttt{M}}$*) When the protocol terminates, all parties in new participating group have the shares of the same secret as the old shares, and the protocol does not reveal any information about the secret. It communicates $O(n^2\kappa \log d + nm^2\kappa \log d' + ((n^2+mn)d + (m^2+mn)d' + (n^3+m^3)\kappa + (m+n)mn\kappa + nm^3) \log |\mathbb{F}|)$ bits and broadcasts $O((n^3+mn^2) \log d + nm^2 \log d' + (n^3 + (n+m)(mn+d) + nd') \log |\mathbb{F}|)$ bits, where $|\mathcal{P}| = n$, $|\mathcal{P}'| = m$, size(M) = d, and size(M') = d'.*

Proof. **Correctness:** Since there exists λ such that $(M'_B)^t \lambda = \mathbf{a}$ for $\forall B \in \Gamma$, $\langle \lambda, \mathbf{s} \rangle = \langle \lambda, M'_B \mathbf{S} \rangle = \langle \lambda, M'_B(\mathbf{X} - \mathbf{R}) \rangle = \langle \lambda, M'_B \mathbf{X} \rangle - \langle \lambda, M'_B \mathbf{R} \rangle = \langle (M'_B)^t \lambda, \mathbf{X} \rangle - \langle (M'_B)^t \lambda, \mathbf{R} \rangle = x - r = (s+r) - r = s$, where $\mathbf{s}$ is the recomposition vector with shares of parties in $\forall B \in \Gamma$, $\mathbf{S}$ is the recomposition vector with shares of all parties in $\mathcal{P}'$, $\mathbf{X}$ is the vector defined in Step 4, and $\mathbf{R}$ is the vector of length e' having r of Step 1 for the first

component and $(e'-1)$ random values for the others. Thus, new sharing reconstructs the same secret s as the input sharing. **Secrecy:** Communicating values are either random value or the share of random value, and the shares of secret s are handled only by local computations. Also, since no one knows the value of r throughout the protocol, reconstruction of x does not reveal any information about the secret value s. **Communication:** Apply $N = n + m$ and $|W| = 2$ for $\texttt{ShareRandom}^{\texttt{M}}$, $l = 1$ for $\texttt{LC-Reconstruct}^{\texttt{M}}$, and $|\mathcal{P}_S| = |\mathcal{P}| = n$ and $|\mathcal{P}_R| = |\mathcal{P}'| = m$ for $\texttt{RobustReshare}^{\texttt{M}}$, as the others are the local computations. Assuming $m = n$ and $d' = d$, the total number of communication bits is $O(n^3\kappa \log d + (n^2 d + n^4 + n^3\kappa) \log |\mathbb{F}|)$ and the total broadcast bits are $O(n^3 \log d + (n^3 + nd) \log |\mathbb{F}|)$. □

Appendix F Protocols and Proofs for Conversions

F.1 Conversion from Additive PMPC to MSP-based PMPC

Protocol $\texttt{ConvertAdditiveIntoMSP}([s]^w, w, \mathcal{S}^w, w+1, \mathcal{S}^{w+1}) \longrightarrow [s]^{w+1}$

Input: the structure $\mathcal{S} := (\mathcal{P}, \Gamma, \Sigma, \Delta, \mathbb{S})$ in phase w and the sharing $\{s_1, ..., s_{|\mathbb{S}|}\}$ of s such that $\sum_{i=1}^{|\mathbb{S}|} s_i = s$

Output: the sharing of s for the structure $\mathcal{S}' := (\mathcal{P}', \Gamma', \Sigma', \Delta', \widehat{M})$ in phase w, where $M \in \mathcal{M}(d \times e)$ is corresponding matrix of the MSP $\widehat{M}$

1. For each $i \in \{1, ..., |\mathbb{S}|\}$ (in parallel):
2. Every party in S_i invokes $\texttt{Share}^{\texttt{M}}(s_i, w+1, \mathcal{P}')$. Denote $|S_i|$ sharings of s_i by $[s_i]^{(1)}, ..., [s_i]^{(|S_i|)}$.
3. Parties in $\mathcal{P}'$ invoke $\texttt{ShareRandom}^{\texttt{M}}(w+1)$ to generate a sharing of a random number, say $r^{(i)}$.
4. Parties in $\mathcal{P}'$ locally compute $[x_i^{(j)}] := [s_i]^{(j)} + [r^{(i)}]$, for $j = 1, ..., |S_i|$.
5. Parties in $\mathcal{P}'$ execute $\texttt{LC-Reconstruct}^{\texttt{M}}(w+1, [x_i^{(j)}])$ (in parallel) $|S_i|$ times for each sharing and choose a set $H \in Honest := \{\mathcal{P} \setminus A | A \in \overline{\Delta}\}$ that $x_i^{(j)} = v$ for all $P_{k_j} \in (S_i \cap H)$. If there exists multiple such sets, they choose the minimal set including P_{id} with lower id.
6. The sharing $[s_i^{(min)}]$ of s_i from $P_{min} \in H$ is chosen as a sharing of s_i, say $[s_i]$.
7. At this point, parties in $\mathcal{P}'$ hold $|\mathbb{S}|$ sharings for each s_i and each party $P_j \in \mathcal{P}'$ holds $|\mathbb{S}|$ vectors of length d_j, for each sharing. Each party locally computes component-wise addition with these vectors and set it as its share of s. i.e., P_j computes $\mathbf{s}_j := \sum_{i=1}^{|\mathbb{S}|} [s_i]_j \in \mathbb{F}^{d_j}$, where each share is the vector of length d_j.
8. Parties in $\mathcal{P}'$ collectively output a sharing of s, $[s]^{w+1} := \{\mathbf{s}_1, ..., \mathbf{s}_m\}$, where $m = |\mathcal{P}'|$.

Theorem F.1. *(Correctness and Secrecy of* $\texttt{ConvertAdditiveIntoMSP}$*) When the protocol terminates, all parties in new participating group have the shares of the same secret as the old shares, and the protocol does not reveal any information about the secret. It communicates* $O(k((m^2 + mn)d + (m^3 + m^2 n)\kappa + nm^3) \log |\mathbb{F}| + k(m^3 + m^2 n)\kappa \log d)$ *bits and broadcasts* $O(k(mnd + m^3 + m^2 n) \log |\mathbb{F}| + k(m^3 + m^2 n) \log d)$ *bits, where* $|\mathcal{P}| = n$*,* $|\mathcal{P}'| = m$*,* $|\mathbb{S}| = k$*, and size(*M*) =* d*.*

Proof. **Correctness:** Since the adversary chooses one set in Δ to corrupt, there exists at least one subset of parties in $Honest$ that includes only honest parties. Thus, parties in $\mathcal{P}'$ can figure out the right sharing of s_i from $|S_i|$ reconstruction values $\{(x_i^{(j)}\}_{j=1}^{|S_i|}$ for each i. In Step 7, each party $P_j \in \mathcal{P}'$ locally computes the summation of $|\mathbb{S}|$ vectors, $\mathbf{s}_j := \sum_{i=1}^{|\mathbb{S}|} [s_i]_j \in \mathbb{F}^{d_j}$, where $[s_i]_j = M_j \mathbf{b}^{(i)}$ for $\mathbf{b}^{(i)} := (s_i, \$, ..., \$)^t \in \mathbb{F}^e$. By definition, $\exists \lambda \in \mathbb{F}^{d_B}$ such that $M_B^t \lambda = \mathbf{a} \in \mathbb{F}^e$, for $\forall B \in \Gamma'$. When parties in $\forall B \in \Gamma'$ reconstruct with their shares, the recomposition vector with B shares is $\mathbf{S}_B = M_B \mathbf{b}$, where $\mathbf{b} = \sum_{i=1}^{|\mathbb{S}|} \mathbf{b}^{(i)}$. Thus, $\langle \lambda, \mathbf{S}_B \rangle = \langle \lambda, M_B \mathbf{b} \rangle = \langle M_B^t \lambda, \mathbf{b} \rangle$ =(the first component of $\mathbf{b}$) $= \sum_{i=1}^{|\mathbb{S}|} s_i = s$. **Secrecy:** Since each $S_i \in \mathbb{S}$ can include one or more parties and some of them might be corrupted, parties who receive the sharing of s_i need to choose the sharing of honest party in S_i. However, if parties in $\mathcal{P}'$ reconstruct s_i, then all parties can compute s by adding all s_i's at the end. Therefore, they generate a sharing of a random element $r^{(i)}$ for each s_i, and the random number will never be reconstructed. In Step 4 of the loop, although each party sees the value of $x_i^{(j)}$, it does not reveal anything about s_i because it is hided by the random number that no one knows. **Communications:** It costs $|\mathbb{S}| * \{n * Cost(\texttt{Share}^{\texttt{M}}) + Cost(\texttt{ShareRandom}^{\texttt{M}}) + n * Cost(\texttt{LC-Reconstruct}^{\texttt{M}})\}$, as $max_i |S_i| = |\mathcal{P}| = n$ when S_i include all parties in $\mathcal{P}$. □

F.2 Conversion from MSP-based PMPC to Additive PMPC

Protocol `ConvertMSPIntoAdditive`$([s]^w, w, \mathcal{S}^w, w+1, \mathcal{S}^{w+1}) \longrightarrow [s]^{w+1}$

Input: the structure $\mathcal{S} := (\mathcal{P}, \Gamma, \Sigma, \Delta, \widehat{M})$ in phase w and the sharing $\{\mathbf{s}_i\}_{i=1}^n$ of s such that $\mathbf{s}_i = M_i\mathbf{b}$ for each i, where $M \in \mathcal{M}(d \times e)$ of the MSP $\widehat{M}$ computes f and accepts Γ and $\mathbf{b} = (s, r_2, ..., r_e)$ for $r_i \xleftarrow{\$} \mathbb{F}$

Output: a sharing of s for $\mathcal{S}' := (\mathcal{P}', \Gamma', \Sigma', \Delta', \mathbb{S}')$ in phase w, where $\mathbb{S}' := \{S_1, S_2, ..., S_k\}$ is the sharing specification in phase $w+1$

1. Each party $P_i \in \mathcal{P}$ invokes $\texttt{Share}^{\texttt{A}}(w+1, s_j^{(i)}, \mathcal{P}')$, for each $s_j^{(i)}$ of $\mathbf{s}_i := (s_1^{(i)}, ..., s_{d_i}^{(i)})$.
2. Every party in $S_j \in \mathbb{S}'$ forms a vector of shares received in Step 1 from the same party P_i, as $\mathbf{s}_{ij} := ((s_1^{(i)})_j, (s_2^{(i)})_j, ..., (s_{d_i}^{(i)})_j)$. i.e., Parties in S_j hold n different vectors $\{\mathbf{s}_{1j}, ..., \mathbf{s}_{nj}\}$ from every party in $\mathcal{P}$ and each vector $\mathbf{s}_{ij}$ has length d_i.
3. Every parties in S_j forms the recomposition vector $\mathbf{Q}_j$ with n vectors $\{\mathbf{s}_{1j}, ..., \mathbf{s}_{nj}\}$ with respect to the indexing function ρ of $\widehat{M}$. Note that the length of $\mathbf{Q}_j$ is d for all $j = 1, ..., k$.
4. Each party in S_j sets $s_j := \langle \lambda, \mathbf{Q}_j \rangle$, where λ is the vector such that $M^t\lambda = \mathbf{a}$, for each $j = 1, ..., |\mathbb{S}|$.
5. Players in $\mathcal{P}'$ collectively output $[s]^{w+1} := \{s_1, ..., s_k\}$.

Theorem F.2. *(Correctness and Secrecy of* `ConvertMSPIntoAdditive`*) When the protocol terminates, all parties in new participating group have the shares of the same secret as the old shares, and the protocol does not reveal any information about the secret. It communicates $O(dkn^3 \log|\mathbb{F}|)$ bits and broadcasts $O(dkn^3 \log|\mathbb{F}|)$ bits, where $|\mathcal{P}| = n$, $|\mathcal{P}'| = m$, $|\mathbb{S}'| = k$, and size(M) = d.*

Proof. **Correctness:** After Step 1, each party in S_j of phase $w+1$ gets n vectors, $\{\mathbf{s}_{1j}, \mathbf{s}_{2j}, ..., \mathbf{s}_{nj}\}$, where $\mathbf{s}_{ij} := ((s_1^{(i)})_j, (s_2^{(i)})_j, ..., (s_{d_i}^{(i)})_j) \in \mathbb{F}^{d_i}$ is the vector of j-th additive shares of the share that $P_i \in \mathcal{P}$ has. Since each party in S_j received each vector $\mathbf{s}_{ij}$ from all n parties in $\mathcal{P}$, it can rearrange n vectors with respect to the indexing function ρ and form a vector of length d. Since parties in S_j have all j-th additive share of shares of phase w, $\mathbf{Q}_j$ is the vector of j-th shares of each component of $\mathbf{s} \in \mathbb{F}^d$, where $\mathbf{s} := M\mathbf{b}$, $\mathbf{b} := (s, r_2, ..., r_e) \in \mathbb{F}^e$ as in Section 5.2. i.e., $\sum_{j=1}^k \mathbf{Q}_j = \mathbf{s}$. By the properties of the inner product (over $\mathbb{R} \supset \mathbb{F} = GF(p)$) that $\langle u, v \rangle = \langle v, u \rangle$ and $\langle au + bv, w \rangle = a\langle u, w \rangle + b\langle v, w \rangle$, $\langle \lambda, \mathbf{Q}_1 + ... + \mathbf{Q}_k \rangle = \langle \lambda, \mathbf{Q}_1 \rangle + ... + \langle \lambda, \mathbf{Q}_k \rangle$. Since $\langle \lambda, \mathbf{s} \rangle = \langle \lambda, M\mathbf{b} \rangle = \langle M^t\lambda, \mathbf{b} \rangle = \langle \mathbf{a}, \mathbf{b} \rangle = s$, and $\langle \lambda, \mathbf{s} \rangle = \langle \lambda, \mathbf{Q}_1 + ... + \mathbf{Q}_k \rangle = \langle \lambda, \mathbf{Q}_1 \rangle + ... + \langle \lambda, \mathbf{Q}_k \rangle$, each $\langle \lambda, \mathbf{Q}_j \rangle$ can be set as the j-th additive share of s. **Secrecy:** It relies on the secrecy of the protocol $\texttt{Share}^{\texttt{A}}$. As what each party receives after the protocol is one additive share of s, it does not reveal any information about the secret until $\texttt{Reconstruct}^{\texttt{A}}$ is executed. **Communications:** As all other steps are local computations, complexities only rely on the $d * Cost(\texttt{Share}^{\texttt{A}})$. □