Sharing Transformation and Dishonest Majority MPC with Packed Secret Sharing

Vipul Goyal
Carnegie Mellon University
and NTT Research
vipul@cmu.edu

Antigoni Polychroniadou
J.P. Morgan AI Research
antigonipoly@gmail.com

Yifan Song
Carnegie Mellon University
yifans2@andrew.cmu.edu

February 4, 2026

Abstract

In the last few years, the efficiency of secure multi-party computation (MPC) in the dishonest majority setting has increased by several orders of magnitudes starting with the SPDZ protocol family which offers a speedy information-theoretic online phase in the prepossessing model. However, state-of-the-art n-party MPC protocols in the dishonest majority setting incur online communication complexity per multiplication gate which is linear in the number of parties, i.e. $O(n)$, per gate across all parties. In this work, we construct the first MPC protocols in the preprocessing model for dishonest majority with sub-linear communication complexity per gate in the number of parties n. To achieve our results, we extend the use of packed secret sharing to the dishonest majority setting. For a constant fraction of corrupted parties (i.e. if 99 percent of the parties are corrupt), we can achieve a communication complexity of $O(1)$ field elements per multiplication gate across all parties.

At the crux of our techniques lies a new technique called *sharing transformation*. The sharing transformation technique allows us to transform shares under one type of linear secret sharing scheme into another, and even perform arbitrary linear maps on the secrets of (packed) secret sharing schemes with optimal communication complexity. This technique can be of independent interest since transferring shares from one type of scheme into another (e.g., for degree reduction) is ubiquitous in MPC. Furthermore, we introduce what we call *sparsely packed Shamir sharing* which allows us to address the issue of network routing efficiently, and *packed Beaver triples* which is an extension of the widely used technique of Beaver triples for packed secret sharing (for dishonest majority).

1 Introduction

In this work we initiate the study of *sharing transformations* which allow us to perform *arbitrary* linear maps on the secrets of (possibly packed) secret-sharing schemes. More specifically, suppose Σ and Σ' are two linear secret sharing schemes over a finite field $\mathbb{F}$. A set of n parties $\{P_1, P_2, \ldots, P_n\}$ start with holding a Σ-sharing $\boldsymbol{X}$. Here $\boldsymbol{X}$ could be the sharing of a single field element or a vector of field elements (e.g., as in packed secret sharing where multiple secrets are stored within a single sharing). The parties wish to compute a Σ'-sharing $\boldsymbol{Y}$ whose secret is a *linear map* of the secret of $\boldsymbol{X}$. Here a linear map means that each output secret is a linear combination of the input secrets (recall that the secret can be a vector in $\mathbb{F}$). We refer to this problem as *sharing transformation*.

Restricted cases of sharing transformations occur frequently in the construction of secure computation protocols based on secret sharing. For example,

- In the well-known BGW protocol [BOGW88] and DN protocol [DN07] and their followups (see [CGH$^+$18, BGIN20, GLO$^+$21] and the citations therein), when evaluating a multiplication gate, all parties first locally compute a Shamir secret sharing of the result with a larger degree. To proceed the computation,

all parties wish to transform it to a Shamir secret sharing of the result with a smaller degree. Here the two linear secret sharing schemes Σ, Σ' are both the Shamir secret sharing schemes but with different degrees.

- A recent line of works [CCXY18, PS21, CRX21] use the notion of reverse multiplication-friendly embeddings (RMFE) to construct efficient information-theoretic MPC protocols over small fields or rings $\mathbb{Z}/p^\ell\mathbb{Z}$. This technique requires all parties to transform a secret sharing of a vector of secrets that are encoded by an encoding scheme to another secret sharing of the same secrets that are encoded by a different encoding scheme.

- A line of works [DIK10, GIP15, GSY21, BGJK21, GPS21] focus on the strong honest majority setting (i.e., $t = (1/2 - \epsilon) \cdot n$) and use the packed secret-sharing technique [FY92] to construct MPC protocols with sub-linear communication complexity in the number of parties. The main technical difficulty is to perform a linear map on the secrets of a single packed secret sharing (e.g., permutation or fan-out). In particular, depending on the circuit, each time the linear map we need to perform can be different.

Unlike the above results, our sharing transformation protocol (1) can perform arbitrary linear maps (2) is not restricted to a specific secret-sharing scheme and (3) can achieve optimal communication complexity[1]. Our transformation can find applications to different protocols based on different secret sharing schemes. In this work we focus on applications to information-theoretic (IT) MPC protocols. Furthermore, since we can handle any linear secret sharing scheme, our sharing transformation works for an arbitrary packing factor k as long as $t \leq n - 2k + 1$ where n is the number of participants and t is the number of corrupted parties by the adversary. This allows us to present the first IT MPC protocols with online communication complexity per gate sub-linear in the number of parties in the circuit-independent prepossessing model for a variety of corruption thresholds based on packed secret sharing. That said, we are able to extend the use of packed secret sharing beyond the strong honest majority setting.

For the case where $t = n - 1$, any function can be computed with IT security in the preprocessing model with online communication complexity of $O(n)$ field elements per gate across all parties [DPSZ12]. Existing protocols in the literature even for $t \in [(n-1)/2, n-1]$ still required communication complexity of $O(n)$ elements per gate. We note that most of these protocols follow the "gate-by-gate" design pattern described in [DNPR16]. In particular, the work [DNPR16] shows that any information-theoretic protocol that works in this design pattern must communicate $\Omega(n)$ for every multiplication gate. However, recent protocols in the strong honest majority setting, based on packed secret-sharing [FY92], where the number of corrupted parties $t = (1/2 - \epsilon) \cdot n$ and $\epsilon \in (0, 1/2)$ [GPS21] do achieve $O(1/\epsilon)$ communication complexity per gate among all parties. Note that the packed secret sharing technique evaluates a batch of multiplication gates in parallel, which differs from the above "gate-by-gate" design pattern in [DNPR16], and therefore does not contradict with the result in [DNPR16]. Our result closes the gap in achieving sub-linear communication complexity per gate in the number of parties for the more popular settings of standard honest majority and dishonest majority.

1.1 Our Contributions

Sharing Transformation. For our arbitrary linear-map transformation on (packed) linear secret sharing schemes we obtain the following informal result focusing on share size 1 (i.e., each share is a single field element).

Theorem 1 (Informal)**.** *Let $k = (n - t + 1)/2$. For all k tuples of $\{(\Sigma_i, \Sigma'_i, f_i)\}_{i=1}^k$ linear secret sharing schemes with injective sharing functions and for all Σ_i-sharings $\{\boldsymbol{X}_i\}_{i=1}^k$, there is an information-theoretic*

[1]To be more precise, our protocol achieves linear communication complexity in the summation of the sharing sizes of the two secret sharing schemes in the transformation. This is optimal (up to a constant factor) since it matches the communication complexity of using an ideal functionality to do sharing transformation: the size of the input is the sharing size of the first secret sharing scheme, the size of the output is the sharing size of the second secret sharing scheme, and the communication complexity is the size of the input and output.

MPC protocol with semi-honest security against t corrupted parties that transforms $\boldsymbol{X}_i$ to a Σ'_i-sharing $\boldsymbol{Y}_i$ such that the secret of $\boldsymbol{Y}_i$ is equal to the result of applying a linear map f_i on the secret of $\boldsymbol{X}_i$ for all $i \in \{1, \ldots, k\}$ (Here the secrets of $\boldsymbol{X}_i$ and $\boldsymbol{Y}_i$ can be vectors). The cost of the protocol is $O(n^3/k^2)$ elements of communication per sharing in a (sharing independent) preprocessing stage leading to preprocessed data of size $O(n^2/k)$, and $O(n^2/k)$ elements of communication per sharing in the online phase. When $t = (1-\epsilon) \cdot n$ for a positive constant ϵ, the overall communication complexity is $O(n)$ elements per sharing transformation.

The formal theorem is stated in Theorem 3. In Section 4, we show that our sharing transformation works for any share size ℓ (with an increase in the communication complexity by a factor ℓ), and is naturally extended to any finite fields and rings $\mathbb{Z}/p^\ell\mathbb{Z}$. The main application of our sharing transformation technique is to construct MPC protocols. And we achieve malicious security by directly compiling our semi-honest MPC protocol instead of relying on a maliciously secure sharing transformation protocol. Therefore, in this work, we do not attempt to achieve malicious security for our sharing transformation technique.

We now turn our attention to constructing general MPC using our sharing transformation technique.

Dishonest majority. In the setting of dishonest majority where the number of corrupted parties $t = (1-\epsilon) \cdot n$ for a positive constant ϵ, our MPC protocol achieves the cost of $O(1/\epsilon^2)$ elements of (the size of) preprocessing data, and $O(1/\epsilon)$ elements of communication per gate among all parties. Thus when ϵ is a constant (e.g., up to 99 percent of all parties may be corrupted), the achieved communication complexity in the online phase is $O(1)$ elements per gate.

Honest Majority. As a corollary of our results in the dishonest majority setting, we can achieve $O(1)$ elements per gate of online communication and $O(1)$ elements of preprocessing data per gate across all parties in the standard honest majority setting (i.e., where the number of corrupted parties t is $(n-1)/2$).

Our main results are summarized below. Note that we have omitted the additive terms of the overhead of the communication complexity in the informal theorems below. The formal theorems are stated in Theorem 4 and Theorem 5, respectively, where the additive terms are dependent on n and the depth of the evaluated circuit. Our first theorem is for the semi-honest setting:

Theorem 2 (Informal)**.** *For an arithmetic circuit C over a finite field $\mathbb{F}$ of size $|\mathbb{F}| \geq |C| + n$, there exists an information-theoretic MPC protocol in the preprocessing model which securely computes the arithmetic circuit C in the presence of a semi-honest adversary controlling up to t parties. The cost of the protocol is $O(|C| \cdot n^2/k^2)$ elements of preprocessing data, and $O(|C| \cdot n/k)$ elements of communication where $k = \frac{n-t+1}{2}$ is the packing parameter. For the case where $k = O(n)$, the achieved communication complexity in the online phase is $O(1)$ elements per gate.*

Our theorem also holds in the presence of a malicious adversary for all $1 \leq k \leq [\frac{n+2}{3}]$. Moreover, using our sharing transformation based on the construction of [GPS21], we can also achieve online communication complexity of $O(1)$ elements per gate for small finite fields of size $|\mathbb{F}| \geq 2n$. See discussion at the end of section 2.2 and Section 7 for more details.

1.2 Related Works

Sharing Transformation and Sharing Conversion. In this work, we study the problem of sharing transformation, where we want to transform the shares of one or multiple secrets under one secret sharing scheme into shares of another secret sharing scheme and apply a function on the secrets. In particular, we require the two secret sharing schemes as well as the function to be linear in the same finite commutative ring.

On the other hand, a line of works [DSZ15, HLOWI16, MR18, RW19, EGK+20, BCG+20, PSSY21, AGJ+21] study the problem of sharing conversion where they focus on the identity function (i.e., keep the secret unchanged) but two secret sharing schemes that are not in the same finite commutative ring, e.g., converting a secret sharing in the binary field to a secret sharing in a prime field. The problem of sharing

conversion appears to be much difficult than the problem of sharing transformation since we need to handle two different rings or fields. In particular, our technique does not work for the problem of sharing conversion.

Sharing Transformation via Pseudo-Random Secret Sharing [BBG$^+$21]. The work [BBG$^+$21] also studies the problem of sharing transformation relying on the technique of PRSS (Pseudo-Random Secret Sharing) initially studied in [CDI05]. At a high level, the idea is to first prepare replicated secret sharings and then transform them to correlated random packed Shamir sharings, which are used for sharing transformations.

The main advantage of this approach is that, after some initial setup, replicated secret sharings can be prepared without interaction relying on pseudo-random generators. Relying on this technique, the work [BBG$^+$21] can realize network routing, a key step of using the packed secret sharing technique in MPC, with no extra cost.

However, both the communication complexity for the setup and the computation complexity for generating each replicated secret sharing grows exponentially with the number of corrupted parties. It restricts the technique in [BBG$^+$21] to be only practical for a small constant number of parties.

Comparison with Techniques in [GPS21]. In [GPS21], the authors obtain an information-theoretic MPC protocol in the strong honest majority setting (i.e., the corruption threshold $t = (1/2 - \epsilon) \cdot n$), which achieves $O(|C|)$ total communication. Despite that [GPS21] focuses on the strong honest majority setting, we note that we can extend their techniques for network routing to the dishonest majority setting. Together with our techniques of packed Beaver triples and adjustment of the packing size, we can obtain a similar result to our main proposal.

On the other hand, our techniques for network routing have the following advantages compared with those in [GPS21].

1. First, we do not need to do circuit preprocessing. In [GPS21], the circuit preprocessing may increase the circuit size by a factor of 2 in the worst case.

2. Second, we provide a general sharing transformation protocol which is simpler and more efficient than that in [GPS21], which only works for restricted classes of sharing transformations. Also, the approach in [GPS21] would require $O(|C| \cdot n^2/k^2)$ communication complexity in the online phase, which is worse than ours $O(|C| \cdot n/k)$.

   Relying on the general sharing transformation protocol, our techniques for network routing are conceptually simple and more efficient for a large enough finite field: The techniques in [GPS21] require 6 sharing transformations per group of gates while our techniques only require 3 sharing transformations. We note that we can instantiate the sharing transformations in [GPS21] by our general sharing transformation protocol. Even with this optimization, the techniques in [GPS21] still require 4 sharing transformations per group of gates.

3. Third, our approach is not based on the Hall's Marriage Theorem. The techniques in [GPS21] require transforming the circuit to a bipartite graph and finding perfect matchings (whose existence is due to the Hall's Marriage Theorem). Finding these linear maps require $O(|C| \cdot \log |C|)$ computation. These steps are not needed in our approach.

The only drawback of our approach is the requirement of a large finite field ($|\mathbb{F}| \geq |C| + n$) while the techniques in [GPS21] works for a small finite field ($|\mathbb{F}| \geq 2n$). Thus, when the finite field is large enough, our main proposal is more efficient. We refer the readers to Section 7 for how our technique for sharing transformation can be used to simplify the protocol in [GPS21] and how to extend their protocol to the dishonest majority setting using our techniques.

Information-Theoretic MPC with Dishonest Majority. In [DPSZ12], Damgård et al. introduced the well-known protocol SPDZ in the all-but-one corruption setting, i.e., the number of corrupted parties is $t = n - 1$. The online phase of SPDZ is information-theoretic and therefore can be viewed as an information-theoretic protocol in the preprocessing model. The cost of the SPDZ protocol is $O(n)$ elements of preprocessing data and $O(n)$ elements of communication per multiplication gate among all parties. Also in the all-but-one corruption setting, a recent breakthrough [Cou19] shows that information-theoretic protocols in the preprocessing model are possible to achieve with sublinear communication complexity in the *circuit size* at a cost of a large amount of preprocessing data. Concretely, Couteau presents an MPC protocol for *layered circuits* in the preprocessing model, which achieves communication complexity of $O(n \cdot |C|/\log\log\log|C|)$ elements, at the cost of $O(|C|^2/\log\log\log|C|)$ elements of preprocessing data. We note that, however, the communication complexity of the protocol in [Cou19] is still linear in the number of parties and the protocol is impractical for a large circuit due to the amount of preprocessing data.

Compared with [DPSZ12, Cou19], we focus on a general corruption threshold where the number of corrupted parties $t = (1 - \epsilon) \cdot n$ for a positive constant ϵ. Our protocol achieves the cost of $O(1)$ elements of preprocessing data and $O(1)$ elements of communication per gate among all parties. One advantage of our protocol is that one may trade the corruption threshold with the real speed-up in the protocol, which is otherwise not possible in [DPSZ12, Cou19].

We note that a folklore solution in our setting is to choose a random small committee and then evaluate the circuit among parties in the small committee. Since there are $\epsilon \cdot n$ honest parties, if each time we add a random party into the committee, the probability of selecting an honest party is ϵ. Let κ be the security parameter. To ensure that with probability $1 - 2^{-\kappa}$, there is at least one honest party in the committee, the size of the committee should be at least $O(\kappa)$. If parties in the selected committee run the protocol in [DPSZ12], the achieved cost is $O(\kappa)$ elements of preprocessing data and $O(\kappa)$ elements of communication per gate among all parties, which is κ times of the cost of our construction. If parties run the protocol in [Cou19], the achieved cost is $O(|C|/\log\log\log|C|)$ elements of preprocessing data and $O(\kappa/(\epsilon \cdot \log\log\log|C|))$ elements of communication per gate among all parties. Note that, however, the circuit size is bounded by a polynomial of the security parameter κ, which means the term $\log\log\log|C|$ is much smaller than κ. Therefore, no matter which protocols are used in the folklore solution, our protocol is always $o(\kappa)$ times faster.

Information-Theoretic MPC with Strong Honest Majority. In the setting of strong honest majority setting where the number of corrupted parties $t = (1/2 - \epsilon) \cdot n$ for a positive constant ϵ, a rich line of works [DIK10, GIP15, IKP+16, GSY21, BGJK21, GPS21, BBG+21] makes use of the packed secret sharing technique to construct efficient multiparty computation protocols. In particular, the recent work [GPS21] gives the first information-theoretic MPC protocol in this setting which achieves $O(1)$ communication complexity per gate among all parties.

Information-Theoretic MPC with Standard Honest Majority. In the setting of the standard honest majority setting where the number of corrupted parties $t = (n - 1)/2$, a rich line of works [DN07, GIP+14, CGH+18, NV18, BBCG+19, GSZ20, BGIN20, GLO+21] focus on malicious security-with-abort and improving the communication efficiency. When assuming the existence of a broadcast channel, the works [BSFO12, GSZ20] have shown that guaranteed output delivery can also be achieved efficiently. The achieved communication complexity in this line of works is $O(n)$ per gate among all parties.

2 Technical Overview

In this section, we give an overview of our techniques. We use bold letters to represent vectors.

Reducing Sharing Transformation to Random Sharing Preparation. Usually, sharing transformation is solved by using a pair of random sharings $(\boldsymbol{R}, \boldsymbol{R}')$ such that $\boldsymbol{R}$ is a random Σ-sharing and $\boldsymbol{R}'$ is a

random Σ'-sharing which satisfies that the secret of $\boldsymbol{R}'$ is equal to the result of applying f on the secret of $\boldsymbol{R}$, where f is the desired linear map. Then all parties can run the following steps to efficiently transform $\boldsymbol{X}$ to $\boldsymbol{Y}$.

1. All parties locally compute $\boldsymbol{X} + \boldsymbol{R}$ and send their shares to the first party P_1.
2. P_1 reconstructs the secret of $\boldsymbol{X}+\boldsymbol{R}$, denoted by $\boldsymbol{w}$. Then P_1 computes $f(\boldsymbol{w})$ and generates a Σ'-sharing of $f(\boldsymbol{w})$, denoted by $\boldsymbol{W}$. Finally, P_1 distributes the shares of $\boldsymbol{W}$ to all parties.
3. All parties locally compute $\boldsymbol{Y} = \boldsymbol{W} - \boldsymbol{R}'$.

If we use $\mathtt{rec}, \mathtt{rec}'$ to denote the reconstruction maps of Σ and Σ' (which are linear by definition) respectively, the correctness follows from that

$$\mathtt{rec}'(\boldsymbol{Y}) = \mathtt{rec}'(\boldsymbol{W}) - \mathtt{rec}'(\boldsymbol{R}') = f(\boldsymbol{w}) - f(\mathtt{rec}(\boldsymbol{R})) = f(\mathtt{rec}(\boldsymbol{X} + \boldsymbol{R}) - \mathtt{rec}(\boldsymbol{R})) = f(\mathtt{rec}(\boldsymbol{X})).$$

And the security follows from the fact that $\boldsymbol{X} + \boldsymbol{R}$ is a random Σ-sharing and thus reveals no information about the secret of $\boldsymbol{X}$. Therefore, the problem of sharing transformation is reduced to preparing a pair of random sharings $(\boldsymbol{R}, \boldsymbol{R}')$. Let $\widetilde{\Sigma} = \widetilde{\Sigma}(\Sigma, \Sigma', f)$ be the secret sharing scheme which satisfies that a $\widetilde{\Sigma}$-sharing of a secret $\boldsymbol{x}$ consists of $\boldsymbol{X}$ which is a Σ-sharing of $\boldsymbol{x}$, and $\boldsymbol{Y}$ which is a Σ'-sharing of $f(\boldsymbol{x})$. Then, the goal becomes to prepare a random $\widetilde{\Sigma}$-sharing.

The generic approach of preparing random sharings of a linear secret sharing scheme over $\mathbb{F}$ is as follows:

1. Each party P_i first samples a random sharing $\boldsymbol{R}_i$ and distributes the shares to all other parties.
2. All parties use a linear randomness extractor over $\mathbb{F}$ to extract a batch of random sharings such that they remain uniformly random even given the random sharings sampled by corrupted parties. For a large finite field, we can use the transpose of a Vandermonde matrix [DN07] as a linear randomness extractor. The use of a randomness extractor is to reduce the communication complexity per random sharing. Alternatively, we can simply add all random sharings $\{\boldsymbol{R}_i\}_{i=1}^n$ and output a single random sharing, which results in quadratic communication complexity in the number of parties.

If t is the number of corrupted parties, all parties can extract $n - t$ random sharings when using a large finite field. Then, the amortized communication cost per sharing is $n^2/(n-t)$ field elements (assuming each share is a single field element). When $n - t = O(n)$, e.g., the honest majority setting, the amortized cost becomes $n^2/(n-t) = O(n)$, which is generally good enough since it matches the communication complexity of delivering a random sharing by a trusted party, which seems like the best we can hope, up to a constant factor.

Thus when we need to prepare many random sharings for the same linear secret sharing scheme, the generic approach is already good enough. And in particular, it is good enough for random $\widetilde{\Sigma}$-sharings which are used for the *same* sharing transformation defined by $\widetilde{\Sigma} = \widetilde{\Sigma}(\Sigma, \Sigma', f)$, since $\widetilde{\Sigma}$ is also a linear secret sharing scheme. This is exactly the case when we need to do degree reduction in [BOGW88, DN07] and change the encoding of the secrets in [CCXY18, PS21, CRX21]. However, it is a different story if we need to prepare random sharings for different linear secret sharing schemes: If only a constant number of random sharings are needed for each linear secret sharing scheme, the amortized cost per sharing becomes $O(n^2)$ field elements. This is exactly the case when we need to perform permutation on the secrets of a packed secret sharing in [DIK10, GIP15, BGJK21, GPS21]. In their setting, the permutations are determined by the circuit structure. In particular, these permutations can all be distinct in the worst case. As a result, the cost of preparing random sharings becomes the dominating term in the communication complexity in the MPC protocols. To avoid it, previous works either restrict the number of different secret sharing schemes they need to prepare random sharings for [DIK10, GIP15, GPS21] or restrict the types of circuits [BGJK21].

This leads to the following fundamental question: *Can we prepare random sharings (used for sharing transformations) for different linear secret sharing schemes with amortized communication complexity $O(n)$?*

2.1 Preparing Random Sharings for Different Linear Secret Sharing Schemes

To better expose our idea, we focus on a large finite field $\mathbb{F}$. In the following, we use n for the number of parties, and t for the number of corrupted parties. We assume semi-honest security in the technical overview.

Linear Secret Sharing Scheme over $\mathbb{F}$. For a linear secret sharing scheme Σ over $\mathbb{F}$, we use $Z = \mathbb{F}^{\tilde{k}}$ to denote the secret space. $\tilde{k}$ is also referred to as the secret size of Σ. For simplicity, we focus on the linear secret sharing schemes that have share size 1 (i.e., each share is a single field element even though the secret is a vector of $\tilde{k}$ elements). Let $\texttt{share} : Z \times \mathbb{F}^{\tilde{r}} \to \mathbb{F}^n$ be the deterministic sharing map which takes as input a secret $\boldsymbol{x}$ and $\tilde{r}$ random field elements, and outputs a Σ-sharing of $\boldsymbol{x}$. We focus on linear secret sharing schemes whose sharing maps are injective, which implies that $\tilde{k} + \tilde{r} \leq n$. Let $\texttt{rec} : \mathbb{F}^n \to Z$ be the reconstruction map which takes as input a Σ-sharing and outputs the secret of the input sharing. As discussed above, we have shown that preparing many random sharings for the same linear secret sharing scheme can be efficiently achieved.

We use the standard Shamir secret sharing scheme over $\mathbb{F}$, and use $[x]_t$ to denote a degree-t Shamir sharing of x. A degree-t Shamir sharing requires $t+1$ shares to reconstruct the secret. And any t shares of a degree-t Shamir sharing are independent of the secret.

2.1.1 Starting Point - Preparing a Random Sharing for a Single Linear Secret Sharing Scheme

Let Σ be an arbitrary linear secret sharing scheme. Although we have already shown how to prepare a random sharing for a single linear secret sharing scheme Σ, we consider the following process which is easy to be extended (discussed later).

1. All parties prepare $\tilde{k} + \tilde{r}$ random degree-t Shamir sharings. Let $\boldsymbol{\tau}$ be the secrets of the first $\tilde{k}$ sharings, and $\boldsymbol{\rho}$ be the secrets of the last $\tilde{r}$ sharings. Our goal is to compute a random Σ-sharing of $\boldsymbol{\tau}$ with random tape $\boldsymbol{\rho}$, i.e., $\texttt{share}(\boldsymbol{\tau}, \boldsymbol{\rho})$.
2. Since $\texttt{share}$ is $\mathbb{F}$-linear, for all $j \in \{1, 2, \ldots, n\}$, the j-th share of $\texttt{share}(\boldsymbol{\tau}, \boldsymbol{\rho})$ is a linear combination of the values in $\boldsymbol{\tau}$ and $\boldsymbol{\rho}$. Thus, all parties can locally compute a degree-t Shamir sharing of the j-th share of $\texttt{share}(\boldsymbol{\tau}, \boldsymbol{\rho})$ by using the degree-t Shamir sharings of the values in $\boldsymbol{\tau}$ and $\boldsymbol{\rho}$ prepared in Step 1 and applying linear combinations on their local shares. Let $[X_j]_t$ denote the resulting sharing.
3. For all $j \in \{1, 2, \ldots, n\}$, all parties send their shares of $[X_j]_t$ to P_j to let P_j reconstruct X_j. All parties take $\boldsymbol{X} = (X_1, \ldots, X_n)$ as output.

Note that $\boldsymbol{\tau}$ and $\boldsymbol{\rho}$ are all uniform field elements, and $\boldsymbol{X} = \texttt{share}(\boldsymbol{\tau}, \boldsymbol{\rho})$. Therefore, the output $\boldsymbol{X}$ is a random Σ-sharing.

We note that this approach requires to prepare $\tilde{k} + \tilde{r} = O(n)$ random degree-t Shamir sharings and communicate n^2 field elements in order to prepare a random Σ-sharing, which is far from $O(n)$. To improve the efficiency, we try to prepare random sharings for a batch of (potentially different) secret sharing schemes each time.

2.1.2 Preparing Random Sharings for a Batch of Different Linear Secret Sharing Schemes

We note that the above vanilla process can be viewed as all parties securely evaluating a circuit for the sharing map $\texttt{share}$ of Σ. In particular, (1) the circuit only involves linear operations, and (2) circuits for different secret sharing schemes (i.e., $\texttt{share}_1, \texttt{share}_2, \ldots, \texttt{share}_k$) all satisfy that each output value is a linear combination of all input values with different coefficients. When we want to prepare random sharings for a batch of different secret sharing schemes, the joint circuit is very similar to a SIMD circuit (which is a circuit that contains many copies of the same sub-circuit). The only difference is that, in our case, each sub-circuit corresponds to a different secret sharing scheme, and therefore the coefficients used in different sub-circuits are distinct. On the other hand, a SIMD circuit would use the same coefficients in all sub-circuits. Thus, it motivates us to explore the packed secret-sharing technique in [FY92], which is originally used to evaluate a SIMD circuit.

Starting Idea. Suppose $\Sigma_1, \Sigma_2, \ldots, \Sigma_k$ are k arbitrary linear secret sharing schemes (Recall that we want to prepare random sharings for different sharing transformations, and every different sharing transformation requires to prepare a random sharing of a different secret sharing scheme). We assume that they all have share size 1 (i.e., each share is a single field element) for simplicity. We consider to use a packed secret sharing scheme that can store k secrets in each sharing. Our attempt is as follows:

1. All parties first prepare n random packed secret sharings (Our construction will use the packed Shamir secret sharings introduced below). The secrets are denoted by $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_n$, where each secret $\boldsymbol{r}_j$ is a vector of k random elements in $\mathbb{F}$.

2. For all $i \in \{1, 2, \ldots, k\}$, we want to use the i-th values of all secret vectors to prepare a random sharing of Σ_i. With more details, suppose Σ_i has secret space $Z_i = \mathbb{F}^{\tilde{k}_i}$, and the sharing map of Σ_i is $\mathtt{share}_i : Z_i \times \mathbb{F}^{\tilde{r}_i} \to \mathbb{F}^n$. Consider the vector $(r_{1,i}, r_{2,i}, \ldots, r_{n,i})$ which contains the i-th values of all secret vectors. We plan to use the first $\tilde{k}_i$ values as the secret $\boldsymbol{\tau}_i$, and the next $\tilde{r}_i$ values as the random tape $\boldsymbol{\rho}_i$. Recall that we require $\mathtt{share}_i$ to be injective. We have $\tilde{k}_i + \tilde{r}_i \leq n$. Therefore, there are enough values for $\boldsymbol{\tau}_i$ and $\boldsymbol{\rho}_i$. The goal is to compute a random Σ_i-sharing $\boldsymbol{X}_i$ of the secret $\boldsymbol{\tau}_i$ with random tape $\boldsymbol{\rho}_i$, i.e., $\boldsymbol{X}_i = \mathtt{share}_i(\boldsymbol{\tau}_i, \boldsymbol{\rho}_i)$.

3. For each party P_j, let $\boldsymbol{u}_j$ denote the j-th shares of $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k$. We want to use the packed secret sharings of $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n$ to compute a single packed secret sharing of $\boldsymbol{u}_j$.

4. After obtaining a packed secret sharing of $\boldsymbol{u}_j$, we can reconstruct the sharing to P_j so that he learns the j-th share of each of $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k$. Thus, we start with n packed secret sharings (of $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n$) of the same secret sharing scheme and end with k sharings $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k$ of k potentially different secret sharing schemes.

Clearly, the main question is how to realize Step 3. We observe that, since Σ_i is a linear secret sharing scheme, the j-th share of $\boldsymbol{X}_i$ can be written as a linear combination of the values in $\boldsymbol{\tau}_i$ and $\boldsymbol{\rho}_i$. Therefore, the j-th share of $\boldsymbol{X}_i$ is a linear combination of the values $(r_{1,i}, r_{2,i}, \ldots, r_{n,i})$. Since it holds for all $i \in \{1, 2, \ldots, k\}$, there exists constant vectors $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n \in \mathbb{F}^k$ such that

$$\boldsymbol{u}_j := \boldsymbol{c}_1 * \boldsymbol{r}_1 + \ldots + \boldsymbol{c}_n * \boldsymbol{r}_n,$$

where $*$ denotes the coordinate-wise multiplication operation. Thus, *what we need is a packed secret sharing scheme that supports efficient coordinate-wise multiplication with a constant vector.* We note that the packed Shamir secret sharing scheme fits our need as we show next.

Packed Shamir Secret Sharing Scheme and Multiplication-Friendliness. The packed Shamir secret sharing scheme [FY92] is a natural generalization of the standard Shamir secret sharing scheme [Sha79]. It allows to secret-share a batch of secrets within a single Shamir sharing. For a vector $\boldsymbol{x} \in \mathbb{F}^k$, we use $[\boldsymbol{x}]_d$ to denote a degree-d packed Shamir sharing, where $k-1 \leq d \leq n-1$. It requires $d+1$ shares to reconstruct the whole sharing, and any $d-k+1$ shares are independent of the secrets. The packed Shamir secret sharing scheme has the following nice properties:

- Linear Homomorphism: For all $d \geq k-1$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x} + \boldsymbol{y}]_d = [\boldsymbol{x}]_d + [\boldsymbol{y}]_d$.
- Multiplicative: For all $d_1, d_2 \geq k-1$ subject to $d_1 + d_2 < n$, and for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x} * \boldsymbol{y}]_{d_1+d_2} = [\boldsymbol{x}]_{d_1} \cdot [\boldsymbol{y}]_{d_2}$, where the multiplications are performed on the corresponding shares.

Note that when $d \leq n-k$, all parties can locally multiply a public vector $\boldsymbol{c} \in \mathbb{F}^k$ with a degree-d packed Shamir sharing $[\boldsymbol{x}]_d$:

1. All parties first locally compute a degree-$(k-1)$ packed Shamir sharing of $\boldsymbol{c}$, denoted by $[\boldsymbol{c}]_{k-1}$. Note that for a degree-$(k-1)$ packed Shamir sharing, all shares are determined by the secret $\boldsymbol{c}$.

2. All parties then locally compute $[\boldsymbol{c} * \boldsymbol{x}]_{n-1} = [\boldsymbol{c}]_{k-1} \cdot [\boldsymbol{x}]_{n-k}$.

We simply write $[\boldsymbol{c}*\boldsymbol{x}]_{n-1} = \boldsymbol{c}\cdot[\boldsymbol{x}]_{n-k}$ to denote the above process. We refer to this property as multiplication-friendliness.

To make sure that the packed Shamir secret sharing scheme is secure against t corrupted parties, we also require $d \geq t+k-1$. When $d = n-k$ and $k = (n-t+1)/2$, the degree-$(n-k)$ packed Shamir secret sharing scheme is both multiplication-friendly and secure against t corrupted parties.

Observe that when we use the degree-$(n-k)$ packed Shamir secret sharing scheme in our attempt, all parties can locally compute a degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{u}_j$ by

$$[\boldsymbol{u}_j]_{n-1} = \boldsymbol{c}_1 \cdot [\boldsymbol{r}_1]_{n-k} + \ldots + \boldsymbol{c}_n \cdot [\boldsymbol{r}_n]_{n-k},$$

which solves the problem.

Summary of Our Construction. In summary, all parties run the following steps to prepare random sharings for k different linear secret sharing schemes $\Sigma_1, \Sigma_2, \ldots, \Sigma_k$.

1. Prepare Packed Shamir Sharings: All parties prepare n random degree-$(n-k)$ packed Shamir sharings, denoted by $[\boldsymbol{r}_1]_{n-k}, \ldots, [\boldsymbol{r}_n]_{n-k}$.
2. Use Packed Secrets as Randomness for Target LSSS: For all $i \in \{1, 2, \ldots, k\}$, let $\boldsymbol{\tau}_i = (r_{1,i}, \ldots, r_{\tilde{k}_i,i})$ and $\boldsymbol{\rho}_i = (r_{\tilde{k}_i+1,i}, \ldots, r_{\tilde{k}_i+\tilde{r}_i,i})$. Let $\boldsymbol{X}_i = \mathtt{share}_i(\boldsymbol{\tau}_i, \boldsymbol{\rho}_i)$.
3. Compute a Single Packed Shamir Sharing for All j-th Shares of Target LSSS via Local Operations: For all $j \in \{1, 2, \ldots, n\}$, let $\boldsymbol{u}_j$ be the j-th shares of $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k)$. All parties locally compute a degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{u}_j$ by using $[\boldsymbol{r}_1]_{n-k}, \ldots, [\boldsymbol{r}_n]_{n-k}$. The resulting sharing is denoted by $[\boldsymbol{u}_j]_{n-1}$.
4. Reconstruct the Single Packed Shamir Sharing of All j-th Shares to P_j: For all $j \in \{1, 2, \ldots, n\}$, all parties reconstruct the sharing $[\boldsymbol{u}_j]_{n-1}$ to P_j to let him learn $\boldsymbol{u}_j = (u_j^{(1)}, \ldots, u_j^{(k)})$. Then all parties take $\{\boldsymbol{X}_i = (u_1^{(i)}, \ldots, u_n^{(i)})\}_{i=1}^k$ as output.

We note that in Step 4, $[\boldsymbol{u}_j]_{n-1}$ is not a random degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{u}_j$. *Directly sending the shares of* $[\boldsymbol{u}_j]_{n-1}$ *to* P_j *may leak the information about honest parties' shares.* To solve it, all parties also prepare n random degree-$(n-1)$ packed Shamir sharings of $\mathbf{0} \in \mathbb{F}^k$, denoted by $[\boldsymbol{o}_1]_{n-1}, \ldots, [\boldsymbol{o}_n]_{n-1}$. Then all parties use $[\boldsymbol{o}_j]_{n-1}$ to refresh the shares of $[\boldsymbol{u}_j]_{n-1}$ by computing $[\boldsymbol{u}_j]_{n-1} := [\boldsymbol{u}_j]_{n-1} + [\boldsymbol{o}_j]_{n-1}$. Now $[\boldsymbol{u}_j]_{n-1}$ is a random degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{u}_j$. All parties send their shares of $[\boldsymbol{u}_j]_{n-1}$ to P_j to let him reconstruct $\boldsymbol{u}_j$.

Communication Complexity. Thus, to prepare random sharings for k linear secret sharing schemes, our construction requires to prepare n random degree-$(n-k)$ packed Shamir sharings and n random degree-$(n-1)$ packed Shamir sharings of $\mathbf{0} \in \mathbb{F}^k$. And the communication complexity is n^2 field elements. On average, each random sharing costs $2n/k$ packed Shamir sharings and n^2/k elements of communication. When we use the generic approach to prepare random packed Shamir sharings, the total communication complexity per random sharing is $O(n^2/k)$ elements.

Recall that $k = (n-t+1)/2$. When $t = (1-\epsilon)\cdot n$ for a positive constant ϵ, the communication complexity per random sharing is $O(n)$ elements, which matches the communication complexity of delivering a random sharing by a trusted party up to a constant factor. In Section 4, we show that our technique works for any share size ℓ (with an increase in the communication complexity by a factor ℓ), and is naturally extended to any finite fields and rings $\mathbb{Z}/p^\ell\mathbb{Z}$.

Efficient Sharing Transformation. Recall that in the problem of sharing transformation, all parties start with holding a sharing $\boldsymbol{X}$ of a linear secret sharing scheme Σ. They want to compute a sharing $\boldsymbol{Y}$ of another linear secret sharing scheme Σ' such that the secret of $\boldsymbol{Y}$ is a linear map of the secret of $\boldsymbol{X}$.

As we discussed above, sharing transformation can be achieved efficiently with the help of a pair of random sharings $(\boldsymbol{R}, \boldsymbol{R}')$ such that $\boldsymbol{R}$ is a random Σ-sharing and $\boldsymbol{R}'$ is a random Σ'-sharing which satisfies that the secret of $\boldsymbol{R}'$ is equal to the result of applying the desired linear map on the secret of $\boldsymbol{R}$. *A key insight is that* $(\boldsymbol{R}, \boldsymbol{R}')$ *can just be seen as a linear secret sharing on its own.* With our technique of preparing random sharings for different linear secret sharing schemes, we can efficiently prepare a pair of random sharings $(\boldsymbol{R}, \boldsymbol{R}')$, allowing efficient sharing transformation from $\boldsymbol{X}$ to $\boldsymbol{Y}$.

When $t = (1-\epsilon)\cdot n$ for a positive constant ϵ, each sharing transformation only requires $O(n)$ field elements of communication.

2.2 Application: MPC via Packed Shamir Secret Sharing Schemes

In this section, we show that our technique for sharing transformation allows us to design an efficient MPC protocol via packed Shamir secret sharing schemes. We focus on the *dishonest majority setting* and information-theoretic setting in the circuit-independent preprocessing model. In the preprocessing model, all parties receive correlated randomness from a trusted party before the computation. The preprocessing model enables the possibility of an information-theoretic protocol in the dishonest majority setting, which otherwise cannot exist in the plain model. The cost of a protocol in the preprocessing model is measured by both the amount of preprocessing data prepared in the preprocessing phase and the amount of communication in the online phase [Cou19, BGIN21].

Let n be the number of parties, and t be the number of corrupted parties. For any positive constant ϵ, we show that there is an information-theoretic MPC protocol in the circuit-independent preprocessing model with semi-honest security (or malicious security) that computes an arithmetic circuit C over a large finite field $\mathbb{F}$ (with $|\mathbb{F}| \geq |C| + n$) against $t = (1-\epsilon)\cdot n$ corrupted parties with $O(|C|)$ field elements of preprocessing data and $O(|C|)$ field elements of communication. Compared with the recent work [GPS21] that achieves $O(|C|)$ communication complexity in the strong honest majority setting (i.e., $t = (1/2-\epsilon)\cdot n$), our construction has the following advantages:

1. Our protocol works in the dishonest majority setting.
2. With our new technique for sharing transformation, we avoid the heavy machinery in [GPS21] for the network routing (see more discussion in Section 1.2).

On the other hand, we note that the protocol in [GPS21] works for a finite field of size $2n$ while our protocol requires the field size to be $|C| + n$. We will discuss how our technique for sharing transformation can be used to simplify the protocol in [GPS21] and how to extend their protocol to the dishonest majority setting using our techniques in Section 7. We refer the readers to Section 1.2 for a more detailed comparison with [GPS21] and other related works.

Review the Packed Shamir Secret Sharing Scheme. We recall the notion of the packed Shamir secret sharing scheme. Let $\alpha_1, \ldots, \alpha_n$ be n distinct elements in $\mathbb{F}$ and $\mathtt{pos} = (p_1, p_2, \ldots, p_k)$ be another k distinct elements in $\mathbb{F}$. A *degree-d* ($d \geq k-1$) packed Shamir sharing of $\boldsymbol{x} = (x_1, \ldots, x_k) \in \mathbb{F}^k$ is a vector $(w_1, \ldots, w_n)$ for which there exists a polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most d such that $f(p_i) = x_i$ for all $i \in \{1, 2, \ldots, k\}$, and $f(\alpha_i) = w_i$ for all $i \in \{1, 2, \ldots, n\}$. The i-th share w_i is held by party P_i.

In our protocol, we will always use the same elements $\alpha_1, \ldots, \alpha_n$ for the positions of the shares of all parties. However, we may use different elements $\mathtt{pos}$ for the secrets. We will use $[\boldsymbol{x} \| \mathtt{pos}]_d$ to denote a degree-d packed Shamir sharing of $\boldsymbol{x} \in \mathbb{F}^k$ stored at positions $\mathtt{pos}$. Let $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)$ be distinct field elements in $\mathbb{F}$ that are different from $\alpha_1, \ldots, \alpha_n$. We will use $\boldsymbol{\beta}$ as the default positions for the secrets, and simply write $[\boldsymbol{x}]_d = [\boldsymbol{x} \| \boldsymbol{\beta}]_d$.

Recall that t is the number of corrupted parties. Let $k = (n-t+1)/2$ and $d = n-k$. As we have shown in Section 2.1, all parties can locally multiply a public vector with a degree-$(n-k)$ packed Shamir sharing, and a degree-$(n-k)$ packed Shamir sharing is secure against t corrupted parties.

An Overview of Our Construction. At a high-level,

1. All parties start with sharing their input values by using packed Shamir sharings.
2. In each layer, addition gates and multiplication gates are divided into groups of size k. Each time we will evaluate a group of k gates:
   (a) For each group of k gates, all parties prepare two packed Shamir sharings, one for the first inputs of all gates, and the other one for the second inputs of all gates. Note that the secrets we want to be in a single sharing can be scattered in different output sharings from previous layers. This step is referred to as *network routing*. Relying on our technique of sharing transformation, we can use a much simpler approach to handle network routing than that in [GPS21].
   (b) After preparing the two input sharings, all parties evaluate these k gates. Addition gates can be locally computed since the packed Shamir secret sharing scheme is linearly homomorphic. For multiplication gates, we extend the technique of Beaver triples [Bea91] to our setting, which we refer to as *packed Beaver triples*. All parties need to prepare packed Beaver triples in the preprocessing phase.
3. After evaluating the whole circuit, all parties reconstruct the sharings they hold to the parties who should receive the result.

Sparsely Packed Shamir Sharings. Our idea is to use a different position to store the output value of each gate. Recall that $|\mathbb{F}| \geq |C| + n$. Let $\beta_1, \beta_2, \ldots, \beta_{|C|}$ be $|C|$ distinct field elements that are different from $\alpha_1, \alpha_2, \ldots, \alpha_n$. (Recall that we have already defined $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)$, which are used as the default positions for a packed Shamir sharing.) We associate the field element β_i with the i-th gate in C. We will use β_i as the position to store the output value of the i-th gate in a degree-$(n-k)$ packed Shamir sharing (see an example below).

Concretely, for each group of k gates, all parties will compute a degree-$(n-k)$ packed Shamir sharing such that the results are stored at the positions associated with these k gates respectively. For example, when $k = 3$, for a batch of 3 gates which are associated with the positions $\beta_1, \beta_3, \beta_6$ respectively, all parties will compute a degree-$(n-k)$ packed Shamir sharing $[(z_1, z_3, z_6) \| (\beta_1, \beta_3, \beta_6)]_{n-k}$ for this batch of gates, where z_1, z_3, z_6 are the output wires of these 3 gates.

As we will see later, it greatly simplifies the protocol for network routing.

2.2.1 Network Routing

In each intermediate layer, for every group of k gates, suppose $\boldsymbol{x}$ are the first inputs of these k gates, and $\boldsymbol{y}$ are the second inputs of these k gates. All parties will prepare two degree-$(n-k)$ packed Shamir sharings $[\boldsymbol{x}]_{n-k}$ and $[\boldsymbol{y}]_{n-k}$ stored at the default positions using the following approach. The reason of choosing the default positions is to use the packed Beaver triples, which use the default positions since the preprocessing phase is circuit-independent (discussed later). We focus on how to obtain $[\boldsymbol{x}]_{n-k}$.

Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$. For simplicity, we assume that $x_1, x_2, \ldots, x_k$ are output wires from k distinct gates. Later on, we will show how to handle the scenario where the same output wire is used multiple times by using fan-out operations. Since we use a different position to store the output of each gate, the positions of these k gates are all different. Let $p_1, \ldots, p_k$ denote the positions of these k gates and $\mathtt{pos} = (p_1, \ldots, p_k)$. We first show that all parties can locally compute a degree-$(n-1)$ packed Shamir sharing $[\boldsymbol{x} \| \mathtt{pos}]_{n-1}$.

Selecting the Correct Secrets. For all $i \in \{1, 2, \ldots, k\}$, let $[\boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]_{n-k}$ be the degree-$(n-k)$ packed Shamir sharing that contains the secret x_i at position p_i from some previous layer. Let $\boldsymbol{e}_i$ be the i-th unit vector in $\mathbb{F}^k$ (i.e., only the i-th term is 1 and all other terms are 0). All parties locally compute a degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{e}_i \| \mathtt{pos}]_{k-1}$. Consider the following degree-$(n-1)$ packed Shamir sharing:

$$[\boldsymbol{e}_i \| \mathtt{pos}]_{k-1} \cdot [\boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]_{n-k}.$$

We claim that, the resulting sharing satisfies that the value stored at position p_i is x_i and the values stored at other positions in $\mathtt{pos}$ are all 0. To see this, recall that each packed Shamir sharing corresponds to a polynomial. Let f be the polynomial corresponding to $[\boldsymbol{e}_i \| \mathtt{pos}]_{k-1}$, and g be the polynomial corresponding to $[\boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]_{n-k}$. Then f satisfies that $f(p_i) = 1$ and $f(p_j) = 0$ for all $j \neq i$, and g satisfies that $g(p_i) = x_i$. Note that $h = f \cdot g$ is the polynomial corresponding to the resulting sharing $[\boldsymbol{e}_i \| \mathtt{pos}]_{k-1} \cdot [\boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]_{n-k}$, which satisfies that $h(p_i) = f(p_i) \cdot g(p_i) = 1 \cdot x_i = x_i$, and $h(p_j) = f(p_j) \cdot g(p_j) = 0 \cdot g(p_j) = 0$ for all $j \neq i$. Thus, the resulting sharing has value x_i in the position p_i and 0 in all other positions in $\mathtt{pos}$. Effectively, we select the secret x_i from $[\boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]_{n-k}$ at position p_i and zero-out the values stored at other positions in $\mathtt{pos}$.

Getting all Secrets into a Single Packed Shamir Sharing. Thus, for the following degree-$(n-1)$ packed Shamir sharing

$$\sum_{i=1}^{k} [\boldsymbol{e}_i \| \mathtt{pos}]_{k-1} \cdot [\boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]_{n-k},$$

it has value x_i stored in the position p_i for all $i \in \{1, 2, \ldots, k\}$, which means that it is a degree-$(n-1)$ packed Shamir sharing $[\boldsymbol{x} \| \mathtt{pos}]_{n-1}$. Therefore, all parties can locally compute $[\boldsymbol{x} \| \mathtt{pos}]_{n-1} = \sum_{i=1}^{k} [\boldsymbol{e}_i \| \mathtt{pos}]_{k-1} \cdot [\boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]_{n-k}$.

Applying Sharing Transformation. Finally, to obtain $[\boldsymbol{x}]_{n-k} = [\boldsymbol{x} \| \boldsymbol{\beta}]_{n-k}$, all parties only need to do a sharing transformation from $[\boldsymbol{x} \| \mathtt{pos}]_{n-1}$ to $[\boldsymbol{x}]_{n-k}$. Relying on our technique for sharing transformation, we can achieve this step with $O(n)$ field elements of communication.

Therefore, our protocol for network routing only requires a local computation for $[\boldsymbol{x} \| \mathtt{pos}]_{n-1}$ and an efficient sharing transformation for $[\boldsymbol{x}]_{n-k}$ with $O(n)$ field elements of communication.

Handling Fan-out Operations. The above solution only works when all the wire values of $\boldsymbol{x}$ come from different gates. In a general case, $\boldsymbol{x}$ may contain many wire values from the same gate. We modify the above protocol as follows:

1. Suppose $x'_1, \ldots, x'_{k'}$ are the different values in $\boldsymbol{x}$. Let $\boldsymbol{x}' = (x'_1, \ldots, x'_{k'}, 0, \ldots, 0) \in \mathbb{F}^k$. For all $i \in \{1, 2, \ldots, k'\}$, let p_i be the position associated with the gate that outputs x'_i. We choose $p_{k'+1}, \ldots, p_k$ to be the first $(k - k')$ unused positions and set $\mathtt{pos} = (p_1, \ldots, p_k)$. Then, all parties follow a similar approach to locally compute a degree-$(n-1)$ packed Shamir sharing of $[\boldsymbol{x}' \| \mathtt{pos}]_{n-1}$.

2. Note that $\boldsymbol{x}'$ contains all different values in $\boldsymbol{x}$. Thus, there is a linear map $f : \mathbb{F}^k \to \mathbb{F}^k$ such that $\boldsymbol{x} = f(\boldsymbol{x}')$. Therefore, relying on our technique for sharing transformation, all parties transform $[\boldsymbol{x}' \| \mathtt{pos}]_{n-1}$ to $[\boldsymbol{x}]_{n-k}$.

The communication complexity remains $O(n)$ field elements.

2.2.2 Evaluating Multiplication Gates Using Packed Beaver Triples

For a group of k multiplication gates, suppose all parties have prepared two degree-$(n-k)$ packed Shamir sharings $[\boldsymbol{x}]_{n-k}$ and $[\boldsymbol{y}]_{n-k}$. Let $\mathtt{pos}$ be the positions associated with these k gates. The goal is to compute a degree-$(n-k)$ packed Shamir sharing of $\boldsymbol{x} * \boldsymbol{y}$ stored at positions $\mathtt{pos}$. To this end, we extend the technique of Beaver triples [Bea91] to our setting, which we refer to as *packed Beaver triples*. We make use of a random

packed Beaver triple $([\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}, [\boldsymbol{c}]_{n-k})$, where $\boldsymbol{a}, \boldsymbol{b}$ are random vectors in $\mathbb{F}^k$ and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. All parties run the following steps:

1. All parties locally compute $[\boldsymbol{x}+\boldsymbol{a}]_{n-k} = [\boldsymbol{x}]_{n-k} + [\boldsymbol{a}]_{n-k}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{n-k} = [\boldsymbol{y}]_{n-k} + [\boldsymbol{b}]_{n-k}$.

2. The first party P_1 collects the whole sharings $[\boldsymbol{x}+\boldsymbol{a}]_{n-k}, [\boldsymbol{y}+\boldsymbol{b}]_{n-k}$ and reconstructs the secrets $\boldsymbol{x}+\boldsymbol{a}, \boldsymbol{y}+\boldsymbol{b}$. Recall that $\boldsymbol{x} = (x_1, \ldots, x_k)$ and $\boldsymbol{a} = (a_1, \ldots, a_k)$ are vectors in $\mathbb{F}^k$, and $\boldsymbol{x}+\boldsymbol{a} = (x_1 + a_1, \ldots, x_k + a_k)$. Similarly, $\boldsymbol{y}+\boldsymbol{b} = (y_1+b_1, \ldots, y_k+b_k)$. P_1 computes the sharings $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ and distributes the shares to other parties.

3. All parties locally compute

$$[\boldsymbol{z}]_{n-1} := [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{b}]_{n-k} - [\boldsymbol{y}+\boldsymbol{b}]_{k-1} \cdot [\boldsymbol{a}]_{n-k} + [\boldsymbol{c}]_{n-k}.$$

   Here the resulting sharing $[\boldsymbol{z}]_{n-1}$ has degree $n-1$ due to the second term and the third term.

4. Finally, all parties transform the sharing $[\boldsymbol{z}]_{n-1}$ to $[\boldsymbol{z} \| \mathtt{pos}]_{n-k}$. Relying on our technique of sharing transformation, this can be done with $O(n)$ field elements of communication.

Note that in the above steps, all parties only reveal $[\boldsymbol{x}+\boldsymbol{a}]_{n-k}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{n-k}$ to P_1. Recall that $[\boldsymbol{a}]_{n-k}$ and $[\boldsymbol{b}]_{n-k}$ are random degree-$(n-k)$ packed Shamir sharings. Therefore, $[\boldsymbol{x}+\boldsymbol{a}]_{n-k}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{n-k}$ are also random degree-$(n-k)$ packed Shamir sharings, which leak no information about $\boldsymbol{x}$ and $\boldsymbol{y}$ to P_1. Thus, the security follows.

Therefore, to evaluate a group of k multiplication gates, all parties need to prepare a random packed Beaver triple $([\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}, [\boldsymbol{c}]_{n-k})$, which is of size $O(n)$ field elements. The communication complexity is $O(n)$ field elements.

2.2.3 Summary

In summary, our protocol works as follows. All parties first prepare enough packed Beaver triples stored at the default positions in the preprocessing phase. Then in the online phase, all parties evaluate the circuit layer by layer. For each layer, all parties first use the protocol for network routing to prepare degree-$(n-k)$ packed Shamir sharings for the inputs of this layer. Then, for every group of addition gates, all parties can compute them locally due to the linear homomorhpism of the packed Shamir secret sharing scheme. For every group of multiplication gates, we use the technique of packed Beaver triple to evaluate these gates. In particular, evaluating each group of multiplication gates will consume one fresh packed Beaver triple prepared in the preprocessing phase.

When $t = (1-\epsilon) \cdot n$ for a positive constant ϵ, we have $k = (n-t+1)/2 = O(n)$. For the amount of preprocessing data, we need to prepare a packed Beaver triple for each group of k multiplication gates. Thus, the amount of preprocessing data is bounded by $O(\frac{|C|}{k} \cdot n) = O(|C|)$. For the amount of communication, note that all parties need to communicate during the network routing and the evaluation of multiplication gates. Both protocols require $O(n)$ elements of communication to process k secrets. Thus, the amount of communication complexity is also bounded by $O(\frac{|C|}{k} \cdot n) = O(|C|)$.

Therefore, we obtain an information-theoretic MPC protocol in the circuit-independent preprocessing model with semi-honest security that computes an arithmetic circuit C over a large finite field $\mathbb{F}$ (with $|\mathbb{F}| \geq |C| + n$) against $t = (1-\epsilon) \cdot n$ corrupted parties with $O(|C|)$ field elements of preprocessing data and $O(|C|)$ field elements of communication.

2.2.4 Other Results

Malicious Security of the Online Protocol. To achieve malicious security, we extend the idea of using information-theoretic MACs introduced in [BDOZ11, DPSZ12] to authenticate packed Shamir sharings. Concretely, at the beginning of the computation, all parties will prepare a random degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{\gamma}]_{n-k}$, where $\boldsymbol{\gamma} = (\gamma, \gamma, \ldots, \gamma) \in \mathbb{F}^k$ and γ is a random field element. The secrets $\boldsymbol{\gamma}$ serve

as the MAC key. To authenticate the secrets of a degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{x}]_{n-k}$, all parties will compute a degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{\gamma} * \boldsymbol{x}]_{n-k}$. We will show that almost all malicious behaviors of corrupted parties can be transformed to additive attacks, i.e., adding errors to the secrets of degree-$(n-k)$ packed Shamir sharings.

Note that if the corrupted parties change the secrets $\boldsymbol{x}$ to $\boldsymbol{x} + \boldsymbol{\delta}_1$, they also need to change the secrets $\boldsymbol{\gamma} * \boldsymbol{x}$ to $\boldsymbol{\gamma} * \boldsymbol{x} + \boldsymbol{\delta}_2$ such that $\boldsymbol{\delta}_2 = \boldsymbol{\gamma} * \boldsymbol{\delta}_1$. However, since γ is a uniform value in $\mathbb{F}$, the probability of a success attack is at most $1/|\mathbb{F}|$. When the field size is large enough, we can detect such an attack with overwhelming probability. See more details in Section 6.

Using the Result of [GPS21] for Small Finite Fields. Recall that our protocol requires the field size to be at least $|C| + n$. On the other hand, the protocol in [GPS21] can use a finite field of size $2n$. This is due to the use of different approaches to handle network routing.

When using a small finite field, we can use the technique in [GPS21] to handle network routing. Our technique for sharing transformation also improves the concrete efficiency of computing fan-out gates and performing permutations in [GPS21]. More details can be found in Section 7.

3 Preliminaries

In this work, we use the *client-server* model for the secure multi-party computation. In the client-server model, clients provide inputs to the functionality and receive outputs, and servers can participate in the computation but do not have inputs nor get outputs. Each party may have different roles in the computation. Note that, if every party plays a single client and a single server, this corresponds to a protocol in the standard MPC model. Let c denote the number of clients and n denote the number of servers. For all clients and servers, we assume that every two of them are connected via a secure (private and authentic) synchronous channel so that they can directly send messages to each other.

We focus on functions that can be represented as arithmetic circuits over a finite field $\mathbb{F}$ with input, addition, multiplication, and output gates.[2] We use κ to denote the security parameter, C to denote the circuit, and $|C|$ for the size of the circuit. In this work, we assume that the field size is $|\mathbb{F}| \geq 2^{\kappa}$. Note that it implies $|\mathbb{F}| \geq |C| + n$ since both the number of parties and the circuit size are bounded by $\mathrm{poly}(\kappa)$.

We are interested in the information-theoretic setting in the (circuit-independent) preprocessing model. The preprocessing model assumes that there is an ideal functionality which can prepare circuit-independent correlated randomness before the computation. Then the correlated randomness is used in a lightweight and fast online protocol. In particular, the preprocessing model enables the possibility of an information-theoretic protocol in the *dishonest majority setting*, which otherwise cannot exist in the plain model. The cost of a protocol in the preprocessing model is measured by both the amount of communication via private channels in the online phase and the amount of preprocessing data prepared in the preprocessing phase [Cou19, BGIN21].

3.1 Security Definition

Let $\mathcal{F}$ be a secure function evaluation functionality. An adversary $\mathcal{A}$ can corrupt at most c clients and t servers, provide inputs to corrupted clients, and receive all messages sent to corrupted clients and servers. In this work, we consider both semi-honest adversaries and fully malicious adversaries.

- If $\mathcal{A}$ is semi-honest, then corrupted clients and servers honestly follow the protocol.
- If $\mathcal{A}$ is fully malicious, then corrupted clients and servers can deviate from the protocol arbitrarily.

[2] In this work, we only focus on deterministic functions. A randomized function can be transformed to a deterministic function by taking as input an additional random tape from each party. The XOR of the input random tapes of all parties is used as the randomness of the randomized function.

Real-World Execution. In the real world, at the beginning of the protocol, all clients and servers receive circuit-independent correlated randomness specified by the protocol from an ideal functionality. Then, the adversary $\mathcal{A}$ controlling corrupted clients and servers interacts with honest clients and servers. At the end of the protocol, the output of the real-world execution includes the inputs and outputs of honest clients and servers and the view of the adversary.

Ideal-World Execution. In the ideal world, a simulator $\texttt{Sim}$ emulates the ideal functionality that provides circuit-independent correlated randomness, simulates honest clients and servers, and interacts with the adversary $\mathcal{A}$. Furthermore, $\texttt{Sim}$ has one-time access to $\mathcal{F}$, which includes providing inputs of corrupted clients and servers to $\mathcal{F}$, receiving the outputs of corrupted clients and servers, and sending instructions specified in $\mathcal{F}$ (e.g., asking $\mathcal{F}$ to abort). The output of the ideal-world execution includes the inputs and outputs of honest clients and servers and the view of the adversary.

We say that a protocol π securely computes $\mathcal{F}$ in the preprocessing model if there exists a simulator $\texttt{Sim}$, such that for all adversary $\mathcal{A}$, the distribution of the output of the real-world execution is *statistically indistinguishable* from the distribution of the output of the ideal-world execution.

3.2 Benefits of the Client-Server Model

In our construction, the clients only participate in the input phase and the output phase. The main computation is conducted by the servers. For simplicity, we use $\{P_1, \ldots, P_n\}$ to denote the n servers, and refer to the servers as parties. Let $\mathcal{C}orr$ denote the set of all corrupted parties and $\mathcal{H}$ denote the set of all honest parties. One benefit of the client-server model is that it is sufficient to only consider maximum adversaries, i.e., adversaries which corrupt t parties. At a high-level, for an adversary $\mathcal{A}$ which controls $t' < t$ parties, we may construct another adversary $\mathcal{A}'$ which controls additional $t - t'$ parties and behaves as follows:

- For a party corrupted by $\mathcal{A}$, $\mathcal{A}'$ follows the instructions of $\mathcal{A}$. This is achieved by passing messages between this party and other $n - t$ honest parties.
- For a party which is not corrupted by $\mathcal{A}$, but controlled by $\mathcal{A}'$, $\mathcal{A}'$ honestly follows the protocol.

Note that, if a protocol is secure against $\mathcal{A}'$, then this protocol is also secure against $\mathcal{A}$ since the additional $t - t'$ parties controlled by $\mathcal{A}'$ honestly follow the protocol in both cases. Thus, we only need to focus on $\mathcal{A}'$ instead of $\mathcal{A}$. Note that in the regular model, each honest party may have input. The same argument does not hold since the input of honest parties controlled by $\mathcal{A}'$ may be compromised. In the following, we assume that there are exactly t corrupted parties.

4 Preparing Random Sharings for Different Arithmetic Secret Sharing Schemes

4.1 Arithmetic Secret Sharing Schemes

Let $\mathcal{R}$ be a finite commutative ring. In this work, we consider the following arithmetic secret sharing schemes from [ACD+20] (with slight modifications).

Definition 1 (Arithmetic Secret Sharing Schemes)**.** *The syntax of an $\mathcal{R}$-arithmetic secret sharing scheme Σ consists of the following data:*

- *A set of parties $\mathcal{I} = \{1, \ldots, n\}$.*
- *A secret space $Z = \mathcal{R}^k$. k is also denoted as the number of secrets packed within Σ.*
- *A share space $U = \mathcal{R}^\ell$. ℓ is also denoted as the share size.*
- *A sharing space $C \subset U^{\mathcal{I}}$, where $U^{\mathcal{I}}$ denotes the indexed Cartesian product $\prod_{i \in \mathcal{I}} U$.*

- *An injective $\mathcal{R}$-module homomorphism:* $\mathtt{share} : Z \times \mathcal{R}^r \to C$*, which maps a secret* $\boldsymbol{x} \in Z$ *and a random tape* $\boldsymbol{\rho} \in \mathcal{R}^r$*, to a sharing* $\boldsymbol{X} \in C$*.* $\mathtt{share}$ *is also denoted as the sharing map of* Σ*.*
- *A surjective $\mathcal{R}$-module homomorphism:* $\mathtt{rec} : C \to Z$*, which takes as input a sharing* $\boldsymbol{X} \in C$ *and outputs a secret* $\boldsymbol{x} \in Z$*.* $\mathtt{rec}$ *is also denoted as the reconstruction map of* Σ*.*

The scheme Σ *satisfies that for all* $\boldsymbol{x} \in Z$ *and* $\boldsymbol{\rho} \in \mathcal{R}^r$*,* $\mathtt{rec}(\mathtt{share}(\boldsymbol{x}, \boldsymbol{\rho})) = \boldsymbol{x}$*. We may refer to* Σ *as the* 6*-tuple* $(n, Z, U, C, \mathtt{share}, \mathtt{rec})$*.*

For a non-empty set $A \subset \mathcal{I}$, the natural projection π_A maps a tuple $u = (u_i)_{i \in \mathcal{I}} \in U^{\mathcal{I}}$ to the tuple $(u_i)_{i \in A} \in U^A$.

Definition 2 (Privacy Set and Reconstruction Set)**.** *Suppose* $A \subset \mathcal{I}$ *is nonempty. We say* A *is a privacy set if for all* $\boldsymbol{x}_0, \boldsymbol{x}_1 \in Z$*, and for all vector* $\boldsymbol{v} \in U^A$*,*

$$\Pr_{\boldsymbol{\rho}}[\pi_A(\mathtt{share}(\boldsymbol{x}_0, \boldsymbol{\rho})) = \boldsymbol{v}] \;=\; \Pr_{\boldsymbol{\rho}}[\pi_A(\mathtt{share}(\boldsymbol{x}_1, \boldsymbol{\rho})) = \boldsymbol{v}].$$

We say A *is a reconstruction set if there is an $\mathcal{R}$-module homomorphism* $\mathtt{rec}_A : \pi_A(C) \to Z$*, such that for all* $\boldsymbol{X} \in C$*,*

$$\mathtt{rec}_A(\pi_A(\boldsymbol{X})) = \mathtt{rec}(\boldsymbol{X}).$$

Intuitively, for a privacy set A, the shares of parties in A are independent of the secret. For a reconstruction set A, the shares of parties in A fully determine the secret.

Threshold Linear Secret Sharing Schemes and Multiplication-friendly Property. In this work, we are interested in threshold arithmetic secret sharing schemes. Concretely, for a positive integer $t < n$, a threshold-t arithmetic secret sharing scheme satisfies that for all $A \subset \mathcal{I}$ with $|A| \leq t$, A is a privacy set.

We are interested in the following property.

Property 1 (Multiplication-Friendliness)**.** *We say* $\Sigma = (n, Z = \mathcal{R}^k, U, C, \mathtt{share}, \mathtt{rec})$ *is multiplication-friendly if there is an $\mathcal{R}$-arithmetic secret sharing scheme* $\Sigma' = (n, Z = \mathcal{R}^k, U', C', \mathtt{share}', \mathtt{rec}')$ *and* n *functions* $\{f_i : \mathcal{R}^k \times U \to U'\}_{i=1}^n$ *such that for all* $\boldsymbol{c} \in \mathcal{R}^k$ *and for all* $\boldsymbol{X} \in C$*,*

- $\boldsymbol{Y} = (f_1(\boldsymbol{c}, X_1), f_2(\boldsymbol{c}, X_2), \ldots, f_n(\boldsymbol{c}, X_n))$ *is in* C'*, i.e., a sharing in* Σ'*. We will use* $\boldsymbol{Y} = \boldsymbol{c} \cdot \boldsymbol{X}$ *to represent the computation process from* $\boldsymbol{c}$ *and* $\boldsymbol{X}$ *to* $\boldsymbol{Y}$*.*
- $\mathtt{rec}'(\boldsymbol{Y}) = \boldsymbol{c} * \mathtt{rec}(\boldsymbol{X})$*, where* $*$ *is the coordinate-wise multiplication operation.*

Intuitively, for a multiplication-friendly scheme Σ, if all parties hold a Σ-sharing of a secret $\boldsymbol{x} \in Z$ and a public vector $\boldsymbol{c} \in \mathcal{R}^k$, they can locally compute a Σ'-sharing of the secret $\boldsymbol{c} * \boldsymbol{x}$, where $*$ denotes the coordinate-wise multiplication operation.

Lemma 1. *If* Σ *is a multiplication-friendly threshold-t $\mathcal{R}$-arithmetic secret sharing scheme, and* Σ' *be the $\mathcal{R}$-arithmetic secret sharing scheme defined in Property 1, then* Σ' *has threshold* t*.*

Proof. We will prove that for all set A of t parties, for all $\boldsymbol{x}_0, \boldsymbol{x}_1 \in Z$, and for all vector $\boldsymbol{v}' \in (U')^A$,

$$\Pr_{\boldsymbol{\rho}'}[\pi_A(\mathtt{share}'(\boldsymbol{x}_0, \boldsymbol{\rho}')) = \boldsymbol{v}'] \;=\; \Pr_{\boldsymbol{\rho}'}[\pi_A(\mathtt{share}'(\boldsymbol{x}_1, \boldsymbol{\rho}')) = \boldsymbol{v}'].$$

Since Σ has threshold t, for all $\boldsymbol{v} \in U^A$, we have

$$\Pr_{\boldsymbol{\rho}}[\pi_A(\mathtt{share}(\boldsymbol{x}_0, \boldsymbol{\rho})) = \boldsymbol{v}] \;=\; \Pr_{\boldsymbol{\rho}}[\pi_A(\mathtt{share}(\boldsymbol{x}_1, \boldsymbol{\rho})) = \boldsymbol{v}].$$

Let $\mathbf{1}$ denote the vector $(1, 1, \ldots, 1) \in Z$ and $\mathbf{0}$ denote the vector $(0, 0, \ldots, 0) \in Z$. Then, for all $\boldsymbol{v}' \in (U')^A$,

$$\Pr_{\boldsymbol{\rho}, \boldsymbol{\rho}'}[\pi_A(\mathbf{1} \cdot \mathtt{share}(\boldsymbol{x}_0, \boldsymbol{\rho}) + \mathtt{share}'(\mathbf{0}, \boldsymbol{\rho}')) = \boldsymbol{v}'] \;=\; \Pr_{\boldsymbol{\rho}, \boldsymbol{\rho}'}[\pi_A(\mathbf{1} \cdot \mathtt{share}(\boldsymbol{x}_1, \boldsymbol{\rho}) + \mathtt{share}'(\mathbf{0}, \boldsymbol{\rho}')) = \boldsymbol{v}'].$$

It is sufficient to show that, for all $\boldsymbol{x} \in Z$, $\mathbf{1} \cdot \mathtt{share}(\boldsymbol{x}, \boldsymbol{\rho}) + \mathtt{share}'(\mathbf{0}, \boldsymbol{\rho}')$ is a random Σ'-sharing of $\boldsymbol{x}$. Note that, $\mathbf{1} \cdot \mathtt{share}(\boldsymbol{x}, \boldsymbol{\rho})$ is a Σ'-sharing of $\boldsymbol{x}$. Then there exists $\boldsymbol{\rho}''$ such that $\mathtt{share}'(\boldsymbol{x}, \boldsymbol{\rho}'') = \mathbf{1} \cdot \mathtt{share}(\boldsymbol{x}, \boldsymbol{\rho})$. Thus $\mathbf{1} \cdot \mathtt{share}(\boldsymbol{x}, \boldsymbol{\rho}) + \mathtt{share}'(\mathbf{0}, \boldsymbol{\rho}') = \mathtt{share}'(\boldsymbol{x}, \boldsymbol{\rho}'') + \mathtt{share}'(\mathbf{0}, \boldsymbol{\rho}') = \mathtt{share}'(\boldsymbol{x}, \boldsymbol{\rho}' + \boldsymbol{\rho}'')$. The last step follows from the fact that $\mathtt{share}'$ is an $\mathcal{R}$-module homomorphism. When $\boldsymbol{\rho}'$ is uniformly random, $\boldsymbol{\rho}' + \boldsymbol{\rho}''$ is also uniformly random. Thus $\mathtt{share}'(\boldsymbol{x}, \boldsymbol{\rho}' + \boldsymbol{\rho}'')$ is a random Σ'-sharing of $\boldsymbol{x}$. □

4.2 Packed Shamir Secret Sharing Scheme

In our work, we are interested in the packed Shamir secret sharing scheme. We use the packed secret-sharing technique introduced by Franklin and Yung [FY92]. This is a generalization of the standard Shamir secret sharing scheme [Sha79]. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2n$. Let n be the number of parties and k be the number of secrets that are packed in one sharing. Let $\alpha_1, \ldots, \alpha_n$ be n distinct elements in $\mathbb{F}$ and $\mathtt{pos} = (p_1, p_2, \ldots, p_k)$ be another k distinct elements in $\mathbb{F}$. A *degree-d* ($d \geq k-1$) packed Shamir sharing of $\boldsymbol{x} = (x_1, \ldots, x_k) \in \mathbb{F}^k$ is a vector $(w_1, \ldots, w_n)$ for which there exists a polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most d such that $f(p_i) = x_i$ for all $i \in \{1, 2, \ldots, k\}$, and $f(\alpha_i) = w_i$ for all $i \in \{1, 2, \ldots, n\}$. The i-th share w_i is held by party P_i. Reconstructing a degree-d packed Shamir sharing requires $d+1$ shares and can be done by Lagrange interpolation. For a random degree-d packed Shamir sharing of $\boldsymbol{x}$, any $d-k+1$ shares are independent of the secret $\boldsymbol{x}$.

In our work, we will always use the same elements $\alpha_1, \ldots, \alpha_n$ for the shares of all parties. However, we may use different elements $\mathtt{pos}$ for the secrets. We will use $[\boldsymbol{x} \| \mathtt{pos}]_d$ to denote a degree-d packed Shamir sharing of $\boldsymbol{x} \in \mathbb{F}^k$ stored at positions $\mathtt{pos}$. In the following, operations (addition and multiplication) between two packed Shamir sharings are coordinate-wise. We recall two properties of the packed Shamir sharing scheme:

- Linear Homomorphism: For all $d \geq k-1$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x} + \boldsymbol{y} \| \mathtt{pos}]_d = [\boldsymbol{x} \| \mathtt{pos}]_d + [\boldsymbol{y} \| \mathtt{pos}]_d$.
- Multiplicative: Let $*$ denote the coordinate-wise multiplication operation. For all $d_1, d_2 \geq k-1$ subject to $d_1 + d_2 < n$, and for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x} * \boldsymbol{y} \| \mathtt{pos}]_{d_1+d_2} = [\boldsymbol{x} \| \mathtt{pos}]_{d_1} \cdot [\boldsymbol{y} \| \mathtt{pos}]_{d_2}$.

These two properties directly follow from the computation of the underlying polynomials.

Note that the second property implies that, for all $k-1 \leq d \leq n-k$, a degree-d packed Shamir secret sharing scheme is multiplication-friendly (defined in Property 1). Concretely, for all $\boldsymbol{x}, \boldsymbol{c} \in \mathbb{F}^k$, all parties can locally compute $[\boldsymbol{c} * \boldsymbol{x} \| \mathtt{pos}]_{d+k-1}$ from $[\boldsymbol{x} \| \mathtt{pos}]_d$ and the public vector $\boldsymbol{c}$. To see this, all parties can locally transform $\boldsymbol{c}$ to a degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{c} \| \mathtt{pos}]_{k-1}$. Then, they can use the property of the packed Shamir sharing scheme to compute $[\boldsymbol{c} * \boldsymbol{x} \| \mathtt{pos}]_{d+k-1} = [\boldsymbol{c} \| \mathtt{pos}]_{k-1} \cdot [\boldsymbol{x} \| \mathtt{pos}]_d$.

Recall that t is the number of corrupted parties. Also recall that a degree-d packed Shamir secret sharing scheme is of threshold $d-k+1$. To ensure that the packed Shamir secret sharing scheme has threshold t and is multiplication-friendly, we choose k such that $t \leq d-k+1$ and $d \leq n-k$. When $d = n-k$ and $k = (n-t+1)/2$, both requirements hold and k is maximal.

4.3 Preparing Random Sharings for Different Arithmetic Secret Sharing Schemes

In this part, we introduce our main contribution: an efficient protocol that prepares random sharings for a batch of different arithmetic secret sharing schemes. Let $\mathcal{R}$ be a finite commutative ring. Let $\Pi = (n, \tilde{Z}, \tilde{U}, \tilde{C}, \mathtt{share}_\Pi, \mathtt{rec}_\Pi)$ be an $\mathcal{R}$-arithmetic secret sharing scheme. Our goal is to realize the functionality $\mathcal{F}_{\text{rand-sharing}}$ presented in Functionality 1.

Initialization. Let $\Sigma = (n, Z = \mathcal{R}^k, U, C, \mathtt{share}, \mathtt{rec})$ be a multiplication-friendly threshold-t $\mathcal{R}$-arithmetic secret sharing scheme. In the following, we will use $[\boldsymbol{x}]$ to denote a Σ-sharing of $\boldsymbol{x} \in \mathcal{R}^k$. Let $\Sigma' = (n, Z' = \mathcal{R}^k, U', C', \mathtt{share}', \mathtt{rec}')$ be the $\mathcal{R}$-arithmetic secret sharing scheme in Property 1. By Lemma 1, Σ' has threshold t. We use $\langle \boldsymbol{y} \rangle$ to denote a Σ'-sharing of $\boldsymbol{y} \in \mathcal{R}^k$. For all $\boldsymbol{c} \in \mathcal{R}^k$, we will write

$$\langle \boldsymbol{c} * \boldsymbol{x} \rangle = \boldsymbol{c} \cdot [\boldsymbol{x}]$$

Functionality 1: $\mathcal{F}_{\text{rand-sharing}}(\Pi)$

1. $\mathcal{F}_{\text{rand-sharing}}$ receives the set of corrupted parties, denoted by $\mathcal{C}orr$.
2. $\mathcal{F}_{\text{rand-sharing}}$ receives from the adversary a set of shares $\{\boldsymbol{u}_j\}_{j\in\mathcal{C}orr}$ where $\boldsymbol{u}_j \in \tilde{U}$ for all $j \in \mathcal{C}orr$.
3. $\mathcal{F}_{\text{rand-sharing}}$ samples a random Π-sharing $\boldsymbol{X}$ such that the shares of $\boldsymbol{X}$ held by corrupted parties are identical to those received from the adversary, i.e., $\pi_{\mathcal{C}orr}(\boldsymbol{X}) = (\boldsymbol{u}_j)_{j\in\mathcal{C}orr}$. If such a sharing does not exist, $\mathcal{F}_{\text{rand-sharing}}$ sends `abort` to all honest parties and halts.
4. Otherwise, $\mathcal{F}_{\text{rand-sharing}}$ distributes the shares of $\boldsymbol{X}$ to honest parties.

to represent the computation process from $\boldsymbol{c}$ and $[\boldsymbol{x}]$ to $\langle \boldsymbol{c} * \boldsymbol{x}\rangle$ in Property 1.

Our construction will use the ideal functionality $\mathcal{F}_{\text{rand}} = \mathcal{F}_{\text{rand-sharing}}(\Sigma)$ that prepares a random Σ-sharing, and the ideal functionality $\mathcal{F}_{\text{randZero}}$ (Functionality 2) that prepares a random Σ'-sharing of $\mathbf{0} \in \mathcal{R}^k$.

Functionality 2: $\mathcal{F}_{\text{randZero}}$

1. Let $\Sigma' = (n, Z' = \mathcal{R}^k, U', C', \texttt{share}', \texttt{rec}')$. $\mathcal{F}_{\text{randZero}}$ receives the set of corrupted parties, denoted by $\mathcal{C}orr$.
2. $\mathcal{F}_{\text{randZero}}$ receives from the adversary a set of shares $\{\boldsymbol{u}'_j\}_{j\in\mathcal{C}orr}$, where $\boldsymbol{u}'_j \in U'$ for all $P_j \in \mathcal{C}orr$.
3. $\mathcal{F}_{\text{randZero}}$ samples a random Σ'-sharing of $\mathbf{0} \in \mathcal{R}^k$, $\langle \mathbf{0}\rangle$, such that the shares of corrupted parties are identical to those received from the adversary, i.e., $\pi_{\mathcal{C}orr}(\langle \mathbf{0}\rangle) = (\boldsymbol{u}'_j)_{j\in\mathcal{C}orr}$. If such a sharing does not exist, $\mathcal{F}_{\text{randZero}}$ sends `abort` to all honest parties and halts.
4. Otherwise, $\mathcal{F}_{\text{randZero}}$ distributes the shares of $\langle \mathbf{0}\rangle$ to honest parties.

Let $\Pi_1, \Pi_2, \ldots, \Pi_k$ be k arbitrary $\mathcal{R}$-arithmetic secret sharing schemes with the restriction that all schemes have the same share size, i.e., the share space $\tilde{U} = \mathcal{R}^{\tilde{\ell}}$. Let $\tilde{Z}_i = \mathcal{R}^{\tilde{k}_i}$ be the secret space of Π_i and $\texttt{share}_i : \tilde{Z}_i \times \mathcal{R}^{\tilde{r}_i} \to \tilde{C}_i$ be the sharing map. Since $\texttt{share}_i$ is injective, and $\tilde{C}_i \subset \tilde{U}^{\mathcal{I}}$, we have $\tilde{k}_i + \tilde{r}_i \le n \cdot \tilde{\ell}$.

The goal is to prepare k random sharings $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_k$ such that $\boldsymbol{X}_i$ is a random Π_i-sharing, i.e., realizing $\{\mathcal{F}_{\text{rand-sharing}}(\Pi_i)\}_{i=1}^k$.

Protocol Description. The construction of our protocol Rand-Sharing appears in Protocol 3. Protocol Rand-Sharing requires $n^2 \cdot \tilde{\ell} \cdot (\ell + \ell')$ ring elements of preprocessing data and $n^2 \cdot \tilde{\ell} \cdot \ell'$ ring elements of communication to prepare k random sharings for $\Pi_1, \Pi_2, \ldots, \Pi_k$, one for each secret sharing scheme. The detailed cost analysis can be found below.

Lemma 2. *For any k $\mathcal{R}$-arithmetic secret sharing schemes $\{\Pi_i\}_{i=1}^k$ such that they have the same share size, Protocol* Rand-Sharing *securely computes $\{\mathcal{F}_{\text{rand-sharing}}(\Pi_i)\}_{i=1}^k$ in the $\{\mathcal{F}_{\text{rand}}, \mathcal{F}_{\text{randZero}}\}$-hybrid model against a semi-honest adversary who controls t parties.*

Proof. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}orr$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties.

The simulator $\mathcal{S}$ works as follows.

1. In Step 2, $\mathcal{S}$ emulates the functionality $\mathcal{F}_{\text{rand}}$: For all $v \in \{1, 2, \ldots, n \cdot \tilde{\ell}\}$, $\mathcal{S}$ receives the shares of $[\boldsymbol{r}_v]$ held by corrupted parties.

Protocol 3: Rand-Sharing

1. Let $\Pi_1, \Pi_2, \ldots, \Pi_k$ be k arbitrary $\mathcal{R}$-arithmetic secret sharing schemes such that they have the same share size. Let $\tilde{U} = \mathcal{R}^{\tilde{\ell}}$ denote the share space. For all $i \in \{1, 2, \ldots, k\}$, let $\tilde{Z}_i = \mathcal{R}^{\tilde{k}_i}$ be the secret space of Π_i, and $\mathtt{share}_i : \tilde{Z}_i \times \mathcal{R}^{\tilde{r}_i} \to \tilde{C}_i$ be the sharing map of Π_i. We have $\tilde{k}_i + \tilde{r}_i \leq n \cdot \tilde{\ell}$.

2. All parties invoke $\mathcal{F}_{\text{rand}}$ $n \cdot \tilde{\ell}$ times and obtain $n \cdot \tilde{\ell}$ random Σ-sharings, denoted by $[\boldsymbol{r}_1], [\boldsymbol{r}_2], \ldots, [\boldsymbol{r}_{n \cdot \tilde{\ell}}]$. For all $i \in \{1, 2, \ldots, k\}$, let $\boldsymbol{\tau}_i = (r_{1,i}, r_{2,i}, \ldots, r_{\tilde{k}_i, i}) \in \mathcal{R}^{\tilde{k}_i}$, and $\boldsymbol{\rho}_i = (r_{\tilde{k}_i+1,i}, r_{\tilde{k}_i+2,i}, \ldots, r_{\tilde{k}_i+\tilde{r}_i,i}) \in \mathcal{R}^{\tilde{r}_i}$. The goal of this protocol is to compute the Π_i-sharing $\boldsymbol{X}_i = \mathtt{share}_i(\boldsymbol{\tau}_i, \boldsymbol{\rho}_i)$.

3. All parties invoke $\mathcal{F}_{\text{randZero}}$ $n \cdot \tilde{\ell}$ times and obtain $n \cdot \tilde{\ell}$ random Σ'-sharings of $\mathbf{0} \in \mathcal{R}^k$, denoted by $\{\langle \boldsymbol{o}_j^{(1)} \rangle, \langle \boldsymbol{o}_j^{(2)} \rangle, \ldots, \langle \boldsymbol{o}_j^{(\tilde{\ell})} \rangle\}_{j=1}^n$.

4. For all $i \in \{1, 2, \ldots, k\}$, $j \in \{1, 2, \ldots, n\}$, and $m \in \{1, 2, \ldots, \tilde{\ell}\}$, let $\mathcal{L}_j^{(i,m)} : \tilde{Z}_i \times \mathcal{R}^{\tilde{r}_i} \to \mathcal{R}$ denote the $\mathcal{R}$-module homomorphism such that for all $\boldsymbol{\tau} \in \tilde{Z}_i$ and $\boldsymbol{\rho} \in \mathcal{R}^{\tilde{r}_i}$, $\mathcal{L}_j^{(i,m)}(\boldsymbol{\tau}, \boldsymbol{\rho})$ outputs the m-th element of the j-th share of the Π_i-sharing $\mathtt{share}_i(\boldsymbol{\tau}, \boldsymbol{\rho})$. Then there exist $c_{j,1}^{(i,m)}, \ldots, c_{j,\tilde{k}_i+\tilde{r}_i}^{(i,m)} \in \mathcal{R}$ such that

$$\mathcal{L}_j^{(i,m)}(\boldsymbol{\tau}, \boldsymbol{\rho}) = \sum_{v=1}^{\tilde{k}_i} c_{j,v}^{(i,m)} \cdot \tau_v + \sum_{v=1}^{\tilde{r}_i} c_{j,\tilde{k}_i+v}^{(i,m)} \cdot \rho_v.$$

   For all $j \in \{1, 2, \ldots, n\}$, $m \in \{1, 2, \ldots, \tilde{\ell}\}$, and $v \in \{1, \ldots, n \cdot \tilde{\ell}\}$, let

$$\boldsymbol{c}_{j,v}^{(\star,m)} = (c_{j,v}^{(1,m)}, c_{j,v}^{(2,m)}, \ldots, c_{j,v}^{(k,m)}) \in \mathcal{R}^k,$$

   where $c_{j,v}^{(i,m)} = 0$ for all $v > \tilde{k}_i + \tilde{r}_i$.

5. For all $i \in \{1, 2, \ldots, k\}$, $j \in \{1, 2, \ldots, n\}$, and $m \in \{1, 2, \ldots, \tilde{\ell}\}$, let $u_j^{(i,m)} = \mathcal{L}_j^{(i,m)}(\boldsymbol{\tau}_i, \boldsymbol{\rho}_i)$. Let $\boldsymbol{u}_j^{(\star,m)} = (u_j^{(1,m)}, u_j^{(2,m)}, \ldots, u_j^{(k,m)})$. For all $j \in \{1, 2, \ldots, n\}$ and $m \in \{1, 2, \ldots, \tilde{\ell}\}$, all parties locally compute a Σ'-sharing

$$\langle \boldsymbol{u}_j^{(\star,m)} \rangle = \langle \boldsymbol{o}_j^{(m)} \rangle + \sum_{v=1}^{n \cdot \tilde{\ell}} \boldsymbol{c}_{j,v}^{(\star,m)} \cdot [\boldsymbol{r}_v].$$

   Then, all parties send their shares of $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ to P_j.

6. For all $j \in \{1, 2, \ldots, n\}$ and $m \in \{1, 2, \ldots, \tilde{\ell}\}$, P_j reconstructs the Σ'-sharing $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ and learns $\boldsymbol{u}_j^{(\star,m)} = (u_j^{(1,m)}, u_j^{(2,m)}, \ldots, u_j^{(k,m)})$. Then for all $i \in \{1, 2, \ldots, k\}$, P_j sets his share of the Π_i-sharing, $\boldsymbol{X}_i$, to be $\boldsymbol{u}_j^{(i)} = (u_j^{(i,1)}, u_j^{(i,2)}, \ldots, u_j^{(i,\tilde{\ell})})$. All parties take $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_k$ as output.

2. In Step 3, $\mathcal{S}$ emulates the functionality $\mathcal{F}_{\text{randZero}}$: For all $j \in \{1, 2, \ldots, n\}$ and $m \in \{1, 2, \ldots, \tilde{\ell}\}$, $\mathcal{S}$ receives the shares of $\langle \boldsymbol{o}_j^{(m)} \rangle$ held by corrupted parties.

3. In Step 4, for all $j \in \{1, 2, \ldots, n\}$, $m \in \{1, 2, \ldots, \tilde{\ell}\}$, and $v \in \{1, \ldots, n \cdot \tilde{\ell}\}$, $\mathcal{S}$ locally compute $\boldsymbol{c}_{j,v}^{(\star,m)}$ by following the protocol.

4. In Step 5, for all $j \in \{1, 2, \ldots, n\}$ and $m \in \{1, 2, \ldots, \tilde{\ell}\}$, $\mathcal{S}$ follows the protocol and computes the

shares of $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ held by corrupted parties. $\mathcal{S}$ simulates the shares that are sent from honest parties to corrupted parties as follows:

(a) For all $i \in \{1, 2, \ldots, k\}$, $\mathcal{S}$ samples a random Π_i-sharing and obtains the shares of corrupted parties $\{\boldsymbol{u}_j^{(i)}\}_{j \in \mathcal{C}orr}$, where $\boldsymbol{u}_j^{(i)} = (u_j^{(i,1)}, u_j^{(i,2)}, \ldots, u_j^{(i,\tilde{\ell})}) \in \mathcal{R}^{\tilde{\ell}}$.

(b) Then for all $j \in \{1, 2, \ldots, n\}$ and $m \in \{1, 2, \ldots, \tilde{\ell}\}$, $\mathcal{S}$ computes $\boldsymbol{u}_j^{(\star,m)}$.

(c) According to Lemma 1, Σ' has threshold t. Therefore, the shares of a random Σ'-sharing are independent of its secret. For all $j \in \mathcal{C}orr$ and $m \in \{1, 2, \ldots, \tilde{\ell}\}$, $\mathcal{S}$ samples a random Σ'-sharing $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ based on the secret $\boldsymbol{u}_j^{(\star,m)}$ and the shares of $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ held by corrupted parties computed by $\mathcal{S}$.

(d) Finally, $\mathcal{S}$ sends the shares of $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ held by honest parties to the adversary.

5. In Step 6, for all $i \in \{1, 2, \ldots, k\}$, $\mathcal{S}$ sends the shares $\{\boldsymbol{u}_j^{(i)}\}_{j \in \mathcal{C}orr}$ to $\mathcal{F}_{\text{rand-sharing}}(\Pi_i)$.

This completes the description of the simulator. We show that the simulator $\mathcal{S}$ perfectly simulates the behaviors of honest parties. Note that all parties only communicate with other parties in Step 5.

- We first show that the shares of corrupted parties, $\{\boldsymbol{u}_j^{(i)}\}_{j \in \mathcal{C}orr}$, sampled by $\mathcal{S}$ have the same distribution as those in the real world. In the ideal world, $\mathcal{S}$ randomly samples the shares of corrupted parties by first sampling random sharings of $\Pi_1, \Pi_2, \ldots, \Pi_k$ and then obtaining the shares of corrupted parties. In the real world, the shares of corrupted parties are computed by following $\mathtt{share}_i(\boldsymbol{\tau}_i, \boldsymbol{\rho}_i)$ for all $i \in \{1, 2, \ldots, k\}$. Recall that $\boldsymbol{\tau}_i, \boldsymbol{\rho}_i$ are generated by $\mathcal{F}_{\text{rand}}$. And Σ has threshold t. Therefore, $\boldsymbol{\tau}_i$ and $\boldsymbol{\rho}_i$ are uniformly random. Thus, $\{\boldsymbol{u}_j^{(i)}\}_{j \in \mathcal{C}orr}$ have the same distribution in both the ideal world and the real world.

- Then, we show that for all $j \in \mathcal{C}orr$ and $m \in \{1, 2, \ldots, \tilde{\ell}\}$, the shares of $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ held by honest parties (which are supposed to send to corrupted parties) generated by $\mathcal{S}$ have the same distribution as those in the real world. Recall that $\langle \boldsymbol{o}_j^{(m)} \rangle$ is a random Σ'-sharing of $\mathbf{0}$. Therefore, in the real world, $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ is a random Σ'-sharing of $\boldsymbol{u}_j^{(\star,m)}$ given the secret and the shares of corrupted parties. Since $\boldsymbol{u}_j^{(\star,m)}$ is determined by $\{\boldsymbol{\tau}_i, \boldsymbol{\rho}_i\}_{i=1}^k$, which are independent of the shares of $\{[\boldsymbol{r}_v]\}_{v=1}^{n \cdot \tilde{\ell}}$ and $\langle \boldsymbol{o}_j^{(m)} \rangle$ held by corrupted parties, $\boldsymbol{u}_j^{(\star,m)}$ is independent of the shares of $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ held by corrupted parties.

  In the ideal world, recall that $\mathcal{S}$ learns the shares of $\{[\boldsymbol{r}_v]\}_{v=1}^{n \cdot \tilde{\ell}}$ held by corrupted parties in Step 2. And $\mathcal{S}$ learns the shares of $\langle \boldsymbol{o}_j^{(m)} \rangle$ held by corrupted parties in Step 3. Therefore, $\mathcal{S}$ can compute the shares of $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ held by corrupted parties. Also recall that we have shown that the secret, $\boldsymbol{u}_j^{(\star,m)}$, which is a part of $\{\boldsymbol{u}_j^{(i)}\}_{j \in \mathcal{C}orr, i \in \{1,\ldots,k\}}$, has the same distribution as that in the real world. Therefore, the secret $\boldsymbol{u}_j^{(\star,m)}$ and the shares of $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ held by corrupted parties have the same distribution in both the ideal world and the real world. Finally, note that $\mathcal{S}$ samples a random Σ'-sharing $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ based on the secret $\boldsymbol{u}_j^{(\star,m)}$ simulated by $\mathcal{S}$ and the shares of $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ held by corrupted parties computed by $\mathcal{S}$. Thus, the shares of honest parties simulated by $\mathcal{S}$ have the same distribution as those in the real world.

  Recall that all parties only communicate with other parties in Step 5. Therefore, the joint view of corrupted parties in the ideal world is identical to that in the real world.

- Finally, we show that the output of honest parties given the joint view of corrupted parties in the ideal world has the same distribution as that in the real world.

  Note that the joint view of corrupted parties is determined by (1) the shares of $[\boldsymbol{r}_v]$ held by corrupted parties for all $v \in \{1, 2, \ldots, n \cdot \tilde{\ell}\}$, (2) the shares of $\langle \boldsymbol{o}_j^{(m)} \rangle$ held by corrupted parties for all $j \in$

$\{1, 2, \ldots, n\}$ and $m \in \{1, 2, \ldots, \tilde{\ell}\}$, and (3) the shares of $\langle \boldsymbol{u}_j^{(\star,m)} \rangle$ held by honest parties for all $j \in \mathcal{C}orr$ and $m \in \{1, 2, \ldots, \tilde{\ell}\}$. In the real world, (1) and (2) are independent of $\{\boldsymbol{\tau}_i, \boldsymbol{\rho}_i\}_{i=1}^k$ since Σ has threshold t. And (3) only depends on the shares $\{\boldsymbol{u}_j^{(i)}\}_{j \in \mathcal{C}orr, i \in \{1,\ldots,k\}}$. Thus, in the real world, for all $i \in \{1, 2, \ldots, k\}$, $\boldsymbol{X}_i = \mathtt{share}_i(\boldsymbol{\tau}_i, \boldsymbol{\rho}_i)$ is a random Π_i-sharing given the shares of corrupted parties. And honest parties simply output their shares of $\boldsymbol{X}_i$.

In the ideal world, $\mathcal{S}$ perfectly simulates the shares of $\boldsymbol{X}_i = \mathtt{share}_i(\boldsymbol{\tau}_i, \boldsymbol{\rho}_i)$ held by corrupted parties and sends them to $\mathcal{F}_{\text{rand-sharing}}(\Pi_i)$. Then $\mathcal{F}_{\text{rand-sharing}}(\Pi_i)$ samples a random Π_i-sharing based on the shares of corrupted parties. The output of honest parties is their shares of $\boldsymbol{X}_i$ for all $i \in \{1, 2, \ldots, k\}$.

Therefore, the output of honest parties given the joint view of corrupted parties in the ideal world has the same distribution as that in the real world.

We conclude that Protocol Rand-Sharing securely computes $\{\mathcal{F}_{\text{rand-sharing}}(\Pi_i)\}_{i=1}^k$ in the $\{\mathcal{F}_{\text{rand}}, \mathcal{F}_{\text{randZero}}\}$-hybrid model against a semi-honest adversary who controls t parties. □

Cost of Protocol 3. We measure the amount of the preprocessing data and the ring elements that all parties need to send.

For the amount of preprocessing data, all parties need to prepare $n \cdot \tilde{\ell}$ random Σ-sharings. Let ℓ be the share size of Σ, i.e., the share space of Σ is $U = \mathcal{R}^\ell$. Thus, all Σ-sharings are of size $n^2 \cdot \tilde{\ell} \cdot \ell$ ring elements. All parties also need to prepare $n \cdot \tilde{\ell}$ random Σ'-sharings of $\mathbf{0} \in \mathcal{R}^k$. Let ℓ' be the share size of Σ', i.e., the share space of Σ' is $U' = \mathcal{R}^{\ell'}$. Thus all Σ'-sharings are of size $n^2 \cdot \tilde{\ell} \cdot \ell'$ ring elements. In total, the amount of preprocessing data is $n^2 \cdot \tilde{\ell} \cdot (\ell + \ell')$ ring elements.

For the communication complexity, all parties only need to communicate in Step 5 of Protocol 3. The communication complexity is $n^2 \cdot \tilde{\ell} \cdot \ell'$ ring elements.

4.4 Instantiating Protocol Rand-Sharing via Packed Shamir Secret Sharing Scheme

For Large Finite Fields $\mathbb{F}$. Recall that when $k = (n - t + 1)/2$, a degree-$(n - k)$ packed Shamir secret sharing has threshold t and is multiplication-friendly. Therefore, we use a degree-$(n - k)$ packed Shamir secret sharing scheme to instantiate Σ in Protocol Rand-Sharing. Then Σ' is a degree-$(n - 1)$ packed Shamir secret sharing scheme. For Σ and Σ',

- The secret space is $\mathbb{F}^k$, where $k = (n - t + 1)/2$.
- The share space is $\mathbb{F}$, i.e., each share is a single field element. Therefore $\ell = \ell' = 1$.

Thus, we obtain a protocol that prepares random sharings for $\Pi_1, \Pi_2, \ldots, \Pi_k$ with $2 \cdot n^2 \cdot \tilde{\ell} = O(n^2 \cdot \tilde{\ell})$ field elements of preprocessing data and $n^2 \cdot \tilde{\ell}$ field elements of communication. On average, the cost per random sharing is $O(\frac{n^2}{n-t+1} \cdot \tilde{\ell})$ field elements of both preprocessing data and communication. Note that when $t = (1-\epsilon) \cdot n$ for a positive constant ϵ, the achieved amortized cost per sharing is $O(n \cdot \tilde{\ell})$ field elements. In particular, $n \cdot \tilde{\ell}$ is the sharing size of Π_i for all $i \in \{1, 2, \ldots, k\}$. Essentially, it costs the same as letting a trusted party generate a random Π_i-sharing and distribute to all parties.

For Small Fields $\mathbb{F}_q$ and Rings $\mathbb{Z}/p^\ell\mathbb{Z}$. For a small field $\mathbb{F}_q$, we can use a large extension field of $\mathbb{F}_q$ so that the packed Shamir secret sharing scheme is available. For a ring $\mathbb{Z}/p^\ell\mathbb{Z}$, we can similarly use a large Galois ring of $\mathbb{Z}/p^\ell\mathbb{Z}$ so that the packed Shamir secret sharing scheme is available [ACD+19].

However, the approach of using a large extension field (or a large Galois ring) leads to a loss in the extension factor: While the share size grows up by the extension factor, the number of secrets that can be packed is still k. To resolve this issue, we can use the notion of reverse multiplication-friendly embedding (RMFE), which is first introduced in [CCXY18] for finite fields, and then extended to rings $\mathbb{Z}/p^\ell\mathbb{Z}$ in [CRX21].

We take an RMFE over a finite field $\mathbb{F}_q$ as example, where q is an exponential of a prime. Informally, a pair of $\mathbb{F}_q$-linear maps (ϕ, ψ) is a $(r, m)_q$-reverse multiplication-friendly embedding if $\phi : \mathbb{F}_q^r \to \mathbb{F}_{q^m}$ and

$\psi : \mathbb{F}_{q^m} \to \mathbb{F}_q^r$ satisfy that for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^r$, $\boldsymbol{x} * \boldsymbol{y} = \psi(\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y}))$. Intuitively, instead of doing a coordinate-wise multiplication over $\mathbb{F}_q$, an RMFE allows us to first map each vector (by ϕ) to an element in the extension field $\mathbb{F}_{q^m}$ and then perform a field multiplication over $\mathbb{F}_{q^m}$. The result can finally be transformed (by ψ) to the coordinate-wise multiplication between the input two vectors.

Now consider an arithmetic secret sharing scheme Σ over $\mathbb{F}_q$:

- The secret space is $Z = \mathbb{F}_q^{k \cdot r}$.
- The share space is $U = \mathbb{F}_{q^m}$, which can be viewed as $\mathbb{F}_q^m$.
- For $\boldsymbol{x} \in Z$, the sharing of $\boldsymbol{x}$ is computed as follows: We first divide $\boldsymbol{x}$ into k vectors of dimension r. Then we map each vector to a field element in $\mathbb{F}_{q^m}$ by using ϕ. Next we use a degree-$(n-k)$ packed Shamir secret sharing scheme over $\mathbb{F}_{q^m}$ to store the k elements in $\mathbb{F}_{q^m}$.
- For a sharing $\boldsymbol{X}$, we first view it as a degree-$(n-k)$ packed Shamir sharing over $\mathbb{F}_{q^m}$ and reconstruct its secret $\boldsymbol{s} \in \mathbb{F}_{q^m}^k$. To recover the secret, we apply ϕ^{-1} on each element in $\boldsymbol{s}$. (See [PS21] for an explicit construction of ϕ^{-1}.)

Note that Σ have threshold t due to the use of the degree-$(n-k)$ packed Shamir secret sharing scheme. Also note that Σ is still multiplication-friendly: To multiply a constant vector $\boldsymbol{c} \in \mathbb{F}_q^{k \cdot r}$ with a Σ-sharing of $\boldsymbol{x}$, we first transform $\boldsymbol{c}$ to a degree-$(k-1)$ packed Shamir sharing over $\mathbb{F}_{q^m}$ similarly as above. Then after multiplying the two packed Shamir sharings, we can reconstruct the secret $\boldsymbol{c} * \boldsymbol{x}$ by using ψ to decode the secrets of the resulting packed Shamir sharing.

Therefore, relying on packed Shamir secret sharing schemes and RMFEs, we can use Σ to instantiate Protocol Rand-Sharing for a small field $\mathbb{F}_q$. For both Σ and Σ',

- The secret space is $\mathbb{F}_q^{k \cdot r}$, where $k = (n - t + 1)/2$.
- The share space is $\mathbb{F}_q^m$, i.e., each share is m field elements. Therefore $\ell = \ell' = m$.

Note that, while the share size of Σ and Σ' grows up by a factor of m, the number of secrets that can be packed becomes $k \cdot r$, which grows up by a factor of r. Thus, we obtain a protocol that prepares random sharings for arbitrary $\mathbb{F}_q$-arithmetic secret sharing schemes $\Pi_1, \Pi_2, \ldots, \Pi_{k \cdot r}$ with $2 \cdot n^2 \cdot \tilde{\ell} \cdot m = O(n^2 \cdot \tilde{\ell} \cdot m)$ field elements of preprocessing data, and $n^2 \cdot \tilde{\ell} \cdot m$ field elements of communication. On average, the cost per random sharing is $O(\frac{n^2}{n-t+1} \cdot \tilde{\ell} \cdot \frac{m}{r})$ field elements of both preprocessing data and communication.

In [CCXY18], Cascudo, et al show that for all $\mathbb{F}_q$, there exists a family of RMFEs with r slowly grows to infinity and m/r is bounded by a constant. Thus, the amortized cost per random sharing becomes $O(\frac{n^2}{n-t+1} \cdot \tilde{\ell})$ field elements, which is the same as that for a large finite field.

A similar result for rings $\mathbb{Z}/p^\ell\mathbb{Z}$ can be achieved by using RMFEs over rings $\mathbb{Z}/p^\ell\mathbb{Z}$ constructed in [CRX21].

4.5 Application of $\mathcal{F}_{\text{rand-sharing}}$

Let Σ and Σ' be two threshold-t $\mathcal{R}$-arithmetic secret sharing schemes. Let $f : Z \to Z'$ be an $\mathcal{R}$-module homomorphism, where Z and Z' are the secret spaces of Σ and Σ' respectively. Suppose given a Σ-sharing, $\boldsymbol{X}$, all parties want to compute a Σ'-sharing, $\boldsymbol{Y}$, subject to $\mathtt{rec}'(\boldsymbol{Y}) = f(\mathtt{rec}(\boldsymbol{X}))$, where $\mathtt{rec}$ and $\mathtt{rec}'$ are reconstruction maps of Σ and Σ', respectively. We refer to this problem as sharing transformation.

As discussed in Section 2, sharing transformation can be efficiently solved with the help of a pair of random sharings $(\boldsymbol{R}, \boldsymbol{R}')$, where $\boldsymbol{R}$ is a Σ-sharing, and $\boldsymbol{R}'$ is a Σ'-sharing subject to $\mathtt{rec}'(\boldsymbol{R}') = f(\mathtt{rec}(\boldsymbol{R}))$. Consider the following $\mathcal{R}$-arithmetic secret sharing scheme $\widetilde{\Sigma} = \widetilde{\Sigma}(\Sigma, \Sigma', f)$:

- The secret space is Z, the same as that of Σ.
- The share space is $U \times U'$, where U is the share space of Σ and U' is the share space of Σ'.
- For a secret $\boldsymbol{x} \in Z$, the sharing of $\boldsymbol{x}$ is the concatenation of a Σ-sharing of $\boldsymbol{x}$ and a Σ'-sharing of $f(\boldsymbol{x})$.

- For a sharing $\boldsymbol{X}$, recall that each share of $\widetilde{\Sigma}$ consists of one share of Σ and one share of Σ'. The secret of $\boldsymbol{X}$ can be recovered by applying $\mathtt{rec}$ of Σ on the sharing which consists of the shares of Σ in $\boldsymbol{X}$.

Then, $(\boldsymbol{R}, \boldsymbol{R}')$ is a random $\widetilde{\Sigma}$-sharing. The problem is reduced to prepare a random $\widetilde{\Sigma}$-sharing, which can be done by $\mathcal{F}_{\text{rand-sharing}}(\widetilde{\Sigma})$.

We summarize the functionality $\mathcal{F}_{\text{tran}}$ in Functionality 4 and the protocol TRAN for $\mathcal{F}_{\text{tran}}$ in Protocol 5.

Functionality 4: $\mathcal{F}_{\text{tran}}$

1. $\mathcal{F}_{\text{tran}}$ receives the set of corrupted parties, denoted by $\mathcal{C}orr$. $\mathcal{F}_{\text{tran}}$ also receives two threshold-t $\mathcal{R}$-arithmetic secret sharing schemes Σ, Σ' and an $\mathcal{R}$-module homomorphism $f : Z \to Z'$.
2. $\mathcal{F}_{\text{tran}}$ receives a Σ-sharing $\boldsymbol{X}$ from all parties and computes $f(\mathtt{rec}(\boldsymbol{X}))$.
3. $\mathcal{F}_{\text{tran}}$ receives from the adversary a set of shares $\{\boldsymbol{u}'_j\}_{j\in\mathcal{C}orr}$, where $\boldsymbol{u}'_j \in U'$ for all $P_j \in \mathcal{C}orr$.
4. $\mathcal{F}_{\text{tran}}$ samples a random Σ'-sharing, $\boldsymbol{Y}$, such that $\mathtt{rec}'(\boldsymbol{Y}) = f(\mathtt{rec}(\boldsymbol{X}))$ and the shares of corrupted parties are identical to those received from the adversary, i.e., $\pi_{\mathcal{C}orr}(\boldsymbol{Y}) = (\boldsymbol{u}'_j)_{j\in\mathcal{C}orr}$. If such a sharing does not exist, $\mathcal{F}_{\text{tran}}$ sends $\mathtt{abort}$ to honest parties and halts.
5. Otherwise, $\mathcal{F}_{\text{tran}}$ distributes the shares of $\boldsymbol{Y}$ to honest parties.

Protocol 5: TRAN

1. Let Σ, Σ' be two threhsold-t $\mathcal{R}$-arithmetic secret sharing schemes and $f : Z \to Z'$ be an $\mathcal{R}$-module homomorphism. All parties hold a Σ-sharing, $\boldsymbol{X}$, at the beginning of the protocol.
2. Let $\widetilde{\Sigma} = \widetilde{\Sigma}(\Sigma, \Sigma', f)$ be the threshold-t $\mathcal{R}$-arithmetic secret sharing scheme defined above. All parties invoke $\mathcal{F}_{\text{rand-sharing}}(\widetilde{\Sigma})$ and obtain a $\widetilde{\Sigma}$-sharing $(\boldsymbol{R}, \boldsymbol{R}')$.
3. All parties locally compute $\boldsymbol{X} + \boldsymbol{R}$ and send their shares to the first party P_1.
4. P_1 reconstructs the secret of $\boldsymbol{X} + \boldsymbol{R}$, denoted by $\boldsymbol{w}$. Then P_1 computes $f(\boldsymbol{w})$ and generates a Σ'-sharing of $f(\boldsymbol{w})$, denoted by $\boldsymbol{W}$. Finally, P_1 distributes the shares of $\boldsymbol{W}$ to all parties.
5. All parties locally compute $\boldsymbol{Y} = \boldsymbol{W} - \boldsymbol{R}'$.

Lemma 3. *For all threshold-t $\mathcal{R}$-arithmetic secret sharing schemes Σ, Σ' and for all $\mathcal{R}$-module homomorphism $f : Z \to Z'$, Protocol* TRAN *securely computes $\mathcal{F}_{tran}$ in the $\mathcal{F}_{rand\text{-}sharing}$-hybrid model against a semi-honest adversary who controls t parties.*

We obtain the following theorem for our sharing transformation protocol.

Theorem 3. *Let n be the number of parties, t be the number of corrupted parties, and $k = (n-t+1)/2$. Let $\mathbb{F}$ be a finite field of size $2n$. Let ℓ_1, ℓ_2 be two positive integers. For all k tuples $\{(\Sigma_i, \Sigma'_i, f_i)\}_{i=1}^k$ and for all $\{\boldsymbol{X}_i\}_{i=1}^k$ such that Σ_i, Σ'_i are $\mathbb{F}$-linear secret sharing schemes with injective sharing functions and with share size ℓ_1, ℓ_2, f_i is a linear map from the secret space of Σ_i to that of Σ'_i, and $\boldsymbol{X}_i$ is a Σ_i-sharing held by all parties, there is an information-theoretic MPC protocol with semi-honest security against t corrupted parties that transforms $\boldsymbol{X}_i$ to a Σ'-sharing $\boldsymbol{Y}_i$ such that the secret of $\boldsymbol{Y}_i$ is equal to the result of applying f_i on the*

secret of $\boldsymbol{X}_i$. The cost of the protocol is $O(n^2 \cdot (\ell_1 + \ell_2))$ elements of preprocessing data and $O(n^2 \cdot (\ell_1 + \ell_2))$ elements of communication.

Remark 1. *We note that the preprocessing phase only prepares random degree-$(n-k)$ packed Shamir sharings and random degree-$(n-1)$ packed Shamir sharings of $\mathbf{0} \in \mathbb{F}^k$. With a variant of the protocol in [DN07] that prepares random degree-t Shamir sharings, the preprocessing can be done with communication complexity $O(\frac{n^3}{k} \cdot (\ell_1 + \ell_2))$ field elements for k sharing transformations.*

Improvement of Protocol Rand-Sharing for a Concrete Sharing Transformation. Our MPC protocol needs to perform the following sharing transformation. Let $\mathbb{F}$ be a large finite field. Recall that for a packed Shamir secret sharing scheme over $\mathbb{F}$, we use fixed $\alpha_1, \ldots, \alpha_n$ for shares of all parties. For two (potentially different) vectors of field elements $\mathtt{pos} = (p_1, \ldots, p_k)$ and $\mathtt{pos}' = (p'_1, \ldots, p'_k)$ such that they are disjoint with $\{\alpha_1, \ldots, \alpha_n\}$, all parties start with a degree-$(n-1)$ packed Shamir sharing $[\boldsymbol{x} \| \mathtt{pos}]_{n-1}$. They want to compute a degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{y} \| \mathtt{pos}']_{n-k}$ such that for all $i \in \{1, 2, \ldots, k\}$, y_i is equal to x_j for some j. In particular, it is possible that for two different indices i and i', y_i and $y_{i'}$ are equal to the same x_j.

Let $f : \mathbb{F}^k \to \mathbb{F}^k$ be an $\mathbb{F}$-linear map which satisfies that $\boldsymbol{y} = f(\boldsymbol{x})$. Then, we need to prepare a pair of random sharings $([\boldsymbol{r} \| \mathtt{pos}]_{n-1}, [f(\boldsymbol{r}) \| \mathtt{pos}']_{n-k})$. In particular, we can view it as a random sharing of the following $\mathbb{F}$-arithmetic secret sharing scheme Π:

- The secret space $Z = \mathbb{F}^k$.
- The share space $U = \mathbb{F}^2$.
- For a secret $\boldsymbol{x} \in Z$, the sharing of $\boldsymbol{x}$ is $([\boldsymbol{x} \| \mathtt{pos}]_{n-1}, [f(\boldsymbol{x}) \| \mathtt{pos}']_{n-k})$.
- For a sharing $\boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2)$, we view $\boldsymbol{X}_1$ as a degree-$(n-1)$ packed Shamir secret sharing scheme and reconstruct the secret $\boldsymbol{x}$.

We note that when directly using Rand-Sharing (instantiated by packed Shamir secret sharing schemes, see Section 4.4) to prepare a random Π-sharing, the communication complexity per sharing is $\frac{2n^2}{k}$ elements. This is because in Step 5 of Rand-Sharing, we need to compute $\tilde{\ell} = 2$ Σ'-sharings for the shares of each party, where $\tilde{\ell}$ is the share size of Π. These sharings are then reconstructed to their corresponding holders, which incurs $2n^2$ elements of communication for k sharings.

We observe that a random degree-d packed Shamir secret sharing scheme has the following nice properties:

- The shares of the first $d+1$ parties are uniformly random.
- The secret and the shares of the rest of $n-d-1$ parties are determined by the shares of the first $d+1$ parties.

When the secret of a random degree-d packed Shamir sharing is given,

- The shares of the first $d-k+1$ parties are uniformly random.
- The shares of the rest of $n-d+k-1$ parties are determined by the secret and the shares of the first $d+1$ parties.

Thus, when preparing a random degree-d packed Shamir sharing, we can view the shares of the first $d+1$ parties as the random tape for generating the whole sharing. In particular, the secret will be determined by the shares of the first $d+1$ parties. In Protocol Rand-Sharing, all parties invoke $\mathcal{F}_{\mathrm{rand}}$ to obtain random Σ-sharings for the random tapes. The secrets of these Σ-sharings can directly be viewed as the shares of the first $d+1$ parties. Thus, we can let the first $d+1$ parties reconstruct their shares in the preprocessing phase. In this way, we only need to reconstruct shares for the rest of $n-d-1$ parties in the online phase. Similarly, when preparing a random degree-d packed Shamir sharing for a given input $\boldsymbol{x}$, we can view the

shares of the first $d-k+1$ parties as the random tape for generating the whole sharing. We can let the first $d-k+1$ parties reconstruct their shares in the preprocessing phase. In this way we only need to reconstruct shares for the rest of $n-d+k-1$ parties in the online phase.

For the sharing transformation we are interested in, we need to prepare random sharings in the form of $([\boldsymbol{r}\|\mathtt{pos}]_{n-1}, [f(\boldsymbol{r})\|\mathtt{pos}']_{n-k})$. For the first sharing, it is a random degree-$(n-1)$ packed Shamir sharing. We can let all parties reconstruct their shares in the preprocessing phase. For the second sharing, it is a random degree-$(n-k)$ packed Shamir sharing given the secret $f(\boldsymbol{r})$. We can let $n-2k+1$ parties reconstruct their shares in the preprocessing phase. Thus, in the online phase, we only need to reconstruct the shares of $[f(\boldsymbol{r})\|\mathtt{pos}']_{n-k}$ of the last $2k-1$ parties.

Concretely, in the preprocessing phase, all parties prepare n random Σ-sharings. The i-th sharing is reconstructed to P_i, and P_i takes the secret as its shares of the degree-$(n-1)$ packed Shamir sharings in $\Pi_1, \ldots, \Pi_k$. (Recall that each time we prepare random sharings for k arithmetic secret sharing schemes. Here we assume that $\Pi_1, \ldots, \Pi_k$ are all in the form of Π constructed above.) Then, all parties prepare $n-2k+1$ random Σ-sharings. The i-th sharing is reconstructed to P_i, and P_i takes the secret as its shares of the degree-$(n-k)$ packed Shamir sharings in $\Pi_1, \ldots, \Pi_k$. After that, all parties prepare $2k-1$ Σ'-sharing of $\mathbf{0} \in \mathbb{F}^k$. Thus, the preprocessing data consists of $2n-2k+1$ random Σ sharings and $2k-1$ random Σ'-sharings of $\mathbf{0} \in \mathbb{F}^k$. Since each Σ-sharing is also reconstructed to a single party, the amount of preprocessing data is $2(2n-2k+1) \cdot n + (2k-1) \cdot n < 4n^2$ field elements.

In the online phase, all parties use the $2n-2k+1$ random Σ-sharings to compute Σ'-sharings of the shares of the last $2k-1$ parties of the degree-$(n-k)$ packed Shamir sharings in $\Pi_1, \ldots, \Pi_k$. Then they use random Σ'-sharings of $\mathbf{0}$ to mask these sharings and let the last $2k-1$ parties reconstruct their shares. Thus, the communication complexity is $(2k-1) \cdot n < 2k \cdot n$ field elements.

Thus, with this improvement, we obtain a protocol that prepares a random sharing for Π with $4n^2/k$ field elements of preprocessing data, and $2n$ field elements of communication. When we use Tran to perform the above sharing transformation, the amortized cost of Tran is $4n^2/k$ field elements of preprocessing data and $4n$ elements of communication.

5 Semi-Honest Protocol

In this section, we focus on the semi-honest security. We show how to use packed Shamir sharing schemes and $\mathcal{F}_{\text{tran}}$ (introduced in Section 4.5) to evaluate a circuit against a semi-honest adversary who controls t parties. Let $k = (n-t+1)/2$.

Recall that we use $[\boldsymbol{x}\|\mathtt{pos}]_d$ to represent a degree-d packed Shamir sharing of $\boldsymbol{x} \in \mathbb{F}^k$ stored at positions $\mathtt{pos} = (p_1, p_2, \ldots, p_k)$. Also recall that the shares of a degree-d packed Shamir sharing are at evaluation points $\alpha_1, \alpha_2, \ldots, \alpha_n$. Let $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_k)$ be k distinct elements in $\mathbb{F}$ that are different from $(\alpha_1, \alpha_2, \ldots, \alpha_n)$. We use $\boldsymbol{\beta}$ as the default positions for a degree-d packed Shamir sharing, and simply write $[\boldsymbol{x}]_d = [\boldsymbol{x}\|\boldsymbol{\beta}]_d$.

5.1 Circuit-Independent Preprocessing Phase

In the circuit-independent preprocessing phase, all parties need to prepare packed Beaver triples. For every group of k multiplication gates, all parties prepare a packed Beaver triple $([\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}, [\boldsymbol{c}]_{n-k})$ where $\boldsymbol{a}, \boldsymbol{b}$ are random vectors in $\mathbb{F}^k$ and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. We will use the technique of packed Beaver triples to compute multiplication gates in the online phase. The functionality $\mathcal{F}_{\text{prep}}$ for the circuit independent preprocessing phase appears in Functionality 6.

5.2 Online Computation Phase

Recall that for the field size it holds that $|\mathbb{F}| \geq |C| + n$, where $|C|$ is the circuit size. Let $\beta_1, \beta_2, \ldots, \beta_{|C|}$ be $|C|$ distinct field elements that are different from $\alpha_1, \alpha_2, \ldots, \alpha_n$. (Recall that we have already defined $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)$, which are used as the default positions for a packed Shamir sharing.) We associate the

Functionality 6: $\mathcal{F}_{\text{prep}}$

For every group of k multiplication gates:

1. $\mathcal{F}_{\text{prep}}$ receives the set of corrupted parties, denoted by $\mathcal{C}orr$.
2. $\mathcal{F}_{\text{prep}}$ receives from the adversary a set of shares $\{(a_j, b_j, c_j)\}_{j\in\mathcal{C}orr}$. $\mathcal{F}_{\text{prep}}$ samples two random vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^k$ and computes $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. Then $\mathcal{F}_{\text{prep}}$ computes three degree-$(n-k)$ packed Shamir sharings $[\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}, [\boldsymbol{c}]_{n-k}$ such that for all $P_j \in \mathcal{C}orr$, the j-th share of $([\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}, [\boldsymbol{c}]_{n-k})$ is (a_j, b_j, c_j).
3. $\mathcal{F}_{\text{prep}}$ distributes the shares of $([\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}, [\boldsymbol{c}]_{n-k})$ to honest parties.

field element β_i with the i-th gate in C. We will use β_i as the position to store the output value of the i-th gate in a degree-$(n-k)$ packed Shamir sharing.

Concretely, for each layer, gates that have the same type are divided into groups of size k. For each group of k gates, all parties will compute a degree-$(n-k)$ packed Shamir sharing such that the results are stored at the positions associated with these k gates respectively.

5.2.1 Input Layer

In the input layer, input gates are divided into groups of size k based on the input holders. For a group of k input gates belonging to the same client, suppose $\boldsymbol{x}$ are the inputs, and $\texttt{pos} = (p_1, p_2, \ldots, p_k)$ are the positions associated with these k gates. The client generates a random degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{x}\|\texttt{pos}]_{n-k}$ and distributes the shares to all parties.

5.2.2 Network Routing

In each intermediate layer, all gates are divided into groups of size k based on their types (i.e., multiplication gates or addition gates). For a group of k gates, all parties prepare two degree-$(n-k)$ packed Shamir sharings, one for the first inputs of all gates, and the other one for the second inputs of all gates.

Concretely, for a group k gates in the current layer, suppose $\boldsymbol{x}$ are the first inputs of these k gates, and $\boldsymbol{y}$ are the second inputs of these k gates. All parties will prepare two degree-$(n-k)$ packed Shamir sharings $[\boldsymbol{x}]_{n-k}$ and $[\boldsymbol{y}]_{n-k}$ stored at the default positions. The reason of choosing the default positions is to use the packed Beaver triples all parties have prepared in the preprocessing phase. Recall that the packed Beaver triples all use the default positions. In the following, we focus on inputs $\boldsymbol{x}$.

Collecting Secrets from Previous Layers. Let $x'_1, x'_2, \ldots, x'_{\ell_1}$ be the different values in $\boldsymbol{x}$ from previous layers. Let $c_1, c_2, \ldots, c_{\ell_2}$ be the constant values in $\boldsymbol{x}$. Then $\ell_1 + \ell_2 \leq k$. For each of the rest of $k - \ell_1 - \ell_2$ values in $\boldsymbol{x}$, it is the same as x'_i for some $i \in \{1, 2, \ldots, \ell_1\}$. In this step, we will prepare a degree-$(n-1)$ packed Shamir sharing that contains the secrets $x'_1, x'_2, \ldots, x'_{\ell_1}$ and $c_1, c_2, \ldots, c_{\ell_2}$.

Note that $\{x'_i\}_{i=1}^{\ell_1}$ are the output values of ℓ_1 different gates in previous layers. Let $p_1, p_2, \ldots, p_{\ell_1}$ be the positions associated with these ℓ_1 gates. We choose another arbitrary $k - \ell_1$ different positions $p_{\ell_1+1}, \ldots, p_k$ which are also different from $\alpha_1, \alpha_2, \ldots, \alpha_n$, and set $\texttt{pos} = (p_1, p_2, \ldots, p_k)$. Suppose for all $1 \leq i \leq \ell_1$, $[\boldsymbol{x}^{(i)}\|\texttt{pos}^{(i)}]_{n-k}$ is the degree-$(n-k)$ packed Shamir sharing from some previous layer that contains the secret x'_i stored at position p_i.

Let $\boldsymbol{e}_i$ be the i-th unit vector in $\mathbb{F}^k$ (i.e., only the i-th term is 1 and all other terms are 0). All parties locally compute a degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{e}_i\|\texttt{pos}]_{k-1}$. Let $\boldsymbol{x}' = (x'_1, \ldots, x'_{\ell_1}, c_1, \ldots, c_{\ell_2}, 0, \ldots, 0)$ be

a vector in $\mathbb{F}^k$. Then all parties locally compute

$$\sum_{i=1}^{\ell_1}[\boldsymbol{e}_i\|\mathtt{pos}]_{k-1}\cdot[\boldsymbol{x}^{(i)}\|\mathtt{pos}^{(i)}]_{n-k}+\sum_{i=1}^{\ell_2}c_i\cdot[\boldsymbol{e}_{\ell_1+i}\|\mathtt{pos}]_{k-1}.$$

We show that this is a degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{x}'$ stored at positions $\mathtt{pos}$. It is clear that the resulting sharing has degree $n-1$. We only need to show the following three points:

- For all $1\le j\le \ell_1$, the secret stored at position p_j is equal to x'_j.
- For all $\ell_1+1\le j\le \ell_1+\ell_2$, the secret stored at position p_j is equal to $c_{j-\ell_1}$.
- For all $\ell_1+\ell_2+1\le j\le k$, the secret stored at position p_j is equal to 0.

For all $1\le i\le \ell_1+\ell_2$, let f_i be the polynomial corresponding to $[\boldsymbol{e}_i\|\mathtt{pos}]_{k-1}$. For all $1\le i\le \ell_1$, let g_i be the polynomial corresponding to $[\boldsymbol{x}^{(i)}\|\mathtt{pos}^{(i)}]_{n-k}$. Then the polynomial corresponding to the resulting sharing is $h=\sum_{i=1}^{\ell_1}f_i\cdot g_i+\sum_{i=1}^{\ell_2}c_i\cdot f_{\ell_1+i}$.

Note that f_i satisfies that $f_i(p_i)=1$ and $f_i(p_j)=0$ for all $j\neq i$. And g_i satisfies that $g_i(p_i)=x'_i$. Therefore, for all $1\le j\le \ell_1$,

$$h(p_j)=\sum_{i=1}^{\ell_1}f_i(p_j)\cdot g_i(p_j)+\sum_{i=1}^{\ell_2}c_i\cdot f_{\ell_1+i}(p_j)=f_j(p_j)\cdot g_j(p_j)=x'_j.$$

For all $\ell_1+1\le j\le \ell_2$,

$$h(p_j)=\sum_{i=1}^{\ell_1}f_i(p_j)\cdot g_i(p_j)+\sum_{i=1}^{\ell_2}c_i\cdot f_{\ell_1+i}(p_j)=c_{j-\ell_1}\cdot f_j(p_j)=c_{j-\ell_1}.$$

For all $\ell_1+\ell_2+1\le j\le k$,

$$h(p_j)=\sum_{i=1}^{\ell_1}f_i(p_j)\cdot g_i(p_j)+\sum_{i=1}^{\ell_2}c_i\cdot f_{\ell_1+i}(p_j)=0.$$

Thus, the resulting sharing is a degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{x}'$ stored at positions $\mathtt{pos}$, denoted by $[\boldsymbol{x}'\|\mathtt{pos}]_{n-1}$.

Transforming to the Desired Sharing. Now all parties hold a degree-$(n-1)$ packed Shamir sharing $[\boldsymbol{x}'\|\mathtt{pos}]_{n-1}$. Recall that $\boldsymbol{x}'$ contains all different values in $\boldsymbol{x}$ from previous layers and all constant values. For each of the rest of values in $\boldsymbol{x}$, it is the same as x'_i for some $i\in\{1,2,\ldots,\ell_1\}$. Then there is a linear map $f:\mathbb{F}^k\to\mathbb{F}^k$ such that $\boldsymbol{x}=f(\boldsymbol{x}')$. Recall that $\boldsymbol{\beta}=(\beta_1,\ldots,\beta_k)$ are the default positions. Let Σ be the degree-$(n-1)$ packed Shamir secret sharing scheme that stores secrets at positions $\mathtt{pos}$. Let Σ' be the degree-$(n-k)$ packed Shamir secret sharing scheme that stores secrets at positions $\boldsymbol{\beta}$. Then $[\boldsymbol{x}'\|\mathtt{pos}]_{n-1}$ is a Σ-sharing, and the sharing we want to prepare, $[\boldsymbol{x}]_{n-k}=[\boldsymbol{x}\|\boldsymbol{\beta}]_{n-k}$, is a Σ'-sharing with $\boldsymbol{x}=f(\boldsymbol{x}')$.

All parties invoke $\mathcal{F}_{\text{tran}}$ with (Σ,Σ',f) and $[\boldsymbol{x}'\|\mathtt{pos}]_{n-1}$, and obtain $[\boldsymbol{x}]_{n-k}$.

Summary of Network Routing. We describe the protocol Network of preparing an input degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{x}]_{n-k}$ in Protocol 7.

5.2.3 Evaluating Addition Gates and Multiplication Gates

Addition Gates. For a group of k addition gates, recall that all parties have prepared two degree-$(n-k)$ packed Shamir sharings $[\boldsymbol{x}]_{n-k},[\boldsymbol{y}]_{n-k}$ where $\boldsymbol{x}$ are the first inputs of these k gates, and $\boldsymbol{y}$ are the second inputs of these k gates. The description of Add appears in Protocol 8. Note that in Step 3 of Protocol Add, we use the fact that a degree-$(n-k)$ packed Shamir sharing can be viewed as a degree-$(n-1)$ packed Shamir sharing.

Protocol 7: NETWORK

1. Suppose all parties want to prepare a degree-$(n-k)$ packed Shamir sharing of $\boldsymbol{x}$ stored at the default positions $\boldsymbol{\beta}$.
2. Let $x'_1, x'_2, \ldots, x'_{\ell_1}$ be the different wire values in $\boldsymbol{x}$ from previous layers. Let $c_1, c_2, \ldots, c_{\ell_2}$ be the constant values in $\boldsymbol{x}$. Let $\boldsymbol{x}' = (x'_1, \ldots, x'_{\ell_1}, c_1, \ldots, c_{\ell_2}, 0, \ldots, 0) \in \mathbb{F}^k$.
3. For all $1 \le i \le \ell_1$, let $[\boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]_{n-k}$ be the degree-$(n-k)$ packed Shamir sharing from some previous layer that contains the secret x'_i stored at position p_i. Let $p_{\ell_1+1}, \ldots, p_k$ be the first $k-\ell_1$ distinct positions that are different from $p_1, \ldots, p_{\ell_1}$ and $\alpha_1, \ldots, \alpha_n$. Let $\mathtt{pos} = (p_1, \ldots, p_k)$.
4. Let $\boldsymbol{e}_i$ be the i-th unit vector in $\mathbb{F}^k$ (i.e., only the i-th term is 1 and all other terms are 0). All parties locally compute a degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{e}_i \| \mathtt{pos}]_{k-1}$.
5. All parties locally compute
$$[\boldsymbol{x}' \| \mathtt{pos}]_{n-1} = \sum_{i=1}^{\ell_1} [\boldsymbol{e}_i \| \mathtt{pos}]_{k-1} \cdot [\boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]_{n-k} + \sum_{i=1}^{\ell_2} c_i \cdot [\boldsymbol{e}_{\ell_1+i} \| \mathtt{pos}]_{k-1}.$$
6. Let $f : \mathbb{F}^k \to \mathbb{F}^k$ be a linear map such that $\boldsymbol{x} = f(\boldsymbol{x}')$. Let Σ be the degree-$(n-1)$ packed Shamir secret sharing scheme that stores secrets at positions $\mathtt{pos}$. Let Σ' be the degree-$(n-k)$ packed Shamir secret sharing scheme that stores secrets at positions $\boldsymbol{\beta}$.

   All parties invoke $\mathcal{F}_{\text{tran}}$ with (Σ, Σ', f) and $[\boldsymbol{x}' \| \mathtt{pos}]_{n-1}$, and output $[\boldsymbol{x}]_{n-k}$.

Protocol 8: ADD

1. Suppose $[\boldsymbol{x}]_{n-k}, [\boldsymbol{y}]_{n-k}$ are the input packed Shamir sharings of the addition gates.
2. All parties locally compute $[\boldsymbol{z}]_{n-k} = [\boldsymbol{x}]_{n-k} + [\boldsymbol{y}]_{n-k}$.
3. Suppose $\mathtt{pos} = (p_1, p_2, \ldots, p_k)$ are the positions associated with these k addition gates. Recall that $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)$ are the default positions. Let Σ be the degree-$(n-1)$ packed Shamir secret sharing scheme that stores secrets at positions $\boldsymbol{\beta}$. Let Σ' be the degree-$(n-k)$ packed Shamir secret sharing scheme that stores secrets at positions $\mathtt{pos}$. Let $I : \mathbb{F}^k \to \mathbb{F}^k$ be the identity map.

   All parties invoke $\mathcal{F}_{\text{tran}}$ with (Σ, Σ', I) and $[\boldsymbol{z}]_{n-k}$, and output $[\boldsymbol{z} \| \mathtt{pos}]_{n-k}$.

Multiplication Gates. For a group of k multiplication gates, recall that all parties have prepared two degree-$(n-k)$ packed Shamir sharings $[\boldsymbol{x}]_{n-k}, [\boldsymbol{y}]_{n-k}$ where $\boldsymbol{x}$ are the first inputs of these k gates, and $\boldsymbol{y}$ are the second inputs of these k gates. Let $([\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}, [\boldsymbol{c}]_{n-k})$ be the packed Beaver triple prepared in the preprocessing phase. We will use the technique of packed Beaver triples to evaluate multiplication gates. The description of MULT appears in Protocol 9.

5.2.4 Output Layer

In the output layer, output gates are divided into groups of size k based on the output receivers. For a group of k output gates belonging to the same client, suppose $\boldsymbol{x}$ are the inputs. All parties invoke the protocol

Protocol 9: MULT

1. Suppose $[\boldsymbol{x}]_{n-k}, [\boldsymbol{y}]_{n-k}$ are the input packed Shamir sharings of the multiplication gates. All parties will use a fresh random packed Beaver triple $([\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}, [\boldsymbol{c}]_{n-k})$ prepared in the preprocessing phase.
2. All parties locally compute $[\boldsymbol{x}+\boldsymbol{a}]_{n-k} = [\boldsymbol{x}]_{n-k} + [\boldsymbol{a}]_{n-k}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{n-k} = [\boldsymbol{y}]_{n-k} + [\boldsymbol{b}]_{n-k}$.
3. The first party P_1 collects the whole sharings $[\boldsymbol{x}+\boldsymbol{a}]_{n-k}, [\boldsymbol{y}+\boldsymbol{b}]_{n-k}$ and reconstructs the secrets $\boldsymbol{x}+\boldsymbol{a}, \boldsymbol{y}+\boldsymbol{b}$. Then, P_1 computes the sharings $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ and distributes the shares to other parties.
4. All parties locally compute
$$[\boldsymbol{z}]_{n-1} := [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{b}]_{n-k} - [\boldsymbol{y}+\boldsymbol{b}]_{k-1} \cdot [\boldsymbol{a}]_{n-k} + [\boldsymbol{c}]_{n-k}.$$
5. Suppose $\mathtt{pos} = (p_1, p_2, \ldots, p_k)$ are the positions associated with these k multiplication gates. Recall that $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)$ are the default positions. Let Σ be the degree-$(n-1)$ packed Shamir secret sharing scheme that stores secrets at positions $\boldsymbol{\beta}$. Let Σ' be the degree-$(n-k)$ packed Shamir secret sharing scheme that stores secrets at positions $\mathtt{pos}$. Let $I : \mathbb{F}^k \to \mathbb{F}^k$ be the identity map.

   All parties invoke $\mathcal{F}_{\text{tran}}$ with (Σ, Σ', I) and $[\boldsymbol{z}]_{n-1}$, and output $[\boldsymbol{z} \| \mathtt{pos}]_{n-k}$.

NETWORK to prepare $[\boldsymbol{x}]_{n-k}$. Then, all parties send their shares to the client to allow him to reconstruct the output.

5.2.5 Main Protocol

Given the above protocols the main semi-honest protocol follows in a straightforward way. The ideal functionality $\mathcal{F}_{\text{main-semi}}$ appears in Functionality 10 and the main protocol is introduced in Protocol 11.

Functionality 10: $\mathcal{F}_{\text{main-semi}}$

1. $\mathcal{F}_{\text{main-semi}}$ receives the input from all clients. Let x denote the input and C denote the circuit.
2. $\mathcal{F}_{\text{main-semi}}$ computes $C(x)$ and distributes the output to all clients.

Lemma 4. *Protocol* MAIN-SEMI *securely computes $\mathcal{F}_{main\text{-}semi}$ in the $(\mathcal{F}_{prep}, \mathcal{F}_{tran})$-hybrid model against a semi-honest adversary who controls t parties.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}orr$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties.

The correctness of MAIN-SEMI follows from (1) the correctness of the protocol NETWORK for network routing, and (2) the correctness of the protocols ADD and MULT for addition gates and multiplication gates respectively.

We now describe the construction of the simulator $\mathcal{S}$.

1. In Step 1, $\mathcal{S}$ emulates the functionality $\mathcal{F}_{\text{prep}}$ and receives the shares of corrupted parties.

Protocol 11: MAIN

1. **Preprocessing Phase**. All parties invoke $\mathcal{F}_{\text{prep}}$ to prepare enough packed Beaver triples in the form of $([\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}, [\boldsymbol{c}]_{n-k})$, where $\boldsymbol{a}, \boldsymbol{b}$ are random vectors in $\mathbb{F}^k$ and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$.

2. **Initialization**. Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be distinct field elements in $\mathbb{F}$ which are used for the shares of all parties in packed Shamir secret sharing schemes. Let $\beta_1, \beta_2, \ldots, \beta_{|C|}$ be $|C|$ distinct field elements that are different from $\alpha_1, \alpha_2, \ldots, \alpha_n$. Let $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_k)$ be the default positions for the packed Shamir secret sharing schemes. We associate the field element β_i with the i-th gate in C.

3. **Input Phase**. Let $\mathsf{Client}_1, \mathsf{Client}_2, \ldots, \mathsf{Client}_\mathsf{c}$ denote the clients who provide inputs. All input gates are divided into groups of size k based on the input holders. For every group of k input gates of Client_i, suppose $\boldsymbol{x}$ are the inputs, and $\mathtt{pos} = (p_1, p_2, \ldots, p_k)$ are the positions associated with these k gates. Client_i generates a random degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{x} \| \mathtt{pos}]_{n-k}$ and distributes the shares to all parties.

4. **Evaluation Phase**. All parties evaluate the circuit layer by layer as follows:

   (a) For the current layer, all gates are divided into groups of size k based on their types (i.e., multiplication gates or addition gates). For each group of k gates, let $\boldsymbol{x}$ be the first inputs of all gates, and $\boldsymbol{y}$ the second inputs of all gates. All parties invoke NETWORK to prepare $[\boldsymbol{x}]_{n-k}$ and $[\boldsymbol{y}]_{n-k}$.

   (b) For each group of k gates, let $\mathtt{pos}$ be the positions associated with these k gates.

   - If they are addition gates, all parties invoke ADD on $([\boldsymbol{x}]_{n-k}, [\boldsymbol{y}]_{n-k})$, and obtain $[\boldsymbol{z} \| \mathtt{pos}]_{n-k}$.
   - If they are multiplication gates, all parties invoke MULT on $([\boldsymbol{x}]_{n-k}, [\boldsymbol{y}]_{n-k})$ with a fresh packed Beaver triple $([\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}, [\boldsymbol{c}]_{n-k})$, and obtain $[\boldsymbol{z} \| \mathtt{pos}]_{n-k}$.

5. **Output Phase**. All output gates are divided into groups of size k based on the output receiver. For every group of k output gates of Client_i, suppose $\boldsymbol{x}$ are the inputs. All parties invoke the protocol NETWORK to prepare $[\boldsymbol{x}]_{n-k}$. Then, all parties send their shares to Client_i to let him reconstruct the result $\boldsymbol{x}$.

2. In Step 2, $\mathcal{S}$ follows the protocol.

3. In Step 3, for every group of k input gates of Client_i, if Client_i is honest, $\mathcal{S}$ samples random elements as the shares of corrupted parties. If Client_i is corrupted, $\mathcal{S}$ learns the inputs $\boldsymbol{x}$ and the shares of corrupted parties (since $\mathcal{S}$ can access to the inputs and random tapes of corrupted clients and corrupted parties). Note that in both cases, $\mathcal{S}$ learns the shares of corrupted parties.

4. In Step 4.(a), $\mathcal{S}$ simulates NETWORK. Note that NETWORK only involves local computation and an invocation of $\mathcal{F}_{\text{tran}}$. $\mathcal{S}$ follows the protocol in NETWORK and computes the shares of corrupted parties. Then $\mathcal{S}$ emulates $\mathcal{F}_{\text{tran}}$ and receives the shares of corrupted parties. At the end of NETWORK, $\mathcal{S}$ learns the shares of $[\boldsymbol{x}]_{n-k}$ and $[\boldsymbol{y}]_{n-k}$ held by corrupted parties.

   In Step 4.(b), for each group of addition gates with input sharings $[\boldsymbol{x}]_{n-k}$ and $[\boldsymbol{y}]_{n-k}$, $\mathcal{S}$ simulates ADD. Note that ADD only involves local computation and an invocation of $\mathcal{F}_{\text{tran}}$. $\mathcal{S}$ follows the protocol in ADD and computes the shares of corrupted parties. Then $\mathcal{S}$ emulates $\mathcal{F}_{\text{tran}}$ and receives the shares of corrupted parties. At the end of ADD, $\mathcal{S}$ learns the shares of $[\boldsymbol{z}]_{n-k}$ held by corrupted parties.

   In Step 4.(b), for each group of multiplication gates with input sharings $[\boldsymbol{x}]_{n-k}$ and $[\boldsymbol{y}]_{n-k}$, $\mathcal{S}$ simulates

MULT as follows:

(a) The protocol MULT consumes a fresh random packed Beaver triple $([\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}, [\boldsymbol{c}]_{n-k})$. Recall that $\mathcal{S}$ learns the shares of these sharings held by corrupted parties when emulating the functionality $\mathcal{F}_{\text{prep}}$.

(b) In Step 2 of MULT, $\mathcal{S}$ computes the shares of $[\boldsymbol{x}+\boldsymbol{a}]_{n-k}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{n-k}$ held by corrupted parties. Then $\mathcal{S}$ generates two random degree-$(n-k)$ packed Shamir sharings as $[\boldsymbol{x}+\boldsymbol{a}]_{n-k}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{n-k}$ based on the shares of corrupted parties.

(c) Since the whole sharings $[\boldsymbol{x}+\boldsymbol{a}]_{n-k}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{n-k}$ have been generated by $\mathcal{S}$, $\mathcal{S}$ honestly follows the protocol in Step 3 of MULT. $\mathcal{S}$ learns the shares of $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ of corrupted parties at the end of this step.

(d) In Step 4 of MULT, $\mathcal{S}$ computes the shares of $[\boldsymbol{z}]_{n-1}$ held by corrupted parties.

(e) In Step 5 of MULT, $\mathcal{S}$ emulates $\mathcal{F}_{\text{tran}}$ and receives the shares of corrupted parties. At the end of this step, $\mathcal{S}$ learns the shares of $[\boldsymbol{z}]_{n-k}$ held by corrupted parties.

5. In Step 5, $\mathcal{S}$ simulates NETWORK in the same way as that for Step 4.(a). $\mathcal{S}$ invokes $\mathcal{F}_{\text{main-semi}}$ with the input of corrupted clients, and receives the output of corrupted clients. For each group of output gates of Client_i, $\mathcal{S}$ has learned the shares of the input sharing $[\boldsymbol{x}]_{n-k}$ held by corrupted parties. If Client_i is honest, $\mathcal{S}$ does nothing. If Client_i is corrupted, $\mathcal{S}$ also learns the secret $\boldsymbol{x}$. $\mathcal{S}$ uses $\boldsymbol{x}$ and the shares of $[\boldsymbol{x}]_{n-k}$ held by corrupted parties, which are together $k + (n-t) = n-k+1$ values (recall that $t = n-2k+1$), to reconstruct the whole sharing $[\boldsymbol{x}]_{n-k}$ and finally sends the shares of $[\boldsymbol{x}]_{n-k}$ of honest parties to Client_i.

This completes the description of $\mathcal{S}$.

We show that $\mathcal{S}$ perfectly simulates the behaviors of honest parties. It is sufficient to focus on the places where honest parties and clients need to communicate with corrupted parties and clients:

- In Step 3, honest clients need to share its input. In the real world, an honest client will generate a random degree-$(n-k)$ packed Shamir sharing and distribute the shares to other parties. By the property of the degree-$(n-k)$ packed Shamir secret sharing scheme, the shares of corrupted parties are uniformly random. Therefore $\mathcal{S}$ perfectly simulates the behaviors of honest clients.

- In Step 4.(b), honest parties need to communicate with corrupted parties when running MULT. In Step 3 of MULT, all parties need to send their shares of $[\boldsymbol{x}+\boldsymbol{a}]_{n-k}, [\boldsymbol{y}+\boldsymbol{b}]_{n-k}$ to P_1. And P_i needs to distribute $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ to all parties.

  Note that $[\boldsymbol{x}+\boldsymbol{a}]_{n-k} = [\boldsymbol{x}]_{n-k} + [\boldsymbol{a}]_{n-k}$, and $[\boldsymbol{a}]_{n-k}$ is a random degree-$(n-k)$ packed Shamir sharing given the shares of corrupted parties. Therefore $[\boldsymbol{x}+\boldsymbol{a}]_{n-k}$ is also a random degree-$(n-k)$ packed Shamir sharing given the shares of corrupted parties. Similarly, $[\boldsymbol{y}+\boldsymbol{b}]_{n-k}$ is a random degree-$(n-k)$ packed Shamir sharing given the shares of corrupted parties. Also note that $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ are fully determined by the secrets $\boldsymbol{x}+\boldsymbol{a}$ and $\boldsymbol{y}+\boldsymbol{b}$.

  In the ideal world, $\mathcal{S}$ generates two random degree-$(n-k)$ packed Shamir sharings as $[\boldsymbol{x}+\boldsymbol{a}]_{n-k}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{n-k}$ based on the shares of corrupted parties. Then, the distribution of these two random degree-$(n-k)$ packed Shamir sharings is identical to that in the real world. Note that it also implies that the distribution of the two secrets $\boldsymbol{x}+\boldsymbol{a}$ and $\boldsymbol{y}+\boldsymbol{b}$ is identical to that in the real world. After sampling $[\boldsymbol{x}+\boldsymbol{a}]_{n-k}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{n-k}$, $\mathcal{S}$ honestly follows Step 3 of MULT. Thus, $\mathcal{S}$ perfectly simulates the behaviors of honest parties.

- Finally, in Step 5, for each group of output gates of a corrupted client, honest parties need to send their shares of the sharing associated with these gates to corrupted clients. Note that a degree-$(n-k)$ packed Shamir sharing is determined by its secret and the shares of corrupted parties. Since $\mathcal{S}$ learns the output of corrupted clients from $\mathcal{F}_{\text{main-semi}}$ and the shares of corrupted parties, $\mathcal{S}$ can compute the shares of honest parties and perfectly simulate the behaviors of honest parties.

□

Analysis of the Communication Complexity. We assume that the number of multiplication gates is the same as the number of addition gates. We also assume that the number of input gates and output gates is much smaller than the number of addition gates and multiplication gates. Let C denote the circuit and DEPTH denote the circuit depth. We use I to denote the input size, G to denote the number of gates, and O to denote the output size. Then $|C| \geq I + G + O$.

- Cost of the Circuit Independent Preprocessing Phase: The size of the preprocessing data per packed Beaver triple is $3n$ field elements. Therefore, the total size of the preprocessing data for multiplication gates is $(\frac{G}{2k} + \mathsf{Depth}) \cdot 3n$. Here $\frac{G}{2}$ is the estimated number of multiplication gates, and $\frac{G}{2k} + \mathsf{Depth}$ is the number of packed Beaver triples required for multiplication gates. The reason of adding Depth is because the multiplication gates are grouped layer by layer. In each layer, if there are m multiplication gates, then we need to prepare $\lceil m/k \rceil < m/k + 1$ packed Beaver triples.

- Cost of the Online Phase:

  - Input Phase: For every group of k input gates, the communication complexity is n field elements. Let I denote the size of input. Then the cost of the input phase is $(\frac{I}{k} + \mathsf{c}) \cdot n$ field elements of communication. The reason of adding c is because the input gates are grouped based on the input holders.
  - Evaluation Phase: For every group of k (addition or multiplication) gates, all parties invoke NETWORK to prepare the two input sharings. Each invocation of NETWORK involves one invocation of $\mathcal{F}_{\text{tran}}$.

    For addition gates, each invocation of ADD involves one invocation of $\mathcal{F}_{\text{tran}}$.

    For multiplication gates, each invocation of MULT involves $4n$ elements of communication and one invocation of $\mathcal{F}_{\text{tran}}$.

    Thus, the evaluation phase cost $(\frac{G}{2k} + \mathsf{Depth}) \cdot 4n$ elements of communication and $(\frac{G}{k} + 2 \cdot \mathsf{Depth}) \cdot 3$ invocations of $\mathcal{F}_{\text{tran}}$. When using TRAN to instantiate $\mathcal{F}_{\text{tran}}$, $(\frac{G}{k} + 2 \cdot \mathsf{Depth}) \cdot 3$ invocations of $\mathcal{F}_{\text{tran}}$ requires $12(\frac{G}{k} + 2 \cdot \mathsf{Depth}) \cdot \frac{n^2}{k}$ field elements of preprocessing data, and $12(\frac{G}{k} + 2 \cdot \mathsf{Depth}) \cdot n$ field elements of communication.

    In total, the cost of the evaluation phase is $12(\frac{G}{k} + 2 \cdot \mathsf{Depth}) \cdot \frac{n^2}{k}$ field elements of preprocessing data, and $14(\frac{G}{k} + 2 \cdot \mathsf{Depth}) \cdot n$ field elements of communication.
  - Output Phase: For every group of k output gates, all parties invoke NETWORK to prepare the input sharing. Then all parties send their shares to the client who should receive the result. Recall that each invocation of NETWORK involves one invocation of $\mathcal{F}_{\text{tran}}$. Thus, the output phase cost $(\frac{O}{k} + \mathsf{c}) \cdot n$ field elements of communication and $\frac{O}{k} + \mathsf{c}$ invocations of $\mathcal{F}_{\text{tran}}$.

    With a similar analysis, the cost of the output phase is $4(\frac{O}{k} + \mathsf{c}) \cdot \frac{n^2}{k}$ field elements of preprocessing data, and $5(\frac{O}{k} + \mathsf{c}) \cdot n$ field elements of communication.

  In summary, the total cost of the online phase is $(\frac{12G}{k} + \frac{4O}{k} + 24 \cdot \mathsf{Depth} + 4\mathsf{c}) \cdot \frac{n^2}{k}$ elements of preprocessing data, and $(\frac{I}{k} + \frac{14G}{k} + \frac{5O}{k} + 28 \cdot \mathsf{Depth} + 6\mathsf{c}) \cdot n$ elements of communication.

Thus, our semi-honest protocol requires $12|C| \cdot \frac{n^2}{k^2} + O((\mathsf{Depth} + \mathsf{c}) \cdot \frac{n^2}{k})$ elements of preprocessing data, and $14|C| \cdot \frac{n}{k} + O((\mathsf{Depth} + \mathsf{c}) \cdot n)$ elements of communication.

Theorem 4. *In the client-server model, let c denote the number of clients, n denote the number of parties (servers), and t denote the number of corrupted parties (servers). Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq |C| + n$. For an arithmetic circuit C over $\mathbb{F}$, there exists an information-theoretic MPC protocol in the preprocessing model which securely computes the arithmetic circuit C in the presence of a semi-honest adversary controlling up to c clients and t parties. The cost of the protocol is $O(|C| \cdot \frac{n^2}{k^2} + (\mathsf{Depth} + \mathsf{c}) \cdot \frac{n^2}{k})$ field elements of preprocessing data and $O(|C| \cdot \frac{n}{k} + (\mathsf{Depth} + \mathsf{c}) \cdot n)$ field elements of communication, where $k = \frac{n-t+1}{2}$ and Depth is the circuit depth.*

6 Maliciously Secure Protocol

In this section, we discuss how to achieve malicious security. One difficulty is that corrupted parties can change the secret of a degree-$(n-k)$ packed Shamir sharing by changing their own shares locally. We extend the idea of using information-theoretic MACs introduced in [BDOZ11, DPSZ12] to authenticate packed Shamir sharings.

Concretely, all parties will prepare a random degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{\gamma}]_{n-k}$, where $\boldsymbol{\gamma} = (\gamma, \gamma, \ldots, \gamma) \in \mathbb{F}^k$ and γ is a random field element in $\mathbb{F}$, in the preprocessing phase (using the default positions). During the computation, a degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{x} \| \mathtt{pos}]_{n-k}$ is authenticated by computing a degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{\gamma} * \boldsymbol{x} \| \mathtt{pos}]_{n-k}$. We use $[\![\boldsymbol{x} \| \mathtt{pos}]\!]_{n-k}$ to denote the pair $([\boldsymbol{x} \| \mathtt{pos}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{x} \| \mathtt{pos}]_{n-k})$. Note that if corrupted parties change the secret $\boldsymbol{x}$ to $\boldsymbol{x}'$, they also need to change the secret $\boldsymbol{\gamma} * \boldsymbol{x}$ to $\boldsymbol{\gamma} * \boldsymbol{x}'$. However, since $\boldsymbol{\gamma}$ is a uniform vector in $\mathbb{F}^k$, the probability of a success attack is at most $1/|\mathbb{F}|$. When the field size is large enough, we can detect such an attack with overwhelming probability. Therefore in the following, we assume $|\mathbb{F}| \geq 2^\kappa$, where κ is the security parameter.

Recall that t is the number of corrupted parties, and $k = (n-t+1)/2$. Then the number of honest parties is $n - t = 2k - 1$. We focus on $\frac{n-1}{3} \leq t \leq n-1$. Then k satisfies that $2k - 2 \leq n - k$. The benefit is that for any $2k-1$ field elements in $\mathbb{F}$, there is a degree-$(n-k)$ packed Shamir sharing such that the shares of honest parties are these $2k-1$ elements. In this way, we can transform any attack that adds errors to the shares of honest parties to an attack that adds errors to the secrets.

6.1 Performing Sharing Transformation with Malicious Security

We will only focus on the sharing transformation we use in our semi-honest protocol: For two (potentially different) vectors of field elements $\mathtt{pos} = (p_1, \ldots, p_k)$ and $\mathtt{pos}' = (p'_1, \ldots, p'_k)$ such that they are disjoint with $\{\alpha_1, \ldots, \alpha_n\}$, all parties start with a degree-$(n-1)$ packed Shamir sharing $[\boldsymbol{x} \| \mathtt{pos}]_{n-1}$. They want to compute a degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{y} \| \mathtt{pos}']_{n-k}$ such that for all $i \in \{1, 2, \ldots, k\}$, y_i is equal to x_j for some j.

We first construct a protocol that allows all parties to prepare a pair of random sharings for the above transformation. It follows from Protocol Rand-Sharing with the concrete improvement discussed in Section 4.5.

6.1.1 Preparing Random Sharings for Sharing Transformations

Let $\Pi = \Pi(\mathtt{pos}, \mathtt{pos}', f)$ be the $\mathbb{F}$-linear secret sharing scheme defined as follows:

- The secret space $Z = \mathbb{F}^k$.
- The share space $U = \mathbb{F}^2$.
- For a secret $\boldsymbol{x} \in Z$, the sharing of $\boldsymbol{x}$ is $([\boldsymbol{x} \| \mathtt{pos}]_{n-1}, [f(\boldsymbol{x}) \| \mathtt{pos}']_{n-k})$.
- For a sharing $\boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2)$, we view $\boldsymbol{X}_1$ as a degree-$(n-1)$ packed Shamir secret sharing scheme and reconstruct the secret $\boldsymbol{x}$.

Our goal is to prepare a pair of random sharings $([\boldsymbol{r} \| \mathtt{pos}]_{n-1}, [f(\boldsymbol{r}) \| \mathtt{pos}']_{n-k})$ where $\boldsymbol{r}$ is a random vector in $\mathbb{F}^k$. In the malicious security setting, we allow an adversary to add a constant vector to the secret of the second sharing. We summarize the functionality $\mathcal{F}_{\text{rand-sharing-mal}}$ in Functionality 12.

Let $\Pi_1, \Pi_2, \ldots, \Pi_k$ be k $\mathbb{F}$-linear secret sharing schemes in the above form. We will prepare k random sharings, one for each secret sharing scheme. Our protocol will use $\mathcal{F}_{\text{rand}}$ to prepare random degree-$(n-k)$ packed Shamir sharings, and $\mathcal{F}_{\text{randZero}}$ to prepare random degree-$(n-1)$ packed Shamir sharings of $\mathbf{0}$. The description of Protocol Rand-Sharing-Mal appears in Protocol 13.

Lemma 5. *For all $k \leq (n+2)/3$, Protocol* Rand-Sharing-Mal *securely computes $\mathcal{F}_{\text{rand-sharing-mal}}$ in the $\{\mathcal{F}_{\text{rand}}, \mathcal{F}_{\text{randZero}}\}$-hybrid model against a fully malicious adversary who controls $t = n - 2k + 1$ parties.*

Functionality 12: $\mathcal{F}_{\text{rand-sharing-mal}}$

1. $\mathcal{F}_{\text{rand-sharing-mal}}$ receives the set of corrupted parties, denoted by $\mathcal{C}orr$. $\mathcal{F}_{\text{rand-sharing-mal}}$ also receives $\Pi = \Pi(\mathtt{pos}, \mathtt{pos}', f)$.
2. $\mathcal{F}_{\text{rand-sharing-mal}}$ receives from the adversary a set of shares $\{(u_j, v_j)\}_{j\in\mathcal{C}orr}$, and a constant vector $\boldsymbol{d} \in \mathbb{F}^k$.
3. $\mathcal{F}_{\text{rand-sharing-mal}}$ samples a random vector $\boldsymbol{r} \in \mathbb{F}^k$ and computes $f(\boldsymbol{r}) + \boldsymbol{d}$.
4. $\mathcal{F}_{\text{rand-sharing-mal}}$ samples a pair of random sharings $([\boldsymbol{r}\|\mathtt{pos}]_{n-1}, [f(\boldsymbol{r}) + \boldsymbol{d}\|\mathtt{pos}']_{n-k})$ such that for all $P_j \in \mathcal{C}orr$, the j-th share of $([\boldsymbol{r}\|\mathtt{pos}]_{n-1}, [f(\boldsymbol{r}) + \boldsymbol{d}\|\mathtt{pos}']_{n-k})$ is (u_j, v_j).
5. $\mathcal{F}_{\text{rand-sharing-mal}}$ distributes the shares of $([\boldsymbol{r}\|\mathtt{pos}]_{n-1}, [f(\boldsymbol{r}) + \boldsymbol{d}\|\mathtt{pos}']_{n-k})$ to honest parties.

Proof. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}orr$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties.

The simulator $\mathcal{S}$ works as follows.

1. In Step 2, $\mathcal{S}$ emulates the functionality $\mathcal{F}_{\text{randZero}}$ and receives the shares of $[\boldsymbol{o}_1]_{n-1}, \ldots, [\boldsymbol{o}_{2k-1}]_{n-1}$ held by corrupted parties.
2. In Step 3, $\mathcal{S}$ emulates the functionality $\mathcal{F}_{\text{rand}}$ and receives the shares of each $[\boldsymbol{r}_j]_{n-k}$ held by corrupted parties.
   - For all corrupted party P_j, $\mathcal{S}$ generates a random degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{r}_j]_{n-k}$ based on the shares held by corrupted parties. Then $\mathcal{S}$ sends the shares held by honest parties to P_j.
   - For all honest party P_j, $\mathcal{S}$ receives from corrupted parties their shares of $[\boldsymbol{r}_j]_{n-k}$. Then $\mathcal{S}$ sets a degree-$(n-1)$ packed Shamir sharing $[\boldsymbol{\delta}_j]_{n-1}$ as follows:
     (a) The share of each honest party is set to be 0.
     (b) The share of each corrupted party is set to be the share of $[\boldsymbol{r}_j]_{n-k}$ received from this corrupted party minus the same share that this corrupted party should hold.

     $\mathcal{S}$ reconstructs $\boldsymbol{\delta}_j$ and views it as an additive attack towards P_j's shares.
3. In Step 4, $\mathcal{S}$ follows the same strategy as that in Step 3 but only for the first $n-2k+1$ parties. For each honest party P_j, $\mathcal{S}$ extracts the additive attack towards P_j's shares, denoted by $\boldsymbol{\delta}'_j$.
4. In Step 7, for all $j \in \{n-2k+2, \ldots, n\}$, $\mathcal{S}$ computes the shares of $[\boldsymbol{r}_{n+j}]_{n-1}$ that corrupted parties should hold. Then, for each $P_j \in \{P_{n-2k+2}, \ldots, P_n\}$,
   - If P_j is corrupted, $\mathcal{S}$ samples a random vector in $\mathbb{F}^k$ as $\boldsymbol{r}_{n+j}$. Based on $\boldsymbol{r}_{n+j}$ and the shares of $[\boldsymbol{r}_{n+j}]_{n-1}$ that corrupted parties should hold, $\mathcal{S}$ randomly samples the shares of honest parties. Finally, $\mathcal{S}$ sends the shares of $\boldsymbol{r}_{n+j}$ held by honest parties to P_j.
   - If P_j is honest, $\mathcal{S}$ receives from corrupted parties their shares of $[\boldsymbol{r}_{n+j}]_{n-1}$. Then $\mathcal{S}$ sets a degree-$(n-1)$ packed Shamir sharing $[\boldsymbol{\delta}'_j]_{n-1}$ as follows:
     (a) The share of each honest party is set to be 0.
     (b) The share of each corrupted party is set to be the share of $[\boldsymbol{r}_{n+j}]_{n-1}$ received from this corrupted party minus the same share that this corrupted party should hold.

     $\mathcal{S}$ reconstructs $\boldsymbol{\delta}'_j$ and views it as an additive attack towards P_j's shares.

Protocol 13: RAND-SHARING-MAL

1. For all $i \in \{1, 2, \ldots, k\}$, let $\Pi_i = \Sigma(\mathtt{pos}_i, \mathtt{pos}'_i, f_i)$.

2. All parties invoke $\mathcal{F}_{\text{randZero}}$ $2k-1$ times and obtain $(2k-1)$ random degree-$(n-1)$ packed Shamir sharings of $\mathbf{0}$, denoted by $[\boldsymbol{o}_1]_{n-1}, \ldots, [\boldsymbol{o}_{2k-1}]_{n-1}$.

3. For all $j \in \{1, 2, \ldots, n\}$, all parties invoke $\mathcal{F}_{\text{rand}}$ and obtain a random degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{r}_j]_{n-k}$. Then all parties send their shares to P_j. P_j views it as a degree-$(n-1)$ packed Shamir sharing and reconstructs $\boldsymbol{r}_j$. P_j will use $\boldsymbol{r}_j$ as his shares of the degree-$(n-1)$ packed Shamir sharings in $\Pi_1, \ldots, \Pi_k$.

4. For all $j \in \{1, 2, \ldots, n-2k+1\}$, all parties invoke $\mathcal{F}_{\text{rand}}$ and obtain a random degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{r}_{n+j}]_{n-k}$. Then all parties send their shares to P_j. P_j views it as a degree-$(n-1)$ packed Shamir sharing and reconstructs $\boldsymbol{r}_{n+j}$. P_j will use the secret $\boldsymbol{r}_{n+j}$ as his shares of the degree-$(n-k)$ packed Shamir sharings in $\Pi_1, \ldots, \Pi_k$.

5. For all $i \in \{1, 2, \ldots, k\}$, the shares of the degree-$(n-1)$ packed Shamir sharing in Π_i, $[\boldsymbol{\tau}_i \| \mathtt{pos}_i]_{n-1}$, are set to be $(r_{1,i}, r_{2,i}, \ldots, r_{n,i})$. And the first $n-2k+1$ shares of the degree-$(n-k)$ packed Shamir sharing in Π_i, $[f_i(\boldsymbol{\tau}_i) \| \mathtt{pos}'_i]_{n-k}$, are set to be $(r_{n+1,i}, \ldots, r_{2n-2k+1,i})$.

6. Note that each value in $\boldsymbol{\tau}_i$ is a linear combination of $(r_{1,i}, r_{2,i}, \ldots, r_{n,i})$. Each value in $f_i(\boldsymbol{\tau}_i)$ is a linear combination of the values in $\boldsymbol{\tau}_i$. And for all $j \in \{n-2k+2, \ldots, n\}$, the j-th share of $[f_i(\boldsymbol{\tau}_i) \| \mathtt{pos}'_i]_{n-k}$ is a linear combination of its first $n-2k+1$ shares $(r_{n+1,i}, \ldots, r_{2n-2k+1,i})$ and the secret $f_i(\boldsymbol{\tau}_i)$. Thus, the j-th share of $[f_i(\boldsymbol{\tau}_i) \| \mathtt{pos}'_i]_{n-k}$ is a linear combination of $(r_{1,i}, r_{2,i}, \ldots, r_{2n-2k+1,i})$. Let $c^{(i)}_{j,1}, \ldots, c^{(j)}_{j,2n-2k+1}$ be the coefficient such that the j-th share of $[f_i(\boldsymbol{\tau}_i) \| \mathtt{pos}'_i]_{n-k}$

$$r_{n+j,i} = \sum_{v=1}^{2n-2k+1} c^{(i)}_{j,v} \cdot r_{v,i}.$$

   For all $j \in \{n-2k+2, \ldots, n\}$ and $v \in \{1, 2, \ldots, 2n-2k+1\}$, let $\boldsymbol{c}_{j,v} = (c^{(1)}_{j,v}, \ldots, c^{(k)}_{j,v})$.

7. For all $j \in \{n-2k+2, \ldots, n\}$, all parties locally compute

$$[\boldsymbol{r}_{n+j}]_{n-1} = [\boldsymbol{o}_{j-n+2k-1}]_{n-1} + \sum_{v=1}^{2n-2k+1} \boldsymbol{c}_{j,v} \cdot [\boldsymbol{r}_v]_{n-k}.$$

   Then, all parties send their shares of $[\boldsymbol{r}_{n+j}]_{n-1}$ to P_j.

8. For all $j \in \{n-2k+2, \ldots, n\}$, P_j reconstructs $[\boldsymbol{r}_{n+j}]_{n-1}$ and learns $\boldsymbol{r}_{n+j}$. Then, for all $i \in \{1, 2, \ldots, k\}$, all parties output $[\boldsymbol{\tau}_i \| \mathtt{pos}_i]_{n-1} = (r_{1,i}, \ldots, r_{n,i})$ and $[f_i(\boldsymbol{\tau}_i) \| \mathtt{pos}'_i]_{n-k} = (r_{n+1,i}, \ldots, r_{2n,i})$.

Now $\mathcal{S}$ transforms the additive attacks towards the shares of honest parties to an additive attack towards the secret of the second sharing in each Π_i. For all $i \in \{1, 2, \ldots, k\}$,

- $\mathcal{S}$ computes a degree-$(2k-2)$ packed Shamir sharing $[\boldsymbol{d}^{(i)}_1 \| \mathtt{pos}_i]_{2k-2}$ which is determined by using $(\delta_{j,i})_{j \in \mathcal{H}}$ as the shares of honest parties. $\boldsymbol{d}^{(i)}_1$ is viewed as an additive attack towards the secret of the first sharing in Π_i.

- $\mathcal{S}$ computes a degree-$(2k-2)$ packed Shamir sharing $[\boldsymbol{d}_2^{(i)}\|\mathtt{pos}'_i]_{2k-2}$ which is determined by using $(\delta'_{j,i})_{j\in\mathcal{H}}$ as the shares of honest parties. $\boldsymbol{d}_2^{(i)}$ is viewed as an additive attack towards the secret of the second sharing in Π_i.

$\mathcal{S}$ computes $\boldsymbol{d}^{(i)} = \boldsymbol{d}_2^{(i)} - f_i(\boldsymbol{d}_1^{(i)})$.

5. Recall that $\mathcal{S}$ has computed $\boldsymbol{r}_j$ and $\boldsymbol{r}_{n+j}$ for all corrupted party P_j. Let $\mathtt{sh}_j([\boldsymbol{d}_1^{(i)}\|\mathtt{pos}_i]_{2k-2})$ denote the j-th share of $[\boldsymbol{d}_1^{(i)}\|\mathtt{pos}_i]_{2k-2}$, and $\mathtt{sh}_j([\boldsymbol{d}_2^{(i)}\|\mathtt{pos}'_i]_{2k-2})$ denote the j-th share of $[\boldsymbol{d}_2^{(i)}\|\mathtt{pos}'_i]_{2k-2}$. For all $i \in \{1,2,\ldots,k\}$, $\mathcal{S}$ sends $\{r_{j,i} + \mathtt{sh}_j([\boldsymbol{d}_1^{(i)}\|\mathtt{pos}_i]_{2k-2}), r_{n+j,i} + \mathtt{sh}_j([\boldsymbol{d}_2^{(i)}\|\mathtt{pos}'_i]_{2k-2})\}_{j\in\mathcal{C}orr}$ and $\boldsymbol{d}^{(i)}$ to $\mathcal{F}_{\text{rand-sharing-mal}}(\Pi_i)$.

This completes the description of the simulator. Now, we show that the simulator we constructed above perfectly simulate the behaviors of honest parties.

We first show that the messages that honest parties send to corrupted parties have the same distribution in both worlds. In Step 3, for each corrupted party P_j, honest parties need to send their shares of $[\boldsymbol{r}_j]_{n-k}$ to P_j. Recall that $[\boldsymbol{r}_j]_{n-k}$ is a random degree-$(n-k)$ packed Shamir sharing generated by $\mathcal{F}_{\text{rand}}$. In the ideal world, $\mathcal{S}$ honestly follows $\mathcal{F}_{\text{rand}}$ to compute the shares of honest parties. Therefore, the shares of $[\boldsymbol{r}_j]_{n-k}$ held by honest parties have the same distribution in both worlds. Similarly, in Step 4, for each corrupted party P_j of the first $n-2k+1$ parties, the shares of $[\boldsymbol{r}_{n+j}]_{n-k}$ held by honest parties have the same distribution in both worlds.

In Step 7, for each corrupted party $P_j \in\in \{P_{n-2k+2},\ldots,P_n\}$, honest parties need to send their shares of $[\boldsymbol{r}_{n+j}]_{n-1}$ to P_j. Recall that $\boldsymbol{r}_{n+j}$ are used as the j-th shares of the degree-$(n-k)$ packed Shamir sharings in $\Pi_1,\ldots,\Pi_k$. Since the degree-$(n-k)$ packed Shamir secret sharing scheme has threshold t, the shares of corrupted parties are uniformly random. Therefore $\boldsymbol{r}_{n+j}$ is uniformly distributed in $\mathbb{F}^k$. Since all parties use a random degree-$(n-1)$ packed Shamir sharing $[\boldsymbol{o}_{j-n+2k-1}]_{n-1}$ as a random mask, $[\boldsymbol{r}_{n+j}]_{n-1}$ is a random degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{r}_{n+j}$ given the secret $\boldsymbol{r}_{n+j}$ and the shares of corrupted parties. In the ideal world, $\mathcal{S}$ randomly samples the secret $\boldsymbol{r}_{n+j}$ and randomly samples the shares of honest parties in $[\boldsymbol{r}_{n+j}]_{n-1}$ based on the secret $\boldsymbol{r}_{n+j}$ and the shares of corrupted parties. Thus, the shares of $[\boldsymbol{r}_{n+j}]_{n-1}$ held by honest parties have the same distribution in both worlds.

Now it is sufficient to show that the shares of the output sharings held by honest parties have the same distribution in both worlds. We note that the only thing that corrupted parties can do is to send incorrect shares to honest parties.

- In the real world, in Step 3 for each honest party P_j, P_j uses the shares he received as a degree-$(n-1)$ packed Shamir sharing and reconstructs $\tilde{\boldsymbol{r}}_j$ (Here we use the tilde notation to distinguish from the correct secret $\boldsymbol{r}_j$ that P_j should obtain.) Compared with the shares that P_j should received, the difference is the degree-$(n-1)$ packed Shamir sharing $[\boldsymbol{\delta}_j]_{n-1}$ we constructed above. Thus, $\boldsymbol{\delta}_j = \tilde{\boldsymbol{r}}_j - \boldsymbol{r}_j$. In the ideal world, $\mathcal{S}$ can compute $[\boldsymbol{\delta}_j]_{n-1}$ as described above. Similarly, in Step 4 and Step 7, $\mathcal{S}$ can extract $\boldsymbol{\delta}'_j = \tilde{\boldsymbol{r}}_{n+j} - \boldsymbol{r}_{n+j}$.

- Now for all $i \in \{1,2,\ldots,k\}$ and for each honest party P_j, the j-th share of the first sharing in Π_i is deviated by $\delta_{j,i}$ due to the malicious behaviors of corrupted parties. Let $[\boldsymbol{\tau}_i\|\mathtt{pos}_i]_{n-1}$ denote the first sharing that all parties should obtain. Then the shares that honest parties actually obtain are equal to the shares of $[\boldsymbol{\tau}_i\|\mathtt{pos}_i]_{n-1} + [\boldsymbol{d}_1^{(i)}\|\mathtt{pos}_i]_{2k-2}$, where recall that $[\boldsymbol{d}_1^{(i)}\|\mathtt{pos}_i]_{2k-2}$ is the degree-$(2k-2)$ packed Shamir sharing determined by using $(\delta_{j,i})_{j\in\mathcal{H}}$ as the shares of honest parties. Note that when $k \le (n+2)/3$, we have $2k-2 \le n-1$. Therefore $[\boldsymbol{\tau}_i\|\mathtt{pos}_i]_{n-1} + [\boldsymbol{d}_1^{(i)}\|\mathtt{pos}_i]_{2k-2} = [\boldsymbol{\tau}_i + \boldsymbol{d}_1^{(i)}\|\mathtt{pos}_i]_{n-1}$. Since $[\boldsymbol{\tau}_i\|\mathtt{pos}_i]_{n-1}$ is a random degree-$(n-1)$ packed Shamir sharing given the shares of corrupted parties, $[\boldsymbol{\tau}_i + \boldsymbol{d}_1^{(i)}\|\mathtt{pos}_i]_{n-1}$ is also a random degree-$(n-1)$ packed Shamir sharing given the shares of corrupted parties. Thus, by sending the shares of $[\boldsymbol{\tau}_i + \boldsymbol{d}_1^{(i)}\|\mathtt{pos}_i]_{n-1}$ of corrupted parties to $\mathcal{F}_{\text{rand-sharing-mal}}$, $\mathcal{F}_{\text{rand-sharing-mal}}$ generates the shares of $[\boldsymbol{\tau}_i + \boldsymbol{d}_1^{(i)}\|\mathtt{pos}_i]_{n-1}$ of honest parties with the same distribution as that in the real world.

Similarly, for the second sharing $[f_i(\boldsymbol{\tau}_i)\|\mathtt{pos}'_i]_{n-k}$ and for each honest party P_j, the j-th share is deviated by $\delta'_{j,i}$ due to the malicious behaviors of corrupted parties. Then the shares that honest parties actually obtain are equal to the shares of $[f_i(\boldsymbol{\tau}_i)\|\mathtt{pos}'_i]_{n-k} + [\boldsymbol{d}_2^{(i)}\|\mathtt{pos}'_i]_{2k-2}$, where recall that $[\boldsymbol{d}_2^{(i)}\|\mathtt{pos}'_i]_{2k-2}$ is the degree-$(2k-2)$ packed Shamir sharing determined by using $(\delta'_{j,i})_{j\in\mathcal{H}}$ as the shares of honest parties. Note that when $k \le (n+2)/3$, we have $2k-2 \le n-k$. Therefore $[f_i(\boldsymbol{\tau}_i)\|\mathtt{pos}'_i]_{n-k} + [\boldsymbol{d}_2^{(i)}\|\mathtt{pos}'_i]_{2k-2} = [f_i(\boldsymbol{\tau}_i)+\boldsymbol{d}_2^{(i)}\|\mathtt{pos}'_i]_{n-k}$. Note that a degree-$(n-k)$ packed Shamir sharing is determined by the secret and the shares of corrupted parties. Since we have provided the shares of $[\boldsymbol{\tau}_i + \boldsymbol{d}_1^{(i)}\|\mathtt{pos}_i]_{n-1}$ of corrupted parties to $\mathcal{F}_{\text{rand-sharing-mal}}$, the secret generated by $\mathcal{F}_{\text{rand-sharing-mal}}$ would correspond to $\boldsymbol{\tau}_i + \boldsymbol{d}_1^{(i)}$. Then the additive errors to the secret becomes $(f_i(\boldsymbol{\tau}_i) + \boldsymbol{d}_2^{(i)}) - f_i(\boldsymbol{\tau}_i + \boldsymbol{d}_1^{(i)}) = \boldsymbol{d}_2^{(i)} - f_i(\boldsymbol{d}_1^{(i)}) = \boldsymbol{d}^{(i)}$. Thus, by sending the shares of $[f_i(\boldsymbol{\tau}_i) + \boldsymbol{d}_2^{(i)}\|\mathtt{pos}'_i]_{n-k}$ of corrupted parties and the additive errors $\boldsymbol{d}^{(i)}$, $\mathcal{F}_{\text{rand-sharing-mal}}$ generates the shares of $[f_i(\boldsymbol{\tau}_i) + \boldsymbol{d}_2^{(i)}\|\mathtt{pos}'_i]_{n-k}$ of honest parties with the same distribution as that in the real world.

We conclude that when $k \le (n+2)/3$, Protocol Rand-Sharing-Mal securely computes $\mathcal{F}_{\text{rand-sharing-mal}}$ in the $\{\mathcal{F}_{\text{rand}}, \mathcal{F}_{\text{randZero}}\}$-hybrid model against a fully malicious adversary who controls $t = n - 2k + 1$ parties. □

6.1.2 Performing Sharing Transformation

Now we present the concrete protocol for performing the sharing transformation that we are interested in. We modify the protocol Tran by requiring that P_1 distributes a degree-$(2k-2)$ packed Shamir sharing of the reconstruction result. In this way, the whole sharing is determined by the shares of honest parties. Note that when $k \le (n+2)/3$, we have $2k-2 \le n-k$. Thus, all parties can still obtain a degree-$(n-k)$ Shamir sharing at the end.

The description of the protocol Tran-Mal appears in Protocol 14.

Protocol 14: Tran-Mal

1. All parties agree on the tuple $(\mathtt{pos}, \mathtt{pos}', f)$ where $\mathtt{pos}, \mathtt{pos}'$ are two vectors of field elements in $\mathbb{F}^k$ and $f : \mathbb{F}^k \to \mathbb{F}^k$ is a linear map. All parties start with a degree-$(n-1)$ packed Shamir sharing $[\boldsymbol{x}\|\mathtt{pos}]_{n-1}$. The goal is to compute $[f(\boldsymbol{x})\|\mathtt{pos}']_{n-k}$.
2. Let $\Pi = \Pi(\mathtt{pos}, \mathtt{pos}', f)$ defined in Section 6.1.1. All parties invoke $\mathcal{F}_{\text{rand-sharing-mal}}(\Pi)$ to prepare a random Π-sharing $([\boldsymbol{r}\|\mathtt{pos}]_{n-1}, [f(\boldsymbol{r})\|\mathtt{pos}']_{n-k})$.
3. All parties locally compute $[\boldsymbol{x}+\boldsymbol{r}\|\mathtt{pos}]_{n-1} = [\boldsymbol{x}\|\mathtt{pos}]_{n-1} + [\boldsymbol{r}\|\mathtt{pos}]_{n-1}$ and send their shares to the first party P_1.
4. P_1 reconstructs the secret $\boldsymbol{x}+\boldsymbol{r}$. Then P_1 computes $f(\boldsymbol{x}+\boldsymbol{r})$ and generates a degree-$(2k-2)$ packed Shamir sharing $[f(\boldsymbol{x}+\boldsymbol{r})\|\mathtt{pos}']_{2k-2}$. Finally, P_1 distributes the shares of $[f(\boldsymbol{x}+\boldsymbol{r})\|\mathtt{pos}']_{2k-2}$ to all parties.
5. All parties locally compute $[f(\boldsymbol{x})\|\mathtt{pos}']_{n-k} = [f(\boldsymbol{x}+\boldsymbol{r})\|\mathtt{pos}']_{2k-2} - [f(\boldsymbol{r})\|\mathtt{pos}']_{n-k}$.

6.2 Circuit-Independent Preprocessing Phase

In the circuit independent preprocessing phase, all parties prepare the following correlated random sharings:

- All parties prepare a random degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{\gamma}]_{n-k}$, where $\boldsymbol{\gamma} = (\gamma, \gamma, \ldots, \gamma) \in \mathbb{F}^k$ and γ is a random field element, which is served as the MAC key.

- For every group of k input gates and output gates:
  1. All parties prepare an authenticated random degree-$(n-k)$ packed Shamir sharings $[\![\boldsymbol{r}]\!]_{n-k} = ([\boldsymbol{r}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k})$. In addition, all parties prepare another random degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{\Delta}]_{n-k}$ and compute the sharing $[\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$. Note that $([\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k})$ can be seen as an authentication of the sharing $[\boldsymbol{r}]_{n-k}$ under the MAC key $[\boldsymbol{\Delta}]_{n-k}$. We will use $([\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k})$ to allow a client to verify the correctness of the secret $\boldsymbol{r}$ in the input protocol and the output protocol.
  2. For every group of k output gates, all parties also prepare a random degree-$(n-1)$ packed Shamir sharing of $\mathbf{0}$, denoted by $[\boldsymbol{o}]_{n-1}$. We will use $[\boldsymbol{o}]_{n-1}$ to protect the shares of honest parties.
- For every group of k multiplication gates:
  1. All parties prepare an authenticated packed Beaver triple $([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{c}]\!]_{n-k})$ where $\boldsymbol{a}, \boldsymbol{b}$ are random vectors in $\mathbb{F}^k$ and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$.
  2. All parties prepare two random degree-$(n-1)$ packed Shamir sharings of $\mathbf{0}$, denoted by $[\boldsymbol{o}^{(1)}]_{n-1}$, $[\boldsymbol{o}^{(2)}]_{n-1}$. In the multiplication protocol, these sharings are used as random masks to protect the shares of honest parties.
- All parties also prepare the following sharings for the verification of the computation:
  1. All parties prepare two random degree-$(n-1)$ packed Shamir sharings of $\mathbf{0}$, denoted by $[\boldsymbol{o}^{(1)}]_{n-1}$ and $[\boldsymbol{o}^{(2)}]_{n-1}$. These two sharings are used to protect the shares of honest parties when checking the correctness of multiplications.
  2. All parties prepare a pair of two random additive sharings $(\langle r \rangle, \langle \gamma \cdot r \rangle)$, where r is a random element in $\mathbb{F}$ and γ is the authentication key.

The functionality $\mathcal{F}_{\text{prep-mal}}$ for the circuit independent preprocessing phase appears in Functionality 15. The size of the preprocessing data among all parties in $\mathcal{F}_{\text{prep-mal}}$ is summarized as follows:

- The sharing of the MAC key $[\boldsymbol{\gamma}]_{n-k}$: n elements.
- Per input gate: $4n/k$ elements.
- Per output gate: $5n/k$ elements.
- Per multiplication gate: $8n/k$ elements.
- For the verification of the computation: $4 \cdot n$ elements.

6.3 Online Computation Phase

6.3.1 Input Layer

For a group of k input gates belonging to the same client, let $\boldsymbol{x}$ be the values associated with these k input gates. Suppose $\texttt{pos}$ are the positions associated with these k gates. The goal is to compute an authenticated sharing $[\![\boldsymbol{x} \| \texttt{pos}]\!]_{n-k}$. Recall that for every group of k input gates, all parties have prepared the random sharings $[\![\boldsymbol{r}]\!]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ in $\mathcal{F}_{\text{prep-mal}}$, where $[\![\boldsymbol{r}]\!]_{n-k} = ([\boldsymbol{r}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k})$. At a high-level, all parties first send the random sharing $[\boldsymbol{r}]_{n-k}$ to the client, and make use of the random sharings $[\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ to allow the client to verify the correctness of the secret $\boldsymbol{r}$. Then, the client samples a random degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{x} + \boldsymbol{r}]_{k-1}$ and distributes the shares to all parties. In this way, all parties can compute

$$\begin{aligned} [\boldsymbol{x}]_{n-k} &= [\boldsymbol{x} + \boldsymbol{r}]_{k-1} - [\boldsymbol{r}]_{n-k} \\ [\boldsymbol{\gamma} * \boldsymbol{x}]_{n-1} &= [\boldsymbol{x} + \boldsymbol{r}]_{k-1} \cdot [\boldsymbol{\gamma}]_{n-k} - [\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k} \end{aligned}$$

which is $[\![\boldsymbol{x}]\!]_{n-1} = ([\boldsymbol{x}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{x}]_{n-1})$. Finally, all parties apply two times of Tran-Mal to transform $[\![\boldsymbol{x}]\!]_{n-1}$ to $[\![\boldsymbol{x} \| \texttt{pos}]\!]_{n-k}$. The description of Input-Mal appears in Protocol 16.

Functionality 15: $\mathcal{F}_{\text{prep-mal}}$

$\mathcal{F}_{\text{prep-mal}}$ receives the set of corrupted parties, denoted by $\mathcal{C}orr$. $\mathcal{F}_{\text{prep-mal}}$ samples a random field element $\gamma \in \mathbb{F}$ and sets $\boldsymbol{\gamma} = (\gamma, \gamma, \ldots, \gamma) \in \mathbb{F}^k$. Let $d \in \{n-k, n-1\}$. We define the following two procedures.

- RandSharing($\boldsymbol{r}$, d): $\mathcal{F}_{\text{prep-mal}}$ receives from the adversary a set of shares $\{r_j\}_{j \in \mathcal{C}orr}$. Then $\mathcal{F}_{\text{prep-mal}}$ samples a random degree-d packed Shamir sharing $[\boldsymbol{r}]_d$ such that for all $P_j \in \mathcal{C}orr$, the j-th share of $[\boldsymbol{r}]_d$ is r_j. Finally, $\mathcal{F}_{\text{prep-mal}}$ distributes the shares of $[\boldsymbol{r}]_d$ to honest parties.
- AuthSharing($\boldsymbol{r}$): $\mathcal{F}_{\text{prep-mal}}$ receives from the adversary a set of shares $\{(r_j, u_j)\}_{j \in \mathcal{C}orr}$. Then $\mathcal{F}_{\text{prep-mal}}$ computes two degree-$(n-k)$ packed Shamir sharings $([\boldsymbol{r}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k})$ such that for all $P_j \in \mathcal{C}orr$, the j-th shares of $([\boldsymbol{r}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k})$ are r_j, u_j respectively. Finally, $\mathcal{F}_{\text{prep-mal}}$ distributes the shares of $[\![\boldsymbol{r}]\!]_{n-k} = ([\boldsymbol{r}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k})$ to honest parties.

The ideal functionality $\mathcal{F}_{\text{prep-mal}}$ runs the following steps.

1. $\mathcal{F}_{\text{prep-mal}}$ invokes RandSharing($\boldsymbol{\gamma}$, $n-k$) to prepare $[\boldsymbol{\gamma}]_{n-k}$.
2. For every group of k input gates and output gates:
   (a) $\mathcal{F}_{\text{prep-mal}}$ samples a random vector $\boldsymbol{r} \in \mathbb{F}^k$ and invokes AuthSharing($\boldsymbol{r}$) to prepare $[\![\boldsymbol{r}]\!]_{n-k}$.
   (b) $\mathcal{F}_{\text{prep-mal}}$ samples a random vector $\boldsymbol{\Delta} \in \mathbb{F}^k$ and invokes RandSharing($\boldsymbol{\Delta}$, $n-k$) and RandSharing($\boldsymbol{\Delta} * \boldsymbol{r}$, $n-k$) to prepare $([\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k})$.
   (c) For every group of k output gates, $\mathcal{F}_{\text{prep-mal}}$ invokes RandSharing($\boldsymbol{0}$, $n-1$) to prepare $[\boldsymbol{o}]_{n-1}$, where $\boldsymbol{o} = \boldsymbol{0}$.
3. For every group of k multiplication gates:
   (a) $\mathcal{F}_{\text{prep-mal}}$ samples two random vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^k$ and computes $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. Then, $\mathcal{F}_{\text{prep-mal}}$ invokes AuthSharing($\boldsymbol{a}$), AuthSharing($\boldsymbol{b}$), and AuthSharing($\boldsymbol{c}$) to prepare $([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{c}]\!]_{n-k})$.
   (b) $\mathcal{F}_{\text{prep-mal}}$ invokes two times of RandSharing($\boldsymbol{0}$, $n-1$) to prepare $[\boldsymbol{o}^{(1)}]_{n-1}$, $[\boldsymbol{o}^{(2)}]_{n-1}$, where $\boldsymbol{o}^{(1)} = \boldsymbol{o}^{(2)} = \boldsymbol{0}$.
4. All parties prepare the following random sharings for the verification of the computation:
   (a) All parties invoke two times of RandSharing($\boldsymbol{0}$, $n-1$) to prepare $[\boldsymbol{o}^{(1)}]_{n-1}, [\boldsymbol{o}^{(2)}]_{n-1}$, where $\boldsymbol{o}^{(1)} = \boldsymbol{o}^{(2)} = \boldsymbol{0}$.
   (b) $\mathcal{F}_{\text{prep-mal}}$ receives from the adversary a set of shares $\{(r_j, r'_j)\}_{j \in \mathcal{C}orr}$. Then $\mathcal{F}_{\text{prep-mal}}$ samples a random field element r and computes $\gamma \cdot r$. $\mathcal{F}_{\text{prep-mal}}$ randomly generates a pair of additive sharings $(\langle r \rangle, \langle \gamma \cdot r \rangle)$ such that for all $P_j \in \mathcal{C}orr$, the j-th shares of $(\langle r \rangle, \langle \gamma \cdot r \rangle)$ are r_j, r'_j respectively. Finally, $\mathcal{F}_{\text{prep-mal}}$ distributes the shares of $(\langle r \rangle, \langle \gamma \cdot r \rangle)$ to honest parties.

6.3.2 Network Routing

We follow the same approach as that in our semi-honest protocol to realize network routing. To obtain an authenticated degree-$(n-k)$ packed Shamir sharing $[\![\boldsymbol{x}]\!]_{n-k}$, we will apply the network routing protocol to compute each of $([\boldsymbol{x}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{x}]_{n-k})$. The description of Network-Mal appears in Protocol 17.

Protocol 16: INPUT-MAL

1. Suppose $\boldsymbol{x} \in \mathbb{F}^k$ is the input associated with the input gates which belongs to Client. Also suppose pos are the positions associated with these gates. Let $[\![\boldsymbol{r}]\!]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ be the random sharings prepared for these input gates in $\mathcal{F}_{\text{prep-mal}}$. Recall that $[\![\boldsymbol{r}]\!]_{n-k} = ([\boldsymbol{r}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k})$.

2. All parties send their shares of $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ to Client.

3. Client checks whether the shares of $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ form valid degree-$(n-k)$ packed Shamir sharings. If not, Client aborts. Otherwise, Client reconstructs the secret $\boldsymbol{r}, \boldsymbol{\Delta}, \boldsymbol{\Delta} * \boldsymbol{r}$ and checks the MAC of $\boldsymbol{r}$, i.e., whether the third secret is equal to the coordinate-wise multiplication of the first two secrets. If not, Client aborts.

4. If both checks pass, Client generates a random degree-$(k-1)$ packed Shamir sharing of $\boldsymbol{x} + \boldsymbol{r}$, denoted by $[\boldsymbol{x} + \boldsymbol{r}]_{k-1}$, and distributes the shares to all parties.

5. All parties locally compute

$$\begin{aligned} [\boldsymbol{x}]_{n-k} &= [\boldsymbol{x} + \boldsymbol{r}]_{k-1} - [\boldsymbol{r}]_{n-k} \\ [\boldsymbol{\gamma} * \boldsymbol{x}]_{n-1} &= [\boldsymbol{x} + \boldsymbol{r}]_{k-1} \cdot [\boldsymbol{\gamma}]_{n-k} - [\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k} \end{aligned}$$

   and set $[\![\boldsymbol{x}]\!]_{n-1} = ([\boldsymbol{x}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{x}]_{n-1})$.

6. Recall that $\boldsymbol{\beta}$ are the default positions for the packed Shamir secret sharing scheme. All parties invoke TRAN-MAL on $[\boldsymbol{x}]_{n-k}$ with $(\boldsymbol{\beta}, \mathtt{pos}, I)$, where I is the identity map, and obtain $[\boldsymbol{x} \| \mathtt{pos}]_{n-k}$. All parties invoke TRAN-MAL on $[\boldsymbol{\gamma} * \boldsymbol{x}]_{n-1}$ with $(\boldsymbol{\beta}, \mathtt{pos}, I)$ and obtain $[\boldsymbol{\gamma} * \boldsymbol{x} \| \mathtt{pos}]_{n-k}$.

7. All parties take $[\![\boldsymbol{x} \| \mathtt{pos}]\!]_{n-k} = ([\boldsymbol{x} \| \mathtt{pos}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{x} \| \mathtt{pos}]_{n-k})$ as output.

6.3.3 Evaluating Addition Gates and Multiplication Gates

Addition Gates. We follow the same approach as that in our semi-honest protocol to evaluate addition gates. To obtain an authenticated degree-$(n-k)$ packed Shamir sharing $[\![\boldsymbol{z} \| \mathtt{pos}]\!]_{n-k}$ (the output sharing), we will invoke TRAN-MAL two times to transform $[\![\boldsymbol{z}]\!]_{n-k}$ to $[\![\boldsymbol{z} \| \mathtt{pos}]\!]_{n-k}$. The description of ADD-MAL appears in Protocol 18.

Multiplication Gates. For a group of k multiplication gates, let $[\![\boldsymbol{x}]\!]_{n-k}, [\![\boldsymbol{y}]\!]_{n-k}$ be the input sharings. Recall that all parties have prepared the following random sharings in $\mathcal{F}_{\text{prep-mal}}$:

- An authenticated packed Beaver triple: $([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{c}]\!]_{n-k})$.
- Three degree-$(n-1)$ packed Shamir sharings of $\mathbf{0}$: $[\boldsymbol{o}^{(1)}]_{n-1}, [\boldsymbol{o}^{(2)}]_{n-1}$.

The goal is to compute an authenticated output sharing $[\![\boldsymbol{z} \| \mathtt{pos}]\!]_{n-k} = [\![\boldsymbol{x} * \boldsymbol{y} \| \mathtt{pos}]\!]_{n-k}$, where pos are the positions associated with these k gates. We follow a similar approach as that in our semi-honest protocol. The first difference is that, all parties will use $[\boldsymbol{o}^{(1)}]_{n-1}, [\boldsymbol{o}^{(2)}]_{n-1}$ to refresh their shares when letting P_1 reconstructs $\boldsymbol{x} + \boldsymbol{a}$ and $\boldsymbol{y} + \boldsymbol{b}$. The second difference is that P_1 not only needs to distribute degree-$(k-1)$ packed Shamir sharings of $\boldsymbol{x} + \boldsymbol{a}, \boldsymbol{y} + \boldsymbol{b}$, but also a degree-$(k-1)$ packed Shamir sharing of $(\boldsymbol{x} + \boldsymbol{a}) \cdot (\boldsymbol{y} + \boldsymbol{b})$. In this way, all parties can locally compute an authenticated degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$. The correctness of the sharings distributed by P_1 will be checked later. Finally, we will invoke TRAN-MAL two times to transform $[\![\boldsymbol{z}]\!]_{n-1}$ to $[\![\boldsymbol{z} \| \mathtt{pos}]\!]_{n-k}$.

The description of MULT-MAL appears in Protocol 19.

Protocol 17: NETWORK-MAL

1. Suppose all parties want to prepare an authenticated degree-$(n-k)$ packed Shamir sharing of $\boldsymbol{x}$ stored at the default positions $\boldsymbol{\beta}$.

2. Let $x'_1, x'_2, \ldots, x'_{\ell_1}$ be the different wire values in $\boldsymbol{x}$ from previous layers. Let $c_1, c_2, \ldots, c_{\ell_2}$ be the constant values in $\boldsymbol{x}$. Let $\boldsymbol{x}' = (x'_1, \ldots, x'_{\ell_1}, c_1, \ldots, c_{\ell_2}, 0, \ldots, 0) \in \mathbb{F}^k$.

3. For all $1 \le i \le \ell_1$, let $[\![\boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]\!]_{n-k}$ be the degree-$(n-k)$ packed Shamir sharing from some previous layer that contains the secret x'_i stored at position p_i. Let $p_{\ell_1+1}, \ldots, p_k$ be the first $k - \ell_1$ distinct positions that are different from $p_1, \ldots, p_{\ell_1}$ and $\alpha_1, \ldots, \alpha_n$. (In particular, $p_i \in \{\beta_1, \ldots, \beta_k\}$ for all $i \in \{\ell_1 + 1, \ldots, k\}$.) Let $\mathtt{pos} = (p_1, \ldots, p_k)$.

4. Let $\boldsymbol{e}_i$ be the i-th unit vector in $\mathbb{F}^k$ (i.e., only the i-th term is 1 and all other terms are 0). All parties locally compute a degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{e}_i \| \mathtt{pos}]_{k-1}$.

5. All parties locally compute

$$\begin{aligned}
[\boldsymbol{x}' \| \mathtt{pos}]_{n-1} &= \sum_{i=1}^{\ell_1} [\boldsymbol{e}_i \| \mathtt{pos}]_{k-1} \cdot [\boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]_{n-k} + \sum_{i=1}^{\ell_2} c_i \cdot [\boldsymbol{e}_{\ell_1+i} \| \mathtt{pos}]_{k-1} \\
[\boldsymbol{\gamma} * \boldsymbol{x}' \| \mathtt{pos}]_{n-1} &= \sum_{i=1}^{\ell_1} [\boldsymbol{e}_i \| \mathtt{pos}]_{k-1} \cdot [\boldsymbol{\gamma} * \boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]_{n-k} \\
&\quad + \sum_{i=1}^{\ell_2} c_i \cdot [\boldsymbol{e}_{\ell_1+i} \| \mathtt{pos}]_{k-1} \cdot [\boldsymbol{\gamma}]_{n-k}
\end{aligned}$$

6. Let $f : \mathbb{F}^k \to \mathbb{F}^k$ be a linear map such that $\boldsymbol{x} = f(\boldsymbol{x}')$. Recall that $\boldsymbol{\beta}$ are the default positions for the packed Shamir secret sharing scheme. All parties invoke TRAN-MAL on $[\boldsymbol{x}' \| \mathtt{pos}]_{n-1}$ with $(\mathtt{pos}, \boldsymbol{\beta}, f)$ and obtain $[\boldsymbol{x}]_{n-k}$. All parties invoke TRAN-MAL on $[\boldsymbol{\gamma} * \boldsymbol{x}' \| \mathtt{pos}]_{n-1}$ with $(\mathtt{pos}, \boldsymbol{\beta}, f)$ and obtain $[\boldsymbol{\gamma} * \boldsymbol{x}]_{n-k}$.

7. All parties take $[\![\boldsymbol{x}]\!]_{n-k} = ([\boldsymbol{x}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{x}]_{n-k})$ as output.

Protocol 18: ADD-MAL

1. Suppose $[\![\boldsymbol{x}]\!]_{n-k}, [\![\boldsymbol{y}]\!]_{n-k}$ are the input packed Shamir sharings of the addition gates.

2. All parties locally compute $[\![\boldsymbol{z}]\!]_{n-k} = [\![\boldsymbol{x}]\!]_{n-k} + [\![\boldsymbol{y}]\!]_{n-k}$.

3. Suppose $\mathtt{pos}$ are the positions associated with these k addition gates. Recall that $\boldsymbol{\beta}$ are the default positions. All parties invoke TRAN-MAL on $[\boldsymbol{z}]_{n-k}$ with $(\boldsymbol{\beta}, \mathtt{pos}, I)$, where I is the identity map, and obtain $[\boldsymbol{z} \| \mathtt{pos}]_{n-k}$. All parties invoke TRAN-MAL on $[\boldsymbol{\gamma} * \boldsymbol{z}]_{n-1}$ with $(\boldsymbol{\beta}, \mathtt{pos}, I)$ and obtain $[\boldsymbol{\gamma} * \boldsymbol{z} \| \mathtt{pos}]_{n-k}$.

4. All parties take $[\![\boldsymbol{z} \| \mathtt{pos}]\!]_{n-k} = ([\boldsymbol{z} \| \mathtt{pos}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{z} \| \mathtt{pos}]_{n-k})$ as output.

Protocol 19: MULT-MAL

1. Suppose $[\![\boldsymbol{x}]\!]_{n-k}, [\![\boldsymbol{y}]\!]_{n-k}$ are the input packed Shamir sharings of the multiplication gates. All parties will use the following random sahrings prepared in the preprocessing phase
   - An authenticated packed Beaver triple: $([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{c}]\!]_{n-k})$.
   - Three degree-$(n-1)$ packed Shamir sharings of $\mathbf{0}$: $[\boldsymbol{o}^{(1)}]_{n-1}, [\boldsymbol{o}^{(2)}]_{n-1}$.
2. All parties locally compute $[\boldsymbol{x}+\boldsymbol{a}]_{n-1} = [\boldsymbol{o}^{(1)}]_{n-1} + [\boldsymbol{x}]_{n-k} + [\boldsymbol{a}]_{n-k}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{n-1} = [\boldsymbol{o}^{(2)}]_{n-1} + [\boldsymbol{y}]_{n-k} + [\boldsymbol{b}]_{n-k}$.
3. The first party P_1 collects the whole sharings $[\boldsymbol{x}+\boldsymbol{a}]_{n-1}, [\boldsymbol{y}+\boldsymbol{b}]_{n-1}$ and reconstructs the secrets $\boldsymbol{x}+\boldsymbol{a}, \boldsymbol{y}+\boldsymbol{b}$. Then, P_1 computes $(\boldsymbol{x}+\boldsymbol{a})\cdot(\boldsymbol{y}+\boldsymbol{b})$ and the sharings $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}, [(\boldsymbol{x}+\boldsymbol{a})\cdot(\boldsymbol{y}+\boldsymbol{b})]_{k-1}$. Finally, P_1 distributes the shares to other parties.
4. All parties locally compute

$$\begin{aligned}
[\boldsymbol{z}]_{n-1} &= [(\boldsymbol{x}+\boldsymbol{a})\cdot(\boldsymbol{y}+\boldsymbol{b})]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1}\cdot[\boldsymbol{b}]_{n-k} \\
&\quad -[\boldsymbol{y}+\boldsymbol{b}]_{k-1}\cdot[\boldsymbol{a}]_{n-k} + [\boldsymbol{c}]_{n-k} \\
[\boldsymbol{\gamma}*\boldsymbol{z}]_{n-1} &= [\boldsymbol{\gamma}]_{n-k}\cdot[(\boldsymbol{x}+\boldsymbol{a})\cdot(\boldsymbol{y}+\boldsymbol{b})]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1}\cdot[\boldsymbol{\gamma}*\boldsymbol{b}]_{n-k} \\
&\quad -[\boldsymbol{y}+\boldsymbol{b}]_{k-1}\cdot[\boldsymbol{\gamma}*\boldsymbol{a}]_{n-k} + [\boldsymbol{\gamma}*\boldsymbol{c}]_{n-k}
\end{aligned}$$

   and set $[\![\boldsymbol{z}]\!]_{n-1} = ([\boldsymbol{z}]_{n-1}, [\boldsymbol{\gamma}*\boldsymbol{z}]_{n-1})$.
5. Suppose $\mathtt{pos}$ are the positions associated with these k multiplication gates. Recall that $\boldsymbol{\beta}$ are the default positions. All parties invoke TRAN-MAL on $[\boldsymbol{z}]_{n-1}$ with $(\boldsymbol{\beta}, \mathtt{pos}, I)$, where I is the identity map, and obtain $[\boldsymbol{z}\|\mathtt{pos}]_{n-k}$. All parties invoke TRAN-MAL on $[\boldsymbol{\gamma}*\boldsymbol{z}]_{n-1}$ with $(\boldsymbol{\beta}, \mathtt{pos}, I)$ and obtain $[\boldsymbol{\gamma}*\boldsymbol{z}\|\mathtt{pos}]_{n-k}$.
6. All parties take $[\![\boldsymbol{z}\|\mathtt{pos}]\!]_{n-k} = ([\boldsymbol{z}\|\mathtt{pos}]_{n-k}, [\boldsymbol{\gamma}*\boldsymbol{z}\|\mathtt{pos}]_{n-k})$ as output.

6.3.4 Output Layer

We will use an ideal functionality $\mathcal{F}_{\text{com}}$ that allows a party to commit values to other parties. The description of $\mathcal{F}_{\text{com}}$ appears in Functionality 20. It has been shown in [DPSZ12] that public commitment in $\mathcal{F}_{\text{com}}$ can be realized with the cost of $2n^2+n$ elements of preprocessing data and $4n^2$ elements of communication. As for private commitment in $\mathcal{F}_{\text{com}}$, it can be realized similarly by restricting the participants between the committer and the receiver.

For a group of k output gates belonging to the same client, let $\boldsymbol{x} \in \mathbb{F}^k$ be the values associated with these k output gates. All parties hold $[\![\boldsymbol{x}]\!]_{n-k}$. Let $[\![\boldsymbol{r}]\!]_{n-k}, ([\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta}*\boldsymbol{r}]_{n-k}), [\boldsymbol{o}]_{n-1}$ be the random sharings prepared for these k output gates in $\mathcal{F}_{\text{prep-mal}}$. To reconstruct the secret $\boldsymbol{x}$ to the client:

1. All parties first invoke $\mathcal{F}_{\text{com}}$ to commit their shares of $([\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta}*\boldsymbol{r}]_{n-k})$ to the client.
2. All parties send to P_1 their shares of $[\boldsymbol{x}+\boldsymbol{r}]_{n-1} := [\boldsymbol{o}]_{n-1} + [\boldsymbol{x}]_{n-k} + [\boldsymbol{r}]_{n-k}$.
3. P_1 reconstructs $\boldsymbol{x}+\boldsymbol{r}$ and distributes a degree-$(2k-2)$ packed Shamir sharing $[\boldsymbol{x}+\boldsymbol{r}]_{2k-2}$.

Before reconstructing the secret to the client, all parties verify the correctness of the computation.

Verification of the Computation. To check the correctness of the computation, it is sufficient to verify the following points:

Functionality 20: $\mathcal{F}_{\text{com}}$

Public Commitment:

1. On input $(\texttt{Commit}, v, i, \tau_v)$ by P_i, $\mathcal{F}_{\text{com}}$ stores (v, i, τ_v) and outputs (i, τ_v) to all parties, where τ_v represents a handle for the commitment.
2. On input $(\texttt{Open}, i, \tau_v)$ by P_i, $\mathcal{F}_{\text{com}}$ outputs (v, i, τ_v) to all parties if exists, or `abort` otherwise.

Private Commitment:

1. On input $(\texttt{Commit}, v, i, R, \tau_v)$ by P_i, $\mathcal{F}_{\text{com}}$ stores (v, i, R, τ_v) and outputs (i, R, τ_v) to the receiver R.
2. On input $(\texttt{Open}, i, R, \tau_v)$ by P_i, $\mathcal{F}_{\text{com}}$ outputs (v, i, R, τ_v) to R if exists, or `abort` otherwise.

1. For Input Phase:
   - Each degree-$(k-1)$ packed Shamir sharing distributed by a client is a valid degree-$(k-1)$ packed Shamir sharing.
   - All parties obtain correct $[\![\boldsymbol{x}\|\texttt{pos}]\!]_{n-k}$ from $[\![\boldsymbol{x}]\!]_{n-1}$ when using Tran.
2. For Network Routing: All parties obtain correct $[\![\boldsymbol{x}]\!]_{n-k}$ from $[\![\boldsymbol{x}'\|\texttt{pos}]\!]_{n-1}$ when using Tran.
3. For Addition Gates: All parties obtain a correct output sharing $[\![\boldsymbol{z}\|\texttt{pos}]\!]_{n-k}$ from $[\![\boldsymbol{z}]\!]_{n-k}$ when using Tran.
4. For Multiplication Gates:
   - $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}, [(\boldsymbol{x}+\boldsymbol{a})\cdot(\boldsymbol{y}+\boldsymbol{b})]_{k-1}$ are valid degree-$(k-1)$ packed Shamir sharings. The first two secrets are identical to the secrets of $[\![\boldsymbol{x}+\boldsymbol{a}]\!]_{n-k} = [\![\boldsymbol{x}]\!]_{n-k} + [\![\boldsymbol{a}]\!]_{n-k}$ and $[\![\boldsymbol{y}+\boldsymbol{b}]\!]_{n-k} = [\![\boldsymbol{y}]\!]_{n-k} + [\![\boldsymbol{b}]\!]_{n-k}$, and the third secret is equal to the coordinate-wise multiplication between the first two secrets.
   - All parties obtain a correct output sharing $[\![\boldsymbol{z}\|\texttt{pos}]\!]_{n-k}$ from $[\![\boldsymbol{z}]\!]_{n-1}$ when using Tran.
5. For Output Gates: The secret of $[\boldsymbol{x}+\boldsymbol{a}]_{2k-2}$ is identical to the secret of $[\![\boldsymbol{x}+\boldsymbol{r}]\!]_{n-k} = [\![\boldsymbol{x}]\!]_{n-k} + [\![\boldsymbol{r}]\!]_{n-k}$.

We will first verify that all degree-$(k-1)$ packed Shamir sharings are valid.

Verification of Degree-$(k-1)$ Packed Shamir Sharings. The verification is done by computing a random linear combination of all degree-$(k-1)$ packed Shamir sharings and checking the correctness of the resulting degree-$(k-1)$ packed Shamir sharing by each party. Note that we do not need to protect the secrecy of these sharings since they are generated by P_1 or a client, who can be corrupted. The description of Check-Consistency appears in Protocol 21. Recall that the cost of $\mathcal{F}_{\text{com}}$ is $2n^2+n$ elements of preprocessing data and $4n^2$ elements of communication. The total cost of Check-Consistency is $2n^3+n^2$ elements of preprocessing data and $4n^3+n^2$ elements of communication.

Lemma 6. *If there exists $i \in \{1, 2, \ldots, m\}$ such that the shares of $[\boldsymbol{x}^{(i)}]_{k-1}$ of honest parties do not correspond to a valid degree-$(k-1)$ packed Shamir sharing, with probability at least $1 - m/|\mathbb{F}|$, all honest parties abort in* Check-Consistency.

Protocol 21: CHECK-CONSISTENCY

1. Let m be the number of degree-$(k-1)$ packed Shamir sharings that all parties need to check. These m sharings are denoted by
$$[\boldsymbol{x}^{(1)}]_{k-1}, [\boldsymbol{x}^{(2)}]_{k-1}, \ldots, [\boldsymbol{x}^{(m)}]_{k-1}.$$
2. Each party P_i samples a random field element $\lambda_i \in \mathbb{F}$ and invokes $\mathcal{F}_{\text{com}}$ with $(\texttt{Commit}, \lambda_i, i, \tau_{\lambda_i})$.
3. Each party P_i invokes $\mathcal{F}_{\text{com}}$ with $(\texttt{Open}, i, \tau_{\lambda_i})$. All parties set $\lambda = \lambda_1 + \lambda_2 + \ldots + \lambda_n$.
4. All parties locally compute
$$[\boldsymbol{x}]_{k-1} = [\boldsymbol{x}^{(1)}]_{k-1} + [\boldsymbol{x}^{(2)}]_{k-1} \cdot \lambda + \ldots + [\boldsymbol{x}^{(m)}]_{k-1} \cdot \lambda^{m-1}.$$
5. All parties send their shares of $[\boldsymbol{x}]_{k-1}$ to all other parties. Then each party P_i checks whether the shares of $[\boldsymbol{x}]_{k-1}$ form a valid degree-$(k-1)$ packed Shamir sharing. If not, P_i aborts. Otherwise, P_i accepts.

Proof. By $\mathcal{F}_{\text{com}}$, all parties obtain a uniform element λ in Step 3. Consider the polynomial
$$[\boldsymbol{F}(\lambda)]_{k-1} = [\boldsymbol{x}^{(1)}]_{k-1} + [\boldsymbol{x}^{(2)}]_{k-1} \cdot \lambda + \ldots + [\boldsymbol{x}^{(m)}]_{k-1} \cdot \lambda^{m-1}.$$
By Lagrange interpolation, for any m different points $\lambda^{(1)}, \lambda^{(2)}, \ldots, \lambda^{(m)}$, there is a one-to-one linear map from $\{[\boldsymbol{F}(\lambda^{(i)})]_{k-1}\}_{i=1}^m$ to $\{[\boldsymbol{x}^{(i)}]_{k-1}\}_{i=1}^m$. Thus, if there exists $i \in \{1, 2, \ldots, m\}$ such that the shares of $[\boldsymbol{x}^{(i)}]_{k-1}$ of honest parties do not correspond to a valid degree-$(k-1)$ packed Shamir sharing, then the number of $\lambda \in \mathbb{F}$ such that the shares of $[\boldsymbol{F}(\lambda)]_{k-1}$ of honest parties correspond to a valid degree-$(k-1)$ packed Shamir sharing is bounded by $m-1$. Since λ is a uniform element in $\mathbb{F}$, the probability of sampling such a λ is bounded by $(m-1)/|\mathbb{F}| \leq m/|\mathbb{F}|$. Note that for every λ where the shares of $[\boldsymbol{F}(\lambda)]_{k-1}$ of honest parties do not correspond to a valid degree-$(k-1)$ packed Shamir sharing, all honest parties will abort since they will always receive the correct shares of honest parties. Thus, the lemma holds. □

Verification of the Secrets of Degree-$(2k-2)$ Packed Shamir Sharings. In this part, for a pair of sharings $([\boldsymbol{x}]_{2k-2}, [\![\boldsymbol{x}]\!]_{n-k})$, we want to verify that the secret of $[\boldsymbol{x}]_{2k-2}$ is identical to that authenticated in $[\![\boldsymbol{x}]\!]_{n-k}$. We assume that we do not need to protect the privacy of the secret.

This can be used to verify the secrets of $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ for multiplication gates (note that we can always view a degree-$(k-1)$ packed Shamir sharing as a degree-$(2k-2)$ packed Shamir sharing), and the secret of $[\boldsymbol{x}+\boldsymbol{r}]_{2k-2}$ for output gates. Also for the correctness of $[(\boldsymbol{x}+\boldsymbol{a})\cdot(\boldsymbol{y}+\boldsymbol{b})]_{k-1}$, note that $[(\boldsymbol{x}+\boldsymbol{a})\cdot(\boldsymbol{y}+\boldsymbol{b})]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ should be a degree-$(2k-2)$ packed Shamir sharing of $\boldsymbol{0}$. We can also verify the secret of $[(\boldsymbol{x}+\boldsymbol{a})\cdot(\boldsymbol{y}+\boldsymbol{b})]_{k-1}$.

At a high-level, we simply compute a random linear combination of all pairs of random sharings and then reconstruct the resulting pair of sharings. The verification protocol will use the two degree-$(n-1)$ packed Shamir sharings of $\boldsymbol{0}$, $[\boldsymbol{o}^{(1)}]_{n-1}$ and $[\boldsymbol{o}^{(2)}]_{n-1}$, prepared in $\mathcal{F}_{\text{prep-mal}}$. The description of CHECK-SECRET appears in Protocol 22. Recall that the cost of $\mathcal{F}_{\text{com}}$ is $2n^2+n$ elements of preprocessing data and $4n^2$ elements of communication. The total cost of CHECK-SECRET is $8n^3+4n^2$ elements of preprocessing data and $16n^3+n^2$ elements of communication.

Informally, the effectiveness of CHECK-SECRET comes from the following two points:

- For all $i \in \{1, 2, \ldots, m\}$, corrupted parties cannot change the secret of $[\boldsymbol{u}^{(i)}]_{2k-2}$ and the secret authenticated by $[\![\boldsymbol{u}^{(i)}]\!]_{n-k}$. The former is because the secret of a degree-$(2k-2)$ packed Shamir sharing is

Protocol 22: CHECK-SECRET

1. Let $\{([\boldsymbol{u}^{(i)}]_{2k-2}, [\![\boldsymbol{u}^{(i)}]\!]_{n-k})\}_{i=1}^{m}$ be the m pairs of sharings that all parties want to check. Let $[\boldsymbol{o}^{(1)}]_{n-1}, [\boldsymbol{o}^{(2)}]_{n-1}$ be the random degree-$(n-1)$ packed Shamir sharings of $\mathbf{0}$ prepared in $\mathcal{F}_{\text{prep-mal}}$, and $[\boldsymbol{\gamma}]_{n-k}$ be the degree-$(n-k)$ packed Shamir sharing of the authentication key prepared in $\mathcal{F}_{\text{prep-mal}}$.
2. Each party P_i samples a random field element $\lambda_i \in \mathbb{F}$ and invokes $\mathcal{F}_{\text{com}}$ with $(\texttt{Commit}, \lambda_i, i, \tau_{\lambda_i})$.
3. Each party P_i invokes $\mathcal{F}_{\text{com}}$ with $(\texttt{Open}, i, \tau_{\lambda_i})$. All parties set $\lambda = \lambda_1 + \lambda_2 + \ldots + \lambda_n$.
4. All parties locally compute

$$\begin{aligned}
[\boldsymbol{u}]_{2k-2} &= [\boldsymbol{u}^{(1)}]_{2k-2} + [\boldsymbol{u}^{(2)}]_{2k-2} \cdot \lambda + \ldots + [\boldsymbol{u}^{(m)}]_{2k-2} \cdot \lambda^{m-1}, \\
[\boldsymbol{u}]_{n-1} &= [\boldsymbol{o}^{(1)}]_{n-1} + [\boldsymbol{u}^{(1)}]_{n-k} + [\boldsymbol{u}^{(2)}]_{n-k} \cdot \lambda + \ldots + [\boldsymbol{u}^{(m)}]_{n-k} \cdot \lambda^{m-1}, \\
[\boldsymbol{\gamma} * \boldsymbol{u}]_{n-1} &= [\boldsymbol{o}^{(2)}]_{n-1} + [\boldsymbol{\gamma} * \boldsymbol{u}^{(1)}]_{n-k} + [\boldsymbol{\gamma} * \boldsymbol{u}^{(2)}]_{n-k} \cdot \lambda + \ldots \\
&\quad + [\boldsymbol{\gamma} * \boldsymbol{u}^{(m)}]_{n-k} \cdot \lambda^{m-1}.
\end{aligned}$$

5. All parties send their shares of $[\boldsymbol{u}]_{2k-2}$ to all other parties. All parties also commit their shares of $[\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma} * \boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}]_{n-k}$ using $\mathcal{F}_{\text{com}}$.
6. All parties open their shares of $[\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma} * \boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}]_{n-k}$ using $\mathcal{F}_{\text{com}}$. Then each party P_i checks the following:
   (a) The shares of $[\boldsymbol{u}]_{2k-2}$ form a valid degree-$(2k-2)$ packed Shamir sharing.
   (b) The shares of $[\boldsymbol{\gamma}]_{n-k}$ form a valid degree-$(n-k)$ packed Shamir sharing and the secret $\boldsymbol{\gamma}$ is in the form of $(\gamma, \gamma, \ldots, \gamma) \in \mathbb{F}^k$.
   (c) The secrets of $[\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma} * \boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}]_{n-k}$ satisfy that the second secret is equal to the coordinate-wise multiplication of the first secret and the third secret.
   (d) The secret of $[\boldsymbol{u}]_{2k-2}$ is the same as the secret of $[\boldsymbol{u}]_{n-1}$.

   If all checks pass, P_i accepts. Otherwise, P_i aborts.

determined by the shares of honest parties. The latter is due to the security of the information-theoretic MAC.

- If there exists $i \in \{1, 2, \ldots, m\}$ such that the secret of $[\boldsymbol{u}^{(i)}]_{2k-2}$ is different from the secret authenticated by $[\![\boldsymbol{u}^{(i)}]\!]_{n-k}$, with overwhelming probability (when the underlying field $\mathbb{F}$ is large enough), the secret of $[\boldsymbol{u}]_{2k-2}$ is different from the secret authenticated by $[\![\boldsymbol{u}]\!]_{n-1} = ([\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma} * \boldsymbol{u}]_{n-1})$.

The formal argument is deferred to the final proof of the whole protocol.

Security of TRAN. In the final proof of the whole protocol, we will show that what an adversary can do in TRAN is to insert an additive error to the secret of the resulting sharing. Since we obtain $[\![f(\boldsymbol{x}) \| \texttt{pos}]\!]_{n-k}$ from $[\![\boldsymbol{x} \| \texttt{pos}]\!]_{n-1}$ by invoking TRAN on $[\boldsymbol{x} \| \texttt{pos}]_{n-1}$ and $[\boldsymbol{\gamma} * \boldsymbol{x} \| \texttt{pos}]_{n-1}$ separately, by the security of the information-theoretic MAC, any additive errors inserted by the adversary will lead to an invalid authenticated degree-$(n-k)$ packed Shamir sharing $[\![f(\boldsymbol{x}) \| \texttt{pos}]\!]_{n-k}$ with overwhelming probability. Such attacks can be detected when verifying that the reconstruction of the authenticated sharings when evaluating multiplication gates and output gates are correct, which is covered by Protocol CHECK-SECRET.

Reconstructing the Output. After all parties verify the computation, they reconstruct the function output to the clients. For a group of k output gates belonging to the same client, let $\boldsymbol{x} \in \mathbb{F}^k$ be the values associated with these k output gates. All parties hold $[\![\boldsymbol{x}]\!]_{n-k}$. Let $[\![\boldsymbol{r}]\!]_{n-k}, ([\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}), [\boldsymbol{o}]_{n-1}$ be the random sharings prepared for these k output gates in $\mathcal{F}_{\text{prep-mal}}$. Recall that all parties have committed their shares of $([\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k})$ to the client and obtained a degree-$(2k-2)$ packed Shamir sharing $[\boldsymbol{x} + \boldsymbol{r}]_{2k-2}$. To reconstruct the secret $\boldsymbol{x}$ to the client:

1. All parties send their shares of $[\boldsymbol{x} + \boldsymbol{r}]_{2k-2}$ to the client to allow him to reconstruct the secret $\boldsymbol{x} + \boldsymbol{r}$.
2. All parties open their commitments of the shares of $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ to the client to allow him to reconstruct and verify the secret $\boldsymbol{r}$.

Finally, the client can compute $\boldsymbol{x} = (\boldsymbol{x} + \boldsymbol{r}) - \boldsymbol{r}$. The description of Output-Mal appears in Protocol 23.

Protocol 23: Output-Mal

1. Suppose $\boldsymbol{x} \in \mathbb{F}^k$ is the output associated with the output gates which belongs to Client. Let $[\![\boldsymbol{r}]\!]_{n-k}, ([\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}), [\boldsymbol{o}]_{n-1}$ be the random sharings prepared for these output gates in $\mathcal{F}_{\text{prep-mal}}$. Recall that $[\![\boldsymbol{r}]\!]_{n-k} = ([\boldsymbol{r}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k})$. All parties hold $[\boldsymbol{x} + \boldsymbol{r}]_{2k-2}$ at the beginning of the protocol and have committed their shares of $([\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k})$ to Client using $\mathcal{F}_{\text{com}}$.
2. All parties send their shares of $[\boldsymbol{x} + \boldsymbol{r}]_{2k-2}$ and open their commitments of shares of $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ to Client using $\mathcal{F}_{\text{com}}$.
3. Client checks whether the shares of $[\boldsymbol{x} + \boldsymbol{r}]_{2k-2}$ form a valid degree-$(2k-2)$ packed Shamir sharing. If not, Client aborts. Otherwise, Client reconstructs the secret $\boldsymbol{x} + \boldsymbol{r}$.
4. Client checks whether the shares of $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ form valid degree-$(n-k)$ packed Shamir sharings. If not, Client aborts. Otherwise, Client reconstructs the secret $\boldsymbol{r}, \boldsymbol{\Delta}, \boldsymbol{\Delta} * \boldsymbol{r}$ and checks the MAC of $\boldsymbol{r}$, i.e., whether the third secret is equal to the coordinate-wise multiplication of the first two secrets. If not, Client aborts.
5. If all checks pass, Client computes $\boldsymbol{x} = (\boldsymbol{x} + \boldsymbol{r}) - \boldsymbol{r}$.

6.4 Main Protocol

Now we are ready to introduce the main protocol. The ideal functionality $\mathcal{F}_{\text{main-mal}}$ appears in Functionality 24. The description of Main-Mal appears in Protocol 25.

Lemma 7. *For all $k \leq (n+2)/3$, Protocol* Main-Mal *securely computes $\mathcal{F}_{main\text{-}mal}$ in the $\{\mathcal{F}_{prep\text{-}mal}, \mathcal{F}_{rand\text{-}sharing\text{-}mal}, \mathcal{F}_{com}\}$-hybrid model against a fully malicious adversary who controls $t = n - 2k + 1$ parties.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}orr$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties.

Throughout the simulator, $\mathcal{S}$ will keep tracking the shares that should be held by corrupted parties. We will maintain the invariant that for all degree-$(2k-2)$ packed Shamir sharings (including degree-$(k-1)$ packed Shamir sharings), $\mathcal{S}$ will learn the whole sharings and their secrets. For every authenticated degree-$(n-k)$ packed Shamir sharing $[\![\boldsymbol{x} \| \mathsf{pos}]\!]_{n-k}$, $\mathcal{S}$ will compute a pair of vectors, which are the additive errors to the secret and its MAC due to the behaviors of corrupted parties.

Functionality 24: $\mathcal{F}_{\text{main-mal}}$

1. $\mathcal{F}_{\text{main-mal}}$ receives the input from all clients. Let x denote the input and C denote the circuit.
2. $\mathcal{F}_{\text{main-mal}}$ computes $C(x)$ and sends the output of corrupted clients to the adversary. $\mathcal{F}_{\text{main-mal}}$ waits for the response of the adversary.
   - If the adversary replies `abort`, $\mathcal{F}_{\text{main-mal}}$ sends `abort` to all clients.
   - If the adversary replies `continue`, $\mathcal{F}_{\text{main-mal}}$ distributes the output to all clients.

Simulation of Main-Mal. In the preprocessing phase, $\mathcal{S}$ emulates the functionality $\mathcal{F}_{\text{prep-mal}}$ and receives the shares of corrupted parties.

Simulating the Input Phase. In Step 3, for every group of k input gates of Client_i, $\mathcal{S}$ simulates Input-Mal. In Step 1 of Input-Mal, recall that $\mathcal{S}$ has learnt the shares of $[\![\boldsymbol{r}]\!]_{n-k}, ([\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k})$ of corrupted parties when emulating $\mathcal{F}_{\text{prep-mal}}$. Starting from Step 2 of Input-Mal, there are two cases:

- If Client_i is honest, in Step 2, $\mathcal{S}$ receives the shares of $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ of corrupted parties.

  In Step 3, for each sharing $[\boldsymbol{p}]_{n-k} \in \{[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}\}$, $\mathcal{S}$ computes a sharing $[\delta(\boldsymbol{p})]_{n-k}$ which is defined as follows:

  1. The shares of all honest parties are set to be 0.
  2. For each corrupted party P_j, the j-th share is equal to the j-th share of $[\boldsymbol{p}]_{n-k}$ received from P_j minus the same share that P_j should hold.

  $\mathcal{S}$ checks whether $[\delta(\boldsymbol{p})]_{n-k}$ is a valid degree-$(n-k)$ packed Shamir sharing of $\mathbf{0}$. If not, $\mathcal{S}$ aborts on behalf of Client_i.

  Otherwise, in Step 4, $\mathcal{S}$ generates a random vector as $\boldsymbol{x} + \boldsymbol{r}$ and honestly follows the protocol by generating a random degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{x} + \boldsymbol{r}]_{k-1}$ and distributing the shares to corrupted parties.

  In Step 5, $\mathcal{S}$ computes the shares of $[\![\boldsymbol{x}]\!]_{n-1}$ held by corrupted parties.

- If Client_i is corrupted, in Step 2, $\mathcal{S}$ samples two random vectors $\boldsymbol{r}, \boldsymbol{\Delta}$ and computes the whole sharings $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$. Then $\mathcal{S}$ sends the shares of honest parties to Client_i on behalf of honest parties.

  In Step 4, $\mathcal{S}$ receives from Client_i the shares of $[\boldsymbol{x} + \boldsymbol{r}]_{k-1}$ of honest parties. $\mathcal{S}$ checks whether the shares of $[\boldsymbol{x} + \boldsymbol{r}]_{k-1}$ held by honest parties form a valid degree-$(k-1)$ packed Shamir sharing. If true, $\mathcal{S}$ recovers the whole sharing $[\boldsymbol{x} + \boldsymbol{r}]_{k-1}$, reconstructs the secret of $[\boldsymbol{x} + \boldsymbol{r}]_{k-1}$, and computes $\boldsymbol{x} = (\boldsymbol{x} + \boldsymbol{r}) - \boldsymbol{r}$. Otherwise, $\mathcal{S}$ sets the shares of $[\boldsymbol{x} + \boldsymbol{r}]_{k-1}$ of corrupted parties to be all 0 and sets $\boldsymbol{x} = \mathbf{0}$. In this case, $\mathcal{S}$ will abort during the verification of degree-$(k-1)$ packed Shamir sharings.

  In Step 5, $\mathcal{S}$ computes the shares of $[\![\boldsymbol{x}]\!]_{n-1}$ held by corrupted parties.

In Step 6 of Input Mal, $\mathcal{S}$ simulates Tran-Mal. The simulation is done as follows:

1. In Step 2, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}_{\text{rand-sharing-mal}}$ and receives the shares of corrupted parties and the additive error $\boldsymbol{d}$. Suppose these two sharings are denoted by $([\boldsymbol{r} \| \mathtt{pos}]_{n-1}, [f(\boldsymbol{r}) + \boldsymbol{d} \| \mathtt{pos}']_{n-k})$.
2. In Step 3, $\mathcal{S}$ generates a random degree-$(n-1)$ packed Shamir sharing as $[\boldsymbol{x} + \boldsymbol{r} \| \mathtt{pos}]_{n-1}$ based on the shares held by corrupted parties. $\mathcal{S}$ reconstructs the secret $\boldsymbol{x} + \boldsymbol{r}$ and computes $f(\boldsymbol{x} + \boldsymbol{r})$.

Protocol 25: MAIN-MAL

1. **Preprocessing Phase**. All parties invoke $\mathcal{F}_{\text{prep-mal}}$ to prepare correlated randomness for the online phase.

2. **Initialization**. Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be distinct field elements in $\mathbb{F}$ which are used for the shares of all parties in packed Shamir secret sharing schemes. Let $\beta_1, \beta_2, \ldots, \beta_{|C|}$ be $|C|$ distinct field elements that are different from $\alpha_1, \alpha_2, \ldots, \alpha_n$. Let $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_k)$ be the default positions for the packed Shamir secret sharing schemes. We associate the field element β_i with the i-th gate in C.

3. **Input Phase**. Let $\mathsf{Client}_1, \mathsf{Client}_2, \ldots, \mathsf{Client}_\mathsf{c}$ denote the clients who provide inputs. All input gates are divided into groups of size k based on the input holders. For every group of k input gates of Client_i, suppose $\boldsymbol{x}$ are the inputs, and $\mathtt{pos} = (p_1, p_2, \ldots, p_k)$ are the positions associated with these k gates. All parties and Client_i invokes INPUT-MAL to obtain $[\![\boldsymbol{x} \| \mathtt{pos}]\!]_{n-k}$.

4. **Evaluation Phase**. All parties evaluate the circuit layer by layer as follows:

   (a) For the current layer, all gates are divided into groups of size k based on their types (i.e., multiplication gates or addition gates). For each group of k gates, let $\boldsymbol{x}$ be the first inputs of all gates, and $\boldsymbol{y}$ the second inputs of all gates. All parties invoke NETWORK-MAL to prepare $[\![\boldsymbol{x}]\!]_{n-k}$ and $[\![\boldsymbol{y}]\!]_{n-k}$.

   (b) For each group of k gates, let $\mathtt{pos}$ be the positions associated with these k gates.

   - If they are addition gates, all parties invoke ADD-MAL on $([\![\boldsymbol{x}]\!]_{n-k}, [\![\boldsymbol{y}]\!]_{n-k})$, and obtain $[\![\boldsymbol{z} \| \mathtt{pos}]\!]_{n-k}$.
   - If they are multiplication gates, all parties invoke MULT-MAL on $([\![\boldsymbol{x}]\!]_{n-k}, [\![\boldsymbol{y}]\!]_{n-k})$ and obtain $[\![\boldsymbol{z} \| \mathtt{pos}]\!]_{n-k}$.

5. **Output Phase**. All output gates are divided into groups of size k based on the output receiver. For every group of k output gates of Client_i, suppose $\boldsymbol{x}$ are the inputs. All parties invoke the protocol NETWORK-MAL to prepare $[\![\boldsymbol{x}]\!]_{n-k}$. Then, all parties run the following steps.

   (a) Let $[\![\boldsymbol{r}]\!]_{n-k}, ([\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}), [\boldsymbol{o}]_{n-1}$ be the random sharings prepared for these k output gates in $\mathcal{F}_{\text{prep-mal}}$. All parties invoke $\mathcal{F}_{\text{com}}$ to commit their shares of $([\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k})$ to Client_i.

   (b) All parties send to P_1 their shares of $[\boldsymbol{x} + \boldsymbol{r}]_{n-1} := [\boldsymbol{o}]_{n-1} + [\boldsymbol{x}]_{n-k} + [\boldsymbol{r}]_{n-k}$.

   (c) P_1 reconstructs $\boldsymbol{x} + \boldsymbol{r}$ and distributes a degree-$(2k-2)$ packed Shamir sharing $[\boldsymbol{x} + \boldsymbol{r}]_{2k-2}$.

   Now all parties verify the computation. (1) All parties invoke CHECK-CONSISTENCY to check the correctness of all degree-$(k-1)$ packed Shamir sharings. (2) All parties invoke CHECK-SECRET to check the correctness of the secrets of the degree-$(2k-2)$ (including degree-$(k-1)$) packed Shamir sharings generated by P_1.

   If all parties accept the verification, all parties and Client_i invoke OUTPUT to reconstruct the output $\boldsymbol{x}$ to Client_i.

3. In Step 4, $\mathcal{S}$ honestly follows the protocol and learns the shares of $[f(\boldsymbol{x}+\boldsymbol{r}) \| \mathtt{pos}']_{2k-2}$ of honest parties. Since it is a degree-$(2k-2)$ packed Shamir sharing, $\mathcal{S}$ recovers the whole sharing by using honest parties' shares. Then $\mathcal{S}$ reconstructs the secret $\overline{f(\boldsymbol{x} + \boldsymbol{r})}$ (to distinguish from the correct secret).

4. In Step 5, $\mathcal{S}$ computes the shares of the resulting sharing held by corrupted parties. Note that the resulting sharing has secret $\overline{f(\boldsymbol{x}+\boldsymbol{r})} - (f(\boldsymbol{r})+\boldsymbol{d})$. On the other hand, the correct secret should be $f(\boldsymbol{x})$. Therefore, effectively, the adversary inserts an additive error

$$\boldsymbol{\delta} = (\overline{f(\boldsymbol{x}+\boldsymbol{r})} - (f(\boldsymbol{r})+\boldsymbol{d})) - f(\boldsymbol{x}) = \overline{f(\boldsymbol{x}+\boldsymbol{r})} - f(\boldsymbol{x}+\boldsymbol{r}) - \boldsymbol{d}.$$

   Note that $\mathcal{S}$ learns $\overline{f(\boldsymbol{x}+\boldsymbol{r})}, f(\boldsymbol{x}+\boldsymbol{r}), \boldsymbol{d}$. $\mathcal{S}$ computes the error $\boldsymbol{\delta}$.

Let $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2$ be the additive errors in the invocation of Tran-Mal on $[\boldsymbol{x}]_{n-1}$ and the invocation of Tran-Mal on $[\boldsymbol{\gamma} * \boldsymbol{x}]_{n-1}$. We associate $(\boldsymbol{\delta}_1, \boldsymbol{\delta}_2)$ with the authenticated sharing $[\![\boldsymbol{x} \| \mathtt{pos}]\!]_{n-k}$.

$\mathcal{S}$ invokes the ideal functionality $\mathcal{F}_{\text{main-mal}}$ and sends to $\mathcal{F}_{\text{main-mal}}$ the input of corrupted parties extracted above. Then $\mathcal{S}$ receives the output of corrupted parties from $\mathcal{F}_{\text{main-mal}}$.

Simulating the Evaluation Phase. In Step 4.(a), $\mathcal{S}$ simulates Network-Mal.

1. The first 5 steps only involve local computation. $\mathcal{S}$ computes the shares of $([\boldsymbol{x}' \| \mathtt{pos}]_{n-1}, [\boldsymbol{\gamma} * \boldsymbol{x}' \| \mathtt{pos}]_{n-1})$ held by corrupted parties. For all $i \in \{1, 2, \ldots, \ell_1\}$, each value x'_i is in the authenticated degree-$(n-k)$ packed Shamir sharing $[\![\boldsymbol{x}^{(i)} \| \mathtt{pos}^{(i)}]\!]_{n-k}$. Recall that for every authenticated degree-$(n-k)$ packed Shamir sharing, $\mathcal{S}$ has computed a pair of vectors, which are the additive errors to the secret and its MAC due to the behaviors of corrupted parties. $\mathcal{S}$ extracts the errors associated with the value x'_i and its MAC for all $i \in \{1, 2, \ldots, \ell_1\}$. Then, for all $i > \ell_1$, $\mathcal{S}$ sets the errors associated with the value x'_i and its MAC to be 0. In this way, $\mathcal{S}$ obtains two vectors $(\boldsymbol{\delta}'_1, \boldsymbol{\delta}'_2)$, which are the additive errors to the secrets of $([\boldsymbol{x}' \| \mathtt{pos}]_{n-1}, [\boldsymbol{\gamma} * \boldsymbol{x}' \| \mathtt{pos}]_{n-1})$.

2. In Step 6, $\mathcal{S}$ simulates Tran-Mal as described above, and extracts the additive errors $\boldsymbol{\delta}''_1, \boldsymbol{\delta}''_2$ in the invocation of Tran-Mal on $[\boldsymbol{x}' \| \mathtt{pos}]_{n-1}$ and the invocation of Tran-Mal on $[\boldsymbol{\gamma} * \boldsymbol{x}' \| \mathtt{pos}]_{n-1}$. Then the additive errors associated with the authenticated sharing $[\![\boldsymbol{x}]\!]_{n-k}$ are $(f(\boldsymbol{\delta}'_1) + \boldsymbol{\delta}''_1, f(\boldsymbol{\delta}'_2) + \boldsymbol{\delta}''_2)$.

In Step 4.(b), for each group of addition gates with input sharings $[\![\boldsymbol{x}]\!]_{n-k}$ and $[\![\boldsymbol{y}]\!]_{n-k}$, $\mathcal{S}$ simulates Add-Mal.

1. In Step 2, $\mathcal{S}$ computes the shares of $[\![\boldsymbol{z}]\!]_{n-k}$ held by corrupted parties. The vector of additive errors associated with $[\![\boldsymbol{z}]\!]_{n-k}$ are the sum of the vectors of additive errors associated with $[\![\boldsymbol{x}]\!]_{n-k}$ and $[\![\boldsymbol{y}]\!]_{n-k}$. Suppose the vector of additive errors associated with $[\![\boldsymbol{z}]\!]_{n-k}$ is denoted by $(\boldsymbol{\delta}'_1, \boldsymbol{\delta}'_2)$.

2. In Step 3, $\mathcal{S}$ simulates Tran-Mal as described above, and extracts the additive errors $\boldsymbol{\delta}''_1, \boldsymbol{\delta}''_2$ in the invocation of Tran-Mal on $[\boldsymbol{z}]_{n-k}$ and the invocation of Tran-Mal on $[\boldsymbol{\gamma} * \boldsymbol{z}]_{n-k}$. Then the additive errors associated with the authenticated sharing $[\![\boldsymbol{z} \| \mathtt{pos}]\!]_{n-k}$ are $(\boldsymbol{\delta}'_1 + \boldsymbol{\delta}''_1, \boldsymbol{\delta}'_2 + \boldsymbol{\delta}''_2)$.

For each group of multiplication gates with input sharings $[\![\boldsymbol{x}]\!]_{n-k}$ and $[\![\boldsymbol{y}]\!]_{n-k}$, $\mathcal{S}$ simulates Mult-Mal.

1. In Step 2 of Mult-Mal, $\mathcal{S}$ computes the shares of $[\boldsymbol{x}+\boldsymbol{a}]_{n-1}, [\boldsymbol{y}+\boldsymbol{b}]_{n-1}$ of corrupted parties and sets the shares of $[\boldsymbol{x}+\boldsymbol{a}]_{n-1}, [\boldsymbol{y}+\boldsymbol{b}]_{n-1}$ of honest parties to be uniform elements. Then $\mathcal{S}$ reconstructs the secrets $\boldsymbol{x}+\boldsymbol{a}$ and $\boldsymbol{y}+\boldsymbol{b}$.

2. In Step 3 of Mult-Mal, since the whole sharings $[\boldsymbol{x}+\boldsymbol{a}]_{n-1}, [\boldsymbol{y}+\boldsymbol{b}]_{n-1}$ have been generated, $\mathcal{S}$ honestly follows the protocol. At the end of Step 3 of Mult-Mal, $\mathcal{S}$ receives the shares of $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}, [(\boldsymbol{x}+\boldsymbol{a}) \cdot (\boldsymbol{y}+\boldsymbol{b})]_{k-1}$ of honest parties from P_1 (which is either honestly simulated by $\mathcal{S}$ or controlled by the adversary). $\mathcal{S}$ checks whether the shares of $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}, [(\boldsymbol{x}+\boldsymbol{a}) \cdot (\boldsymbol{y}+\boldsymbol{b})]_{k-1}$ of honest parties form valid degree-$(k-1)$ packed Shamir sharings.

   - If true, $\mathcal{S}$ recovers the whole sharings $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}, [(\boldsymbol{x}+\boldsymbol{a}) \cdot (\boldsymbol{y}+\boldsymbol{b})]_{k-1}$ using the shares of honest parties and then reconstructs the secrets $\overline{\boldsymbol{x}+\boldsymbol{a}}, \overline{\boldsymbol{y}+\boldsymbol{b}}, \overline{(\boldsymbol{x}+\boldsymbol{a}) \cdot (\boldsymbol{y}+\boldsymbol{b})}$. Then $\mathcal{S}$ computes three vectors

$$\begin{aligned}
\boldsymbol{\eta}^{(1)} &= (\overline{\boldsymbol{x}+\boldsymbol{a}}) - (\boldsymbol{x}+\boldsymbol{a}) \\
\boldsymbol{\eta}^{(2)} &= (\overline{\boldsymbol{y}+\boldsymbol{b}}) - (\boldsymbol{y}+\boldsymbol{b}) \\
\boldsymbol{\eta}^{(3)} &= \overline{(\boldsymbol{x}+\boldsymbol{a}) \cdot (\boldsymbol{y}+\boldsymbol{b})} - (\overline{\boldsymbol{x}+\boldsymbol{a}}) \cdot (\overline{\boldsymbol{y}+\boldsymbol{b}})
\end{aligned}$$

- Otherwise, $\mathcal{S}$ sets the shares of $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ of corrupted parties to be all 0. In this case, $\mathcal{S}$ will abort during the verification of degree-$(k-1)$ packed Shamir sharings.

3. In Step 4 of Mult-Mal, $\mathcal{S}$ computes the shares of $[\boldsymbol{z}]_{n-1}, [\boldsymbol{\gamma} * \boldsymbol{z}]_{n-1}$ of corrupted parties.

4. In Step 5, $\mathcal{S}$ simulates Tran-Mal as described above, and extracts the additive errors $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2$ in the invocation of Tran-Mal on $[\boldsymbol{z}]_{n-1}$ and the invocation of Tran-Mal on $[\boldsymbol{\gamma} * \boldsymbol{z}]_{n-1}$. Then the additive errors associated with the authenticated sharing $[\![\boldsymbol{z} \| \mathtt{pos}]\!]_{n-k}$ are $(\boldsymbol{\delta}_1, \boldsymbol{\delta}_2)$.

Simulating the Output Phase. For every group of k output gates of Client_i, suppose $\boldsymbol{x}$ are the inputs. $\mathcal{S}$ first simulates Network-Mal as described above and compute the additive errors associated with the authenticated degree-$(n-k)$ Shamir sharing $[\![\boldsymbol{x}]\!]_{n-k}$.

In Step 5.(a), for each honest party P_j, $\mathcal{S}$ emulates $\mathcal{F}_{\text{com}}$ by outputting the handle of the commitment to Client_i. For each corrupted party P_j, $\mathcal{S}$ honestly emulates $\mathcal{F}_{\text{com}}$ and learns P_j's shares of $([\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k})$.

In Step 5.(b), $\mathcal{S}$ computes the shares of $[\boldsymbol{x}+\boldsymbol{r}]_{n-1}$ of corrupted parties and sets the shares of $[\boldsymbol{x}+\boldsymbol{r}]_{n-1}$ of honest parties to be uniform elements. Then $\mathcal{S}$ reconstructs the secrets $\boldsymbol{x}+\boldsymbol{r}$.

In Step 5.(c), $\mathcal{S}$ honestly follows the protocol and learns the shares of $[\boldsymbol{x}+\boldsymbol{r}]_{2k-2}$ held by honest parties. $\mathcal{S}$ recovers the whole sharings $[\boldsymbol{x}+\boldsymbol{r}]_{2k-2}$ using the shares of honest parties and then reconstructs the secrets $\overline{\boldsymbol{x}+\boldsymbol{r}}$. Then $\mathcal{S}$ computes $\boldsymbol{\eta} = \overline{\boldsymbol{x}+\boldsymbol{r}} - (\boldsymbol{x}+\boldsymbol{r})$.

Then $\mathcal{S}$ simulates the verification process. For Check-Consistency,

1. In Step 2, for each honest party P_i, $\mathcal{S}$ emulates $\mathcal{F}_{\text{com}}$ by outputting (i, τ_{λ_i}) to all parties. For each corrupted party P_i, $\mathcal{S}$ honestly emulates $\mathcal{F}_{\text{com}}$ and learns λ_i.

2. In Step 3, without loss of generality, assume that P_n is an honest party. $\mathcal{S}$ samples a random element as λ. For each honest party $P_i \neq P_n$, $\mathcal{S}$ samples a random element as λ_i. Then λ_n is set to be $\lambda - \sum_{i=1}^{n-1} \lambda_i$. For each honest party P_i, $\mathcal{S}$ emulates $\mathcal{F}_{\text{com}}$ by outputting $(\lambda_i, i, \tau_{\lambda_i})$. For each corrupted party P_i, $\mathcal{S}$ honestly emulates $\mathcal{F}_{\text{com}}$.

3. Recall that for all degree-$(k-1)$ packed Shamir sharings generated in Mult-Mal, $\mathcal{S}$ has already learnt the shares of honest parties. Therefore, $\mathcal{S}$ honestly follows the rest of steps in Check-Consistency. If no party aborts at the end of Mult-Mal but there exists a degree-$(k-1)$ packed Shamir sharing such that the shares of honest parties do not form a valid degree-$(k-1)$ packed Shamir sharing, $\mathcal{S}$ aborts.

For Check-Secret,

1. In Step 1, a pair of sharings $([\boldsymbol{u}^{(i)}]_{2k-2}, [\![\boldsymbol{u}^{(i)}]\!]_{n-k})$ may come from three places:

   - It may be $([\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\![\boldsymbol{x}+\boldsymbol{a}]\!]_{n-k})$ in Mult-Mal. In this case, $\mathcal{S}$ has computed the error $\boldsymbol{\eta}^{(1)}$ which is the secret of $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}$ minus the secret of $[\![\boldsymbol{x}+\boldsymbol{a}]\!]_{n-k}$. Note that, $\mathcal{S}$ learns the secrets of both sharings.
   - It may be $([(\boldsymbol{x}+\boldsymbol{a}) \cdot (\boldsymbol{y}+\boldsymbol{b})]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1}, 0)$ in Mult-Mal, where the second sharing is all-0 sharing. In this case, $\mathcal{S}$ has computed the error $\boldsymbol{\eta}^{(3)}$, which is the secret of $[(\boldsymbol{x}+\boldsymbol{a}) \cdot (\boldsymbol{y}+\boldsymbol{b})]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$. Note that, $\mathcal{S}$ learns the secrets of both sharings.
   - It may be $([\boldsymbol{x}+\boldsymbol{r}]_{k-1}, [\![\boldsymbol{x}+\boldsymbol{r}]\!]_{n-k})$ in Step 5 of Main-Mal. In this case, $\mathcal{S}$ has computed the error $\boldsymbol{\eta}$ which is the secret of $[\boldsymbol{x}+\boldsymbol{r}]_{k-1}$ minus the secret of $[\![\boldsymbol{x}+\boldsymbol{r}]\!]_{n-k}$. Note that, $\mathcal{S}$ learns the secrets of both sharings.

   Thus, in any case, $\mathcal{S}$ learns the secrets of both sharings. $\mathcal{S}$ also learns the shares held by corrupted parties, and the additive errors associated with $[\![\boldsymbol{u}^{(i)}]\!]_{n-k}$.

2. In Step 2 and Step 3, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}_{\text{com}}$ in the same way as that in Check-Consistency.

3. In Step 4,

   - $\mathcal{S}$ computes the whole sharing $[\boldsymbol{u}]_{2k-2}$ and the secret $\overline{\boldsymbol{u}}$
   - $\mathcal{S}$ computes the shares of $[\boldsymbol{u}]_{n-1}$ held by corrupted parties, and the secret $\boldsymbol{u}$. Then $\mathcal{S}$ randomly samples the shares of honest parties based on the secret $\boldsymbol{u}$ and the shares of corrupted parties.
   - $\mathcal{S}$ samples a random element as γ and sets $\boldsymbol{\gamma} = (\gamma, \gamma, \ldots, \gamma) \in \mathbb{F}^k$. Then $\mathcal{S}$ computes the shares of $[\boldsymbol{\gamma}]_{n-k}$ based on the secret $\boldsymbol{\gamma}$ and the shares of corrupted parties.
   - $\mathcal{S}$ computes the shares of $[\boldsymbol{\gamma} * \boldsymbol{u}]_{n-1}$ held by corrupted parties. Recall that $\mathcal{S}$ has computed a vector of additive errors for each $[\![\boldsymbol{u}^{(i)}]\!]_{n-k}$, denoted by $(\boldsymbol{\delta}_1^{(i)}, \boldsymbol{\delta}_2^{(i)})$. $\mathcal{S}$ computes the additive errors to $([\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma} * \boldsymbol{u}]_{n-1})$, denoted by $(\boldsymbol{\delta}_1, \boldsymbol{\delta}_2)$. Then $\mathcal{S}$ computes the secret of $[\boldsymbol{\gamma} * \boldsymbol{u}]_{n-1}$ by $\overline{\boldsymbol{\gamma} * \boldsymbol{u}} = \boldsymbol{\gamma} * \boldsymbol{u} + \boldsymbol{\delta}_2 - \boldsymbol{\gamma} * \boldsymbol{\delta}_1$. $\mathcal{S}$ randomly samples the shares of $[\boldsymbol{\gamma} * \boldsymbol{u}]_{n-1}$ of honest parties based on the secret $\overline{\boldsymbol{\gamma} * \boldsymbol{u}}$ and the shares of corrupted parties.

4. In Step 5 and Step 6, since the whole sharings $[\boldsymbol{u}]_{2k-2}, [\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma} * \boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}]_{n-k}$ have been generated, $\mathcal{S}$ honestly follows the protocol and honestly emulates $\mathcal{F}_{\text{com}}$. If no party aborts at the end of Check-Secret but the following case occurs, $\mathcal{S}$ aborts.

   - There exists $([\boldsymbol{u}^{(i)}]_{2k-2}, [\![\boldsymbol{u}^{(i)}]\!]_{n-k})$ such that their secrets do not satisfy

   $$\bar{\boldsymbol{u}}^{(i)} = \boldsymbol{u}^{(i)} - \boldsymbol{\delta}_1^{(i)}.$$

After the verification, $\mathcal{S}$ simulates Output-Mal.

1. If Client_i is corrupted, $\mathcal{S}$ learns the output $\boldsymbol{x}$ from $\mathcal{F}_{\text{main-mal}}$. Recall that $\mathcal{S}$ learns the whole sharing of $[\boldsymbol{x} + \boldsymbol{r}]_{2k-2}$ and the secret $\boldsymbol{x} + \boldsymbol{r}$. $\mathcal{S}$ computes $\boldsymbol{r} = (\boldsymbol{x} + \boldsymbol{r}) - \boldsymbol{x}$. Then $\mathcal{S}$ recovers the whole sharing $[\boldsymbol{r}]_{n-k}$ using the secret $\boldsymbol{r}$ and the shares of corrupted parties. $\mathcal{S}$ samples a random vector as $\boldsymbol{\Delta}$ and computes $\boldsymbol{\Delta} * \boldsymbol{r}$. Then $\mathcal{S}$ recovers the whole sharings $[\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ using the secrets $\boldsymbol{\Delta}, \boldsymbol{\Delta} * \boldsymbol{r}$ and the shares of corrupted parties.

   Since the whole sharings $[\boldsymbol{x} + \boldsymbol{r}]_{2k-2}, [\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ have been generated, $\mathcal{S}$ emulates $\mathcal{F}_{\text{com}}$ to open each honest party P_j's shares of $([\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k})$ to Client_i and honestly follows the rest of the protocol.

2. If Client_i is honest, $\mathcal{S}$ receives the shares of $[\boldsymbol{x} + \boldsymbol{r}]_{2k-2}$ of corrupted parties and emulates $\mathcal{F}_{\text{com}}$ honestly when corrupted parties open their commitments of shares of $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ to Client_i. For $[\boldsymbol{x} + \boldsymbol{r}]_{2k-2}$, if the shares received from corrupted parties are not equal to the shares that corrupted parties should hold, $\mathcal{S}$ aborts on behalf of Client_i.

   For each sharing $[\boldsymbol{p}]_{n-k} \in \{[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}\}$, $\mathcal{S}$ computes a sharing $[\delta(\boldsymbol{p})]_{n-k}$ which is defined as follows:

   (a) The shares of all honest parties are set to be 0.

   (b) For each corrupted party P_j, the j-th share is equal to the j-th share of $[\boldsymbol{p}]_{n-k}$ received from P_j minus the same share that P_j should hold.

   $\mathcal{S}$ checks whether $[\delta(\boldsymbol{p})]_{n-k}$ is a valid degree-$(n-k)$ packed Shamir sharing of $\boldsymbol{0}$. If not, $\mathcal{S}$ aborts on behalf of Client_i.

This completes the description of the simulator $\mathcal{S}$.

Hybrid Argument. Now we show that $\mathcal{S}$ perfectly simulates the behaviors of honest parties with overwhelming probability. Consider the following hybrids.

Hybrid$_0$: The execution in the real world.

Hybrid$_1$: In this hybrid, $\mathcal{S}$ simulates Check-Consistency. Compared with **Hybrid**$_0$, there are two changes:

- The functionality $\mathcal{F}_{\text{com}}$ is emulated by $\mathcal{S}$ and the random element λ is chosen by $\mathcal{S}$. Note that in $\mathbf{Hybrid}_0$, corrupted parties only receive from $\mathcal{F}_{\text{com}}$ (i, τ_{λ_i}) for each honest party in Step 2 of Check-Consistency. Therefore, for each corrupted party P_i, the value λ_i is independent of the values of honest parties. It means that the element λ computed in Step 3 of Check-Consistency is uniformly random. In $\mathbf{Hybrid}_1$, $\mathcal{S}$ simply outputs (i, τ_{λ_i}) on behalf of $\mathcal{F}_{\text{com}}$ for each honest party in Step 2 of Check-Consistency. Then $\mathcal{S}$ samples a random element λ and then samples the value λ_i for each honest party such that the summation of all λ_i's is λ. Therefore, the distribution of these two steps is identical to that in $\mathbf{Hybrid}_0$.

- $\mathcal{S}$ will abort if no party aborts in Check-Consistency but there exists a degree-$(k-1)$ packed Shamir sharing such that the shares of honest parties do not form a valid degree-$(k-1)$ packed Shamir sharing. By Lemma 6, it happens with probability at most $m/|\mathbb{F}|$.

Recall that $|\mathbb{F}| \geq 2^{\kappa}$, where κ is the security parameter. Therefore, the distribution of $\mathbf{Hybrid}_1$ is statistically close to the distribution of $\mathbf{Hybrid}_0$.

$\mathbf{Hybrid}_2$: In this hybrid, $\mathcal{S}$ computes the shares that corrupted parties should hold as described above. For each authenticated degree-$(n-k)$ packed Shamir sharing, $\mathcal{S}$ also computes a pair of vectors, which are the additive errors to the secret and its MAC. For each degree-$(2k-2)$ (including degree-$(k-1)$) packed Shamir sharing, $\mathcal{S}$ computes the whole sharing by using the shares of honest parties. Note that we do not change the way that $\mathcal{S}$ simulates the behaviors of honest parties. Therefore the distribution of $\mathbf{Hybrid}_2$ is identical to the distribution of $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_3$: In this hybrid, $\mathcal{S}$ simulates Check-Secret (including the simulation of the authentication key). Compared with $\mathbf{Hybrid}_2$, there are three changes:

- The functionality $\mathcal{F}_{\text{com}}$ is simulated by $\mathcal{S}$ and the random element λ is chosen by $\mathcal{S}$. Following the same argument as that in $\mathbf{Hybrid}_1$, the distribution of these two steps is identical to that in $\mathbf{Hybrid}_2$.

- $\mathcal{S}$ samples a random element as γ and sets $\boldsymbol{\gamma} = (\gamma, \gamma, \ldots, \gamma)$. Then $\mathcal{S}$ computes the shares of $[\boldsymbol{\gamma}]_{n-k}$ of honest parties based on the secret and the shares of corrupted parties.

  In $\mathbf{Hybrid}_2$, we argue that the messages that honest parties send to corrupted parties before Check-Secret are independent of the shares of $[\boldsymbol{\gamma}]_{n-k}$ held by honest parties. The messages sent from honest parties to corrupted parties have 3 different kinds:

  - The shares of a random degree-$(n-k)$ packed Shamir sharing of honest parties directly learnt from $\mathcal{F}_{\text{prep-mal}}$. This kind includes Step 2 of Input-Mal.
  - The shares of a random degree-$(n-1)$ packed Shamir sharing of honest parties, which are uniformly random field elements. This kind includes Step 3 of Tran-Mal, the first half of Step 3 of Mult-Mal, and Step 5.(a) of Main-Mal.
  - The degree-$(2k-2)$ (including degree-$(k-1)$) packed Shamir sharings whose secrets are uniformly random. This kind includes Step 4 of Tran-Mal, Step 4 of Input-Mal, the second half of Step 3 of Mult-Mal, and Step 5.(b) of Main-Mal.
  - The linear combinations of shares of degree-$(k-1)$ packed Shamir sharings that are directly received from other parties. This kind includes Step 5 of Check-Consistency.

  One can verify that all above messages are independent of the shares of $[\boldsymbol{\gamma}]_{n-k}$ held by honest parties. Since γ is uniformly random given the shares of $[\boldsymbol{\gamma}]_{n-k}$ held by corrupted parties, the distribution of the shares of $[\boldsymbol{\gamma}]_{n-k}$ held by honest parties is identical in both $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_3$.

- For $[\boldsymbol{u}]_{2k-2}$, $\mathcal{S}$ has already computed the whole sharing. Let $\bar{\boldsymbol{u}}$ denote the secret of $[\boldsymbol{u}]_{2k-2}$. For $([\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma} * \boldsymbol{u}]_{n-1})$, the secrets can be computed from the secrets and the MACs of $\{[\![\boldsymbol{u}^{(i)}]\!]_{n-k}\}_{i=1}^{m}$. For each $[\![\boldsymbol{u}^{(i)}]\!]_{n-k}$, $\mathcal{S}$ computes the secret by using the shares that corrupted parties should hold and the shares of honest parties. Later on, we will show that $\mathcal{S}$ can compute the secret $\boldsymbol{u}^{(i)}$ without the shares of honest parties. Note that $\mathcal{S}$ also computes a pair of vectors $(\boldsymbol{\delta}_1^{(i)}, \boldsymbol{\delta}_2^{(i)})$ which are additive errors to

the secret and its MAC of $[\![\boldsymbol{u}^{(i)}]\!]_{n-k}$. Then, the secret of $[\boldsymbol{\gamma}*\boldsymbol{u}^{(i)}]_{n-k}$ is $\overline{\boldsymbol{\gamma}*\boldsymbol{u}^{(i)}} = \boldsymbol{\gamma}*\boldsymbol{u}^{(i)}+\boldsymbol{\delta}_2^{(i)}-\boldsymbol{\gamma}*\boldsymbol{\delta}_1^{(i)}$. $\mathcal{S}$ computes the secret of $[\boldsymbol{\gamma}*\boldsymbol{u}^{(i)}]_{n-k}$. Now $\mathcal{S}$ can compute the secrets of $([\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}*\boldsymbol{u}]_{n-1})$. Given the secrets of $[\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}*\boldsymbol{u}]_{n-1}$ and the shares of corrupted parties, $\mathcal{S}$ randomly samples the shares of honest parties.

In $\mathbf{Hybrid}_2$, for $[\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}*\boldsymbol{u}]_{n-1}$, since all parties use random degree-$(n-1)$ packed Shamir sharings of $\mathbf{0}$, $[\boldsymbol{o}^{(1)}]_{n-1}, [\boldsymbol{o}^{(2)}]_{n-1}$, as random masks, they are random degree-$(n-1)$ packed Shamir sharings given the secrets $\boldsymbol{u}, \overline{\boldsymbol{\gamma}*\boldsymbol{u}}$ and the shares of corrupted parties. Note that $\boldsymbol{u}$ is identical to that in $\mathbf{Hybrid}_3$, and $\overline{\boldsymbol{\gamma}*\boldsymbol{u}}$ is determined by $\boldsymbol{u}$ and the pair of additive errors. Therefore, $\boldsymbol{u}, \overline{\boldsymbol{\gamma}*\boldsymbol{u}}$ are identical to those in $\mathbf{Hybrid}_3$. The shares of $[\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}*\boldsymbol{u}]_{n-1}$ of honest parties have the same distribution in both $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_3$.

- If no party aborts at the end of Check-Secret but the following case occurs, $\mathcal{S}$ aborts.
  - There exists $([\boldsymbol{u}^{(i)}]_{2k-2}, [\![\boldsymbol{u}^{(i)}]\!]_{n-k})$ such that their secrets do not satisfy
  $$\bar{\boldsymbol{u}}^{(i)} = \boldsymbol{u}^{(i)} - \boldsymbol{\delta}_1^{(i)}.$$
  Let $(\boldsymbol{\delta}_1, \boldsymbol{\delta}_2)$ be the pair of additive errors to $[\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}*\boldsymbol{u}]_{n-1}$ computed from $\{(\boldsymbol{\delta}_1^{(i)}, \boldsymbol{\delta}_2^{(i)})\}_{i=1}^m$. Note that
  - Following the same argument as that in Lemma 6, if there exists a pair $([\boldsymbol{u}^{(i)}]_{2k-2}, [\![\boldsymbol{u}^{(i)}]\!]_{n-k})$ such that $\bar{\boldsymbol{u}}^{(i)} \neq \boldsymbol{u}^{(i)} - \boldsymbol{\delta}_1^{(i)}$, then with overwhelming probability, for the final pair $([\boldsymbol{u}]_{2k-2}, [\![\boldsymbol{u}]\!]_{n-1})$, $\bar{\boldsymbol{u}} \neq \boldsymbol{u} - \boldsymbol{\delta}_1$.
  - Corrupted parties cannot change their shares of $[\boldsymbol{u}]_{2k-2}$ without being detected since the whole sharing is determined by the shares of honest parties.
  - While corrupted parties may change their shares of $[\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}*\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}]_{n-k}$, any change of the shares of corrupted parties either will be detected or is equivalent to additive errors to the secrets $\boldsymbol{u}, \overline{\boldsymbol{\gamma}*\boldsymbol{u}}, \boldsymbol{\gamma}$. To see this, for each of the sharing $[\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}*\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}]_{n-k}$, we may compute the difference between the sharing that uses the shares that corrupted parties commit and the sharing that uses the shares that corrupted parties should hold. For $[\boldsymbol{u}]_{n-1}$ or $[\boldsymbol{\gamma}*\boldsymbol{u}]_{n-1}$, the difference corresponds to a degree-$(n-1)$ packed Shamir sharing of the additive error. For $[\boldsymbol{\gamma}]_{n-k}$, the difference is either not a valid degree-$(n-k)$ packed Shamir sharing, which will be detected when checking $[\boldsymbol{\gamma}]_{n-k}$, or a degree-$(n-k)$ packed Shamir sharing of the additive error. Note that the additive errors are independent of $\boldsymbol{\gamma}, \overline{\boldsymbol{\gamma}*\boldsymbol{u}}$ since the adversary needs to decide the additive errors before reconstructing $[\boldsymbol{\gamma}*\boldsymbol{u}]_{n-1}, [\boldsymbol{\gamma}]_{n-k}$.
    When $\overline{\boldsymbol{u}} \neq \boldsymbol{u} - \boldsymbol{\delta}_1$, the only possibility that no party aborts at the end of Check-Secret is that the adversary adds $\boldsymbol{\epsilon}_0, \boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2$ to $\boldsymbol{u}, \overline{\boldsymbol{\gamma}*\boldsymbol{u}}, \boldsymbol{\gamma}$ respectively such that (1) $\overline{\boldsymbol{u}} = \boldsymbol{u} + \boldsymbol{\epsilon}_0$, and (2) $(\boldsymbol{\gamma}+\boldsymbol{\epsilon}_2)*(\boldsymbol{u}+\boldsymbol{\epsilon}_0) = \overline{\boldsymbol{\gamma}*\boldsymbol{u}} + \boldsymbol{\epsilon}_1$. Recall that $\overline{\boldsymbol{\gamma}*\boldsymbol{u}} = \boldsymbol{\gamma}*\boldsymbol{u} + \boldsymbol{\delta}_2 - \boldsymbol{\gamma}*\boldsymbol{\delta}_1$. From (1), $\boldsymbol{\epsilon}_0 \neq -\boldsymbol{\delta}_1$. From (2), $\boldsymbol{\gamma}*(\boldsymbol{\epsilon}_0+\boldsymbol{\delta}_1) = \boldsymbol{\delta}_2 + \boldsymbol{\epsilon}_1 - \boldsymbol{u}*\boldsymbol{\epsilon}_2 - \boldsymbol{\epsilon}_0*\boldsymbol{\epsilon}_2$. Since γ is a random field element and $\boldsymbol{\gamma} = (\gamma, \gamma, \ldots, \gamma) \in \mathbb{F}^k$, the probability that the adversary can find $\boldsymbol{\epsilon}_0, \boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2$ that satisfy (1) and (2) is negligible.

In Summary, the distribution of $\mathbf{Hybrid}_3$ is statistically close to the distribution of $\mathbf{Hybrid}_2$. Note that in $\mathbf{Hybrid}_3$, $\mathcal{S}$ will always abort if the computation is incorrect.

$\mathbf{Hybrid}_4$: In this hybrid, $\mathcal{S}$ extracts the input of corrupted parties as described above. Then $\mathcal{S}$ invokes $\mathcal{F}_{\text{main-mal}}$ and sends the input of corrupted parties to $\mathcal{F}_{\text{main-mal}}$. $\mathcal{S}$ simulates Output. We first consider the case that the output gates belong to an honest client. Compared with $\mathbf{Hybrid}_3$, the change is the following:

- In $\mathbf{Hybrid}_4$, after $\mathcal{S}$ receives the shares of $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta}*\boldsymbol{r}]_{n-k}$ of corrupted parties, for each sharing, $\mathcal{S}$ computes the difference of the sharing that uses the shares that corrupted send to $\mathcal{S}$ and the sharing that uses the shares that corrupted parties should hold. $\mathcal{S}$ aborts if the difference is not a valid degree-$(n-k)$ packed Shamir sharing of $\mathbf{0}$.

- In $\mathbf{Hybrid}_3$, Client reconstructs the secret of $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ and checks whether the third secret is equal to the coordinate-wise multiplication of the first two secrets. Following the same argument as that in $\mathbf{Hybrid}_3$, the change of the shares of $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ of corrupted parties either will lead to invalid degree-$(n-k)$ packed Shamir sharings or is equivalent to additive errors to the secrets $\boldsymbol{r}, \boldsymbol{\Delta}, \boldsymbol{\Delta} * \boldsymbol{r}$. If the adversary changes the shares of corrupted parties but Client does not abort, the only possibility is that the adversary adds $\boldsymbol{\epsilon}_0, \boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2$ to $\boldsymbol{r}, \boldsymbol{\Delta}, \boldsymbol{\Delta} * \boldsymbol{r}$ respectively such that $(\boldsymbol{\Delta} + \boldsymbol{\epsilon}_1) * (\boldsymbol{r} + \boldsymbol{\epsilon}_0) = \boldsymbol{\Delta} * \boldsymbol{r} + \boldsymbol{\epsilon}_2$. It means that $\boldsymbol{\Delta} * \boldsymbol{\epsilon}_0 + \boldsymbol{\epsilon}_1 * \boldsymbol{r} + \boldsymbol{\epsilon}_1 * \boldsymbol{\epsilon}_0 = \boldsymbol{\epsilon}_2$. If $\boldsymbol{\epsilon}_0 = \boldsymbol{\epsilon}_1 = \boldsymbol{\epsilon}_2 = \mathbf{0}$, then in both $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_4$, Client (or $\mathcal{S}$) accepts. If $\boldsymbol{\epsilon}_0 = \mathbf{0}$ but $\boldsymbol{\epsilon}_1 \neq \mathbf{0}$ or $\boldsymbol{\epsilon}_2 \neq \mathbf{0}$, then $\boldsymbol{\epsilon}_1$ and $\boldsymbol{\epsilon}_2$ should satisfy that $\boldsymbol{\epsilon}_1 * \boldsymbol{r} = \boldsymbol{\epsilon}_2$. Note that due to the use of $\mathcal{F}_{\text{com}}$, $\boldsymbol{\epsilon}_1$ and $\boldsymbol{\epsilon}_2$ are determined in Step 5.(a) and in this case $\boldsymbol{r}$ is uniformly random and not known to the adversary. Then the probability that the adversary can find such $\boldsymbol{\epsilon}_1$ and $\boldsymbol{\epsilon}_2$ is negligible. Similarly, if $\boldsymbol{\epsilon}_0 \neq \mathbf{0}$, since $\boldsymbol{\Delta}$ is a random vector, the probability that the adversary can find $\boldsymbol{\epsilon}_0, \boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2$ that satisfy the above requirement is negligible.

Thus, when the client is honest, the distribution of $\mathbf{Hybrid}_4$ is statistically close to $\mathbf{Hybrid}_3$.

Now we consider the case that the output gates belong to a corrupted client. Compared with $\mathbf{Hybrid}_3$, the change is that $\mathcal{S}$ computes $\boldsymbol{r}$ from $\boldsymbol{x} + \boldsymbol{r}$ and $\boldsymbol{x}$, and $\mathcal{S}$ samples a random vector as $\boldsymbol{\Delta}$. Then based on the secrets and the shares of corrupted parties, $\mathcal{S}$ recovers the whole sharings $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$. Note that Output is only invoked after Check-Consistency and Check-Secret. Recall that we have changed these protocols by the simulation of $\mathcal{S}$, which will abort if the computation is incorrect. Thus, $\boldsymbol{x}$ received from $\mathcal{F}_{\text{main-mal}}$ is the same as $\boldsymbol{x}$ in $\mathbf{Hybrid}_3$. Therefore, the vector $\boldsymbol{r}, \boldsymbol{\Delta}$ computed by $\mathcal{S}$ have the same distribution as those in $\mathbf{Hybrid}_3$. Since the sharings $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ are determined by the secrets and the shares of corrupted parties, the distribution of $\mathbf{Hybrid}_4$ is identical to $\mathbf{Hybrid}_3$ when the client is corrupted.

In summary, the distribution of $\mathbf{Hybrid}_4$ is statistically close to the distribution of $\mathbf{Hybrid}_3$.

$\mathbf{Hybrid}_5$: In this hybrid, $\mathcal{S}$ simulates Tran-Mal, Mult-Mal, Add-Mal, and Step 5.(a), 5.(b) of Mult-Mal. The only difference is that when all parties need to send a degree-$(n-1)$ packed Shamir sharing to P_1, $\mathcal{S}$ samples random elements as the shares of honest parties. Recall that we either use a random degree-$(n-1)$ packed Shamir sharing $[\boldsymbol{r}]_{n-1}$, or use a pair of random sharings $([\boldsymbol{r}]_{n-k}, [\boldsymbol{o}]_{n-1})$ where $\boldsymbol{o} = \mathbf{0}$, as a random mask. Note that $[\boldsymbol{r}]_{n-k} + [\boldsymbol{o}]_{n-1}$ has the same distribution as $[\boldsymbol{r}]_{n-1}$. Therefore, the sharings that P_1 collects from all parties are always random degree-$(n-1)$ packed Shamir sharings. Given the shares of corrupted parties, the shares of a random degree-$(n-1)$ packed Shamir sharing of honest parties are uniformly random. Thus, the distribution of $\mathbf{Hybrid}_5$ is identical to the distribution of $\mathbf{Hybrid}_4$. Note that the shares generated for honest parties together with the shares of corrupted parties can be used to compute the secret of $[\![\boldsymbol{u}^{(i)}]\!]_{n-k}$ in Check-Secret.

$\mathbf{Hybrid}_6$: In this hybrid, $\mathcal{S}$ simulates Input. We first consider the case that the input gates belong to an honest client. Compared with $\mathbf{Hybrid}_5$, the difference is the following:

- In $\mathbf{Hybrid}_6$, after $\mathcal{S}$ receives the shares of $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ of corrupted parties, for each sharing, $\mathcal{S}$ computes the difference of the sharing that uses the shares that corrupted parties send to $\mathcal{S}$ and the sharing that uses the shares that corrupted parties should hold. If the difference is not a valid degree-$(n-k)$ packed Shamir sharing of $\mathbf{0}$, $\mathcal{S}$ aborts. Otherwise $\mathcal{S}$ generates a random vector as $\boldsymbol{x} + \boldsymbol{r}$ and honestly follows the protocol by generating a random degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{x} + \boldsymbol{r}]_{k-1}$ and distributing the shares to corrupted parties.

- In $\mathbf{Hybrid}_5$, Client reconstructs the secret of $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ and checks whether the third secret is equal to the coordinate-wise multiplication of the first two secrets. Following the same argument as that in $\mathbf{Hybrid}_5$, if corrupted parties change their shares of an authenticated sharing (which causes the the change of the secret), with overwhelming probability, the change will be detected and Client will abort. If Client accepts, Client will generate and distribute a random degree-$(k-1)$ packed Shamir sharing of $\boldsymbol{x} + \boldsymbol{r}$. Since $\boldsymbol{r}$ is a random vector, $\boldsymbol{x} + \boldsymbol{r}$ is also a random vector.

Thus, when the client is honest, the distribution of $\mathbf{Hybrid}_6$ is statistically close to $\mathbf{Hybrid}_5$.

Now we consider the case that the input gates belong to a corrupted client. Compared with $\mathbf{Hybrid}_5$, the change is that $\mathcal{S}$ samples two random vector as $\boldsymbol{r}, \boldsymbol{\Delta}$. Then based on the secrets and the shares of corrupted

parties, $\mathcal{S}$ recovers the whole sharings $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$. Note that $\boldsymbol{r}, \boldsymbol{\Delta}$ are indeed random vectors in $\mathbf{Hybrid}_5$. Since the sharings $[\boldsymbol{r}]_{n-k}, [\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ are determined by the secrets and the shares of corrupted parties, the distribution of $\mathbf{Hybrid}_6$ is identical to $\mathbf{Hybrid}_5$ when the client is corrupted.

In summary, the distribution of $\mathbf{Hybrid}_6$ is statistically close to the distribution of $\mathbf{Hybrid}_5$.

Note that $\mathbf{Hybrid}_6$ is the execution in the ideal world, and the distribution of $\mathbf{Hybrid}_6$ is statistically close to the distribution of $\mathbf{Hybrid}_0$, the execution in the real world. □

Theorem 5. *In the client-server model, let* c *denote the number of clients,* n *denote the number of parties (servers), and* t *denote the number of corrupted parties (servers). Let* κ *be the security parameter and* $\mathbb{F}$ *be a finite field of size* $|\mathbb{F}| \geq 2^\kappa$*. For an arithmetic circuit* C *over* $\mathbb{F}$ *and for all* $\frac{n-1}{3} \leq t \leq n-1$*, there exists an information-theoretic MPC protocol in the circuit-independent preprocessing model which securely computes the arithmetic circuit* C *in the presence of a fully malicious adversary controlling up to* c *clients and* t *parties. The cost of the protocol is* $O(|C| \cdot \frac{n^2}{k^2} + \mathit{poly}(\mathsf{Depth}, \mathsf{c}, n))$ *field elements of preprocessing data and* $O(|C| \cdot \frac{n}{k} + \mathit{poly}(\mathsf{Depth}, \mathsf{c}, n))$ *field elements of communication, where* $k = \frac{n-t+1}{2}$ *and* Depth *is the circuit depth.*

7 Using [GPS21] for Small Finite Fields

In this section, we discuss how to use the technique in [GPS21] to relax the requirement of the field size.

In essence, we will always use the default positions for the packed Shamir secret sharing scheme in our construction. Then, for network routing, we will use the technique in [GPS21] instead of our approach. We describe the technique in [GPS21] as follows.

Approach in [GPS21] for the Network Routing. The work [GPS21] focuses on the strong honest majority setting, where the number of corrupted parties is bounded by $t = n/2 - k$, and shows how to use packed Shamir sharings to evaluate a single circuit with $O(n/k)$ communication per gate. While the corruption threshold of this work is different from that in [GPS21], we note that their method can be extended to our setting. In [GPS21], the network routing is done by two steps:

1. For each input sharing that we want to prepare, all parties first prepare a degree-$(n-1)$ packed Shamir sharing such that it contains all the secrets we need but the order may be incorrect.

2. To obtain the input sharing we need, all parties perform a permutation on the secrets of the sharing that all parties have prepared in the first step.

Note that the second step can be simply achieved by using our technique for sharing transformation since a permutation is a linear map over $\mathbb{F}^k \to \mathbb{F}^k$. For the first step, the main observation in [GPS21] is that, if the secrets we need are coming from different positions of the output sharings from previous layers, then all parties can locally obtain a degree-$(n-1)$ packed Shamir sharing that contains the secrets we want.

Concretely, suppose $[\boldsymbol{x}^{(1)}]_{n-k}, \ldots, [\boldsymbol{x}^{(k)}]_{n-k}$ are k degree-$(n-k)$ packed Shamir sharings from previous layers (which do not need to be distinct) and all parties want to prepare a sharing that contains the i_j-th secret of $\boldsymbol{x}^{(j)}$ for all $j \in \{1, 2, \ldots, k\}$. If $i_1, i_2, \ldots, i_k$ are distinct, then all parties can locally compute a degree-$(n-1)$ packed Shamir sharing that contains the secrets we want as follows:

1. For all $j \in \{1, 2, \ldots, k\}$, let $\boldsymbol{e}_j$ denote a vector in $\mathbb{F}^k$ where all entries of $\boldsymbol{e}_j$ are 0 except the i_j-th entry is 1. All parties locally transform $\boldsymbol{e}_j$ to a degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{e}_j]_{k-1}$.

2. All parties locally compute

$$[\boldsymbol{x}]_{n-1} = \sum_{j=1}^{k} [\boldsymbol{e}_j]_{k-1} \cdot [\boldsymbol{x}^{(j)}]_{n-k}.$$

The correctness follows from the facts that $[\boldsymbol{e}_j]_{k-1} \cdot [\boldsymbol{x}^{(j)}]_{n-k} = [\boldsymbol{e}_j * \boldsymbol{x}^{(j)}]_{n-1}$ and $\boldsymbol{e}_j * \boldsymbol{x}^{(j)}$ is a vector in $\mathbb{F}^k$ where all entries are 0 except the i_j-th entry is $x_{i_j}^{(j)}$.

Following from this observation, Goyal, et al. [GPS21] consider the so-called non-collision property.

Property 2 (Non-collision [GPS21])**.** *For each input sharing of each layer, the secrets of this input sharing come from different positions in the output sharings of previous layers.*

The non-collision property can be achieved by the following two steps:

1. For each of input gates, addition gates, and multiplication gates, we insert a fan-out gate that copies the output wire the number of times it is used in later layers. In this way, *every output wire of the input layer and all intermediate layers is used exactly once as an input wire of a later layer* (which may not be the next layer). Note that this step has not achieved the non-collision property yet: If the original circuit has already satisfied that for each of input gates, addition gates, and multiplication gates, the output wire is just used once in later layers, then this step essentially does nothing. It is still possible, for example, that the secrets that need to be in a single input sharing are all coming from the first position of k different sharings.

2. Let m denote the number of output packed Shamir sharings of the input layer and all intermediate layers in the circuit after the introduction of the fan-out gate in Step 1. Then the number of input packed Shamir sharings of the output layer and all intermediate layers is also m. A collision refers to the scenario that two secrets that are *in the same position* of two different output sharings want to go to the same input sharing. One way to remove a collision is to perform a permutation on one of the output sharing so that these two secrets are in different positions. The non-collision property requires us to remove all collisions at the same time. In [GPS21], Goyal, et al. show that there exists m permutations, such that *all collisions can be removed* by applying the i-th permutation to the secrets of the i-th output packed Shamir sharing for all $i \in \{1, 2, \ldots, m\}$.

   Specifically, consider a matrix $\boldsymbol{N} \in \{1, 2, \ldots, m\}^{m \times k}$ where $\boldsymbol{N}_{i,j}$ is the index of the input sharing that the j-th secret of the i-th output sharing wants to go to. We have the following theorem from [GPS21].

   Theorem 6. *Let $m \geq 1, k \geq 1$ be integers. Let $\boldsymbol{N}$ be a matrix of dimension $m \times k$ in $\{1, 2, \ldots, m\}^{m \times k}$ such that there are exactly k entries that are equal to ℓ for all $\ell \in \{1, 2, \ldots, m\}$ (Here an entry $\boldsymbol{N}_{i,j} = \ell$ means that the the j-th secret of the i-th output sharing wants to go to the ℓ-th input sharing). Then, there exists m permutations $p_1, p_2, \ldots, p_m$ over $\{1, 2, \ldots, k\}$ such that after performing the permutation p_i on the i-th row of $\boldsymbol{N}$, the new matrix $\boldsymbol{N}'$ satisfies that each column of $\boldsymbol{N}'$ is a permutation over $(1, 2, \ldots, m)$. Furthermore, the permutations $p_1, p_2, \ldots, p_m$ can be found within polynomial time.*

   Note that after applying the permutation p_i to the i-th output sharing for all $i \in \{1, 2, \ldots, m\}$, for all $j \in \{1, 2, \ldots, k\}$, the secrets in the j-th column want to go to different input sharings. I.e., for each input sharing, the secrets come from different positions in the output sharings of previous layers.

Relying Sharing Transformation to Perform Linear Maps. To achieve the non-collision property, after evaluating a group of k (input, addition, or multiplication) gates, we first copy each output wire the number of times that it is used in the later layers. Let $[\boldsymbol{x}]_{n-k}$ denote the output packed Shamir sharing, and let n_i denote the number of times that the i-th secret x_i is used in later layers for all $i \in \{1, 2, \ldots, k\}$. We first transform $\boldsymbol{x}$ to $m = \lceil \frac{n_1 + n_2 + \ldots + n_k}{k} \rceil$ vectors $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}$ in $\mathbb{F}^k$ such that they contain n_i copies of the value x_i for all $i \in \{1, 2, \ldots, k\}$. All parties use the following algorithm to locally determine what values should be in each of these m vectors.

1. All parties locally initiate an empty list L. From $i = 1$ to k, all parties locally insert n_i times of x_i into L.

2. From $i = 1$ to m, all parties locally set $\boldsymbol{x}^{(i)}$ to be the vector of the first k elements in L, and then remove these elements from L. In this way, all parties determine the values that should be in the m vectors $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}$.

For each $\boldsymbol{x}^{(i)} \in \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$, to obtain $[\boldsymbol{x}^{(i)}]_{n-k}$ from $[\boldsymbol{x}]_{n-k}$, all parties set $\mathcal{L}'_i$ to be the $\mathbb{F}$-linear map that maps $\boldsymbol{x}$ to $\boldsymbol{x}^{(i)}$, and then perform a sharing transformation to transform $[\boldsymbol{x}]_{n-k}$ to $[L'_i(\boldsymbol{x})]_{n-k}$. To achieve the non-collision property, we need to perform the permutation computed from the algorithm in Theorem 6 on $\boldsymbol{x}^{(i)}$. Let $\mathcal{L}''_i$ denote this permutation. Then, all parties can achieve the non-collision property by performing a sharing transformation from $[\boldsymbol{x}^{(i)}]_{n-k}$ to $[\mathcal{L}''_i(\boldsymbol{x}^{(i)})]_{n-k}$.

We note that all parties can obtain $[\mathcal{L}''_i(\boldsymbol{x}^{(i)})]_{n-k}$ directly from $[\boldsymbol{x}]_{n-k}$: Since $\mathcal{L}'_i, \mathcal{L}''_i$ are linear maps, $\mathcal{L}_i = \mathcal{L}'_i \circ \mathcal{L}''_i$ is also a linear map. Thus, all parties can perform a single sharing transformation from $[\boldsymbol{x}]_{n-k}$ to $[\mathcal{L}_i(\boldsymbol{x})]_{n-k}$.

Summary. By using the technique in [GPS21], we only need to use the default positions for the packed Shamir secret sharing scheme. Therefore, the protocol works for any finite field $\mathbb{F}$ of size $|\mathbb{F}| \geq 2n$.

Theorem 7. *In the client-server model, let c denote the number of clients, n denote the number of parties (servers), and t denote the number of corrupted parties (servers). Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2n$. For an arithmetic circuit C over $\mathbb{F}$, there exists an information-theoretic MPC protocol in the preprocessing model which securely computes the arithmetic circuit C in the presence of a semi-honest adversary controlling up to c clients and t parties. The cost of the protocol is $O(|C| \cdot \frac{n^2}{k^2} + (\mathsf{Depth} + c) \cdot \frac{n^2}{k})$ field elements of preprocessing data and $O(|C| \cdot \frac{n}{k} + (\mathsf{Depth} + c) \cdot n)$ field elements of communication, where $k = \frac{n-t+1}{2}$ and Depth is the circuit depth.*

Acknowledgements. V. Goyal, Y. Song—Supported by the NSF award 1916939, DARPA SIEVE program under Agreement No. HR00112020025, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award. Y. Song was also supported by a Cylab Presidential Fellowship.
A. Polychroniadou—This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful. 2020 JPMorgan Chase & Co. All rights reserved.

References

[ACD+19] Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, and Chen Yuan. Efficient Information-Theoretic Secure Multiparty Computation over $\mathbb{Z}/p^k\mathbb{Z}$ via Galois Rings. In *Theory of Cryptography*, pages 471–501, Cham, 2019. Springer International Publishing.

[ACD+20] Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, Matthieu Rambaud, Chaoping Xing, and Chen Yuan. Asymptotically Good Multiplicative LSSS over Galois Rings and Applications to MPC over $\mathbb{Z}/p^k\mathbb{Z}$. In *Advances in Cryptology – ASIACRYPT 2020*, pages 151–180, Cham, 2020. Springer International Publishing.

[AGJ+21] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. Cryptology ePrint Archive, Paper 2021/576, 2021. `https://eprint.iacr.org/2021/576`.

[BBCG+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In *Advances in Cryptology – CRYPTO 2019*, pages 67–97, Cham, 2019. Springer International Publishing.

[BBG+21] Fabrice Benhamouda, Elette Boyle, Niv Gilboa, Shai Halevi, Yuval Ishai, and Ariel Nof. Generalized pseudorandom secret sharing and efficient straggler-resilient secure computation. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography*, pages 129–161, Cham, 2021. Springer International Publishing.

[BCG+20] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. Cryptology ePrint Archive, Paper 2020/1392, 2020. `https://eprint.iacr.org/2020/1392`.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 169–188, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

[BGIN20] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Efficient fully secure computation via distributed zero-knowledge proofs. In *Advances in Cryptology – ASIACRYPT 2020*, pages 244–276, Cham, 2020. Springer International Publishing.

[BGIN21] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Sublinear GMW-Style Compiler for MPC with Preprocessing. In *Advances in Cryptology – CRYPTO 2021*, pages 457–485, Cham, 2021. Springer International Publishing.

[BGJK21] Gabrielle Beck, Aarushi Goel, Abhishek Jain, and Gabriel Kaptchuk. Order-c secure multiparty computation for highly repetitive circuits. In *Advances in Cryptology – EUROCRYPT 2021*, pages 663–693, Cham, 2021. Springer International Publishing.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988.

[BSFO12] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *Advances in Cryptology – CRYPTO 2012*, pages 663–680, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[CCXY18] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized Complexity of Information-Theoretically Secure MPC Revisited. In *Advances in Cryptology – CRYPTO 2018*, pages 395–426, Cham, 2018. Springer International Publishing.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *Theory of Cryptography*, pages 342–362, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[CGH+18] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority mpc for malicious adversaries. In *Annual International Cryptology Conference*, pages 34–64. Springer, 2018.

[Cou19] Geoffroy Couteau. A Note on the Communication Complexity of Multiparty Computation in the Correlated Randomness Model. In *Advances in Cryptology – EUROCRYPT 2019*, pages 473–503, Cham, 2019. Springer International Publishing.

[CRX21] Ronald Cramer, Matthieu Rambaud, and Chaoping Xing. Asymptotically-Good Arithmetic Secret Sharing over $\mathbb{Z}/p^\ell\mathbb{Z}$ with Strong Multiplication and Its Applications to Efficient MPC. In *Advances in Cryptology – CRYPTO 2021*, pages 656–686, Cham, 2021. Springer International Publishing.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 445–465. Springer, 2010.

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Annual International Cryptology Conference*, pages 572–590. Springer, 2007.

[DNPR16] Ivan Damgård, Jesper Buus Nielsen, Antigoni Polychroniadou, and Michael Raskin. On the communication required for unconditionally secure multiplication. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 459–488, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology–CRYPTO 2012*, pages 643–662. Springer, 2012.

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - a framework for efficient mixed-protocol secure two-party computation. NDSS, 2015.

[EGK+20] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for mpc over mixed arithmetic-binary circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 823–852, Cham, 2020. Springer International Publishing.

[FY92] Matthew Franklin and Moti Yung. Communication Complexity of Secure Computation (Extended Abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, page 699–710, New York, NY, USA, 1992. Association for Computing Machinery.

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj M. Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 495–504, New York, NY, USA, 2014. ACM.

[GIP15] Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. Efficient multi-party computation: from passive to active security via secure simd circuits. In *Annual Cryptology Conference*, pages 721–741. Springer, 2015.

[GLO+21] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. Atlas: Efficient and scalable mpc in the honest majority setting. In *Advances in Cryptology – CRYPTO 2021*, pages 244–274, Cham, 2021. Springer International Publishing.

[GPS21] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Unconditional Communication-Efficient MPC via Hall's Marriage Theorem. In *Advances in Cryptology – CRYPTO 2021*, pages 275–304, Cham, 2021. Springer International Publishing.

[GSY21] S. Dov Gordon, Daniel Starin, and Arkady Yerukhimovich. The More the Merrier: Reducing the Cost of Large Scale MPC. In *Advances in Cryptology – EUROCRYPT 2021*, pages 694–723, Cham, 2021. Springer International Publishing.

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority mpc. In *Advances in Cryptology – CRYPTO 2020*, pages 618–646, Cham, 2020. Springer International Publishing.

[HLOWI16] Brett Hemenway, Steve Lu, Rafail Ostrovsky, and William Welser IV. High-precision secure computation of satellite collision probabilities. In Vassilis Zikas and Roberto De Prisco, editors, *Security and Cryptography for Networks*, pages 169–187, Cham, 2016. Springer International Publishing.

[IKP+16] Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. Secure protocol transformations. In *Advances in Cryptology – CRYPTO 2016*, pages 430–458, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[MR18] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 35–52, New York, NY, USA, 2018. Association for Computing Machinery.

[NV18] Peter Sebastian Nordholt and Meilof Veeningen. Minimising communication in honest-majority mpc by batchwise multiplication verification. In *Applied Cryptography and Network Security*, pages 321–339, Cham, 2018. Springer International Publishing.

[PS21] Antigoni Polychroniadou and Yifan Song. Constant-Overhead Unconditionally Secure Multiparty Computation Over Binary Fields. In *Advances in Cryptology – EUROCRYPT 2021*, pages 812–841, Cham, 2021. Springer International Publishing.

[PSSY21] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved Mixed-Protocol secure Two-Party computation. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2165–2182. USENIX Association, August 2021.

[RW19] Dragos Rotaru and Tim Wood. Marbled circuits: Mixing arithmetic and boolean circuits with active security. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *Progress in Cryptology – INDOCRYPT 2019*, pages 227–249, Cham, 2019. Springer International Publishing.

[Sha79] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, November 1979.