Sharing Transformation and Dishonest Majority MPC with Packed Secret Sharing

Vipul Goyal[1,2], Antigoni Polychroniadou[3], and Yifan Song[1]

[1] Carnegie Mellon University, Pittsburgh, PA
goyal@cs.cmu.edu, yifans2@andrew.cmu.edu
[2] NTT Research, Sunnyvale, CA
[3] J.P. Morgan AI Research, New York, NY
antigonipoly@gmail.com

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 $\{P_1, P_2, \dots, P_n\}$ 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[4]. 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

[4] 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.

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 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 the full version of this paper [GPS22]. 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 in the full version of this paper [GPS22], we show that it 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 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 $1 \leq k \leq [\frac{n+2}{3}]$. The formal theorem for semi-honest security is stated in Theorem 3. We refer the readers to the full version of this paper [GPS22] for the formal theorem for malicious security. 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 $|\mathbb{F}| \geq 2n$. We refer the readers to the full version of this paper [GPS22] for more details.

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

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

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.

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.

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 $\boldsymbol{0} \in \mathbb{F}^k$, 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 the full version of this paper [GPS22]).

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 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 the full version of this paper [GPS22]. We also refer the readers to the full version of this paper [GPS22] 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 $\texttt{pos} = (p_1, p_2, \ldots, p_k)$ 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 $\texttt{pos}$ for the secrets. We will use $[\boldsymbol{x} \| \texttt{pos}]_d$ to denote a degree-d packed Shamir sharing of $\boldsymbol{x} \in \mathbb{F}^k$ stored at positions $\texttt{pos}$. 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.

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.

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 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 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 $[\boldsymbol{z}\|\mathsf{pos}]_{n-k}$. 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.

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.

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 the full version of this paper [GPS22].

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 the full version of this paper [GPS22].

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.[5] 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 $\text{poly}(\kappa)$.

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].

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. Corrupted clients and servers can deviate from the protocol arbitrarily. One benefit of the client-server model is that it is sufficient to only consider maximum adversaries, i.e., adversaries which corrupt exactly t parties. We refer the readers to the full version of this paper [GPS22] for more details about the security definition and the benefit of the client-server model. 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 $\mathcal{I} = \{1, \dots, n\}$.*

[5] 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.

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

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

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 $\boldsymbol{v} \in U^A$,*

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

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

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

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 $|A| \le t$, A is a privacy set.

We are interested in the following property.

Property 1 (Multiplication-Friendliness). We say $\Sigma = (n, Z = \mathcal{R}^k, U, C, \texttt{share}, \texttt{rec})$ is multiplication-friendly if there is an $\mathcal{R}$-arithmetic secret sharing scheme $\Sigma' = (n, Z = \mathcal{R}^k, U', C', \texttt{share}', \texttt{rec}')$ 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 $\boldsymbol{Y}$.

– $\texttt{rec}'(\boldsymbol{Y}) = \boldsymbol{c} * \texttt{rec}(\boldsymbol{X})$, 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. We prove Lemma 1 in the full version of this paper [GPS22].

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.*

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 $\texttt{pos} = (p_1, p_2, \ldots, p_k)$ 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 $\texttt{pos}$ for the secrets. We will use $[\boldsymbol{x}\|\texttt{pos}]_d$ to denote a degree-d packed Shamir sharing of $\boldsymbol{x} \in \mathbb{F}^k$ stored at positions $\texttt{pos}$. 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 $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x}+\boldsymbol{y}\|\texttt{pos}]_d = [\boldsymbol{x}\|\texttt{pos}]_d + [\boldsymbol{y}\|\texttt{pos}]_d$.
– 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 $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x} * \boldsymbol{y}\|\texttt{pos}]_{d_1+d_2} = [\boldsymbol{x}\|\texttt{pos}]_{d_1} \cdot [\boldsymbol{y}\|\texttt{pos}]_{d_2}$.

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 $[\boldsymbol{c} * \boldsymbol{x}\|\texttt{pos}]_{d+k-1}$ from $[\boldsymbol{x}\|\texttt{pos}]_d$ and the public vector $\boldsymbol{c}$. To see this, all parties can locally transform $\boldsymbol{c}$ to a degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{c}\|\texttt{pos}]_{k-1}$.

Then, they can use the property of the packed Shamir sharing scheme to compute $[\boldsymbol{c} * \boldsymbol{x} \| \texttt{pos}]_{d+k-1} = [\boldsymbol{c} \| \texttt{pos}]_{k-1} \cdot [\boldsymbol{x} \| \texttt{pos}]_d$.

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 $\Pi = (n, \tilde{Z}, \tilde{U}, \tilde{C}, \texttt{share}_\Pi, \texttt{rec}_\Pi)$ be an $\mathcal{R}$-arithmetic secret sharing scheme. Our goal is to realize the functionality $\mathcal{F}_{\text{rand-sharing}}$ presented in Functionality 1.

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.

Initialization. Let $\Sigma = (n, Z = \mathcal{R}^k, U, C, \texttt{share}, \texttt{rec})$ 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 $\Sigma' = (n, Z' = \mathcal{R}^k, U', C', \texttt{share}', \texttt{rec}')$ 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

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

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 $\boldsymbol{0} \in \mathcal{R}^k$.

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

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.

$\tilde{U} = \mathcal{R}^{\tilde{\ell}}$. 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 $\tilde{k}_i + \tilde{r}_i \leq n \cdot \tilde{\ell}$.

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. We prove Lemma 2 in the full version of this paper [GPS22]. 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 also be found in the full version of this paper [GPS22].

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 $\{\mathcal{F}_{\textit{rand-sharing}}(\Pi_i)\}_{i=1}^k$ in the $\{\mathcal{F}_{\textit{rand}}, \mathcal{F}_{\textit{randZero}}\}$-hybrid model against a semi-honest adversary who controls t parties.*

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

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

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 $\texttt{share}_i : \tilde{Z}_i \times \mathcal{R}^{\tilde{r}_i} \to \tilde{C}_i$ 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 $\boldsymbol{X}_i = \texttt{share}_i(\boldsymbol{\tau}_i, \boldsymbol{\rho}_i)$.
3. All parties invoke $\mathcal{F}_{\text{randZero}}$ $n \cdot \tilde{\ell}$ times and obtain $n \cdot \tilde{\ell}$ random Σ'-sharings of $\boldsymbol{0} \in \mathcal{R}^k$, 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 $\texttt{share}_i(\boldsymbol{\tau}, \boldsymbol{\rho})$. 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.

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.

In the full version of this paper [GPS22], we discuss how to instantiate Protocol Rand-Sharing for small fields $\mathbb{F}_q$ and rings $\mathbb{Z}/p^\ell\mathbb{Z}$.

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

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 $\texttt{rec}'(\boldsymbol{Y}) = f(\texttt{rec}(\boldsymbol{X}))$, where $\texttt{rec}$ and $\texttt{rec}'$ 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 $\texttt{rec}'(\boldsymbol{R}') = f(\texttt{rec}(\boldsymbol{R}))$. 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 $\texttt{rec}$ 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.

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.*

A formal theorem of Theorem 1 can be found in the full version of this paper [GPS22].

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

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 $f(\texttt{rec}(\boldsymbol{X}))$.
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 $\texttt{rec}'(\boldsymbol{Y}) = f(\texttt{rec}(\boldsymbol{X}))$ 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 `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 $\boldsymbol{Y} = \boldsymbol{W} - \boldsymbol{R}'$.

circuit against a semi-honest adversary who controls t parties. Let $k = (n - t + 1)/2$.

Recall that we use $[\boldsymbol{x}\|\texttt{pos}]_d$ to represent a degree-d packed Shamir sharing of $\boldsymbol{x} \in \mathbb{F}^k$ stored at positions $\texttt{pos} = (p_1, p_2, \ldots, p_k)$. 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.

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.

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 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.

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.

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

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

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

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

For all $1 \leq i \leq \ell_1 + \ell_2$, let f_i be the polynomial corresponding to $[\boldsymbol{e}_i \| \texttt{pos}]_{k-1}$. For all $1 \leq i \leq \ell_1$, let g_i be the polynomial corresponding to $[\boldsymbol{x}^{(i)} \| \texttt{pos}^{(i)}]_{n-k}$. 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 $1 \leq j \leq \ell_1$,

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

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

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

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

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

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 $\mathcal{F}_{\mathrm{tran}}$ 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.

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.

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.

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

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 $[\boldsymbol{x}^{(i)} \| \mathsf{pos}^{(i)}]_{n-k}$ 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 $\mathsf{pos} = (p_1, \ldots, p_k)$.
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 $[\boldsymbol{e}_i \| \mathsf{pos}]_{k-1}$.
5. All parties locally compute
$$[\boldsymbol{x}' \| \mathsf{pos}]_{n-1} = \sum_{i=1}^{\ell_1} [\boldsymbol{e}_i \| \mathsf{pos}]_{k-1} \cdot [\boldsymbol{x}^{(i)} \| \mathsf{pos}^{(i)}]_{n-k} + \sum_{i=1}^{\ell_2} c_i \cdot [\boldsymbol{e}_{\ell_1+i} \| \mathsf{pos}]_{k-1}.$$
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 pos. Let Σ' be the degree-$(n-k)$ packed Shamir secret sharing scheme that stores secrets at positions $\boldsymbol{\beta}$.
   All parties invoke $\mathcal{F}_{\mathrm{tran}}$ with (Σ, Σ', f) and $[\boldsymbol{x}' \| \mathsf{pos}]_{n-1}$, 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 $\mathsf{pos} = (p_1, p_2, \ldots, p_k)$ 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 pos. Let $I : \mathbb{F}^k \to \mathbb{F}^k$ be the identity map.
   All parties invoke $\mathcal{F}_{\mathrm{tran}}$ with (Σ, Σ', I) and $[\boldsymbol{z}]_{n-k}$, and output $[\boldsymbol{z} \| \mathsf{pos}]_{n-k}$.

Main Protocol Given the above protocols the main semi-honest protocol follows in a straightforward way. We refer the readers to the full version of this paper [GPS22] for the description of our main protocol, the security proof, and the analysis of the cost. Overall we obtain the following theorem.

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 $\texttt{pos} = (p_1, p_2, \ldots, p_k)$ 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 $\texttt{pos}$. 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 $[\boldsymbol{z}\|\texttt{pos}]_{n-k}$.

Theorem 3. *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.*

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

References

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.

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.

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.

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.

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 699âĂŞ710, New York, NY, USA, 1992. Association for Computing Machinery.

GIP15. Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. Efficient multiparty 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 inÂăthe 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.

GPS22. Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority mpc with packed secret sharing. Cryptology ePrint Archive, 2022.

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.

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.

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