Unconditional Communication-Efficient MPC via Hall's Marriage Theorem

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

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

Abstract. The best known n party unconditional multiparty computation protocols with an optimal corruption threshold communicates $O(n)$ field elements per gate. This has been the case even in the semi-honest setting despite over a decade of research on communication complexity in this setting. Going to the slightly sub-optimal corruption setting, the work of Damgård, Ishai, and Krøigaard (EUROCRYPT 2010) provided the first protocol for a single circuit achieving communication complexity of $O(\log |C|)$ elements per gate. While a number of works have improved upon this result, obtaining a protocol with $O(1)$ field elements per gate has been an open problem.

In this work, we construct the first unconditional multi-party computation protocol evaluating a single arithmetic circuit with amortized communication complexity of $O(1)$ elements per gate.

1 Introduction

Secure Multi-Party Computation (MPC) enables a set of n parties to mutually run a protocol that computes some function f on their private inputs without compromising the privacy of their inputs or the correctness of the outputs [Yao82,GMW87,CCD88,BOGW88]. An important distinction in designing

V. Goyal, Y. Song—Supported in part by the NSF award 1916939, DARPA SIEVE program, 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.

MPC protocols is that of the power of the adversary. An adversary in a semi-honest protocol follows the protocols specification but tries to learn information from the received messages, and an adversary in a malicious protocol is allowed to deviate from the protocols specification in arbitrary ways.

In this work, our focus is on the communication complexity of information theoretic protocols evaluating an arithmetic circuit in the presence of semi-honest or malicious adversaries. The "dream" in the unconditional setting is to get as close to $|C|$ as possible (or even below) where $|C|$ is the circuit size. The best known protocols in the so called optimal threshold regime tolerating $t = (n-1)/2$ corrupted parties require communicating $O(n \cdot |C|)$ field elements (ignoring circuit independent terms) [DN07,GIP+14,CGH+18,NV18,BBCG+19,GSZ20,BGIN20]. There are no constructions known beating this barrier even in the semi-honest setting despite over a decade of research.

Moving to Sub-optimal Corruption Threshold. In a remarkable result, Damgård et al. [DIK10] showed an unconditional MPC protocol with communication complexity of $O(\log |C| \cdot n/k)$ per gate (ignoring circuit independent terms) tolerating $t' = (n-1)/3-k+1$ corrupted parties. This was later extended by Genkin et al. [GIP15] to obtain a construction tolerating $t' = (n-1)/2-k+1$ corrupted parties with also a constant factor improvement in the communication complexity. These works rely on the packed secret sharing technique introduced by Franklin and Yung [FY92] where k secrets are packed into a single secret sharing. An incomparable result was given by Garay et al. [GIOZ17] who obtained a protocol with communication complexity $O(\log^{1+\delta} n \cdot |C|)$ where δ is any positive constant. If one was interested in evaluating the same circuit multiple times on different inputs, *Franklin and Yung* [FY92] showed how to use packed secret sharing to evaluate k copies of the circuit with *amortized* communication complexity of $O(n/k)$ elements per gate or $O(1)$ elements when $k = O(n)$. However in case of a single circuit evaluation, the works mentioned [GIP15,GIOZ17] remains the best known.

To our knowledge, there is no known unconditional MPC protocol which only requires communicating $O(1)$ field elements per gate for any corruption threshold (assuming the number of corrupted parties is at least super-constant). This raises the following natural question:

Is it possible to construct information theoretic MPC protocols for computing a single *arithmetic circuit with communication complexity $O(1)$ field elements per gate?*

We answer the above question in the affirmative by constructing an information theoretic n-party protocol based on packed secret sharing for an arithmetic circuit over a finite field $\mathbb{F}$ of size $|\mathbb{F}| \geq 2n$. Our communication complexity amortized over the multiplication gates within the same circuit (rather than amortized over multiple circuits) is $O(n/k)$ field elements per multiplication gate. Informally, we prove the following:

Theorem 1 (informal). *Assume a point-to-point channel between every pair or parties. For all $1 \leq k \leq t$ where $t = \lfloor (n-1)/2 \rfloor$, there exists an information theoretic n-party MPC protocol which securely computes a single arithmetic circuit in the presence of a semi-honest (malicious) adversary controlling up to $t-k+1$ parties with an communication complexity of $O(n/k)$ field elements per multiplication gate. For the case where $k = O(n)$, the achieved communication complexity is $O(1)$ elements per gate. In addition, our finite field $\mathbb{F}$ is of size $|\mathbb{F}| \geq 2n$.*

Our formal theorem for semi-honest security with perfect security can be found in Theorem 6 and we refer the readers to the full version of this paper [GPS21] for the formal theorem for malicious security (with abort and statistical security). In order to achieve these results, we introduce a set of combinatorial lemmas which could be of independent interest. In particular, we marry packed secret sharing with techniques from graph theory. A key technical challenge with using packed secret sharing in the context of a single circuit is to make sure that all the required secrets for a batch of gates appear in a single packed secret sharing. In addition, one needs to ensure that these secrets appear in the correct order. Our key technical contributions in this paper relate to performing secure permutations of the secrets efficiency by using techniques from perfect matching in bipartite graphs. In particular, we make an extensive use of Hall's Marriage Theorem.

2 Technical Overview

In the following, we will use $n = 2t + 1$ to denote the number of parties. Let $1 \leq k \leq t$ be an integer. We consider the scenario where an adversary is allowed to corrupt $t' = t - k + 1$ parties. For simplicity, we focus on the semi-honest setting. We will discuss how to achieve malicious security at a later point.

Our construction will use the packed secret-sharing technique introduced by Franklin and Yung [FY92]. This is a generalization of the standard Shamir secret sharing scheme [Sha79]. It allows to secret-share a batch of secrets within a single Shamir sharing. In the case that $t' = t - k + 1$, we can use a degree-t Shamir sharing, which requires $t + 1$ shares to reconstruct the whole sharing, to store k secrets such that any t' shares are independent of the secrets. We refer to such a sharing as a degree-t packed Shamir sharing. Let $\boldsymbol{x}$ be a vector of dimension k. We use $[\boldsymbol{x}]$ to denote a degree-t packed Shamir sharing of the secrets $\boldsymbol{x}$.

In this work, we are interested in the information-theoretic setting. Our goal is to construct a semi-honest MPC protocol for a *single* arithmetic circuit over a finite field $\mathbb{F}$ (of size $|\mathbb{F}| \geq 2n$), such that the amortized communication complexity (of each party) per gate is $O(n/k)$ elements. Note that when $k = O(n)$, the amortized communication complexity per gate becomes $O(1)$ elements.

2.1 Background: Using the Packed Secret-sharing Technique in MPC

In the information-theoretic setting, a general approach to construct an MPC protocol is to compute a secret sharing for each wire of the circuit. The circuit is evaluated gate by gate, and the problem is reduced to compute the output sharing of an addition gate or a multiplication gate given the input sharings. When the corruption threshold can be relaxed to $t' = t - k + 1$ where $t = \frac{n-1}{2}$, a natural way of using the packed secret-sharing technique [FY92] is to compute $k \geq 1$ copies of the same circuit (i.e., a SIMD circuit): by storing the value related to the i-th copy in the i-th position of the secret sharing for each wire, all copies of the same circuit are evaluated simultaneously. Moreover, the communication complexity of a single operation for packed secret sharings is usually the same as that for standard secret sharings. Effectively, the amortized communication complexity per copy is reduced by a factor of k.

In 2010, Damgård et al. [DIK10] provided the first protocol of using packed secret-sharing technique to evaluate *a single circuit*. The original work focuses on the corruption threshold $t' < (1/3 - \epsilon)n$ and perfect security. It is later extended by [GIP15] to the setting of security with abort against $t' < (1/2 - \epsilon)n$ corrupted parties with a constant factor improvement in the communication complexity[4]. At a high-level, the idea is to divide the gates of the same type in each layer into groups of k. Each group of gates will be evaluated at the same time. For each group of gates, all parties need to prepare the input sharings by using the output sharings from previous layers. Unlike the case when evaluating a SIMD circuit, input sharings for each group of gates do not come for free:

- The secrets needed to be in a single sharing may be scattered in different output sharings of previous layers.
- Even if we have all the secrets in a single sharing, we need the secrets to be in the correct order so that the i-th secret is the input of the i-th gate.

The naive approach of preparing a single input sharing by collecting the secret one by one would require $O(k)$ operations, which eliminates the benefit of using the packed secret-sharing technique. In [DIK10], they solve this problem by compiling the circuit into a special form of a universal circuit such that it can be viewed as k copies of the same circuit. In particular, the compilation uses the so-called Beneš network, which increases the circuit size by a factor of $\log |C|$, where $|C|$ is the circuit size. As a result, the amortized communication complexity per gate is $O(\log |C| \cdot n/k)$ elements.

Our work aims to remove the $\log |C|$ factor in the communication complexity and achieves the same communication efficiency as that for the evaluation of many copies of the same circuit. In this paper, we describe our idea from the bottom up:

[4] While the semi-honest version of the protocol in [GIP15] can use a field $\mathbb{F}$ of size $O(n)$, the maliciously secure protocol requires to use a large enough field since the error probability is proportional to the field size.

1. We start with the basic protocols to evaluate input gates, addition gates, multiplication gates, and output gates using the packed Shamir sharing scheme. These protocols are simple variants of the protocols in [DN07], which focuses on the adversary that can corrupt t parties.
2. To use these protocols to evaluate addition gates and multiplication gates, we need the secrets in the input packed Shamir sharings to have the correct order. Assuming each input sharing contains all the secrets we want, we discuss how to permute the secrets in each input sharing to the correct order.
3. Next, we show how to collect the secrets of an input packed Shamir sharing from the output sharings of previous layers. Our solution requires that each output wire from each layer is only used once in the computation, as an input wire to a single layer. This requirement can be met by further requiring that there is a fan-out gate right after each gate that copies the output wire the number of times it is used in later layers.
4. After that, we discuss how to evaluate fan-out gates efficiently.
5. Finally, we discuss how to achieve malicious security.

Our key techniques lie on the second point and the third point. We will focus on these two points in the technical overview, which are in Section 2.2 and Section 2.3. We will briefly discuss the last two points in Section 2.4 and Section 2.6.

2.2 Performing an Arbitrary Permutation on the Secrets of a Single Sharing

During the computation, we may encounter the scenario that the order of the secrets is not what we want. For example, when $k = 2$ and we want to compute two multiplication gates with input secrets $(x_1, y_1), (x_2, y_2)$, ideally we want all parties to hold two packed Shamir sharings of $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{y} = (y_1, y_2)$ so that when we use the multiplication protocol with these two packed Shamir sharing, we can obtain a packed Shamir sharing of the secret $\boldsymbol{x} * \boldsymbol{y} = (x_1 \cdot y_1, x_2 \cdot y_2)$. During the computation, however, all parties may hold two packed Shamir sharings of $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{y}' = (y_2, y_1)$. In particular, the secrets in the second sharing are not in the order we want. Using these two packed Shamir sharings in the multiplication protocol, we can only obtain a packed Shamir sharing of $\boldsymbol{x} * \boldsymbol{y}' = (x_1 \cdot y_2, x_2 \cdot y_1)$ instead of the correct result $\boldsymbol{x} * \boldsymbol{y} = (x_1 \cdot y_1, x_2 \cdot y_2)$.

To solve it, we need to construct a protocol which allows all parties to perform an arbitrary permutation on the secrets of a single sharing. Let $p(\cdot)$ be a permutation over $\{1, 2, \ldots, k\}$. We use F_p to denote the linear map which maps $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$ to $\tilde{\boldsymbol{x}} = (x_{p(1)}, x_{p(2)}, \ldots, x_{p(k)})$. Given the input sharing $[\boldsymbol{x}]$, the goal is to compute a degree-t packed Shamir sharing $[F_p(\boldsymbol{x})]$.

We first review the approach in [DIK10] for permuting the secrets of $[\boldsymbol{x}]$:

1. All parties prepare two random degree-t packed Shamir sharings $([\boldsymbol{r}], [\tilde{\boldsymbol{r}}])$, where $\tilde{\boldsymbol{r}} = F_p(\boldsymbol{r})$ and $p(\cdot)$ is the permutation we want to perform.

2. All parties locally compute $[\boldsymbol{e}] := [\boldsymbol{x}] + [\boldsymbol{r}]$ and send their shares to the first party P_1.
3. P_1 reconstructs the secrets $\boldsymbol{e}$ and computes $\tilde{\boldsymbol{e}} = F_p(\boldsymbol{e})$. P_1 generates a random degree-t packed Shamir sharing $[\tilde{\boldsymbol{e}}]$ and distributes the shares to other parties.
4. All parties locally compute $[\tilde{\boldsymbol{x}}] := [\tilde{\boldsymbol{e}}] - [\tilde{\boldsymbol{r}}]$.

To see the correctness, note that in the second step we have $\boldsymbol{e} = \boldsymbol{x}+\boldsymbol{r}$. Therefore,

$$\tilde{\boldsymbol{x}} = F_p(\boldsymbol{x}) = F_p(\boldsymbol{e} - \boldsymbol{r}) = F_p(\boldsymbol{e}) - F_p(\boldsymbol{r}) = \tilde{\boldsymbol{e}} - \tilde{\boldsymbol{r}}.$$

The communication complexity of this protocol is $O(n/k)$ elements per secret (excluding the cost for the preparation of $([\boldsymbol{r}], [\tilde{\boldsymbol{r}}])$).

As noted in [DIK10], the main issue of this approach is how to *efficiently* prepare a pair of random sharings $([\boldsymbol{r}], [\tilde{\boldsymbol{r}}])$. Although there are known techniques to prepare random sharings $([\boldsymbol{r}], [\tilde{\boldsymbol{r}}])$ for *a fixed* permutation p such that the amortized communication complexity per pair is $O(n)$ elements where in turn the amortized cost per secret is $O(n/k)$ elements, these techniques suffer a large overhead (at least $O(n^2)$ elements) that is independent of the number of sharings we want to prepare. It means that the overhead of preparing random sharings depends on the number of different permutations we want to perform. In the worst case where each time we need to perform a different permutation, the overhead of each pair of random sharings is as large as $O(n^2)$ elements, which eliminates the benefit of using the packed Shamir sharing scheme. In [DIK10], this issue is solved by compiling the circuit such that only $O(\log n)$ different permutations are needed in the computation with the cost of blowing up the circuit size by a factor of $O(\log |C|)$, where $|C|$ is the circuit size. This approach does not achieve our goal since the amortized communication complexity per gate becomes $O(\log |C| \cdot n/k)$ elements. To generate random sharings for m permutations, our idea is to first generate random sharings for a limited number ($O(n^2)$) of different permutations which are related to the input permutations, and then transform them to the random sharings for the desired permutations (the input permutations). In this way, since we only need to prepare random sharings for $O(n^2)$ different permutations, we do not suffer the quadratic overhead in the communication complexity even if all the input permutations are different. Moreover, we do not need to compile the circuit and therefore do not suffer the $O(\log |C|)$ factor in the communication complexity as that in [DIK10]. As a result, the amortized communication complexity of our permutation protocol is $O(n/k)$ elements per secret.

Before introducing our idea, we first introduce a useful functionality $\mathcal{F}_{\text{select}}$, which selects secrets from one or more packed Shamir sharings and outputs a single sharing which contains the chosen secrets. Later on, we will use $\mathcal{F}_{\text{select}}$ to solve the above issue of preparing random sharings for permutations. Concretely, $\mathcal{F}_{\text{select}}$ takes as input k degree-t packed Shamir sharings $\{[\boldsymbol{x}^{(i)}]\}_{i=1}^{k}$ (which do not need to be distinct) and outputs a degree-t packed Shamir sharing of $\boldsymbol{y}$ such that $y_i = x_i^{(i)}$. Effectively, $\mathcal{F}_{\text{select}}$ chooses the i-th secret of $[\boldsymbol{x}^{(i)}]$ and generates a new degree-t packed Shamir sharing $[\boldsymbol{y}]$ that contains the chosen secrets. Note that *the secrets we choose are from different positions and the positions of these*

secrets remain unchanged in the output sharing. To realize $\mathcal{F}_{\text{select}}$, we observe that $\boldsymbol{y}$ can be computed by $\sum_{i=1}^{k} \boldsymbol{e}^{(i)} * \boldsymbol{x}^{(i)}$, where $\boldsymbol{e}^{(i)}$ is a constant vector where the i-th entry is 1 and all other entries are 0, and $*$ denotes the coordinate-wise multiplication operation. We realize $\mathcal{F}_{\text{select}}$ by extending the basic protocol for multiplication gates as described in Section 4.2. The amortized communication complexity of $\mathcal{F}_{\text{select}}$ is $O(n/k)$ elements per secret.

Using $\mathcal{F}_{select}$ to Generate Random Sharings for Permuting Secrets. For all $i, j \in \{1, 2, \ldots, k\}$, we say a pair of degree-t packed Shamir sharings $([\boldsymbol{x}], [\boldsymbol{y}])$ contains an (i, j)-component if $x_i = y_j$. To perform a permutation $p(\cdot)$, we need to prepare two random degree-t packed Shamir sharings $([\boldsymbol{r}], [F_p(\boldsymbol{r})])$. We can view $([\boldsymbol{r}], [F_p(\boldsymbol{r})])$ as a composition of an $(i, p(i))$-component for all $i \in [k]$.

Now we introduce a new approach for preparing random sharings $([\boldsymbol{r}], [F_p(\boldsymbol{r})])$:

1. Let $q_1, q_2, \ldots, q_k$ be k different permutations over $\{1, 2, \ldots, k\}$ such that for all $i \in [k]$, $q_i(i) = p(i)$.
2. All parties prepare a pair of random sharings for each permutation q_i, denoted by $([\boldsymbol{r}^{(i)}], [F_{q_i}(\boldsymbol{r}^{(i)})])$. Since $q_i(i) = p_i$, $([\boldsymbol{r}^{(i)}], [F_{q_i}(\boldsymbol{r}^{(i)})])$ contains an $(i, p(i))$-component.
3. To prepare $([\boldsymbol{r}], [F_p(\boldsymbol{r})])$, we can use $\mathcal{F}_{\text{select}}$ to select the $(i, p(i))$-component from $([\boldsymbol{r}^{(i)}], [F_{q_i}(\boldsymbol{r}^{(i)})])$ for all $i \in [k]$. More concretely, for $[\boldsymbol{r}]$, we use $\mathcal{F}_{\text{select}}$ to select the i-th secret of $[\boldsymbol{r}^{(i)}]$ for all $i \in [k]$. For $[F_p(\boldsymbol{r})]$, we use $\mathcal{F}_{\text{select}}$ to select the $p(i)$-th secret of $[F_{q_i}(\boldsymbol{r}^{(i)})]$ for all $i \in [k]$.

While this way of preparing a single pair of random sharings for the permutation p requires k pairs of random sharings for k permutations $q_1, \ldots, q_k$, we note that *the unused components of* $([\boldsymbol{r}^{(i)}], [F_{q_i}(\boldsymbol{r}^{(i)})])$ can potentially be used to prepare random sharings for other permutations.

In general, when we want to prepare random sharings for m permutations $p_1(\cdot), p_2(\cdot), \ldots, p_m(\cdot)$, relying on $\mathcal{F}_{\text{select}}$, it is sufficient to alternatively prepare random sharings for m permutations $q_1(\cdot), q_2(\cdot), \ldots, q_m(\cdot)$ such that:

- For all $i, j \in \{1, 2, \ldots, k\}$, the number of permutations $p \in \{p_1, p_2, \ldots, p_m\}$ which satisfies that $p(i) = j$ is equal to the number of permutations $q \in \{q_1, q_2, \ldots, q_m\}$ which satisfies that $q(i) = j$.

Then, from $i = 1$ to m, a pair of random sharings for the permutation p_i can be prepared by using $\mathcal{F}_{\text{select}}$ to choose the first unused $(j, p_i(j))$-component for all $j \in [k]$.

The major benefit of this approach is that *we can limit the number of different permutations in* $\{q_1, q_2, \ldots, q_m\}$ as we show in Theorem 2.

Theorem 2. *Let $m, k \geq 1$ be integers. For all m permutations $p_1, p_2, \ldots, p_m$ over $\{1, 2, \ldots, k\}$, there exists m permutations $q_1, q_2, \ldots, q_m$ over $\{1, 2, \ldots, k\}$ such that:*

- *For all $i, j \in \{1, 2, \ldots, k\}$, the number of permutations $p \in \{p_1, p_2, \ldots, p_m\}$ such that $p(i) = j$ is the same as the number of permutations $q \in \{q_1, q_2, \ldots, q_m\}$ such that $q(i) = j$.*

– $q_1, q_2, \ldots, q_m$ *contain at most* k^2 *different permutations.*

Moreover, $q_1, q_2, \ldots, q_m$ *can be found within polynomial time given* $p_1, p_2, \ldots, p_m$.

Recall that the issue of using known techniques to prepare random sharings for $p_1, p_2, \ldots, p_m$ is that there will be an overhead of $O(n^2)$ elements per different permutation in $p_1, p_2, \ldots, p_m$. Relying on $\mathcal{F}_{\text{select}}$, we only need to prepare random sharings for permutations $q_1, \ldots, q_m$, which contain at most $k^2 \leq n^2$ different permutations. In this way, the overhead is independent of the number of permutations and the circuit size. Recall that the amortized communication complexity for each pair of random sharings is $O(n/k)$ elements per secret, and our protocol for $\mathcal{F}_{\text{select}}$ and the permutation protocol from [DIK10] also have the same amortized communication complexity, i.e., $O(n/k)$ elements per secret. Therefore, the overall communication complexity to perform an arbitrary permutation on the secrets of a single secret sharing is $O(n/k)$ elements per secret.

Using Hall's Marriage Theorem to Prove Theorem 2. We note that Theorem 2 has a close connection to graph theory. We first introduce two basic notions.

- For a graph $G = (V, E)$, we say G is a bipartite graph if there exists a partition (V_1, V_2) of V such that all edges are between vertices in V_1 and vertices in V_2. Such a graph is denoted by $G = (V_1, V_2, E)$.
- For a bipartite graph $G = (V_1, V_2, E)$ where $|V_1| = |V_2|$, a perfect matching is a subset of edges $\mathcal{E} \in E$ which satisfies that each vertex in the sub-graph $(V_1, V_2, \mathcal{E})$ has degree exactly 1.

Note that a permutation p over $\{1, 2, \ldots, k\}$ corresponds to a perfect matching in a bipartite graph: the set of vertices are $V_1 = V_2 = \{1, 2, \ldots, k\}$, and the set of edges are $\mathcal{E} = \{(i, p(i))\}_{i=1}^{k}$.

We first construct a bipartite graph $G = (V_1, V_2, E)$ where $V_1 = V_2 = \{1, 2, \ldots, k\}$ and E contains all edges in the perfect matching that $p_1, p_2, \ldots, p_m$ correspond to. Strictly speaking, G is a multi-graph since a pair of vertices may have multiple edges. Note that Theorem 2 is equivalent to decomposing G into m perfect matching such that the number of different perfect matching is bounded by k^2. Our idea of finding these m perfect matching is to repeat the following steps until E becomes empty:

1. We first find a perfect matching $\mathcal{E} \subset E$ in G.
2. We repeatedly remove $\mathcal{E}$ from E until $\mathcal{E}$ is no longer a subset of E. The number of times that $\mathcal{E}$ is removed from E is the number of times that $\mathcal{E}$ appears in the output perfect matching.

Note that the number of different perfect matches is the same as the number of iterations of the above two steps. Suppose the first step always succeeds. The second step guarantees that in each iteration, we will completely use up the edges between one pair of vertices in E. Since there are at most k^2 different pairs of vertices, the above process will terminate within k^2 iterations.

For the first step, we use Hall's Marriage Theorem to prove the existence of a perfect matching.

Theorem 3 (Hall's Marriage Theorem). *For a bipartite graph (V_1, V_2, E) such that $|V_1| = |V_2|$, there exists a perfect matching iff for all subset $V_1' \subset V_1$, the number of the neighbors of vertices in V_2 is at least $|V_1'|$.*

Hall's Marriage Theorem is a well-known theorem in graph theory which has many applications in mathematics and computer science. It provides a necessary and sufficient condition of the existence of a perfect matching in a bipartite graph. In addition, there are known efficient polynomial-time algorithms to find a perfect matching in a bipartite graph, e.g. the Hopcroft-Karp algorithm.

To prove the existence of a perfect matching, we show that the graph G at the beginning of each iteration satisfies the necessary and sufficient condition in Hall's Marriage Theorem. We say a bipartite graph $G' = (V_1', V_2', E')$ is d-regular if the degree of each vertex in $V_1' \bigcup V_2'$ is d. A well-known corollary of Hall's Marriage Theorem states that:

Corollary 1. *There exists a perfect matching in a d-regular bipartite graph.*

Therefore, it is sufficient to show that the graph G at the beginning of each iteration is a d-regular bipartite graph. Recall that in the beginning, the set of edges E contains all edges in the perfect matching that $p_1, p_2, \ldots, p_m$ correspond to. Since by definition, the degree of each vertex in a perfect matching is exactly 1, the degree of each vertex in G is m, which means that G is a m-regular bipartite graph. In each iteration, we first find a perfect matching in Step 1 and then repeatedly remove this perfect matching from E in Step 2. Each time of removing a perfect matching reduces the degree of each vertex in G by 1. Thus, G is still a d-regular bipartite graph after each remove of a perfect matching. Therefore, the graph G at the beginning of each iteration is a d-regular bipartite graph.

2.3 Obtaining Input Sharings for Multiplication Gates and Addition Gates

So far, we have introduced how to perform a permutation to the secrets of a single sharing to obtain the correct order. However, this only solves the problem when we have all the values we want in a single sharing. During the computation, such a sharing does not come for free since the values we want may be scattered in one or more output sharings of previous layers. This requires us to collect the secrets from those sharings and generate a single sharing for these secrets efficiently.

Our starting point is the functionality $\mathcal{F}_{\text{select}}$. Recall that $\mathcal{F}_{\text{select}}$ allows us to select secrets from one or more sharings and generate a new sharing for the chosen secrets if the secrets we select are *in different positions*. To use $\mathcal{F}_{\text{select}}$, we consider what we call the *non-collision* property stated in Property 1.

Property 1 (Non-collision). For each input sharing of each layer, the secrets of this input sharing come from different positions in the output sharings of previous layers.

Note that if we can guarantee the non-collision property, then we can use $\mathcal{F}_{\text{select}}$ to generate the input sharing we want. Unfortunately, this property does not hold in general. A counterexample is that we need the same secret twice in a single input sharing. Then these two secrets will always come from the same position. To solve this problem, we require that

- every output wire of the input layer and all intermediate layers is used exactly once as an input wire of a later layer (which may not be the next layer).

Note that this requirement can be met without loss of generality by assuming that there is a fan-out gate right after each (input, addition, or multiplication) gate that copies the output wire the number of times it is used in later layers. In the next subsection, we will discuss how to evaluate fan-out gates efficiently. With this requirement, there is a bijective map between the output wires (of the input layer and all intermediate layers) and the input wires (of the output layer and all intermediate layers).

Note that only meeting this requirement is not enough: it is still possible that two secrets of a single input sharing come from the same position but in two different output sharings. Our idea is to perform a permutation on each output sharing to achieve the non-collision property.

Since every output wire from every layer is only used once as an input wire of another layer, the number of output sharings in the circuit is the same as the number of input sharings in the circuit. Let m denote the number of output packed Shamir sharings of the input layer and all intermediate layers in the circuit. Then the number of input packed Shamir sharings of the output layer and all intermediate layers is also m. We label all the output sharings by $1, 2, \ldots, m$ and all the input sharings also by $1, 2, \ldots, m$. Consider a matrix $\boldsymbol{N} \in \{1, 2, \ldots, m\}^{m \times k}$ where $\boldsymbol{N}_{i,j}$ is the index of the input sharing that the j-th secret of the i-th output sharing wants to go to. Then for all $\ell \in \{1, 2, \ldots, m\}$, there are exactly k entries of $\boldsymbol{N}$ which are equal to ℓ. We will prove the following theorem.

Theorem 4. *Let $m \geq 1, k \geq 1$ be integers. Let $\boldsymbol{N}$ be a matrix of dimension $m \times k$ in $\{1, 2, \ldots, m\}^{m \times k}$ such that for all $\ell \in \{1, 2, \ldots, m\}$, the number of entries of $\boldsymbol{N}$ which are equal to ℓ is k. Then, there exists m permutations $p_1, p_2, \ldots, p_m$ over $\{1, 2, \ldots, k\}$ such that after performing the permutation p_i on the i-th row of $\boldsymbol{N}$, the new matrix $\boldsymbol{N}'$ satisfies that each column of $\boldsymbol{N}'$ is a permutation over $(1, 2, \ldots, m)$. Furthermore, the permutations $p_1, p_2, \ldots, p_m$ can be found within polynomial time.*

Jumping ahead, when we apply p_i to the i-th output sharing for all $i \in \{1, 2, \ldots, m\}$, Theorem 4 guarantees that for all $j \in \{1, 2, \ldots, k\}$ the j-th secrets of all output sharings want to go to different input sharings. Note that this ensures the non-collision property. During the computation, we will perform the permutation p_i on the i-th output sharing right after it is computed. Note that when preparing an input sharing, the secrets we need only come from the output sharings which have been computed. The secrets of these output sharings have

been properly permuted such that the secrets we want are in different positions. Therefore, we can use $\mathcal{F}_{\text{select}}$ to choose these secrets and obtain the desired input sharing.

Using Hall's Marriage Theorem to Prove Theorem 4. Let $\boldsymbol{N}$ be the matrix in Theorem 4. Our idea is to repeat the following steps:

1. In the ℓ-th iteration, for each row of $\boldsymbol{N}$, we pick a value in the last $k-\ell+1$ entries of this row (so that the first $\ell-1$ entries will not be chosen), such that the values we pick in all rows form a permutation over $\{1, 2, \ldots, m\}$.
2. For each row of $\boldsymbol{N}$, we swap the ℓ-th entry with the value we picked in this row. In this way, the ℓ-th column of $\boldsymbol{N}$ is a permutation over $\{1, 2, \ldots, m\}$.

Note that in each iteration, we switch two elements in each row. At the end of the above process, we can compute the permutation for each row based on the elements we switched in each iteration.

To make this idea work, we need to show that we can always find the values which form a permutation over $\{1, 2, \ldots, m\}$ in Step 1. We transform this problem to finding a perfect matching in a bipartite graph. We explain our solution for the first iteration.

Consider a graph $G = (V_1, V_2, E)$ where $V_1 = V_2 = \{1, 2, \ldots, m\}$. For each entry $N_{i,j}$, there is an edge $(i, N_{i,j})$ in E. Then picking a value in each row is equivalent to picking an edge for each vertex in V_1. The chosen values forming a permutation over $\{1, 2, \ldots, m\}$ is equivalent to the chosen edges forming a perfect matching in G. To prove the existence of a perfect matching, we show that the graph G is a k-regular bipartite graph and rely on the corollary (Corollary 1) of Hall's Marriage Theorem. For all vertex $i \in V_1$, there is an edge $(i, N_{i,j})$ in E for each entry in the i-th row of $\boldsymbol{N}$. Therefore, the degree of the vertex i is k. For all vertex $j \in V_2$, the degree of j equals to the number of entries in $\boldsymbol{N}$ which equal to j. Note that there are exactly k entries which equals to j. Thus, the degree of the vertex j is k. Therefore G is a k-regular graph. By Corollary 1, there exists a perfect matching in G. The same arguments work for other iterations. We refer the readers to Section 4.3 for more details.

It is worth noting that we use Hall's Marriage Theorem to solve two different problems:

- In Theorem 2, we use Hall's Marriage Theorem to find a different set of permutations $q_1, q_2, \ldots, q_m$ given the permutations $p_1, p_2, \ldots, p_m$ and limit the number of different permutations in $q_1, q_2, \ldots, q_m$.
- In Theorem 4, we use Hall's Marriage Theorem to find a permutation for each output sharing to achieve the non-collision property (Property 1).

2.4 Handling Fan-out Gates

We briefly discuss how to evaluate fan-out gates efficiently. We first model the problem as follows: given a degree-t packed Shamir sharing $[\boldsymbol{x}]$ along with a vector $(n_1, n_2, \ldots, n_k) \in \mathbb{N}^k$, where $n_i \geq 1$ is the number of times that x_i is

used in later layers, the goal is to compute $\frac{n_1+n_2+\ldots+n_k}{k}$ degree-t packed Shamir sharings which contain n_i copies of the value x_i for all $i \in \{1, 2, \ldots, k\}$. (For simplicity, we assume that $n_1 + n_2 + \ldots + n_k$ is a multiple of k. We refer the readers to Section 5.1 for how we handle the edge case.)

Our idea is to compute the output sharings one by one. For each output sharing $[\boldsymbol{y}]$, all values of $\boldsymbol{y}$ come from $\boldsymbol{x}$, which means that we may write $\boldsymbol{y}$ as a linear function of $\boldsymbol{x}$. Let F be a linear map such that $\boldsymbol{y} = F(\boldsymbol{x})$. To compute $[\boldsymbol{y}]$, we can prepare a pair of random sharings $([\boldsymbol{r}], [F(\boldsymbol{r})])$ and use the same method to compute $[\boldsymbol{y}]$ as that for permutations. Then we face the same problem that naively preparing the random sharings $([\boldsymbol{r}], [F(\boldsymbol{r})])$ suffer an overhead which depends on the number of different linear maps F. In the worst case where we need a different linear map for different output packed Shamir sharing, the overhead of preparing each pair of random sharings is as large as $O(n^2)$ elements, which eliminate the benefit of using the packed Shamir sharing scheme.

We follow the same idea as that for permutation to prepare the random sharings $([\boldsymbol{r}], [F(\boldsymbol{r})])$: Given m different linear maps $F_1, F_2, \ldots, F_m$, we will prepare random sharings for m other linear maps $G_1, G_2, \ldots, G_m$ and then recompose the components in the random sharings for $G_1, G_2, \ldots, G_m$ to obtain random sharings for $F_1, F_2, \ldots, F_m$. The main difficulty is that it is unclear how to define a component. Our solution includes the following additional steps:

- We require the secrets of the output packed Shamir sharings to be in a specific order.
- To compute each output packed Shamir sharing $[\boldsymbol{y}]$, we first permute the secrets of $[\boldsymbol{x}]$ based on $\boldsymbol{y}$.

These two steps allow us to properly define a component in a way that we can efficiently find $G_1, G_2, \ldots, G_m$ such that the above idea works. The description of the ideal functionality for fan-out gates is presented in Section 4.4. We refer the readers to the full version of this paper [GPS21] for more details about our protocol for fan-out gates.

2.5 Overview of Our Semi-honest Protocol

So far, we have introduced all the building blocks we need in our semi-honest protocol. To evaluate a single circuit:

1. All parties first transform the circuit to a good form in the sense that the number of gates of each type in each layer is a multiple of k. The transformation is done locally by running a deterministic algorithm. Unlike [DIK10], our transformation only increases the circuit size by a constant factor and an additive term $O(k \cdot \mathsf{Depth})$, where the latter term comes from the fact that the number of gates in each layer is a multiple of k after the transformation. The same term (or a larger term) also exists in [DIK10,GIP15]. We refer the readers to Section 5.1 for more details.
2. All parties preprocess the circuit to determine how the wire values should be packed. Also, all parties compute a permutation for each output sharing

for the non-collision property (see Property 1 in Section 2.3). This step is also done locally. We refer the readers to Section 5.2 for more details.
3. Finally, all parties evaluate the circuits using the protocols we described above. We refer the readers to Section 5.3 for more details.

Note that only the third step requires communication. We briefly analyze the communication complexity. For each group of k gates, all parties use the basic protocol to evaluate these gates. The communication complexity of the basic protocol is $O(n)$ elements. To prepare the input sharings for this group of k (addition, multiplication, or output) gates, we need to evaluate fan-out gates, perform permutations to achieve the non-collision property, use $\mathcal{F}_{\text{select}}$ to collect the secrets of the input sharings, and perform permutations again to obtain the correct orders. Since each operation requires $O(n)$ elements, the amortized communication complexity per gate is $O(n/k)$ elements.

2.6 Achieving Malicious Security

We briefly discuss how to compile our semi-honest protocol to a fully malicious one. Our main observation is that most of our semi-honest protocols have already achieved perfect privacy *against a fully malicious adversary*, namely the executions of these protocols do not leak any information to the adversary. Also, the deviation of a fully malicious adversary can be reduced to the following two kinds of attacks:

- An adversary can distribute an inconsistent degree-t packed Shamir sharing.
- An adversary can add additive errors to the secrets of the output sharing.

To achieve malicious security, our idea is to first run our semi-honest protocol before the output phase, check whether the above two kinds of attacks are launched by the adversary, and finally reconstruct the output.

To this end, for each semi-honest protocol, we first construct a functionality which allows the adversary to launch the above two kinds of attacks, and prove that our semi-honest protocol securely (with abort) computes the new functionality against a fully malicious adversary. Then we construct protocols to check whether the above two kinds of attacks are launched by the adversary. We view the computation as a composition of two parts: (1) evaluation of the basic gates, i.e., addition gates and multiplication gates, and (2) network routing, i.e., computing input sharings of each layer using the output sharings from previous layers.

- For the first part, since addition gates are computed without interaction, it is sufficient to only check the correctness of multiplications. We extend the recent sub-linear verification techniques [BBCG$^+$19,GSZ20] which are used in the honest majority setting (i.e., the corruption threshold $t' = t$) to our setting (i.e., the corruption threshold $t' = t - k + 1$).
- For the second part, it includes evaluating fan-out gates, performing permutations to achieve the non-collision property, using $\mathcal{F}_{\text{select}}$ to collect the

secrets of the input sharings, performing permutations again to obtain the correct orders. We note that the network routing does not change the secret values. Instead, its goal is to create new sharings which contain the secret values we want in the correct positions. Thus, it is sufficient to only focus on the front sharing before the network routing and the end sharing after the network routing, and check whether they have the same values.

Finally, when both checks pass, all parties reconstruct the output as the semi-honest protocol. We refer the readers to the full version of this paper [GPS21] for more details.

Remark 1. We note that the multiplication protocol is an exception in the sense that it cannot be reduced directly to the additive attacks we mention above. In fact, the work [GIP15] showed that a malicious attack can only be reduced to a linear attack, where the error in the output secret can depend on the input secrets. Our observation is that the linear attack is due to the inconsistency of the input sharings. If the input sharings are consistent, then the linear attack in [GIP15] degenerates to an additive attack to the final result. To model such a security property, we use a weaker functionality for the multiplication protocol, which does not guarantee the correctness of the multiplication result when the input sharings are inconsistent. The verification is done by first checking the consistency of all sharings. If the verification passes, then the attack of an adversary degenerates to additive attacks, which allows us to use the efficient verification protocol for multiplication gates in previous works. We refer the readers to the full version of this paper [GPS21] for more discussion.

3 Preliminaries

3.1 The Model

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 or 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 $n = 2t + 1$ 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. The communication complexity is measured by the number of bits via private channels.

We focus on functions that can be represented as arithmetic circuits over a finite field $\mathbb{F}$ with input, addition, multiplication, and output gates. We use κ to denote the security parameter, C to denote the circuit, and $|C|$ for the size of the circuit. We assume that the field size is $|\mathbb{F}| \geq 2n$.

Let $1 \leq k \leq t$ be an integer. An adversary $\mathcal{A}$ can corrupt at most c clients and $t' = t - k + 1$ 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 $t' = t - k + 1$ parties. We refer the readers to the full version of this paper [GPS21] for more details about the security definition and the benefit of the client-server model. In the following, we assume that there are exactly $t' = t - k + 1$ corrupted parties.

3.2 Packed Shamir Secret Sharing Scheme

In this work, we will 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 n be the number of parties and k be the number of secrets that are packed in one sharing. Let $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_k$ be $n + k$ distinct non-zero 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 $(w_1, \ldots, w_n)$ which satisfies that, there exists a polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most d such that $\forall i \in [k], f(\beta_i) = x_i$ and $\forall i \in [n], f(\alpha_i) = w_i$. 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 $\boldsymbol{x}$.

We will use $[\boldsymbol{x}]$ to denote a degree-t packed Shamir sharing of $\boldsymbol{x} \in \mathbb{F}^k$, and $\langle \boldsymbol{x} \rangle$ to denote a degree-$2t$ packed Shamir sharing. Recall that the number of corrupted parties is at most $t - k + 1$. Therefore, using degree-t packed Shamir sharings is sufficient to protect the privacy of the secrets. 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 $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x} + \boldsymbol{y}] = [\boldsymbol{x}] + [\boldsymbol{y}]$.
- Multiplication: Let $*$ denote coordinate-wise multiplication. For all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $\langle \boldsymbol{x} * \boldsymbol{y} \rangle = [\boldsymbol{x}] \cdot [\boldsymbol{y}]$.

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

For a constant vector $\boldsymbol{v} \in \mathbb{F}^k$ which is known by all parties, sometimes it is convenient to transform it to a degree-t packed Shamir sharing. This can be done by constructing a polynomial $f(\cdot) \in F[X]$ of degree $k - 1$ such that for all $i \in [k]$, $f(\beta_i) = v_i$. The i-th share of $[\boldsymbol{v}]$ is defined to be $f(\alpha_i)$ as usual.

3.3 Generating Random Sharings

In our work, we adopt the notion of an abstract definition of a general linear secret sharing scheme (GLSSS) in [CCXY18]. We will make use of a functionality $\mathcal{F}_{\text{rand}}$ introduced in [PS21], which allows all parties to prepare a random sharing for a given $\mathbb{F}$-linear secret sharing scheme.

In [PS21], Polychroniadou and Song proposed an instantiation of $\mathcal{F}_{\text{rand}}$ which is secure against an adversary that corrupts t parties. We note that their protocol can be extended to any corruption threshold (with different communication complexity). In particular, in our setting where $t' = t - k + 1$, if the share size of the given $\mathbb{F}$-linear secret sharing scheme is sh field elements in $\mathbb{F}$, the communication complexity of generating N random sharings is $O(N \cdot n \cdot \mathsf{sh} + n^3 \cdot \kappa)$ elements in $\mathbb{F}$. We refer the readers to the full version of this paper [GPS21] for more details.

3.4 Permutation Matrix, Bipartite Graph and Hall's Marriage Theorem

Definition 1 (Permutation Matrix). *Let $k \geq 1$ be an integer. A matrix $\boldsymbol{M} \in \{0,1\}^{k \times k}$ is a permutation matrix if for each row and each column, there is exactly one entry which is* 1*.*

For a permutation $p(\cdot)$ over $\{1, 2, \ldots, k\}$, let $\boldsymbol{M}_p$ be a permutation matrix such that for all $i, j \in \{1, \ldots, k\}$, $(\boldsymbol{M}_p)_{i,j} = 1$ iff $p(i) = j$. Note that for each permutation matrix $\boldsymbol{M}'$, there exists a permutation $p(\cdot)$ such that $\boldsymbol{M}' = \boldsymbol{M}_p$.

Definition 2 (Balanced Matrix). *Let $k \geq 1$ be an integer. A matrix $\boldsymbol{M} \in \mathbb{N}^{k \times k}$ is a balanced matrix if for each row and each column, the summation of all the entries is the same.*

Note that for all permutations $p(\cdot)$ over $\{1, 2, \ldots, k\}$, the permutation matrix $\boldsymbol{M}_p$ is a balanced matrix since the summation of the entries in each row and each column is 1.

Definition 3 (Bipartite Graph). *A graph $G = (V, E)$ is a bipartite graph if there exists a partition (V_1, V_2) of V such that for all edge $(v_i, v_j) \in E$, $v_i \in V_1$ and $v_j \in V_2$.*

In the following, we will use (V_1, V_2, E) to denote a bipartite graph. We say a bipartite graph (V_1, V_2, E) is d-regular if the degree of each vertex in $V_1 \bigcup V_2$ is d.

Definition 4 (Perfect Matching). *For a bipartite graph (V_1, V_2, E) such that $|V_1| = |V_2|$, a perfect matching is a subset of edges $\mathcal{E} \in E$ which satisfies that each vertex in the sub-graph $(V_1, V_2, \mathcal{E})$ has degree* 1*.*

Theorem 3 (Hall's Marriage Theorem). *For a bipartite graph (V_1, V_2, E) such that $|V_1| = |V_2|$, there exists a perfect matching iff for all subset $V_1' \subset V_1$, the number of the neighbors of vertices in V_2 is at least $|V_1'|$.*

In this work, we will make use of the following two well-known corollaries of Hall's Marriage Theorem. For completeness, we also provide proofs for the corollaries in the full version of this paper [GPS21].

Corollary 1. *There exists a perfect matching in a d-regular bipartite graph.*

Corollary 2. *Let $k \geq 1$ be an integer. For all non-zero balanced matrix $\boldsymbol{N} \in \mathbb{N}^{k \times k}$, there exists a permutation matrix $\boldsymbol{M}$ such that for all $i, j \in \{1, 2, \ldots, k\}$, $\boldsymbol{N}_{i,j} \geq \boldsymbol{M}_{i,j}$.*

4 Circuit Evaluation - Against a Semi-honest Adversary

In this section, we discuss how to evaluate a general circuit by using the packed Shamir sharing scheme. For simplicity, we assume the adversary is semi-honest. Recall that we are in the client-server model where there are c clients and $n = 2t + 1$ parties (servers). Recall that $1 \leq k \leq t$ is an integer. An adversary is allowed to corrupt $t' = t - k + 1$ parties. We will use the degree-t packed Shamir sharing scheme, which can store k secrets within one sharing. Recall that $\mathcal{C}$ denotes the set of corrupted parties and $\mathcal{H}$ denotes the set of honest parties.

4.1 Basic Protocols for Input Gates, Addition Gates, Multiplication Gates, and Output Gates

We distinguish input gates and output gates belonging to different clients. For each client, we assume the number of input gates belonging to this client and the number of output gates belonging to this client are multiples of k. For each layer, we assume that the number of addition gates and the number of multiplication gates are multiples of k. In Section 5, we will show how to compile a general circuit to meet this requirement.

Evaluating Input Gates and Output Gates. The functionalities $\mathcal{F}_{\text{input-semi}}$ and $\mathcal{F}_{\text{output-semi}}$ are described in Functionality 1 and Functionality 2 respectively. We refer the readers to the full version of this paper [GPS21] for the protocols that realize $\mathcal{F}_{\text{input-semi}}$ and $\mathcal{F}_{\text{output-semi}}$. The communication complexity of each protocol is $O(n)$ field elements.

Functionality 1: $\mathcal{F}_{\text{input-semi}}$

1. Suppose $\boldsymbol{x} \in \mathbb{F}^k$ is the input associated with the input gate which belongs to the Client. $\mathcal{F}_{\text{input-semi}}$ receives the input $\boldsymbol{x}$ from the Client.
2. $\mathcal{F}_{\text{input-semi}}$ receives from the adversary a set of shares $\{s_i\}_{i \in \mathcal{C}}$. $\mathcal{F}_{\text{input-semi}}$ samples a random degree-t packed Shamir sharing $[\boldsymbol{x}]$ such that for all $P_i \in \mathcal{C}$, the i-th share of $[\boldsymbol{x}]$ is s_i.
3. $\mathcal{F}_{\text{input-semi}}$ distributes the shares of $[\boldsymbol{x}]$ to honest parties.

Functionality 2: $\mathcal{F}_{\text{output-semi}}$

1. Suppose $[\boldsymbol{x}]$ is the sharing associated with the output gate which belongs to the Client. $\mathcal{F}_{\text{output-semi}}$ receives the shares of $[\boldsymbol{x}]$ from honest parties.
2. $\mathcal{F}_{\text{output-semi}}$ recovers the whole sharing $[\boldsymbol{x}]$, and sends the shares of corrupted parties to the adversary.
3. $\mathcal{F}_{\text{output-semi}}$ reconstructs $\boldsymbol{x}$ and sends it to the Client.

Evaluating Addition Gates and Multiplication Gates. In the following, we use $[\boldsymbol{x}], [\boldsymbol{y}]$ to denote the input degree-t packed Shamir sharings.

For an addition gate, all parties want to compute $[\boldsymbol{x}+\boldsymbol{y}]$. Note that this can be done by computing $[\boldsymbol{x}+\boldsymbol{y}] := [\boldsymbol{x}]+[\boldsymbol{y}]$, i.e., each party locally adds its shares of $[\boldsymbol{x}], [\boldsymbol{y}]$. The correctness follows from the property that packed Shamir sharing is linearly homomorphic.

Recall that $*$ denotes the coordinate-wise multiplication. For a multiplication gate, all parties want to compute a degree-t packed Shamir sharing of $\boldsymbol{z} := \boldsymbol{x} * \boldsymbol{y}$. We summarize the functionality $\mathcal{F}_{\text{mult-semi}}$ in Functionality 3.

Functionality 3: $\mathcal{F}_{\text{mult-semi}}$

1. Suppose $[\boldsymbol{x}], [\boldsymbol{y}]$ are the input degree-t packed Shamir sharings. $\mathcal{F}_{\text{mult-semi}}$ receives the shares of $[\boldsymbol{x}], [\boldsymbol{y}]$ from honest parties.
2. $\mathcal{F}_{\text{mult-semi}}$ recovers the whole sharings $[\boldsymbol{x}], [\boldsymbol{y}]$ and reconstructs the secrets $\boldsymbol{x}, \boldsymbol{y}$. $\mathcal{F}_{\text{mult-semi}}$ computes $\boldsymbol{z} := \boldsymbol{x} * \boldsymbol{y}$.
3. $\mathcal{F}_{\text{mult-semi}}$ receives from the adversary a set of shares $\{s_i\}_{i\in\mathcal{C}}$. $\mathcal{F}_{\text{mult-semi}}$ samples a random degree-t packed Shamir sharing $[\boldsymbol{z}]$ such that for all $P_i \in \mathcal{C}$, the i-th share of $[\boldsymbol{z}]$ is s_i.
4. $\mathcal{F}_{\text{mult-semi}}$ distributes the shares of $[\boldsymbol{z}]$ to honest parties.

A multiplication gate can be evaluated by a natural extension of the DN multiplication protocol in [DN07]. The main observation is that all parties can locally compute a degree-$2t$ packed Shamir sharing $\langle \boldsymbol{z} \rangle = \langle \boldsymbol{x} * \boldsymbol{y} \rangle = [\boldsymbol{x}] \cdot [\boldsymbol{y}]$. The only task is to reduce the degree of $\langle \boldsymbol{z} \rangle$. Following the approach in [DN07], this can be achieved by preparing a pair of two random sharings $([\boldsymbol{r}], \langle \boldsymbol{r} \rangle)$ of the same secrets $\boldsymbol{r}$. We refer the readers to the full version of this paper [GPS21] for the protocol that realizes $\mathcal{F}_{\text{mult-semi}}$. The communication complexity of m invocations of $\mathcal{F}_{\text{mult-semi}}$ is $O(m \cdot n + n^3 \cdot \kappa)$ field elements.

4.2 Performing an Arbitrary Permutation on the Secrets of a Single Sharing

During the computation, we may encounter the scenario that the order of the secrets is not what we want (see more discussion in Section 2.2). To solve it, we need a functionality which allows us to perform an arbitrary permutation on the secrets of a single sharing. Let $p(\cdot)$ be a permutation over $\{1, 2, \ldots, k\}$. Recall that each permutation $p(\cdot)$ maps to a permutation matrix $\boldsymbol{M}_p \in \{0,1\}^{k\times k}$ where $(\boldsymbol{M}_p)_{i,j} = 1$ iff $p(i) = j$. To permute a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$ to $\tilde{\boldsymbol{x}} = (x_{p(1)}, x_{p(2)}, \ldots, x_{p(k)})$, it is equivalent to computing $\tilde{\boldsymbol{x}} = \boldsymbol{M}_p \cdot \boldsymbol{x}$. We model the functionality $\mathcal{F}_{\text{permute-semi}}$ in Functionality 4.

Functionality 4: $\mathcal{F}_{\text{permute-semi}}$

1. $\mathcal{F}_{\text{permute-semi}}$ receives a permutation p and the shares of a degree-t packed Shamir sharing $[\boldsymbol{x}]$ from honest parties.
2. $\mathcal{F}_{\text{permute-semi}}$ reconstructs the secrets $\boldsymbol{x}$ from the shares of honest parties, and computes $\tilde{\boldsymbol{x}} = \boldsymbol{M}_p \cdot \boldsymbol{x}$.
3. $\mathcal{F}_{\text{permute-semi}}$ receives from the adversary a set of shares $\{s_i\}_{i\in\mathcal{C}}$. $\mathcal{F}_{\text{permute-semi}}$ samples a random degree-t packed Shamir sharing $[\tilde{\boldsymbol{x}}]$ such that for all $P_i \in \mathcal{C}$, the i-th share of $[\tilde{\boldsymbol{x}}]$ is s_i.
4. $\mathcal{F}_{\text{permute-semi}}$ distributes the shares of $[\tilde{\boldsymbol{x}}]$ to honest parties.

We follow the techniques in [DIK10] to realize $\mathcal{F}_{\text{permute-semi}}$ by making use of a pair of random degree-t packed Shamir sharings $([\boldsymbol{r}], [\boldsymbol{M}_p \cdot \boldsymbol{r}])$. We refer the readers to Section 2.2 for an overview of these techniques. As we discussed in Section 2.2, the main issue of this approach is how to how to *efficiently* prepare a pair of random sharings $([\boldsymbol{r}], [\tilde{\boldsymbol{r}}])$. To generate random sharings for m permutations, our idea is to first generate random sharings for a limited number ($O(n^2)$) of different permutations which are related to the input permutations, and then transform them to the random sharings for the desired permutations (the input permutations).

Before moving forward, we first introduce a useful functionality $\mathcal{F}_{\text{select}}$, which selects secrets from one or more packed Shamir sharings and outputs a single sharing which contains the chosen secrets. Later on, we will use $\mathcal{F}_{\text{select}}$ to solve the above issue of preparing random sharings for permutations.

Selecting Secrets from One or More Packed Shamir Sharings. Concretely, we want to realize the functionality $\mathcal{F}_{\text{select-semi}}$, which takes as input k degree-t packed Shamir sharings $[\boldsymbol{x}^{(1)}], [\boldsymbol{x}^{(2)}], \ldots, [\boldsymbol{x}^{(k)}]$ (which do not need to be distinct) and a permutation $p(\cdot)$ over $\{1, 2, \ldots, k\}$, and outputs a degree-t packed Shamir sharing $[\boldsymbol{y}]$ such that for all $i \in [k]$, $y_{p(i)} = x^{(i)}_{p(i)}$, where $x^{(i)}_j$ is the j-th value

of $\boldsymbol{x}^{(i)}$. Effectively, $\mathcal{F}_{\text{select-semi}}$ chooses the $p(1)$-th secret of $[\boldsymbol{x}^{(1)}]$, the $p(2)$-th secret of $[\boldsymbol{x}^{(2)}]$, ..., the $p(k)$-th secret of $[\boldsymbol{x}^{(k)}]$ and generates a new degree-t packed Shamir sharing $[\boldsymbol{y}]$ which contains the chosen secrets. Note that the positions of the chosen secrets remain the same. Therefore, we require p to be a permutation so that the chosen secrets come from different positions. The description of $\mathcal{F}_{\text{select-semi}}$ appears in Functionality 5.

Functionality 5: $\mathcal{F}_{\text{select-semi}}$

1. $\mathcal{F}_{\text{select-semi}}$ receives from honest parties their shares of k degree-t packed Shamir sharings $[\boldsymbol{x}^{(1)}], [\boldsymbol{x}^{(2)}], \ldots, [\boldsymbol{x}^{(k)}]$. $\mathcal{F}_{\text{select-semi}}$ also receives a permutation p from honest parties.
2. $\mathcal{F}_{\text{select-semi}}$ reconstructs $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(k)}$. Then $\mathcal{F}_{\text{select-semi}}$ sets $\boldsymbol{y} = (y_1, y_2, \ldots, y_k)$ such that for all $i \in [k]$, $y_{p(i)} = x^{(i)}_{p(i)}$, where $x^{(i)}_j$ is the j-th value of $\boldsymbol{x}^{(i)}$.
3. $\mathcal{F}_{\text{select-semi}}$ receives from the adversary a set of shares $\{s_i\}_{i \in \mathcal{C}}$. $\mathcal{F}_{\text{select-semi}}$ samples a random degree-t packed Shamir sharing $[\boldsymbol{y}]$ such that for all $P_i \in \mathcal{C}$, the i-th share of $[\boldsymbol{y}]$ is s_i.
4. $\mathcal{F}_{\text{select-semi}}$ distributes the shares of $[\boldsymbol{y}]$ to honest parties.

For all $i \in [k]$, let $\boldsymbol{e}_i \in \{0,1\}^k$ denote the vector where the i-th entry is 1 and for all $j \neq i$, the j-th entry is 0. Recall that in Section 3.2 we show how to transform a constant vector to a degree-t packed Shamir sharing. Let $[\boldsymbol{e}_i]$ denote the degree-t packed Shamir sharing of $\boldsymbol{e}_i$.

To realize $\mathcal{F}_{\text{select-semi}}$, note that $[\boldsymbol{e}_{p(i)}] \cdot [\boldsymbol{x}^{(i)}]$ is a degree-$2t$ packed Shamir sharing of $\boldsymbol{e}_{p(i)} * \boldsymbol{x}^{(i)}$. Also note that $\boldsymbol{y} = \sum_{i=1}^{k} \boldsymbol{e}_{p(i)} * \boldsymbol{x}^{(i)}$. Therefore, all parties can locally compute $\langle \boldsymbol{y} \rangle = \sum_{i=1}^{k} [\boldsymbol{e}_{p(i)}] \cdot [\boldsymbol{x}^{(i)}]$. And the only task is to reduce the degree of $\langle \boldsymbol{y} \rangle$. Note that this can be achieved by the same technique as Mult. The description of the protocol Select appears in Protocol 6. The communication complexity of m invocations of Select is $O(m \cdot n + n^3 \cdot \kappa)$ field elements.

Lemma 1. *Protocol* Select *securely computes $\mathcal{F}_{\text{select-semi}}$ in the $\mathcal{F}_{\text{rand}}$-hybrid model against a semi-honest adversary who controls $t' = t - k + 1$ parties.*

We refer the readers to the full version of this paper [GPS21] for more discussion about Lemma 1.

Using $\mathcal{F}_{\text{select-semi}}$ to Generate Random Sharings for Permuting Secrets. For all $i, j \in \{1, 2, \ldots, k\}$, we say a pair of degree-t packed Shamir sharings $([\boldsymbol{x}], [\boldsymbol{y}])$ contains an (i, j)-component if the secrets of these two sharings satisfy that $x_i = y_j$.

Protocol 6: SELECT

1. Let $[\boldsymbol{x}^{(1)}], [\boldsymbol{x}^{(2)}], \ldots, [\boldsymbol{x}^{(k)}]$ denote the k input packed Shamir sharings and $p(\cdot)$ denote the permutation. The goal is to generate a degree-t packed Shamir sharing $[\boldsymbol{y}]$ such that for all $i \in [k]$, $y_{p(i)} = x^{(i)}_{p(i)}$. Recall that $\boldsymbol{e}_i \in \{0,1\}^k$ denote the vector where the i-th entry is 1 and for all $j \neq i$, the j-th entry is 0. For all $i \in [k]$, all parties agree on the whole sharing $[\boldsymbol{e}_i]$ based on the transformation in Section 3.2.
2. All parties invoke $\mathcal{F}_{\text{rand}}$ to prepare a pair of random sharings $([\boldsymbol{r}], \langle \boldsymbol{r} \rangle)$.
3. All parties locally compute $\langle \boldsymbol{e} \rangle := \sum_{i=1}^{k} [\boldsymbol{e}_{p(i)}] \cdot [\boldsymbol{x}^{(i)}] + \langle \boldsymbol{r} \rangle$.
4. All parties send their shares of $\langle \boldsymbol{e} \rangle$ to the first party P_1.
5. P_1 reconstructs the secrets $\boldsymbol{e}$, generates a random degree-t packed Shamir sharing $[\boldsymbol{e}]$, and distributes the shares to other parties.
6. All parties locally compute $[\boldsymbol{y}] := [\boldsymbol{e}] - [\boldsymbol{r}]$.

To perform a permutation $p(\cdot)$, we need to prepare two random degree-t packed Shamir sharings $([\boldsymbol{r}], [\boldsymbol{M}_p \cdot \boldsymbol{r}])$. We can view $([\boldsymbol{r}], [\boldsymbol{M}_p \cdot \boldsymbol{r}])$ as a composition of a $(1, p(1))$-component, a $(2, p(2))$-component, ..., and a $(k, p(k))$-component.

Let m denote the number of permutations we want to prepare random sharings for. These permutations are denoted by $p_1(\cdot), p_2(\cdot), \ldots, p_m(\cdot)$. Our idea is as follows:

1. We first find m permutations $q_1(\cdot), q_2(\cdot), \ldots, q_m(\cdot)$ such that:
   - For all $i, j \in \{1, 2, \ldots, k\}$, the number of permutations $p \in \{p_1, p_2, \ldots, p_m\}$ which satisfies that $p(i) = j$ is equal to the number of permutations $q \in \{q_1, q_2, \ldots, q_m\}$ which satisfies that $q(i) = j$.
2. All parties prepare random sharings for permutations $q_1, q_2, \ldots, q_m$.
3. From $i = 1$ to m, a pair of random sharings for the permutation p_i is prepared by using $\mathcal{F}_{\text{select-semi}}$ to choose the first unused $(j, p_i(j))$-component from the random sharings for $q_1, q_2, \ldots, q_m$ for all $j \in [k]$.

We refer the readers to Section 2.2 for a more detailed explanation.

The major benefit of this approach is that *we can limit the number of different permutations in* $\{q_1, q_2, \ldots, q_m\}$ as we show below. More concretely, we will prove the following theorem:

Theorem 2. *Let $m, k \geq 1$ be integers. For all m permutations $p_1, p_2, \ldots, p_m$ over $\{1, 2, \ldots, k\}$, there exists m permutations $q_1, q_2, \ldots, q_m$ over $\{1, 2, \ldots, k\}$ such that:*

- *For all $i, j \in \{1, 2, \ldots, k\}$, the number of permutations $p \in \{p_1, p_2, \ldots, p_m\}$ such that $p(i) = j$ is the same as the number of permutations $q \in \{q_1, q_2, \ldots, q_m\}$ such that $q(i) = j$.*
- *$q_1, q_2, \ldots, q_m$ contain at most k^2 different permutations.*

Moreover, $q_1, q_2, \ldots, q_m$ can be found within polynomial time given $p_1, p_2, \ldots, p_m$.

The proof of Theorem 2 can be found in the full version of this paper [GPS21].

Preparing Random Sharings for Different Permutations. We are ready to introduce the functionality and its implementation for preparing random sharings for different permutations. The functionality $\mathcal{F}_{\text{rand-perm-semi}}$ appears in Functionality 7.

Functionality 7: $\mathcal{F}_{\text{rand-perm-semi}}$

1. $\mathcal{F}_{\text{rand-perm-semi}}$ receives from honest parties m permutations $p_1, p_2, \ldots, p_m$ over $\{1, 2, \ldots, k\}$.
2. For all $i \in [m]$, $\mathcal{F}_{\text{rand-perm-semi}}$ receives from the adversary a set of shares $\{(u_j^{(i)}, v_j^{(i)})\}_{j \in \mathcal{C}}$. $\mathcal{F}_{\text{rand-perm-semi}}$ samples a random vector $\boldsymbol{r}^{(i)} \in \mathbb{F}^k$ and samples two degree-t packed Shamir sharings $([\boldsymbol{r}^{(i)}], [\boldsymbol{M}_{p_i} \cdot \boldsymbol{r}^{(i)}])$ such that for all $P_j \in \mathcal{C}$, the j-th share of $([\boldsymbol{r}^{(i)}], [\boldsymbol{M}_{p_i} \cdot \boldsymbol{r}^{(i)}])$ is $(u_j^{(i)}, v_j^{(i)})$.
3. For all $i \in [m]$, $\mathcal{F}_{\text{rand-perm-semi}}$ distributes the shares of $([\boldsymbol{r}^{(i)}], [\boldsymbol{M}_{p_i} \cdot \boldsymbol{r}^{(i)}])$ to honest parties.

For a fixed permutation $p(\cdot)$ over $\{1, 2, \ldots, k\}$, we show how to use $\mathcal{F}_{\text{rand}}$ to prepare a pair of random sharings $([\boldsymbol{r}], [\boldsymbol{M}_p \cdot \boldsymbol{r}])$ in the full version of this paper [GPS21]. The communication complexity of preparing m pairs of random sharings in the form of $([\boldsymbol{r}], [\boldsymbol{M}_p \cdot \boldsymbol{r}])$ for a fixed permutation $p(\cdot)$ is $O(m \cdot n + n^3 \cdot \kappa)$ elements in $\mathbb{F}$. We describe the protocol for $\mathcal{F}_{\text{rand-perm-semi}}$ in Protocol 8. The communication complexity of using Rand-Perm to prepare random sharings for m permutations is $O(m \cdot n + n^5 \cdot \kappa)$ field elements.

Lemma 2. *Protocol* Rand-Perm *securely computes* $\mathcal{F}_{\textit{rand-perm-semi}}$ *in the* ($\mathcal{F}_{\textit{rand}}$, $\mathcal{F}_{\textit{select-semi}}$)*-hybrid model against a semi-honest adversary who controls* $t' = t - k + 1$ *parties.*

The proof of Lemma 2 can be found in the full version of this paper [GPS21].

Realizing $\mathcal{F}_{\textit{permute-semi}}$. Now we are ready to present the protocol for $\mathcal{F}_{\text{permute-semi}}$. The protocol Permute uses $\mathcal{F}_{\text{rand-perm-semi}}$ to prepare the random sharings for the permutation we want to perform and then follows the techniques in [DIK10]. In Permute, we will prepare a random degree-$2t$ packed Shamir sharing of $\mathbf{0} \in \mathbb{F}^k$, which is used as a random mask for the shares of honest parties (see the proof of Lemma 3). This is not needed for semi-honest security but will be helpful when we consider a fully malicious adversary at a later point.

We show how to use $\mathcal{F}_{\text{rand}}$ to prepare a random degree-$2t$ packed Shamir sharing of $\mathbf{0} \in \mathbb{F}^k$ in the full version of this paper [GPS21]. The communication complexity of preparing m random degree-$2t$ packed Shamir sharings of $\mathbf{0}$ is $O(m \cdot n + n^3 \cdot \kappa)$ elements in $\mathbb{F}$. The description of Permute appears in Protocol 9. The communication complexity of m invocations of Permute is $O(m \cdot n + n^5 \cdot \kappa)$ field elements.

Protocol 8: RAND-PERM

1. Let $p_1, p_2, \ldots, p_m$ be the permutations over $\{1, 2, \ldots, k\}$ that all parties want to prepare random sharings for.
2. All parties use a deterministic algorithm that all parties agree on to compute m permutations $q_1, q_2, \ldots, q_m$ such that
   - For all $i, j \in \{1, 2, \ldots, k\}$, the number of permutations $p \in \{p_1, p_2, \ldots, p_m\}$ such that $p(i) = j$ is the same as the number of permutations $q \in \{q_1, q_2, \ldots, q_m\}$ such that $q(i) = j$.
   - $q_1, q_2, \ldots, q_m$ contain at most k^2 different permutations.

   The existence of such an algorithm is guaranteed by Theorem 2.
3. Suppose $q'_1, q'_2, \ldots, q'_{k^2}$ denote the different permutations in $q_1, q_2, \ldots, q_m$. For all $i \in \{1, 2, \ldots, k^2\}$, let n'_i denote the number of times that q'_i appears in $q_1, q_2, \ldots, q_m$. All parties invoke $\mathcal{F}_{\text{rand}}$ to prepare n'_i pairs of random sharings in the form $([\boldsymbol{r}], [\boldsymbol{M}_{q'_i} \cdot \boldsymbol{r}])$ for all $i \in \{1, 2, \ldots, k^2\}$. Note that we have prepared a pair of random sharings for each permutation q_i for all $i \in [m]$. Let $([\boldsymbol{r}^{(i)}], [\boldsymbol{M}_{q_i} \cdot \boldsymbol{r}^{(i)}])$ denote the random sharings for the permutation q_i.
4. For all $i, j \in \{1, 2, \ldots, k\}$, all parties initiate an empty list $L_{i,j}$. From $\ell = 1$ to m, for all $i, j \in \{1, 2, \ldots, k\}$, if $([\boldsymbol{r}^{(\ell)}], [\boldsymbol{M}_{q_\ell} \cdot \boldsymbol{r}^{(\ell)}])$ contains an (i, j)-component, all parties insert $([\boldsymbol{r}^{(\ell)}], [\boldsymbol{M}_{q_\ell} \cdot \boldsymbol{r}^{(\ell)}])$ into the list $L_{i,j}$.
5. From $\ell = 1$ to m, all parties prepare a pair of random sharings for p_ℓ as follows:
   - From $i = 1$ to k, let $([\boldsymbol{r}^{(\ell_i)}], [\boldsymbol{M}_{q_{\ell_i}} \cdot \boldsymbol{r}^{(\ell_i)}])$ denote the first pair of sharings in the list $L_{i,p_\ell(i)}$, and then remove it from $L_{i,p_\ell(i)}$. Note that $([\boldsymbol{r}^{(\ell_i)}], [\boldsymbol{M}_{q_{\ell_i}} \cdot \boldsymbol{r}^{(\ell_i)}])$ contains an $(i, p_\ell(i))$-component, which is not used when preparing random sharings for $p_1, p_2, \ldots, p_{\ell-1}$.
   - Let I denote the identity permutation over $\{1, 2, \ldots, k\}$.
     - All parties invoke $\mathcal{F}_{\text{select-semi}}$ with
     $$[\boldsymbol{r}^{(\ell_1)}], [\boldsymbol{r}^{(\ell_2)}], \ldots, [\boldsymbol{r}^{(\ell_k)}]$$
     and the permutation I. The output is denoted by $[\boldsymbol{v}^{(\ell)}]$.
     - All parties invoke $\mathcal{F}_{\text{select-semi}}$ with
     $$[\boldsymbol{M}_{q_{\ell_1}} \cdot \boldsymbol{r}^{(\ell_1)}], [\boldsymbol{M}_{q_{\ell_2}} \cdot \boldsymbol{r}^{(\ell_2)}], \ldots, [\boldsymbol{M}_{q_{\ell_k}} \cdot \boldsymbol{r}^{(\ell_k)}]$$
     and the permutation p_ℓ. The output is denoted by $[\tilde{\boldsymbol{v}}^{(\ell)}]$. Note that for all $i \in [k]$, $v_i^{(\ell)} = r_i^{(\ell_i)} = (\boldsymbol{M}_{q_{\ell_i}} \cdot \boldsymbol{r}^{(\ell_i)})_{q_{\ell_i}(i)} = (\boldsymbol{M}_{q_{\ell_i}} \cdot \boldsymbol{r}^{(\ell_i)})_{p_\ell(i)} = \tilde{v}^{(\ell)}_{p_\ell(i)}$.
6. All parties take $([\boldsymbol{v}^{(1)}], [\tilde{\boldsymbol{v}}^{(1)}]), ([\boldsymbol{v}^{(2)}], [\tilde{\boldsymbol{v}}^{(2)}]), \ldots, ([\boldsymbol{v}^{(m)}], [\tilde{\boldsymbol{v}}^{(m)}])$ as output.

Lemma 3. *Protocol* PERMUTE *securely computes* $\mathcal{F}_{\text{permute-semi}}$ *in the* ($\mathcal{F}_{\text{rand}}$, $\mathcal{F}_{\text{rand-perm-semi}}$)*-hybrid model against a semi-honest adversary who controls* $t' = t - k + 1$ *parties.*

The proof of Lemma 3 can be found in the full version of this paper [GPS21].

Protocol 9: PERMUTE

1. Let $[\boldsymbol{x}]$ denote the input degree-t packed Shamir sharing and $p(\cdot)$ denote the permutation all parties want to perform on $\boldsymbol{x}$.
2. All parties invoke $\mathcal{F}_{\text{rand-perm-semi}}$ with p to prepare a pair of random sharings $([\boldsymbol{r}], [\boldsymbol{M}_p \cdot \boldsymbol{r}])$. All parties invoke $\mathcal{F}_{\text{rand}}$ to prepare a random degree-$2t$ packed Shamir sharing $\langle \boldsymbol{0} \rangle$.
3. All parties locally compute $\langle \boldsymbol{e} \rangle := [\boldsymbol{x}] + [\boldsymbol{r}] + \langle \boldsymbol{0} \rangle$.
4. All parties send their shares of $\langle \boldsymbol{e} \rangle$ to the first party P_1.
5. P_1 reconstructs the secrets $\boldsymbol{e}$, and computes $\tilde{\boldsymbol{e}} = \boldsymbol{M}_p \cdot \boldsymbol{e}$. Then P_1 generates a random degree-t packed Shamir sharing $[\tilde{\boldsymbol{e}}]$, and distributes the shares to other parties.
6. All parties locally compute $[\tilde{\boldsymbol{x}}] := [\tilde{\boldsymbol{e}}] - [\boldsymbol{M}_p \cdot \boldsymbol{r}]$.

4.3 Obtaining Input Sharings for Multiplication Gates and Addition Gates

So far, we have introduced how to evaluate multiplication gates and addition gates using the packed Shamir sharing scheme. In the case that the secrets of an input sharing are not in the correct order, we have shown how to efficiently perform a permutation to obtain the correct order. During the computation, however, input sharings of multiplication gates and addition gates do not come for free. When evaluating the multiplication gates and addition gates in some layer, the secrets we want to be in a single sharing may be scattered in one or more output sharings from the previous layers. This requires us to collect the secrets from those sharings and generate a single sharing for these secrets efficiently. Our idea is to achieve the *non-collision* property:

Property 1 (Non-collision). For each input sharing of each layer, the secrets of this input sharing come from different positions in the output sharings of previous layers.

As we discussed in Section 2.3, with this property, we can use $\mathcal{F}_{\text{select-semi}}$ to choose the secrets and generates the input sharing we want. To avoid the case that we need the same secret twice in a single input sharing, which makes the non-collision property impossible to achieve, we further require that

- every output wire of the input layer and all intermediate layers is used exactly once as an input wire of a later layer (which may not be the next layer).

Note that this requirement can be met without loss of generality by assuming that there is a fan-out gate right after each (input, addition, or multiplication) gate that copies the output wire the number of times it is used in later layers. To achieve the non-collision property, our idea is to perform a permutation on each output sharing.

In the following, when we use the term "output sharings", we refer to the output sharings from the input layer and all intermediate layers. When we use the term "input sharings", we refer to the input sharings of the output layer and all intermediate layers. We further assume that the number of the input wires and the number of the output wires of each layer are multiples of k, where recall that k is the number of secrets we can store in a single packed Shamir sharing. In Section 5, we will show how to compile a general circuit to meet this requirement.

Let m denote the number of output sharings in the circuit. Then the number of input sharings is also m. We will label all the output sharings by $1, 2, \ldots, m$ and all the input sharings also by $1, 2, \ldots, m$. Consider a matrix $\boldsymbol{N} \in \{1, 2, \ldots, m\}^{m \times k}$ where $\boldsymbol{N}_{i,j}$ is the index of the input sharing that the j-th secret of the i-th output sharing wants to go to. Then for all $\ell \in \{1, 2, \ldots, m\}$, there are exactly k entries of $\boldsymbol{N}$ which are equal to ℓ. And the secrets at those positions are the secrets we want to collect for the ℓ-th input sharing. We will prove the following theorem.

Theorem 4. *Let $m \geq 1, k \geq 1$ be integers. Let $\boldsymbol{N}$ be a matrix of dimension $m \times k$ in $\{1, 2, \ldots, m\}^{m \times k}$ such that for all $\ell \in \{1, 2, \ldots, m\}$, the number of entries of $\boldsymbol{N}$ which are equal to ℓ is k. Then, there exists m permutations $p_1, p_2, \ldots, p_m$ over $\{1, 2, \ldots, k\}$ such that after performing the permutation p_i on the i-th row of $\boldsymbol{N}$, the new matrix $\boldsymbol{N}'$ satisfies that each column of $\boldsymbol{N}'$ is a permutation over $(1, 2, \ldots, m)$. Furthermore, the permutations $p_1, p_2, \ldots, p_m$ can be found within polynomial time.*

The proof of Theorem 4 can be found in the full version of this paper [GPS21].

When we apply p_i to the i-th output sharing for all $i \in \{1, 2, \ldots, m\}$, Theorem 4 guarantees that for all $j \in \{1, 2, \ldots, k\}$ the j-th secrets of all output sharings need to go to different input sharings. Note that this ensures the non-collision property. During the computation, we will perform the permutation p_i on the i-th output sharing right after it is computed. Note that when preparing an input sharing, the secrets we need only come from the output sharings which have been computed. The secrets of these output sharings have been properly permuted such that the secrets we want are in different positions. Therefore, we can use $\mathcal{F}_{\text{select-semi}}$ to choose these secrets and obtain the desired input sharing.

4.4 Handling Fan-out Gates

In the last subsection, we discussed how to prepare the input sharings for multiplication gates and addition gates. Our solution requires that

- every output wire of the input layer and all intermediate layers is used exactly once as an input wire of a later layer (which may not be the next layer).

This requirement can be met by inserting fan-out gates in each layer, which copy each output wire the number of times it is used in later layers. Specifically, we consider a functionality $\mathcal{F}_{\text{fan-out-semi}}$ which takes as input a degree-t packed Shamir sharing of $\boldsymbol{x} = (x_1, x_2, \ldots, x_k) \in \mathbb{F}^k$ along with a vector

$(n_1, n_2, \ldots, n_k) \in \mathbb{N}^k$, where $n_i \geq 1$ is the number of times that x_i is used in later layers, and outputs $\frac{n_1+n_2+\ldots+n_k}{k}$ degree-t packed Shamir sharings which contain n_i copies of the value x_i for all $i \in \{1, 2, \ldots, k\}$. We assume that $\sum_{i=1}^{k} n_i$ is a multiple of k. In Section 5, we will show how to compile a general circuit to meet this requirement. The description of $\mathcal{F}_{\text{fan-out-semi}}$ appears in Functionality 10.

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

1. $\mathcal{F}_{\text{fan-out-semi}}$ receives from honest parties the shares of $[\boldsymbol{x}]$ and a vector $(n_1, n_2, \ldots, n_k)$.
2. $\mathcal{F}_{\text{fan-out-semi}}$ reconstructs the secrets $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$. Then $\mathcal{F}_{\text{fan-out-semi}}$ initiates an empty list L. From $i = 1$ to k, $\mathcal{F}_{\text{fan-out-semi}}$ inserts n_i times of x_i into L.
3. Let $m = \frac{n_1+n_2+\ldots+n_k}{k}$. From $i = 1$ to m,
   (a) $\mathcal{F}_{\text{fan-out-semi}}$ sets $\boldsymbol{x}^{(i)}$ to be the vector of the first k elements in L, and then removes the first k elements in L.
   (b) $\mathcal{F}_{\text{fan-out-semi}}$ receives from the adversary a set of shares $\{s_j^{(i)}\}_{j \in \mathcal{C}}$. $\mathcal{F}_{\text{fan-out-semi}}$ generates a degree-t packed Shamir sharing $[\boldsymbol{x}^{(i)}]$ such that the j-th share of $[\boldsymbol{x}^{(i)}]$ is $s_j^{(i)}$.
   (c) $\mathcal{F}_{\text{fan-out-semi}}$ distributes the shares of $[\boldsymbol{x}^{(i)}]$ to honest parties.

We refer the readers to the full version of this paper [GPS21] for the protocol that realizes $\mathcal{F}_{\text{fan-out-semi}}$. Our protocol prepares the output sharings of $\mathcal{F}_{\text{fan-out-semi}}$ one by one. Therefore, the communication complexity only depends on the number of output sharings *even if the output sharings come from different invocations with different input sharings.* The communication complexity of computing m output sharings is $O(m \cdot n + n^5 \cdot \kappa)$ field elements.

5 Main Protocol - Against a Semi-honest Adversary

In this section, we will introduce our main protocol of using packed Shamir sharing to evaluate a general circuit C against a *semi-honest* adversary. We first discuss how to compile a general circuit to meet the requirements we assume in Section 4. Then we give the main protocol and analyze its security and communication complexity.

5.1 Transforming a General Circuit C

We will prove the following theorem.

Theorem 5. *Given an arithmetic circuit C with input coming from c clients, there exists an efficient algorithm which takes C as input and outputs an arithmetic circuit C' with the following properties:*

- *For all input x, $C(x) = C'(x)$.*
- *In the input layer and the output layer, the number of input gates belonging to each client and the number of output gates belonging to each client are multiples of k. In each intermediate layer, the number of addition gates and the number of multiplication gates are multiples of k.*
- *After grouping the gates that have the same type in each layer, the number of times that the output wires of each group are used in later layers is a multiple of k.*
- *Circuit size: $|C'| = O(|C| + k \cdot (c + \mathsf{Depth}))$, where c is the number of clients that provide inputs and Depth is the depth of C.*

In the full version of this paper [GPS21], we explain why the properties we assume in Section 4 can be met by applying the transformation in Theorem 5 and then formally prove this theorem.

5.2 Preprocessing Phase

In this part, we describe how parties preprocess the circuit before doing the computation. During the computation phase, a batch of k wire values are stored in a single packed Shamir sharing. The main task of the preprocessing phase is to determine how the wire values should be packed. Also, all parties need to compute a permutation for each output sharing using the algorithm in Theorem 4. These permutations are used to achieve the non-collision property. See Section 4.3 for more details. The preprocessing phase only depends on the circuit C and does not need any communication. We refer the readers to the full version of this paper [GPS21] for the description of the protocol Preprocess.

5.3 Main Protocol - Against Semi-honest Adversary

We are ready to introduce our main protocol. At a high-level, given the preprocessed circuit,

- all parties use $\mathcal{F}_{\text{input-semi}}, \mathcal{F}_{\text{output-semi}}, \mathcal{F}_{\text{mult-semi}}$ (see Section 4.1) to evaluate input gates, output gates, multiplication gates, and addition gates in each layer;
- for the input layer and all intermediate layers, all parties use $\mathcal{F}_{\text{fan-out-semi}}$ to evaluate fan-out gates (see Section 4.4);
- for each output sharing, all parties use $\mathcal{F}_{\text{permute-semi}}$ to perform the permutation associated with this sharing (see Section 4.2) to achieve the non-collision property (see Section 4.3);
- to prepare each input sharing for the next layer, all parties use $\mathcal{F}_{\text{select-semi}}$ to choose the secrets it wants from the output sharings from previous layers (see Section 4.2), and then use $\mathcal{F}_{\text{permute-semi}}$ to permute the secrets to the correct order (see Section 4.2).

The ideal functionality $\mathcal{F}_{\text{main-semi}}$ appears in Functionality 11. The main protocol is introduced in Protocol 12.

Functionality 11: $\mathcal{F}_{\text{main-semi}}$

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

Lemma 4. *Protocol* Main-semi *securely computes* $\mathcal{F}_{main\text{-}semi}$ *in the* ($\mathcal{F}_{input\text{-}semi}$, $\mathcal{F}_{fan\text{-}out\text{-}semi}, \mathcal{F}_{permute\text{-}semi}, \mathcal{F}_{select\text{-}semi}, \mathcal{F}_{mult\text{-}semi}, \mathcal{F}_{output\text{-}semi}$)*-hybrid model against a semi-honest adversary who controls* $t' = t - k + 1$ *parties.*

The proof of Lemma 4 can be found in the full version of this paper [GPS21].

The overall communication complexity of our protocol Main-semi is $O(|C| \cdot n/k + n \cdot (c + \mathsf{Depth}) + n^5 \cdot \kappa)$ field elements. In the full version of this paper [GPS21], we provide the analysis of the communication complexity of our protocol in details. Together with Lemma 4, we have the following theorem.

Theorem 6. *In the client-server model, let* c *denote the number of clients, and* $n = 2t + 1$ *denote the number of parties (servers). Let* κ *denote the security parameter, and* $\mathbb{F}$ *denote a finite field. For an arithmetic circuit* C *over* $\mathbb{F}$ *and for all* $1 \leq k \leq t$*, there exists an information-theoretic MPC protocol which securely computes the arithmetic circuit* C *in the presence of a semi-honest adversary controlling up to* c *clients and* $t - k + 1$ *parties. The communication complexity of this protocol is* $O(|C| \cdot n/k + n \cdot (c + \mathsf{Depth}) + n^5 \cdot \kappa)$ *elements in* $\mathbb{F}$*.*

References

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

BGIN20. Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Efficient fully secure computation via distributed zero-knowledge proofs. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 244–276, Cham, 2020. 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.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19. ACM, 1988.

CCXY18. Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure mpc revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology –*

Protocol 12: MAIN-SEMI

1. **Circuit Transformation Phase**. Let C denote the evaluated circuit. All parties preprocess the circuit by running the PREPROCESS protocol. Let C' denote the circuit after transformation.
2. **Input Phase**. Let $\mathsf{Client}_1, \mathsf{Client}_2, \ldots, \mathsf{Client}_c$ denote the clients who provide inputs.
   (a) Input Secret-sharing Phase: For every group of k input gates of Client_i, Client_i invokes $\mathcal{F}_{\text{input-semi}}$ to share its inputs $\boldsymbol{x}^{(i)}$ to the parties.
   (b) Handling Fan-out Gates: For the output sharing $[\boldsymbol{x}]$ of each group of input gates, let n_i denote the number of times that the i-th secret of $\boldsymbol{x}$ is used in later layers. All parties invoke $\mathcal{F}_{\text{fan-out-semi}}$ with input $[\boldsymbol{x}]$ and $(n_1, n_2, \ldots, n_k)$.
   (c) Achieving Non-Collision Property for the next layers: For each output sharing $[\boldsymbol{y}]$ of the input layer, let p denote the permutation associated with it. All parties invoke $\mathcal{F}_{\text{permute-semi}}$ with input $[\boldsymbol{y}]$ and p.
3. **Evaluation Phase**. All parties evaluate the circuit layer by layer as follows:
   (a) Permute Input Sharings from Previous Layers: For each input sharing $[\boldsymbol{x}]$, let $[\boldsymbol{x}^{(i)}]$ denote the output sharing from previous layers which contains the i-th secret x_i, and let q_i denote the position of x_i in $[\boldsymbol{x}^{(i)}]$. According to the non-collision property, $q_1, q_2, \ldots, q_k$ is a permutation of $(1, 2, \ldots, k)$. Let $q(\cdot)$ be a permutation over $\{1, 2, \ldots, k\}$ such that $q(i) = q_i$. All parties invoke $\mathcal{F}_{\text{select-semi}}$ on $[\boldsymbol{x}^{(1)}], [\boldsymbol{x}^{(2)}], \ldots, [\boldsymbol{x}^{(k)}]$ and the permutation q. Let $[\boldsymbol{x}']$ denote the output of $\mathcal{F}_{\text{select-semi}}$. Then, all parties invoke $\mathcal{F}_{\text{permute-semi}}$ with input $[\boldsymbol{x}']$ and q to obtain $[\boldsymbol{x}]$.
   (b) Evaluating Multiplication Gates and Addition Gates: For each group of multiplication gates with input sharings $[\boldsymbol{x}], [\boldsymbol{y}]$, all parties invoke $\mathcal{F}_{\text{mult-semi}}$ with input $[\boldsymbol{x}], [\boldsymbol{y}]$. For each group of addition gates with input sharings $[\boldsymbol{x}], [\boldsymbol{y}]$, all parties locally compute $[\boldsymbol{x} + \boldsymbol{y}] = [\boldsymbol{x}] + [\boldsymbol{y}]$.
   (c) Handling Fan-out Gates: For the output sharing $[\boldsymbol{x}]$ of each group of multiplication gates or addition gates, all parties follow the same step as Step 2.(b) to handle fan-out gates.
   (d) Achieving Non-Collision Property: Follow Step 2.(c).
4. **Output Phase**.
   (a) Permute Input Sharings from Previous Layers: For each input sharing $[\boldsymbol{x}]$, all parties follow the same step as Step 3.(a) to prepare $[\boldsymbol{x}]$.
   (b) Reconstruct the Output: For each group of output gates belonging to Client_i ($i \geq 1$), let $[\boldsymbol{x}]$ denote the input sharing. All parties invoke $\mathcal{F}_{\text{output-semi}}$ with input $[\boldsymbol{x}]$ to let Client_i learn the result $\boldsymbol{x}$.

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.

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.

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 699710, New York, NY, USA, 1992. Association for Computing Machinery.

GIOZ17. Juan Garay, Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. The price of low communication in secure multi-party computation. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 420–446, Cham, 2017. Springer International Publishing.

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

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

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

GPS21. Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Unconditional communication-efficient mpc via hall's marriage theorem. Cryptology ePrint Archive, Report 2021/834, 2021. https://eprint.iacr.org/2021/834.

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

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

PS21. Antigoni Polychroniadou and Yifan Song. Constant-overhead unconditionally secure multiparty computation over binary fields. Appears in Eurocrypt, 2021. https://eprint.iacr.org/2020/1412.

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

Yao82. Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.