Scalable Secure Multiparty Computation with Perfect Security from Preprocessing

Yifan Song[1] and Xiaxi Ye[2]

[1] Tsinghua University and Shanghai Qi Zhi Institute
yfsong@mail.tsinghua.edu.cn
[2] Institute for Network Sciences and Cyberspace, Tsinghua University
yexx23@mails.tsinghua.edu.cn

Abstract. In this work, we study the communication complexity of MPC achieving perfect security with optimal resilience ($t < n/3$). We ask the question: "Is it possible to build a perfectly secure MPC for arithmetic circuits of size $|C|$ with optimal resilience with communication of $o(|C| \cdot n)$ field elements?"
On the positive side, we construct a perfectly secure MPC protocol for general arithmetic circuits with communication complexity of $O(|C|)$ elements assuming preprocessing data of size $O(|C|)$, where the preprocessing data consists of packed Beaver triples over bivariate polynomials. Furthermore, we show that packed Beaver triples over bivariate polynomials can be prepared at an amortized cost of $O(1)$ elements plus $O(1)$ three-party Beaver triples per secret.
On the negative side, we establish a communication lower bound proving that preparing packed Beaver triples over bivariate polynomials requires at least $\Omega(n)$ elements of communication per secret. This lower bound is derived by first proving a communication lower bound for verifying the correctness of packed Beaver triples with perfect security with abort, and then efficiently reducing the task of verifying packed Beaver triples to preparing packed Beaver triples over bivariate polynomials. To match this bound, we give a concrete construction for preparing packed Beaver triples over bivariate polynomials with $O(n)$ elements per secret, demonstrating the tightness of our lower bound.
Our proof technique also extends to show that for the task of computing the inner-product of two length-$|C|$ vectors, any MPC protocol that achieves perfect security with abort requires either $\Omega(|C| \cdot n)$ elements of communication or $\Omega(|C|)$ elements of preprocessing data.

Table of Contents

1 Introduction . . . 3
1.1 Our Contribution . . . 3
1.2 Related Works . . . 5
2 Technical Overview . . . 5
2.1 Starting Point: Perfect Security with Abort in the Preprocessing Model . . . 5
2.2 Perfect Security with GOD from Bivariate Polynomials . . . 7
2.3 Negative Result of Preparing Packed Beaver Triples . . . 9
2.4 Organization of the Document . . . 11
3 Preliminary . . . 12
3.1 Security Definition . . . 12
3.2 Packed Secret Sharing via Bivariate Polynomials . . . 12
3.3 Byzantine Agreement . . . 13
3.4 Party Elimination Framework . . . 13
4 Perfectly Secure MPC in the Preprocessing Model . . . 13
4.1 Preprocessing Data . . . 14
4.2 Identifiability from Bivariate Polynomials . . . 14
4.3 Computation of Multiplication Gates . . . 17
4.4 Towards General Circuits . . . 19
4.5 Summary for Segment Computation . . . 23
4.6 Main Protocol . . . 24
5 Lower Bounds . . . 40
5.1 Warm-up: Impossibility Result of Computing AND of Two Bits . . . 40
5.2 Communication Complexity of Verifying Packed Triples . . . 42
5.3 Communication Complexity of Preparing Packed Triples over Bivariate Polynomials . . . 45
5.4 Communication Complexity of Inner Product Functions . . . 50
6 Preparing Packed Beaver Triples . . . 56
7 Packed Beaver Triples from 3-Party Beaver Triples . . . 58
A Additional Preliminary . . . 72
A.1 Properties of Packed Shamir Sharings over Bivariate Polynomials . . . 72
A.2 Building Blocks . . . 74
B Proofs of Propositions in Lemma 3 . . . 76
C The Figures . . . 82

1 Introduction

Secure multiparty computation (MPC) allows a set of mutually distrustful parties to jointly compute a function on their private inputs. Informally, an MPC protocol guarantees that each party learns only his own input and output, and nothing else. Since the concept was introduced by [Yao82], initial feasibility results were obtained in the computational setting, where adversaries are assumed to have bounded computational power [Yao82,GMW87]. Later works [BGW88,CCD88] extended these results to the unconditional (information-theoretic) setting, showing that MPC is achievable against up to $t < n/3$ corrupted parties, assuming secure point-to-point communication channels.

In this work, we focus on the communication complexity of perfectly secure MPC protocols with optimal resilience $t < n/3$. The first feasibility result was given by [BGW88] in 1988. Since then, a line of works focuses on improving the communication complexity of perfectly secure MPC protocols. In the literature, it is known from [BH08,GLS19,AAPP23] that linear communication of $O(|C| \cdot n)$ field elements can be achieved to compute a circuit of size $|C|$ among n parties. The recent work [SY24] further improves the online communication complexity to $O(|C|)$ field elements while maintaining the overall linear communication complexity, inspired by Turbopack [EGPS22] that achieves a similar result in the honest majority setting. Despite the great improvement made by the previous works, there is no construction that achieves $o(|C| \cdot n)$ field elements and we still do not know whether a linear communication complexity is necessary for perfectly secure MPC protocols. The only evidence is from [DLS21] which proves a lower bound of $\Theta(|C| \cdot n)$ field elements of communication for certain types[3] of perfectly secure MPC protocols targeting for a specific function. This leads to our following question.

"Can we build a perfectly secure MPC protocol with optimal resilience ($t < n/3$) that achieves $o(|C| \cdot n)$ field elements of communication complexity when computing a circuit of size $|C|$?"

1.1 Our Contribution

In this work we make progress towards answering the above question from two aspects. On the positive side, we show that in the preprocessing model, there is a perfectly secure MPC protocol with optimal resilience that requires both $O(|C|)$ field elements of preprocessing data and communication among all parties.

Theorem 1 (informal). *Let n be the number of parties and $t = (n-1)/3$. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2n$. For an arithmetic circuit C over $\mathbb{F}$, there exists a perfectly secure MPC protocol in the circuit-independent preprocessing model computing C in the presence of a fully malicious adversary controlling at most t parties. The cost is $O(|C| + D \cdot n + C_O \cdot n + n^5)$ elements of communication along with preprocessing data of size $O(|C|)$, where $|C|$, D, and C_O denote the circuit size, circuit depth, and the number of output gates, respectively.*

The formal theorem is stated as Theorem 6. As a byproduct of this result, a variant of our construction yields a perfectly secure MPC protocol for general arithmetic circuits with constant communication overhead per gate in the suboptimal corruption threshold setting $t = (1/3 - \epsilon)n$ for any constant $\epsilon \in (0, 1/3)$.

At a first glance, the readers may feel that a similar result can be obtained from [SY24]: If we view the output in the offline phase of [SY24] as preprocessing data, we directly obtain a construction in the preprocessing model with $O(|C|)$ field elements of communication. However, the size of the output in the offline phase of [SY24] is $\Theta(|C| \cdot n)$ field elements. This is because their construction requires to prepare a random degree-t Shamir sharing for each wire in the circuit. From the technique side, these random degree-t Shamir sharings allow them to compute a degree-t Shamir sharing for each wire value in the online phase and the error-correction property allows them to always reconstruct the whole degree-t Shamir sharing. In our case, we have to always use packed Shamir sharings to keep $O(|C|)$ field elements of preprocessing data and communication, in which case parties can only detect inconsistency but cannot reconstruct the secrets

[3] The lower bounds in [DLS21] apply to the so-called two-phase protocols, in which a part of the inputs is committed during the input phase, before any information about the rest of the inputs and the output are revealed.

with guarantee. This brings difficulties when the reconstruction fails. We will elaborate our techniques in Section 2.

Going beyond, we show that the required preprocessing data in Theorem 1 can be prepared from three-party Beaver triples.

Theorem 2 (informal). *Let n be the number of parties and $t = (n-1)/3$. There exists a protocol that, assuming three-party Beaver triples, generates the preprocessing data of size $O(|C|)$ used in Theorem 1 with perfect security against a fully malicious adversary corrupting at most t corrupted parties. The total cost is $O(|C|)$ elements of communication plus $O(|C|)$ three-party Beaver triples.*

The formal statement appears in Lemma 6.

On the negative side, we show that the preprocessing data we need in Theorem 1 cannot be prepared with $o(|C| \cdot n)$ communication *in the plain model*. To be more concrete, the preprocessing data we need is a variant of packed Beaver triples based on packed Shamir sharings. Our negative result shows that no perfectly secure MPC protocol can prepare such packed Beaver triples with sub-linear communication.

Theorem 3 (informal). *Let $n = 3t + 1$. Any protocol generating packed Beaver triples over bivariate polynomials of size $O(|C|)$ with perfect security with abort against a fully malicious adversary controlling at most t corrupted parties communicates at least $\Omega(|C| \cdot n)$ field elements.*

The formal theorem is stated as Theorem 7. To match this lower bound, we give a concrete construction for preparing packed Beaver triples over bivariate polynomials in Section 6, demonstrating the tightness of our lower bound.

Theorem 4 (informal). *Let n be the number of parties and $t = (n-1)/3$. There exists a protocol generating packed Beaver triples over bivariate polynomials of size $O(|C|)$ with perfect security against a fully malicious adversary controlling at most t corrupted parties. The communication cost is $O(|C| \cdot n)$ elements.*

The formal statement is presented in Lemma 5. We further extend our proof technique to show that for the task of computing the inner-product of two vectors of length $|C|$, a perfectly secure MPC protocol requires either $\Omega(|C| \cdot n)$ elements of communication or $\Omega(|C|)$ elements of preprocessing data.

Theorem 5 (informal). *Let $n = 3t + 1$. Any protocol that computes the inner-product of two vectors of length $|C|$ with perfect security with abort in the preprocessing model, against a fully malicious adversary controlling at most t corrupted parties, either communicates $\Omega(|C| \cdot n)$ elements or requires preprocessing data of size $\Omega(|C|)$.*

The formal theorem is stated as Theorem 8 and Corollary 3. This result implies that there exists a circuit $\tilde{C}$ with input size $|\tilde{C}|$ and output size $O(1)$ for which a linear communication of $\Theta(|\tilde{C}| \cdot n)$ field elements is necessary for perfectly secure MPC protocols. It also suggests that there may not exist a perfectly secure MPC protocol with $o(|C| \cdot n)$ communication for a general circuit C.

Comparison with [DPPR26]. We note that a concurrent work [DPPR26] obtains a similar communication lower bound. To be more concrete, in the optimal resilience ($t < n/3$) setting, they show that for the task of computing a variant of oblivious transfer (OT) functionality with perfect security with abort, there is no MPC protocol with $o(I \cdot n)$ communication, where I is the message length. Note that such a string OT functionality can be computed with a circuit of size $\Theta(I)$. In comparison, our work targets the task of preparing packed Beaver triples, which leads to a perfectly secure MPC protocol with both $O(|C|)$ communication and $O(|C|)$ preprocessing data, and the task of computing the inner-product of two vectors.

From the technique side, the authors in [DPPR26] begin with a four-party protocol that computes the string OT functionality with perfect security with abort. They then derive a three-party protocol for the same functionality by randomly emulating the fourth party, while preserving the communication complexity. This three-party protocol is shown to imply a secure two-party protocol for private message transfer over authenticated and insecure channels in the correlated randomness model, where the amount of correlated

randomness required corresponds to the communication complexity of the three-party protocol. Then they show a lower bound that, in any such two-party private message transfer protocol, the correlated randomness must be long enough to one-time pad the entire private message. This in turn implies a communication lower bound for the original four-party protocol that securely realizes the string OT functionality. Finally, the result is extended to the general n-party setting with optimal resilience ($t < n/3$).

In contrast, our technique starts from a negative result in the 3-party setting for perfect security with abort. Very informally, we show that there is a strategy for the corrupted party that causes honest parties to end up with an incorrect output for the AND function with non-zero probability. Then in the 4-party setting, we manage to show that if the communication of a party is too small, the same attack can be applied, which violates the perfect correctness. This implies a communication lower bound for each party. Finally, we show that this result naturally extends to the n-party setting, which gives the lower bound we want.

1.2 Related Works

Lower bounds. In honest majority setting with semi-honest and statistical security, the result of [DNOR16] implies that, for a special class of Boolean circuits with $|C| = n$, $\Omega(n^2)$ bits of communication is required. [DNPR16] shows that for protocols following the "gate-by-gate" paradigm, communicating $\Omega(n)$ bits per multiplication gate is necessary. [DLN19] proves that any protocols that computes an inner-product like function with $|C|$ as its input and circuit size must communicate $\Omega(|C| \cdot n)$ bits. Focusing on perfect malicious security, for optimal resilience $n = 3t + 1$, [DLS21] proves that for a family of Boolean circuits, any two-phase reactive protocol must communicate on average at least $\Omega(n)$ bits per gate. Its proof only applies to two-phase protocols, where a part of the inputs has been committed in the input phase before receiving rest of the inputs and the outputs. We emphasize that both the work [DPPR26] and our work establish lower bounds for perfect security with abort, which is a weaker security notion.

Upper bounds. Focusing on perfect security with optimal resilience ($n = 3t + 1$) and ignoring circuit-independent costs, [BH08] achieves $O(|C| \cdot n + D \cdot n^2)$ elements of communication. Later, [GLS19] eliminates the circuit depth dependent term and achieves communication of $O(|C| \cdot n)$ elements. To obtain constant online communication per gate, [SY24] relies on packed secret sharings and achieves $O(|C| + D \cdot n)$ elements for the online phase, $O(|C| \cdot n + D \cdot n^2)$ elements for the preprocessing phase. Aiming for $O(D)$ round complexity, [AAY21] achieves $O(|C| \cdot n^3)$ elements of communication. This is later improved by [AAPP23], which achieves linear communication of $O(|C| \cdot n + D \cdot n^2)$ elements.

2 Technical Overview

In this section, we give a high-level overview of our main techniques. Let $\mathbb{F}$ be a finite field such that $|\mathbb{F}| \geq 2n$. Recall that in this work we focus on the optimal resilience setting in which $n = 3t + 1$. We will use $[\boldsymbol{x}]_d$ to denote a degree-d packed Shamir sharing of the secret vector $\boldsymbol{x} \in \mathbb{F}^k$, where k is the number of secrets packed in a single sharing. To achieve privacy against t corrupted parties, we will use a degree-$(t + k - 1)$ packed Shamir sharing to store k secrets. In addition, we set $k = \left\lceil \frac{n+5}{6} \right\rceil$ so that $t + 2k - 2 = 2t$. Jumping ahead, we will use the technique of packed Beaver triple [GPS22] to compute a group of k multiplications where parties need to reconstruct a degree-$(t + 2k - 2)$ packed Shamir sharing. Setting k in the above way ensures that the whole sharing is determined by shares held by all honest parties.

In the following, we start with our positive result. For simplicity, we focus on a SIMD circuit.

2.1 Starting Point: Perfect Security with Abort in the Preprocessing Model

Our first step is to understand what kind of preprocessing data is sufficient for us to build a perfectly secure MPC protocol with $O(|C|)$ communication. As we have mentioned in Section 1.1, we may interpret the construction in [SY24] as a protocol in the preprocessing model, by viewing the output in the offline

phase as preprocessing data. This gives us a perfectly secure MPC protocol with $O(|C|n)$ field elements of preprocessing data, and $O(|C|)$ field elements of communication.

When focusing on a SIMD circuit, the preprocessing data in [SY24] just contains Beaver triples over standard degree-t (non-packed) Shamir sharings. These standard Beaver triples are packed to degree-$(t+k-1)$ packed Beaver triples and then used in the online phase. If the online phase fails due to an inconsistent degree-$(t+k-1)$ packed Shamir sharing, parties may use the degree-t Shamir sharings in the preprocessing data to recover the correct sharing and locate a corrupted party.

We notice that if we only focus on perfect security with abort, it is sufficient to only assume packed Beaver triples. This is because during the online computation, parties only need to reconstruct degree-$(t+k-1)$ and degree-$(t+2k-2)$ packed Shamir sharings. In both cases, the whole sharing is fully determined by honest parties' shares. Thus corrupted parties cannot insert error without being detected. To be more concrete, suppose all parties want to compute a group of k multiplication gates. Parties start with input sharings $[\boldsymbol{x}]_{t+k-1}, [\boldsymbol{y}]_{t+k-1}$ and hold a random packed Beaver triple $[\boldsymbol{a}]_{t+k-1}, [\boldsymbol{b}]_{t+k-1}, [\boldsymbol{c}]_{t+k-1}$ satisfying that $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$, where $*$ denotes the coordinate-wise multiplication operation. All parties in addition prepare a pair of random packed Shamir sharings $([\boldsymbol{r}]_{t+k-1}, [\boldsymbol{r}]_{t+2k-2})$, which can be prepared with perfect security [SY24].

1. All parties locally compute $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$.
2. A designated party P_{king} receives $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$ and reshares $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ to all parties.
3. All parties locally compute

$$\begin{aligned}[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2} =& [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{b}]_{t+k-1} \\ & - [\boldsymbol{a}]_{t+k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} + [\boldsymbol{c}]_{t+k-1} + [\boldsymbol{r}]_{t+2k-2}.\end{aligned}$$

4. The party P_{king} receives $[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$ and reshares $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$.
5. All parties locally compute $[\boldsymbol{z}]_{t+k-1} = [\boldsymbol{z}+\boldsymbol{r}]_{k-1} - [\boldsymbol{r}]_{t+k-1}$.

In the above process, an honest P_{king} can detect inconsistency in Step 2 and Step 4. To protect against a corrupted P_{king}, additional verifications are required which we omit here for simplicity.

Thus, from the construction in [SY24], we may obtain a perfectly secure MPC protocol with abort that achieves $O(|C|)$ elements of both preprocessing data and communication by just providing packed Beaver triples as preprocessing data.

Difficulty of Achieving Guaranteed Output Delivery. To achieve guaranteed output delivery, a generic approach is to use the party-elimination framework [HMP00]. At a high level, when the computation fails, all parties will together find out a corrupted party who deviates from the computation, or a pair of conflicted parties where they do not agree on the message sent from one party to the other. Notice that for the latter case, at least one party is corrupted. Such a pair of parties is also referred to as a disputed pair. This process is referred to as the *fault-localization* process. Then the identified corrupted party or the identified pair of conflicted parties are removed. The whole computation restarts with the remaining parties. Note that the corruption threshold of the remaining parties is still bounded by $1/3$. Since each time at least one corrupted party is eliminated, the number of failure attempts is bounded by $t = O(n)$. To avoid the communication blowup due to the failure attempts and retries, the whole computation is separated into $\Omega(n)$ segments that are evaluated sequentially. In case the computation of the current segment fails, all parties will only restart the computation of the current segment.

To make the above idea work, we need to construct a secure protocol that realizes the fault-localization process while protecting the secrecy of their inputs. This is usually not a difficult task in the plain model:

– First, if we do not care about the secrecy of their inputs, such a corrupted party or a disputed pair can be easily located by letting each party broadcast his view. Then each party may locally emulate the whole protocol and verify whether each party follows the protocol. Now because the computation fails, there must be either a corrupted party who does not follow the computation, or a pair of parties who do not agree on the message sent from one party to the other.

– To protect the secrecy of their inputs, we can simply use a generic MPC protocol where each party uses his view as input. To avoid the communication cost of the fault localization process dominating the communication complexity of the final protocol, we may choose a smaller segment size (and thus the number of segments may become $\mathtt{poly}(n)$).

Unfortunately the above generic solution does not work in the preprocessing model even if we do not care about the secrecy of parties' inputs. This is because the preprocessing data is not generated by any party and if corrupted parties lie about their preprocessing data, we may not be able to find out the corrupted party or a disputed pair. For example, after opening all parties' view, we find that the first packed Shamir sharing $[\boldsymbol{a}]_{t+k-1}$ of some packed Beaver triple is inconsistent. Then we do not know which parties provide wrong shares. This is unavoidable since a degree-$(t+k-1)$ packed Shamir sharing cannot be reconstructed with guarantee when there are t incorrect shares as long as $k \geq 2$.

2.2 Perfect Security with GOD from Bivariate Polynomials

To resolve the above issue, our idea is to extend packed Shamir sharings which are based on *univariate* polynomials to *bivariate* polynomials. To be more concrete, let $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_k$ be $n+k$ different field elements. To share a vector $\boldsymbol{x} \in \mathbb{F}^k$ using the packed Shamir sharing scheme, the dealer samples a random degree-$(t+k-1)$ polynomial $f(\cdot)$ such that $f(\beta_i) = x_i$ for all $i \in \{1, \ldots, k\}$ and sets $[\boldsymbol{x}]_{t+k-1} = (f(\alpha_1), \ldots, f(\alpha_n))$. When extending to bivariate polynomials, the secret $\boldsymbol{X}$ is a matrix of size $k \times k$. The dealer samples a random degree-$(t+k-1, t+k-1)$ bivariate polynomial $F(\cdot,\cdot)$ such that $F(\beta_i, \beta_j) = X_{i,j}$. Then each party P_i receives $f_i(\cdot) = F(\cdot, \alpha_i)$ and $g_i(\cdot) = F(\alpha_i, \cdot)$ as his shares. Effectively, this can be viewed as a linear packed secret sharing with packing size k^2. We denote such a packed sharing as $[\![\boldsymbol{X}]\!]_{t+k-1}$. Later we show that we can extend the technique of Beaver triple to such a linear packed secret sharing scheme and then use it to evaluate a group of k^2 multiplication gates in parallel.

The main benefit of using bivariate polynomials is the pairwise consistency: every pair of parties (P_i, P_j) hold two common points $F(\alpha_i, \alpha_j)$ and $F(\alpha_j, \alpha_i)$. Now when $[\![\boldsymbol{X}]\!]_{t+k-1}$ is inconsistent, each party P_i broadcasts his share $f_i(\cdot), g_i(\cdot)$. Then we must observe at least one of the two cases:

– For some party P_i, $f_i(\cdot)$ or $g_i(\cdot)$ is not a valid degree-$(t+k-1)$ polynomial;
– For some pair of parties (P_i, P_j), $f_i(\alpha_j) \neq g_j(\alpha_i)$ or $g_i(\alpha_j) \neq f_j(\alpha_i)$.

For the first case, P_i is a corrupted party. For the second case, (P_i, P_j) is a pair of disputed parties. To see why this is the case, first notice that all honest parties' shares are consistent with the underlying bivariate polynomial $F(\cdot,\cdot)$. Now suppose a corrupted party P_i provides an incorrect share, say $f_i(\cdot) \not\equiv F(\cdot, \alpha_i)$. Either $f_i(\cdot)$ is not a valid degree-$(t+k-1)$ polynomial or $f_i(\alpha_j) = F(\alpha_j, \alpha_i)$ holds for at most $t+k-1 \leq 2t$ different α_j. In the latter case, there is an honest party P_j such that $f_i(\alpha_j) \neq F(\alpha_j, \alpha_i) = g_j(\alpha_i)$. Thus the pairwise consistency does not hold for (P_i, P_j), which will be identified as a disputed pair.

Thus, although there is still no guarantee that $[\![\boldsymbol{X}]\!]_{t+k-1}$ can be always reconstructed when there are t incorrect shares, we are able to identify either a corrupted party or a disputed pair when the reconstruction fails *without the help from the dealer*.

Other Difficulties and Our Solutions. Now we sketch our construction for a SIMD circuit based on packed sharings from bivariate polynomials. At a high level, we follow the party-elimination framework and utilize the pairwise consistency of packed sharings based on bivariate polynomials to locate disputed pairs.

In the beginning, all parties receive packed Beaver triples in the form of $([\![\boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{B}]\!]_{t+k-1}, [\![\boldsymbol{C}]\!]_{t+k-1})$ where $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{F}^{k \times k}$ are random matrices and $\boldsymbol{C} = \boldsymbol{A} * \boldsymbol{B}$. For each group of k^2 multiplication gates, all parties also prepare $([\![\boldsymbol{R}]\!]_{t+k-1}, [\![\boldsymbol{R}]\!]_{t+2k-2})$ which can be done with an amortized communication of $O(n^2)$ field elements following the technique in [BH08].

In the online phase, for each group of k^2 addition gates with input sharings $[\![\boldsymbol{X}]\!]_{t+k-1}$ and $[\![\boldsymbol{Y}]\!]_{t+k-1}$, all parties locally add up their shares to obtain $[\![\boldsymbol{X}+\boldsymbol{Y}]\!]_{t+k-1} = [\![\boldsymbol{X}]\!]_{t+k-1} + [\![\boldsymbol{Y}]\!]_{t+k-1}$. For each group of k^2 multiplication gates with input sharings $[\![\boldsymbol{X}]\!]_{t+k-1}$ and $[\![\boldsymbol{Y}]\!]_{t+k-1}$, a fresh Beaver triple is consumed.

1. All parties locally compute $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{B}+\boldsymbol{Y}]\!]_{t+k-1}$.
2. A designated party P_{king} receives $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{B}+\boldsymbol{Y}]\!]_{t+k-1}$ and reshares $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{k-1}, [\![\boldsymbol{B}+\boldsymbol{Y}]\!]_{k-1}$ to all parties.
3. All parties locally compute

$$\begin{aligned}[\![\boldsymbol{Z}+\boldsymbol{R}]\!]_{t+2k-2} =& [\![\boldsymbol{X}+\boldsymbol{A}]\!]_{k-1} \cdot [\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{k-1} - [\![\boldsymbol{X}+\boldsymbol{A}]\!]_{k-1} \cdot [\![\boldsymbol{B}]\!]_{t+k-1} \\ & - [\![\boldsymbol{A}]\!]_{t+k-1} \cdot [\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{k-1} + [\![\boldsymbol{C}]\!]_{t+k-1} + [\![\boldsymbol{R}]\!]_{t+2k-2}.\end{aligned}$$

4. The party P_{king} receives $[\![\boldsymbol{Z}+\boldsymbol{R}]\!]_{t+2k-2}$ and reshares $[\![\boldsymbol{Z}+\boldsymbol{R}]\!]_{k-1}$.
5. All parties locally compute $[\![\boldsymbol{Z}]\!]_{t+k-1} = [\![\boldsymbol{Z}+\boldsymbol{R}]\!]_{k-1} - [\![\boldsymbol{R}]\!]_{t+k-1}$.

For each sharing P_{king} received, if the sharing is inconsistent, P_{king} verifies the pairwise consistency and finds (P_i, P_j) whose common shares do not match. Then P_{king} broadcasts the identities of these two parties together with the received shares from P_i and P_j, and P_i, P_j broadcast the shares they sent to P_{king}. The two parties who broadcast different values are identified as a disputed pair. (To protect against a corrupted P_{king}, additional verifications are required for sharings distributed by P_{king}, which we omit here for simplicity and refer the readers to Section 4 for more details.) If a disputed pair of parties is identified, all parties eliminate these two parties and re-evaluate the current segment following the party-elimination framework.

However, to make the above fault localization process work, we need the number of remaining honest parties to be at least $t+2k-1$ so that $[\![\boldsymbol{Z}+\boldsymbol{R}]\!]_{t+2k-2}$ is fully determined by honest parties' shares. Recall that $t+2k-1 = 2t+1$. This means that after we remove a disputed pair of parties (which may contain one honest party), we cannot rely on the above process to locate a disputed pair.

To resolve this issue, in our construction, we use a smaller packing parameter $k = n/12 = \Theta(n)$, which still preserves $O(|C|)$ field elements of preprocessing data and online communication. On one hand, the above sketch still works as long as the number of eliminated honest parties is bounded by $(2t+1)-(t+2k-1) \approx n/6$. On the other hand, after eliminating $n/6$ pairs of disputed parties, the corruption threshold of the remaining parties is reduced to $1/4$. In this case, the remaining parties may restart the whole computation using a perfectly secure MPC protocol against $1/4$ corruption, which achieves $O(|C|)$ communication using packed Shamir sharings, as detailed in Section 4.6.

Towards General Circuits. Directly using packed Beaver triples over bivariate polynomials ($[\![\boldsymbol{A}]\!]_{t+k-1}$, $[\![\boldsymbol{B}]\!]_{t+k-1}$, $[\![\boldsymbol{C}]\!]_{t+k-1}$) to compute general circuits requires computing a batch of k^2 multiplication gates in a single layer. This could result in an $O(D \cdot n^2)$ overhead on the communication cost, limiting its applicability to circuits within a "thin" shape.

To avoid this limitation, instead of using the whole packed Beaver triple over bivariate polynomials $([\![\boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{B}]\!]_{t+k-1}, [\![\boldsymbol{C}]\!]_{t+k-1})$ to compute $O(n^2)$ multiplication gates in a row, the parties will decompose each degree-$(t+k-1, t+k-1)$ packed Shamir sharing into k degree-$(t+k-1)$ packed Shamir sharings over univariate polynomials to obtain k packed Beaver triples $\{[\![\boldsymbol{a}_\ell]\!]_{t+k-1}, [\![\boldsymbol{b}_\ell]\!]_{t+k-1}, [\![\boldsymbol{c}_\ell]\!]_{t+k-1}\}_{\ell=1}^{k}$ and use a packed Beaver triple to compute a group of k multiplication gates, where $\boldsymbol{a}_\ell, \boldsymbol{b}_\ell, \boldsymbol{c}_\ell \in \mathbb{F}^k$ are the ℓ-th rows of the secret matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}^{k\times k}$, respectively. However, doing this would add difficulty for parties to identify a disputed pair in case a packed sharing $[\![\boldsymbol{x}]\!]_{t+k-1}$ is inconsistent as secrets coming from different rows of the secret matrices can mix together. For instance, when using such a packed Beaver triple to compute multiplication gates, if in Step 2, P_{king} receives an inconsistent degree-$(t+k-1)$ packed Shamir sharing $[\![\boldsymbol{x}+\boldsymbol{a}]\!]_{t+k-1}$, a disputed pair is expected to be identified. The difficulty comes from the fact that $\boldsymbol{x}+\boldsymbol{a}$ can be a linear combination of vectors lying in different rows of secret matrices. To ensure the secrecy, parties cannot open all the involved bivariate polynomials to locate a disputed pair. At a high level, we resolve this issue by first writing $[\![\boldsymbol{x}+\boldsymbol{a}]\!]_{t+k-1}$ as $[\![\boldsymbol{u}_1]\!]_{t+k-1} + \cdots + [\![\boldsymbol{u}_k]\!]_{t+k-1}$ with $\boldsymbol{u}_\ell$ being the ℓ-th row of the secret matrix $\boldsymbol{U}^{(\ell)}$, and then investigating each sharing $[\![\boldsymbol{U}^{(\ell)}]\!]_{t+k-1}$ to locate a disputed pair as well as using the sacrifice technique to protect the secrecy of each $[\![\boldsymbol{U}^{(\ell)}]\!]_{t+k-1}$. As an invariant, throughout the computation, parties compute a packed Shamir sharing $[\boldsymbol{w}]_{t+k-1}$ for each group of k wires $\boldsymbol{w}$, where $[\boldsymbol{w}]_{t+k-1}$ can always be decomposed as $[\boldsymbol{w}_0]_{k-1} + [\![\boldsymbol{w}]\!]_{t+k-1}$. The verifiability of the first component $[\boldsymbol{w}_0]_{k-1}$ relies on error correcting property of degree-$(k-1)$ Shamir sharings as $k = n/12$ and thus $k-1 < t$ while the verifiability of the second one $[\![\boldsymbol{w}_1]\!]_{t+k-1}$ is derived as just mentioned. We refer the readers to Section 4.2 for more details.

To compute input sharings for each group of gates, parties should collect secrets from different packed sharings and align the secrets according to correct orders, which is referred to as *network routing* in [GPS21,GPS22]. To handle network routing, we follow [GPS21,GPS22] and adapt it to our setting of bivariate polynomials to prepare random linear masks for sharing transformation of the form $([\![\boldsymbol{R}]\!]_{t+k-1}, [\![\vec{\boldsymbol{L}}(\boldsymbol{R})]\!]_{t+k-1})$, where $\vec{\boldsymbol{L}} = (\boldsymbol{L}_1, \ldots, \boldsymbol{L}_k)$ is a vector of k linear transformations from $\mathbb{F}^k$ to $\mathbb{F}^k$. We will discuss in more details in Section 4.3.

2.3 Negative Result of Preparing Packed Beaver Triples

Our positive result sheds light on achieve $o(|C|n)$ communication for perfectly secure MPC as it is sufficient to focus on preparing packed Beaver triples over bivariate polynomials with $O(n^2)$ field elements of communication per triple (so that the amortized communication per secret is $O(1)$ field elements).

Unfortunately, in this section we show that the task of preparing packed Beaver triples over bivariate polynomials cannot be achieved with an amortized communication complexity of $o(n^3)$ field elements, even just for perfect security with abort. Our negative result shows that

- there exists a concrete computation task C that cannot be realized with $o(|C|n)$ field elements of communication,
- and for a given circuit $\tilde{C}$, computing $\tilde{C}$ with perfect security may inherently require $O(|C|n)$ a linear communication.

Warm-up: Impossibility Result of 3PC AND. To better explain our idea, we start with a three-party computation task $f_{\mathtt{AND}}(\perp, x_2, x_3) \to (\perp, x_2 \cdot x_3, x_2 \cdot x_3)$. We show that there is no 3-party protocol that computes $f_{\mathtt{AND}}$ with perfect security with abort. Later, we will extend the proof technique and use it to show our negative result.

For the sake of contradiction, suppose Π is a 3-party protocol that computes $f_{\mathtt{AND}}$ with perfect security with abort. The parties are denoted by P_1, P_2, P_3. At a high level, we want to construct an execution where $x_2 = x_3 = 1$ and P_1 is corrupted. In this case, both P_2 and P_3 should take either 1 or $\perp$ as the output. We will construct a corrupted P_1 so that P_2 and P_3 both take 0 as output, thus leading to a contradiction. The same technique idea is also used in [PP25] for network-agnostic MPC with perfect security.

We start with an honest execution with input $x_2 = x_3 = 0$ and each party's randomness r_1, r_2, r_3, which we refer to as $\mathtt{exec}_0$. Now consider the perfect privacy against a corrupted P_2. When $x_2 = 0$, we always have $x_2 \cdot x_3 = 0$. Thus a semi-honest P_2 should not distinguish whether x_3 is 0 or 1. This implies that there should exist an honest execution with input $x_2 = 0, x_3 = 1$ and each party's randomness r_1', r_2, r_3', denoted by $\mathtt{exec}_1$, such that P_2's view is identical to that in $\mathtt{exec}_0$. Similarly, from the perfect privacy against a corrupted P_3, there exists an honest execution with input $x_2 = 1, x_3 = 0$ and each party's randomness r_1'', r_2'', r_3, denoted by $\mathtt{exec}_2$, such that P_3's view is identical to that in $\mathtt{exec}_0$. Note that both P_2's view and P_3's view contain the communication transcript between P_2 and P_3. This shows that in all three executions, the communication transcripts between P_2 and P_3 are identical.

Now consider the following execution where P_1 is corrupted: P_2 uses $(x_2 = 1, r_2'')$ as his input and randomness while P_3 uses $(x_3 = 1, r_3')$ as his input and randomness. The corrupted P_1 first emulates $\mathtt{exec}_1$ and $\mathtt{exec}_2$ and obtains the messages he would send to P_2 and P_3 in both executions. Now in the actual execution, P_1 sends the messages he would send in $\mathtt{exec}_2$ when interacting with P_2 and sends the messages he would send in $\mathtt{exec}_1$ when interacting with P_3. By induction, we can show that in each round, P_2 would see the same messages from P_1 and P_3 as those in $\mathtt{exec}_2$, and P_3 would see the same messages from P_1 and P_2 as those in $\mathtt{exec}_1$. Then P_2 would obtain the same output as that in $\mathtt{exec}_2$ and P_3 would obtain the same output as that in $\mathtt{exec}_1$, which are both 0. This leads to a contradiction.

Intuition of Our Communication Lower Bound. Now we attempt to extend the above proof to the 4-party setting for the computation task $f_{\mathtt{AND}}(\perp, \perp, x_2, x_3) \to (\perp, \perp, x_2 \cdot x_3, x_2 \cdot x_3)$ by adding a party P_0. Notice that such an attempt must fail due to the positive result from [BGW88]. However, we try to obtain some intuition from this attempt about the communication required by each party.

Following the same analysis, we may obtain $\mathtt{exec}_0, \mathtt{exec}_1, \mathtt{exec}_2$ that use inputs $(0,0),(0,1),(1,0)$ respectively such that P_2's view is identical in $\mathtt{exec}_0$ and $\mathtt{exec}_1$ while P_3's view is identical in $\mathtt{exec}_0$ and $\mathtt{exec}_2$, and then build the last execution $\mathtt{exec}_3$ as above (where P_1 is corrupted). The major difference here is that we have another honest party P_0. Recall that our strategy for P_1 is to use the same messages as those in $\mathtt{exec}_2$ when interacting with P_2 so that P_2 cannot distinguish $\mathtt{exec}_2$ and $\mathtt{exec}_3$. But the messages from P_0 to P_2 may be different in these two executions, which makes the same argument no longer work. On the other hand, *if the messages P_0 receives and sends in all three executions $\mathtt{exec}_0, \mathtt{exec}_1, \mathtt{exec}_2$ are identical*, we can show that the messages P_0 receives and sends in $\mathtt{exec}_3$ would be identical as well when P_1 follows our strategy.

Thus, given any fixed P_0's received and sent messages, there should be no $\mathtt{exec}_0, \mathtt{exec}_1, \mathtt{exec}_2$ that satisfy the above conditions, or otherwise we may construct $\mathtt{exec}_3$ that breaks the perfect correctness. To derive a communication lower bound for perfectly secure MPC, our idea is to classify all possible honest executions based on P_0's received and sent messages. Intuitively, if P_0's message size is small, then for some assignment of P_0's messages, there will be sufficiently many executions that allow us to find $\mathtt{exec}_0, \mathtt{exec}_1, \mathtt{exec}_2$.

Negative Result of Verifying Packed Beaver Triples. Instead of focusing on the AND function, our negative result targets for the verification of packed Beaver triples over packed Shamir sharings for two reasons.

- First, it is easy to extend to the general n-party setting and allows us to derive a communication lower bound for preparing packed Beaver triples over bivariate polynomials.
- Second, the packed Shamir sharings reveal no information about the secrets to any t parties. Thus, given that the input packed Beaver triples are correct, we may freely choose the secrets after fixing the shares of any t (corrupted) parties.

To be more concrete, we focus on verifying C degree-$(t+k-1)$ packed Beaver triples $([\boldsymbol{a}_i]_{t+k-1}, [\boldsymbol{b}_i]_{t+k-1}, [\boldsymbol{c}_i]_{t+k-1})$. Let P_0 be a single party and each of P_1, P_2, P_3 be a collection of t parties. In the following, we abuse the notation by using P_0's view to denote his input and all messages he receives and sends. I.e., we exclude the internal randomness in P_0's view. Suppose m is the size of P_0's view. Then there are at most $|\mathbb{F}|^m$ possible assignments for P_0's view. Our goal is to find $\mathtt{exec}_0, \mathtt{exec}_1, \mathtt{exec}_2$ such that P_0's view is identical in all three executions.

We start with an arbitrary honest execution $\mathtt{exec}_0$ with correct packed Beaver triples. Suppose the inputs are denoted by (I_0, I_1, I_2, I_3). Now for any other inputs (I_0', I_1', I_2, I_3') such that P_2's input remains unchanged and all parties' inputs form correct packed Beaver triples, by the perfect privacy against P_2, there exists an execution such that P_2's view remains the same. Then there are at least $|\mathbb{F}|^{2Ck}$ executions that keep P_2's view unchanged since for each packed Beaver triple, after fixing P_2's input, we may freely choose the secrets of the first two sharings, and then compute the secrets of the last sharing. All these executions are classified based on P_0's view. Then we may pick any $\mathtt{exec}_1$ that shares the same P_0's view as $\mathtt{exec}_0$ if exists. While the existence of $\mathtt{exec}_1$ is not guaranteed for any $\mathtt{exec}_0$, we note that if we pick a random $\mathtt{exec}_0$, then with probability $1 - |\mathbb{F}|^{m-2Ck}$, there exists such an execution $\mathtt{exec}_1$.

Similarly, when we pick a random $\mathtt{exec}_0$, with probability $1 - |\mathbb{F}|^{m-2Ck}$, there exists an execution $\mathtt{exec}_2$ such that P_3's view remains unchanged and $\mathtt{exec}_2$ shares the same P_0's view as $\mathtt{exec}_0$. By union bound, with probability $1 - 2|\mathbb{F}|^{m-2Ck}$, we may find $\mathtt{exec}_0, \mathtt{exec}_1, \mathtt{exec}_2$. Now we construct $\mathtt{exec}_3$ as:

- P_2 uses the same input and randomness as that in $\mathtt{exec}_2$.
- P_3 uses the same input and randomness as that in $\mathtt{exec}_1$.
- P_0 uses the same input and randomness as that in $\mathtt{exec}_0$.
- The corrupted P_1 first emulates $\mathtt{exec}_0, \mathtt{exec}_1, \mathtt{exec}_2$ and obtains the messages he would send to P_0, P_2, P_3. Then when interacting with P_0, P_1 uses messages in $\mathtt{exec}_0$; when interacting with P_2, P_1 uses messages in $\mathtt{exec}_2$; when interacting with P_3, P_1 uses messages in $\mathtt{exec}_1$.

We can show that (1) P_0's view remains the same as that in $\mathtt{exec}_0, \mathtt{exec}_1, \mathtt{exec}_2$, (2) P_2's view is identical to that in $\mathtt{exec}_2$, and (3) P_3's view is identical to that in $\mathtt{exec}_1$. Thus, all honest parties would output accept. To derive a contradiction, we want to show that the input packed Beaver triples are *incorrect* in $\mathtt{exec}_3$. First

note that for each party P_i, his input in $\texttt{exec}_3$ can be computed by the summation of his inputs in $\texttt{exec}_1$ and $\texttt{exec}_2$, and then subtracting his input in $\texttt{exec}_0$. Thus, if we use $\boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)}, \boldsymbol{c}^{(i)}$ to denote the secrets of the input packed Beaver triples in $\texttt{exec}_i$, we have

$$(\boldsymbol{a}^{(0)}, \boldsymbol{b}^{(0)}, \boldsymbol{c}^{(0)}) + (\boldsymbol{a}^{(3)}, \boldsymbol{b}^{(3)}, \boldsymbol{c}^{(3)}) = (\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}) + (\boldsymbol{a}^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{c}^{(2)}).$$

Now for the sake of contradiction, suppose $\boldsymbol{c}^{(i)} = \boldsymbol{a}^{(i)} * \boldsymbol{b}^{(i)}$ for all $i \in \{0, 1, 2, 3\}$. Then it implies that

$$(\boldsymbol{a}^{(0)} - \boldsymbol{a}^{(1)}) * (\boldsymbol{b}^{(0)} - \boldsymbol{b}^{(2)}) + (\boldsymbol{a}^{(0)} - \boldsymbol{a}^{(2)}) * (\boldsymbol{b}^{(0)} - \boldsymbol{b}^{(1)}) = \boldsymbol{0}.$$

Unfortunately we may not obtain any contradiction from this requirement. In our proof we make the following two changes:

- First when finding $\texttt{exec}_1$, we further restrict that $\boldsymbol{b}^{(1)} = \boldsymbol{b}^{(0)}$. I.e., we only change the secrets of the first sharing in $\texttt{exec}_1$. Similarly, when finding $\texttt{exec}_2$, we restrict that $\boldsymbol{a}^{(2)} = \boldsymbol{a}^{(0)}$. I.e., we only change the secrets of the second sharing in $\texttt{exec}_2$. In this way, the above requirement becomes $(\boldsymbol{a}^{(0)} - \boldsymbol{a}^{(1)}) * (\boldsymbol{b}^{(0)} - \boldsymbol{b}^{(2)}) = \boldsymbol{0}$ since the second term is always $\boldsymbol{0}$. However this is still not sufficient for us to derive a contradiction.
- For the second change, we want to make sure there is an index i such that $a_i^{(0)} \neq a_i^{(1)}$ and $b_i^{(0)} \neq b_i^{(2)}$. In this way, $(a_i^{(0)} - a_i^{(1)}) \cdot (b_i^{(0)} - b_i^{(2)}) \neq 0$, which contradicts with the above requirement. To this end, when finding $\texttt{exec}_1$, we sample a random $\texttt{exec}_0$ such that 1) there exists a set of executions in which the views of P_2 and P_0 remain unchanged, and 2) this set provides more than $Ck/2$ distinct indices from which we can choose an $\texttt{exec}_1$ such that $\boldsymbol{a}^{(1)}$ differs from $\boldsymbol{a}^{(0)}$ at the selected index. The same change is applied when finding $\texttt{exec}_2$.

With the above changes, we obtain a contradiction as long as $m < Ck/2$, which implies that the size of messages P_0 sends or receives should be at least $\Theta(Ck)$.

Recall that P_1 is a collection of t parties and each party in P_1 is equivalent to P_0. Thus we can show that any party in P_1 also needs to communicate at least $\Theta(Ck)$ field elements, leading to $\Theta(Ckn) = \Theta(Cn^2)$ communication lower bound for verifying packed Beaver triples.

Negative Result of Preparing Packed Beaver Triples. Finally, to show the communication lower bound for preparing packed Beaver triples, we construct a verification protocol for C packed Beaver triples with $O(Cn)$ communication in the preprocessing model assuming $O(C/n)$ correct packed Beaver triples over bivariate polynomials. At a high level, we first depack the prepared $O(C/n)$ packed Beaver triples over bivariate polynomials to $O(C)$ packed Beaver triples over standard packed Shamir sharings, and then apply the sacrifice technique to check each input packed Beaver triple. We refer the readers to Section 5.3 for more details. Since we have shown that $\Theta(Cn^2)$ elements of communication is necessary for the verification of packed Beaver triples, it implies that preparing each packed Beaver triple over bivariate polynomial requires at least $\Theta(n^3)$ elements of communication.

Moreover, we construct a perfectly secure MPC protocol for packed Beaver triples over bivariate polynomials that matches the communication lower bound, showing the tightness of our result. Finally, following a similar technique in [SY25], we prepare packed Beaver triples over bivariate polynomials assuming three-party Beaver triples, with amortized cost of $O(1)$ elements and $O(1)$ three-party Beaver triples per secret. This efficiently reduces the generation of packed Beaver triples to simpler correlations. Fore more details, we refer the reader to Section 7.

2.4 Organization of the Document

In Section 3, we present the necessary preliminaries, including security definition, packed secret sharing via bivariate polynomials, Byzantine agreement, and the party elimination framework. In Section 4, we first present the preprocessing functionality that generates packed Beaver triples over bivariate polynomials. We then construct an MPC protocol that computes general circuits with perfect security from this preprocessing

functionality. In Section 5, we first present, as a warm-up, an impossibility result regarding computing AND of two bits with perfect security *among* 3 *parties*. We then establish a communication lower bound for verifying packed Beaver triples, which in turn implies a communication lower bound for preparing packed Beaver triples over bivariate polynomials. Moreover, we prove a communication lower bound of computing an inner product function, together with a lower bound on the size of preprocessing data to compute the inner product function. Section 6 presents a realization of preparing packed Beaver triples over bivariate polynomials, matching the lower bound. Section 7 efficiently reduces the preprocessing functionality to simper correlations, namely three-party Beaver triples.

Supplementary material. Section A presents additional preliminaries, including properties of packed sharings over bivariate polynomials and the underlying building blocks. Section B complements the proofs of propositions used in Section 5. For ease of reference, Figures 1 and 2 summarize our perfectly secure MPC protocol in the preprocessing model and proof strategy used to establish the communication lower bound.

3 Preliminary

3.1 Security Definition

We consider a set of n parties $\{P_1, \ldots, P_n\}$ where each party can provide inputs, receive outputs, and participate in the computation. For every pair of parties, there exists a secure (private and authenticated) 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.

In this work, we focus on functions that can be represented as arithmetic circuits over a finite field $\mathbb{F}$ with $|\mathbb{F}| \geq 2n$, consisting of input, addition, multiplication, and output gates. We adopt the standard simulation-based definition of MPC [Can00]. An adversary is able to corrupt at most $t = (n-1)/3$ parties, provide inputs to corrupted parties, and receive all messages sent to the corrupted parties. Corrupted parties can deviate from the protocol arbitrarily. Our goal is to achieve perfect security.

3.2 Packed Secret Sharing via Bivariate Polynomials

A degree-(d,d) bivariate polynomial S over a finite field $\mathbb{F}$ is of the form $S(x,y) = \sum_{i=0}^{d}\sum_{j=0}^{d} a_{i,j} \cdot x^i y^j$, where $a_{i,j} \in \mathbb{F}$ for every $i,j \in \{0,1,\ldots,d\}$. Let k denote the packing parameter. Let $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_k$ be $n+k$ distinct field elements.

The sharing algorithm of a degree-(d,d) packed secret sharing via bivariate polynomials with a $k \times k$ matrix of k^2 secrets $\boldsymbol{X} = (x_{ij})_{ij} \in \mathbb{F}^{k\times k}$ is defined as:

> $\texttt{Share}(\boldsymbol{X})$: Sample a random degree-(d,d) bivariate polynomial $S(x,y)$ such that $S(\beta_i, \beta_j) = x_{ij}$ for every $i,j \in [k]$. Then i-th share is defined as two degree-d polynomials $(f_i(x), g_i(y))$, where $f_i(x) := S(x, \alpha_i)$ and $g_i(y) := S(\alpha_i, y)$. Output $\{f_i(x), g_i(y)\}_{i\in[n]}$.

We use $[\![\boldsymbol{X}]\!]_d$ to denote such a degree-(d,d) packed Shamir secret sharing over bivariate polynomials with secret $\boldsymbol{X} \in \mathbb{F}^{k\times k}$, which satisfies two properties:

- Linearly homomorphic. For all $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{F}^{k\times k}$, $[\![\boldsymbol{X}+\boldsymbol{Y}]\!]_d = [\![\boldsymbol{X}]\!]_d + [\![\boldsymbol{Y}]\!]_d$.
- Multiplicative. Let $*$ denote coordinate-wise multiplication. For all $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{F}^{k\times k}$ and $k-1 \leq d_1, d_2 \leq n-1$ such that $d_1 + d_2 \leq n-1$, $[\![\boldsymbol{X} * \boldsymbol{Y}]\!]_{d_1+d_2} = [\![\boldsymbol{X}]\!]_{d_1} \cdot [\![\boldsymbol{Y}]\!]_{d_2}$.

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

The properties of degree-(d,d) bivariate polynomials implies two useful properties of degree-(d,d) packed Shamir sharings: 1) the whole sharing is fully determined by $d+1$ parties' shares when pairwise consistency is assumed and 2) any set of shares of $d+1-k$ parties gives no information about the secret, which we refer the reader to Section A.1 for more details.

3.3 Byzantine Agreement

Our MPC protocol uses Byzantine agreement for both broadcast and consensus. Broadcast channels are implemented over authenticated and synchronous which allow a sender to distribute a message with a guarantee that all parties will receive the same value. Consensus allows parties, each holding an input x_i, to agree on a value x' such that $x' = x$ if all honest parties hold the same input $x_i = x$. With $t < n/3$, both consensus and broadcast can be achieved by a perfect byzantine agreement protocol communicating $O(n^2)$ bits [BGP92,CW92]. Alternatively, we can use an expected constant-round broadcast protocol in [AC24] which achieves communication complexity of $O(n \cdot L)$ bits plus expected $O(n^3 \cdot \log^2 n)$ bits for broadcasting a message of L bits.

We denote the communication complexity of a protocol by $\mathsf{P2P}(M) + N_1 \times \mathsf{BA}(L_1) + N_2 \times \mathsf{BC}(L_2)$, which means the protocol costs M *elements* in total over the point-to-point private channels, calls the Byzantine agreement N_1 times to reach an agreement on a message of L_1 *bits*, and calls the broadcast channel N_2 times with a message of L_2 *bits*.

3.4 Party Elimination Framework

To achieve perfect security, we follow the party elimination framework [HMP00]. In this framework, the computation task is uniformly divided into n segments which are evaluated sequentially. In each segment, parties first evaluate the segment and then verify the computation. After the verification, all active parties reach an agreement on whether the segment is successfully evaluated. In the case of success, all parties proceed to evaluate the next segment. In the case of failure, a new disputed pair containing at least one corrupted party is identified, and will be eliminated. The rest of the parties will re-evaluate the current segment. Since each failure would lead to a reduction on the number of active corrupted parties, the failure can occur at most $O(n)$ times.

We use $\mathcal{P}_{\texttt{active}}$ to denote the set of all active parties in the current segment. Let n' denote the number of active parties. Let $\mathcal{C}$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties. Let t' denote the number of active corrupted parties. In the beginning, all parties initialize $\mathcal{P}_{\texttt{active}} \leftarrow \{P_1, \ldots, P_n\}$. Each time a fault occurs, a disputes pair $E \subset \mathcal{P}_{\texttt{active}}$[4] with $E \cap \mathcal{C} \neq \emptyset$ is identified. All parties update $\mathcal{P}_{\texttt{active}} \leftarrow \mathcal{P}_{\texttt{active}} \setminus E$, $n' \leftarrow n' - |E|$, and $t' \leftarrow t' - |E \cap \mathcal{C}|$. Then we have the invariant that $n' - 2t' \geq n - 2t \geq t + 1$.

Importantly, as mentioned in Section 2.2, we require at least $t + 2k - 1$ honest parties are active so that a degree-$(t+2k-2, t+2k-2)$ packed Shamir sharing is fully determined by the shares of honest parties. Setting $k = n/12$, if the number of eliminated parties is fewer than $n/3$, all parties continue the party elimination framework. However, if a number of $n/3$ parties have been eliminated, the corruption ratio for the remaining parties is reduced to at most $1/4$. The parties will restart the computation by invoking an MPC protocol achieving perfect security against a fully malicious adversary controlling at most $1/4$ fraction of parties, as detailed in Section 4.6.

4 Perfectly Secure MPC in the Preprocessing Model

In this section, we construct a perfectly secure MPC for general circuits in the preprocessing model. We first present the preprocessing data of size $O(|C|)$ and then design an online protocol consuming the preprocessing data to compute a circuit, by extending the high-level idea introduced in Section 2.2.

Notations. $[\![\boldsymbol{U}]\!]_d$ denotes a degree-(d, d) packed Shamir sharing over bivariate polynomials with secret $\boldsymbol{U} \in \mathbb{F}^{k \times k}$. $[\![\boldsymbol{U}_{\ell,*}]\!]_d$ denotes the degree-d packed Shamir sharing with secret $\boldsymbol{U}_{\ell,*} \in \mathbb{F}^k$ being the ℓ-th row of the secret matrix $\boldsymbol{U}$. Accordingly, $[\![\boldsymbol{U}_{*,\ell}]\!]_d$ denotes the degree-d packed Shamir sharing with secret $\boldsymbol{U}_{*,\ell} \in \mathbb{F}^k$ being the ℓ-th column of the secret matrix $\boldsymbol{U}$.

[4] In our construction, sometimes a disputed pair only contains a corrupted party.

4.1 Preprocessing Data

Following [GPS22], our construction requires preprocessing data of packed Beaver triples. With packed Beaver triples, the computation of multiplication only involves degree reduction. As discussed in Section 2.2, to identify a disputed pair when faults occur, we use secret sharings via bivariate polynomials, which allows a party to verify its shares by checking common points held jointly with another party. The preprocessing functionality appears in $\mathcal{F}_{\text{packed-triple}}(N, k)$, which prepares N packed triples over bivariate polynomials, each packing k^2 secrets.

We emphasize that $\mathcal{F}_{\text{packed-triple}}$ achieves security with identifiable abort, in which a disputed pair is guaranteed to be identified when the functionality decides to abort.

Functionality 1: $\mathcal{F}_{\text{packed-triple}}(N, k)$

1. Receive the set of corrupted parties $\mathcal{C}$, the set of active parties $\mathcal{P}_{\texttt{active}}$, and public parameters N, k.
2. Receive from the adversary shares of corrupted parties $\{f^a_{\ell,i}(x),\ g^a_{\ell,i}(y),\ f^b_{\ell,i}(x),\ g^b_{\ell,i}(y),\ f^c_{\ell,i}(x),\ g^c_{\ell,i}(y)\}_{\ell\in[N],i\in\mathcal{C}\cap\mathcal{P}_{\texttt{active}}}$ such that
   - $f^z_{\ell,i}(x), g^z_{\ell,i}(y)$ are degree-$(t+k-1)$ polynomials for $\ell \in [N], i \in \mathcal{C}\cap\mathcal{P}_{\texttt{active}}, z \in \{a,b,c\}$,
   - $f^z_{\ell,i}(\alpha_j) = g^z_{\ell,j}(\alpha_i)$ for every $i, j \in \mathcal{C}\cap\mathcal{P}_{\texttt{active}}, z \in \{a,b,c\}$.
3. For $\ell \in [N]$, $\mathcal{F}_{\text{packed-triple}}$ randomly samples a packed triple over bivariate polynomials $([\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}, [\![\boldsymbol{C}_\ell]\!]_{t+k-1})$, where $\boldsymbol{C}_\ell = \boldsymbol{A}_\ell * \boldsymbol{B}_\ell$, $\boldsymbol{A}_\ell = (A_{\ell,ij})_{ij}$, $\boldsymbol{B}_\ell = (B_{\ell,ij})_{ij}$, and $\boldsymbol{C}_\ell = (C_{\ell,ij})_{ij}$, according to shares of corrupted parties. Concretely, $\mathcal{F}_{\text{packed-triple}}$ samples three degree-$(t+k-1, t+k-1)$ bivariate polynomials $S^a_\ell(x,y), S^b_\ell(x,y), S^c_\ell(x,y)$ such that
   - $S^a_\ell(\beta_i,\beta_j) = A_{\ell,ij}$, $S^b_\ell(\beta_i,\beta_j) = B_{\ell,ij}$, and $S^c_\ell(\beta_i,\beta_j) = C_{\ell,ij}$ for $i, j \in [k]$,
   - $S^z_\ell(x,\alpha_i) = f^z_{\ell,i}(x), S^z_\ell(\alpha_i,y) = g^z_{\ell,i}(y)$ for $i \in \mathcal{C}\cap\mathcal{P}_{\texttt{active}}, z \in \{a,b,c\}$.
4. $\mathcal{F}_{\text{packed-triple}}$ waits for an instruction from the adversary.
   - Upon receiving `accept`, for each $\ell \in [N], i \in \mathcal{P}_{\texttt{active}}, z \in \{a,b,c\}$, $\mathcal{F}_{\text{packed-triple}}$ computes $f^z_{\ell,i}(x) := S^z_\ell(x,\alpha_i), g^z_{\ell,i}(y) := S^z_\ell(\alpha_i,y)$ and sends to P_i $\{f^z_{\ell,i}(x), g^z_{\ell,i}(y)\}_{\ell\in[N],z\in\{a,b,c\}}$ as its shares of $\{[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}, [\![\boldsymbol{C}_\ell]\!]_{t+k-1}\}_{\ell\in[N]}$.
   - Upon receiving `reject`, $\mathcal{F}_{\text{packed-triple}}$ receives a new disputed pair $E \subset \mathcal{P}_{\texttt{active}}$ containing at least one corrupted party from the adversary, and outputs `abort` together with E to all active parties.

4.2 Identifiability from Bivariate Polynomials

In this section, we start with the basic identifiability of a packed Shamir sharing over bivariate polynomials $[\![\boldsymbol{X}]\!]_{t+k-1}$ and extend this to the identifiability of $[\![\boldsymbol{x}]\!]_{t+k-1}$, where $\boldsymbol{x} \in \mathbb{F}^k$ can be a linear combinations of different rows from different secret matrices. This is rather crucial to locate disputed pairs when computing a general circuit.

Identifiability of $[\![\boldsymbol{U}]\!]_d$. To verify whether a party P_r distributes consistent degree-(d,d) packed Shamir sharings and locate a disputed pair in case of inconsistency, we first present a procedure in Check-All$(P_r, [\![\boldsymbol{U}]\!]_d)$ that allows all parties to check whether a party P_r distributed a valid degree-(d,d) packed Shamir sharing. This ensures the establishment of pairwise consistency. Then we show a procedure in Check-One$(P_r, [\![\boldsymbol{U}]\!]_d)$ that allows a party P_r to check the consistency of the shares of $[\![\boldsymbol{U}]\!]_d$ with $d \le t+2k-2$ received from other parties and locate a disputed pair in case of inconsistency.

Procedure 1: CHECK-ALL($P_r, [\![\boldsymbol{U}]\!]_d$)

All active parties receive their shares of $[\![\boldsymbol{U}]\!]_d$ from an active party P_r and check whether the shares of all (honest) parties form a valid degree-(d,d) packed Shamir sharing. In case of failure, locate a new disputed pair.
Suppose $[\![\boldsymbol{U}]\!]_d$ is shared via a degree-(d,d) bivariate polynomial $S(x,y)$, and P_r distributes $\big(f_i(x), g_i(y)\big) := \big(S(x,\alpha_i), S(\alpha_i,y)\big)$ to each active P_i.

1. For every $i \in \mathcal{P}_{\texttt{active}}$, party P_i checks whether f_i, g_i are two degree-d polynomials and $f_i(\alpha_i) = g_i(\alpha_i)$.
   - If this check fails, P_i complains by broadcasting $(\texttt{Incorrect}, P_i)$. All parties take the disputed pair $\{P_r, P_i\}$ as output.
2. For $(i,j) \in \mathcal{P}_{\texttt{active}} \times \mathcal{P}_{\texttt{active}}$ such that $i \neq j$, P_i sends $(f_i(\alpha_j), g_i(\alpha_j))$ to P_j. P_j checks whether $(f_i(\alpha_j), g_i(\alpha_j)) = (g_j(\alpha_i), f_j(\alpha_i))$.
   - If this check fails, P_j complains by broadcasting $(\texttt{Inconsistent}, P_j, P_i, (f_j(\alpha_i), g_j(\alpha_i)), (f_i(\alpha_j), g_i(\alpha_j)))$.
     - If P_j broadcasts values such that $f_j(\alpha_i) = g_i(\alpha_j)$ and $g_j(\alpha_i) = f_i(\alpha_j)$, all parties output the corrupted party $\{P_j\}$.

     Then P_r and P_i also broadcast their values. Two adjacent parties broadcasting different values is regarded as a disputed pair.
3. If all checks pass, each active party P_i takes $f_i(x), g_i(y)$ as output.

Procedure 2: CHECK-ONE($P_r, [\![\boldsymbol{U}]\!]_d$)

P_r receives from all active parties their shares of $[\![\boldsymbol{U}]\!]_d$ and checks whether the shares received form a valid degree-(d,d) packed Shamir sharing. In case of failure, a new disputed pair is figured out.

1. Suppose $[\![\boldsymbol{U}]\!]_d$ is shared by a degree-(d,d) bivariate polynomial $S(x,y)$. Then each active party P_i sent shares $f_i(x) := S(x,\alpha_i)$ and $g_i(y) := S(\alpha_i,y)$ to P_r.
2. P_r checks whether $f_i(x)$ and $g_i(y)$ are degree-d polynomials such that $f_i(\alpha_i) = g_i(\alpha_i)$.
   - If this check fails, P_r complains by broadcasting $(P_i, \texttt{Corrupt})$. All parties take as output the disputed pair $\{P_r, P_i\}$.
3. For every $(i,j) \in \mathcal{P}_{\texttt{active}} \times \mathcal{P}_{\texttt{active}}$ such that $i \neq j$, P_r checks whether $(f_i(\alpha_j), g_i(\alpha_j)) = (g_j(\alpha_i), f_j(\alpha_i))$.
   - If this check fails, P_r complains by broadcasting $(\texttt{Inconsistent}, (f_i(\alpha_j), g_i(\alpha_j)), (g_j(\alpha_i), f_j(\alpha_i)), P_i, P_j)$.
     - If P_r broadcasts values such that $\big(f_i(\alpha_j), g_i(\alpha_j)\big) = \big(f_j(\alpha_i), g_j(\alpha_i)\big)$, all parties output the corrupted party $\{P_r\}$.

     Then both P_i and P_j broadcast their values they had sent to P_r. Two adjacent parties broadcasting different values is regarded as a disputed pair.
4. If all checks pass, each active party P_i takes $f_i(x), g_i(y)$ as output.

Identifiability of $[\![\boldsymbol{u}]\!]_d$. Based on procedure CHECK-ONE, we now extend the identifiability to a packed Shamir sharing $[\![\boldsymbol{u}]\!]_d$ with $d \leq t + 2k - 2$. As mentioned in Section 2.2, the difficulty originates from the fact that the secrets of $\boldsymbol{u}$ may come from different rows of secret matrices shared via packed Shamir sharings over bivariate polynomials. To leverage pairwise consistency, we decompose the packed sharing $[\![\boldsymbol{u}]\!]_d$ into k packed sharings $[\![\boldsymbol{U}^{(1)}_{1,*}]\!]_d, \ldots, [\![\boldsymbol{U}^{(k)}_{k,*}]\!]_d$ with each $\boldsymbol{U}^{(\ell)}_{\ell,*}$ representing the ℓ-th row of the secret matrix $\boldsymbol{U}^{(\ell)}$, i.e.,

$$[\![\boldsymbol{u}]\!]_d = [\![\boldsymbol{U}^{(1)}_{1,*}]\!]_d + \cdots + [\![\boldsymbol{U}^{(k)}_{k,*}]\!]_d.$$

Since $[\![\boldsymbol{u}]\!]_d$ is inconsistent, then at least one packed sharing $[\![\boldsymbol{U}^{(\ell)}_{\ell,*}]\!]_d$ and hence $[\![\boldsymbol{U}^{(\ell)}]\!]_d$ are inconsistent for some $\ell \in [k]$, which implies P_r can locate a disputed pair by investigating the consistency of the sharings $[\![\boldsymbol{U}^{(\ell)}]\!]_d$ for each $\ell \in [k]$. However, revealing the whole sharing $[\![\boldsymbol{U}^{(\ell)}]\!]_d$ to P_r could break the secrecy. To resolve this issue, we resort to the sacrifice technique. In particular, the parties first prepare random packed Shamir sharings $[\![\boldsymbol{W}^{(1)}]\!]_d, \ldots, [\![\boldsymbol{W}^{(k)}]\!]_d$ and reveal the sharing $[\![\boldsymbol{w}]\!]_d \leftarrow [\![\boldsymbol{W}^{(1)}_{1,*}]\!]_d + \cdots + [\![\boldsymbol{W}^{(k)}_{k,*}]\!]_d$ to P_r, which is the sum of k random packed sharings. Since $[\![\boldsymbol{u}]\!]_d$ is inconsistent, at least one of $[\![\boldsymbol{w}]\!]_d$ and $[\![\boldsymbol{u}]\!]_d + [\![\boldsymbol{w}]\!]_d$ must be inconsistent. Suppose $[\![\boldsymbol{w}]\!]_d$ is inconsistent; the other case is analogous. This in turn implies at least one of the sharings $[\![\boldsymbol{W}^{(1)}]\!]_d, \ldots, [\![\boldsymbol{W}^{(k)}]\!]_d$ is inconsistent. To catch a disputed pair, P_r checks the consistency for each sharing $[\![\boldsymbol{W}^{(\ell)}]\!]_d$ by invoking Check-One. This preserves the secrecy as revealing one of $[\![\boldsymbol{W}^{(\ell)}]\!]_d$ and $[\![\boldsymbol{U}^{(\ell)} + \boldsymbol{W}^{(\ell)}]\!]_d$ does not leak any information regarding $\boldsymbol{U}^{(\ell)}$.

We present the protocol in Decompose-Sharing to locate a disputed pair in case a packed sharing $[\![\boldsymbol{u}]\!]_d$ is inconsistent. We then extend this to show a protocol in Analyze-Sharing that locates a disputed pair in case a packed sharing $[\boldsymbol{u}]_{t+2k-2}$ is inconsistent, where the packed sharing can be written of the form $[\boldsymbol{u}]_{t+2k-2} = [\boldsymbol{d}_0]_{2k-2} + [\![\boldsymbol{u}_0]\!]_{t+2k-2} + \sum_{\ell=1}^{s} [\boldsymbol{d}_\ell]_{k-1} \cdot [\![\boldsymbol{u}_\ell]\!]_{t+k-1}$, by further leveraging error-correcting property of degree-$(2k-2)$ Shamir sharings.

The cost of Analyze-Sharing is $\mathsf{P2P}(O(k \cdot n^2)) + O(1) \times \mathsf{BC}(O(\log n + \log |\mathbb{F}|))$. Notice that Analyze-Sharing is only invoked when faults occur.

Protocol 1: Decompose-Sharing$(P_r, [\![\boldsymbol{u}]\!]_d)$

P_r has received the sharing $[\![\boldsymbol{u}]\!]_d$ from all other active parties, which is an *inconsistent* degree-d packed Shamir sharing. In the preprocessing phase, all parties have prepared random sharings of the form $([\![\boldsymbol{W}^{(1)}]\!]_d, \ldots, [\![\boldsymbol{W}^{(k)}]\!]_d)$. All parties locate a new disputed pair as follows.

1. Suppose $[\![\boldsymbol{u}]\!]_d$ can be decomposed as the sum of $[\![\boldsymbol{U}^{(1)}_{1,*}]\!]_d, [\![\boldsymbol{U}^{(2)}_{2,*}]\!]_d, \ldots, [\![\boldsymbol{U}^{(k)}_{k,*}]\!]_d$, i.e., $[\![\boldsymbol{u}]\!]_d = [\![\boldsymbol{U}^{(1)}_{1,*}]\!]_d + [\![\boldsymbol{U}^{(2)}_{2,*}]\!]_d + \cdots + [\![\boldsymbol{U}^{(k)}_{k,*}]\!]_d$, where $[\![\boldsymbol{U}^{(\ell)}_{\ell,*}]\!]_d$ is the ℓ-th row of a degree-(d,d) packed Shamir sharing $[\![\boldsymbol{U}^{(\ell)}]\!]_d$ for each $\ell \in [k]$.
2. All parties pick k random degree-(d,d) packed Shamir sharings $[\![\boldsymbol{W}^{(1)}]\!]_d, [\![\boldsymbol{W}^{(2)}]\!]_d, \ldots, [\![\boldsymbol{W}^{(k)}]\!]_d$ prepared in the preprocessing phase.
3. All parties locally compute $[\![\boldsymbol{w}]\!]_d \leftarrow [\![\boldsymbol{W}^{(1)}_{1,*}]\!]_d + [\![\boldsymbol{W}^{(2)}_{2,*}]\!]_d + \cdots + [\![\boldsymbol{W}^{(k)}_{k,*}]\!]_d$ and send their shares of $[\![\boldsymbol{w}]\!]_d$ to P_r.
4. P_r checks whether the received sharing $[\![\boldsymbol{w}]\!]_d$ is a valid degree-(d) packed Shamir sharing.
   - If not, P_r broadcasts `Inconsistent`. This means at least one of $\{[\![\boldsymbol{W}^{(\ell)}_{\ell,*}]\!]_d\}_{\ell \in [k]}$ is inconsistent. All parties locate a disputed pair by invoking Check-One$(P_r, [\![\boldsymbol{W}^{(\ell)}]\!]_d)$ for each $\ell \in [k]$.
     - At least one invocation will result in a disputed pair E. All parties take E as output.
   - Otherwise, P_r broadcasts `Consistent`. This means $[\![\boldsymbol{u} + \boldsymbol{w}]\!]_d$ must be inconsistent and at least one of $\{[\![\boldsymbol{U}^{(\ell)}_{\ell,*} + \boldsymbol{W}^{(\ell)}_{\ell,*}]\!]_d\}_{\ell \in [k]}$ is inconsistent. Then all parties locate a disputed pair by invoking Check-One$(P_r, [\![\boldsymbol{U}^{(\ell)} + \boldsymbol{W}^{(\ell)}]\!]_d)$ for each $\ell \in [k]$.
     - At least one invocation will result in a disputed pair E. All parties take E as output.

Protocol 2: Analyze-Sharing$(P_r, [\boldsymbol{u}]_{t+2k-2})$

P_r receives the sharing $[\boldsymbol{u}]_{t+2k-2}$ from all other active parties, which is an *inconsistent* degree-$(t + 2k - 2)$ packed Shamir sharing. In the preprocessing phase, all parties have prepared

- random sharings of the form $([\![\boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{v}_k]\!]_{t+k-1})$,
- random sharings of the form $([\![\boldsymbol{U}^{(1)}]\!]_{t+k-1}, \ldots, [\![\boldsymbol{U}^{(k)}]\!]_{t+k-1})$,
- and random sharings of the form $([\![\boldsymbol{W}^{(1)}]\!]_{t+2k-2}, \ldots, [\![\boldsymbol{W}^{(k)}]\!]_{t+2k-2})$.

P_r and other parties aim to locate a disputed pair as follows.

1. Suppose $[\boldsymbol{u}]_{t+2k-2} = [\boldsymbol{d}_0]_{2k-2} + [\![\boldsymbol{u}_0]\!]_{t+2k-2} + \sum_{\ell=1}^{s} [\boldsymbol{d}_\ell]_{k-1} \cdot [\![\boldsymbol{u}_\ell]\!]_{t+k-1}$, where $[\boldsymbol{d}_0]_{2k-2}$ is a degree-$(2k-2)$ packed Shamir sharing, $[\boldsymbol{d}_1]_{k-1}, \ldots, [\boldsymbol{d}_s]_{k-1}$ are degree-$(k-1)$ packed Shamir sharings, and $s \leq \max\{3, k\}$.
2. All parties pick $s+1$ random packed Shamir sharings $[\![\boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{v}_s]\!]_{t+k-1}$ prepared in the preprocessing phase.
3. All parties locally compute $[\boldsymbol{v}]_{t+2k-2} \leftarrow [\![\boldsymbol{v}_0]\!]_{t+2k-2} + \sum_{\ell=1}^{s} [\boldsymbol{d}_\ell]_{k-1} \cdot [\![\boldsymbol{v}_\ell]\!]_{t+k-1}$ and send their shares of $[\boldsymbol{v}]_{t+2k-2}$ to P_r. Additionally, all parties send their shares of $[\boldsymbol{d}_0]_{2k-2}, [\boldsymbol{d}_1]_{k-1}, \ldots, [\boldsymbol{d}_s]_{k-1}$ to P_r.
4. On receiving shares from other parties, P_r first uses the error correcting property of degree-$(2k-2)$ packed Shamir sharings to recover the whole sharings of $[\boldsymbol{d}_0]_{2k-2}, [\boldsymbol{d}_1]_{k-1}, \ldots, [\boldsymbol{d}_s]_{k-1}$. Then P_r checks whether the receiving $[\boldsymbol{v}]_{t+2k-2}$ forms a valid degree-$(t+2k-2)$ packed Shamir sharing.
   - If this check fails, P_r broadcasts `Inconsistent`, meaning that the computation of $[\boldsymbol{v}]_{t+2k-2}$ is incorrect. P_r identifies a disputed pair as follows.
     (a) All parties send their shares of $[\![\boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{v}_s]\!]_{t+k-1}$ to P_r.
     (b) P_r checks whether each active party P_i correctly computes its share of $[\boldsymbol{v}]_{t+2k-2}$ from their shares of $[\![\boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{v}_s]\!]_{t+k-1}$, i.e., $[\boldsymbol{v}]_{t+2k-2} = [\![\boldsymbol{v}_0]\!]_{t+2k-2} + \sum_{\ell=1}^{s} [\boldsymbol{d}_\ell]_{k-1} \cdot [\![\boldsymbol{v}_\ell]\!]_{t+k-1}$.
        - If not, P_r broadcasts $(P_i, \texttt{Incorrect})$ to indicate the computation by P_i is incorrect. All parties take $E \leftarrow \{P_r, P_i\}$ as output.
     (c) Then at least one of the sharings $[\![\boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{v}_s]\!]_{t+k-1}$ is inconsistent, say $[\![\boldsymbol{v}_j]\!]_d$. P_r broadcasts $(j, \texttt{Inconsistent})$. Then P_r and other parties invoke Decompose-Sharing$(P_r, [\![\boldsymbol{v}_j]\!]_d)$ and take the disputed pair E as output.
   - If this check passes, P_r broadcasts `Consistent`, meaning the sharing $[\boldsymbol{u}+\boldsymbol{v}]_{t+2k-2} \leftarrow [\boldsymbol{u}]_{t+2k-2} + [\boldsymbol{v}]_{t+2k-2}$ is inconsistent. P_r identifies a disputed pair as follows.
     (a) All parties send their shares of $[\![\boldsymbol{u}_0+\boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{u}_1+\boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{u}_s+\boldsymbol{v}_s]\!]_{t+k-1}$ to P_r.
     (b) P_r checks whether each active party P_i correctly computes its share of $[\boldsymbol{u}+\boldsymbol{v}]_{t+2k-2}$ from their shares of $[\![\boldsymbol{u}_0+\boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{u}_1+\boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{u}_s+\boldsymbol{v}_s]\!]_{t+k-1}$, i.e., $[\boldsymbol{u}+\boldsymbol{v}]_{t+2k-2} = [\boldsymbol{d}_0]_{2k-2} + [\![\boldsymbol{u}_0+\boldsymbol{v}_0]\!]_{t+2k-2} + \sum_{\ell=1}^{s} [\boldsymbol{d}_\ell]_{k-1} \cdot [\![\boldsymbol{u}_\ell+\boldsymbol{v}_\ell]\!]_{t+k-1}$.
        - If not, P_r broadcasts $(P_i, \texttt{Incorrect})$. All parties take $E \leftarrow \{P_r, P_i\}$ as output.
     (c) Then at least one of the sharings $[\![\boldsymbol{u}_0+\boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{u}_1+\boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{u}_s+\boldsymbol{v}_s]\!]_{t+k-1}$ is inconsistent, say $[\![\boldsymbol{u}_j+\boldsymbol{v}_j]\!]_d$. P_r broadcasts $(j, \texttt{Inconsistent})$. Then P_r and other parties invoke Decompose-Sharing$(P_r, [\![\boldsymbol{u}_j+\boldsymbol{v}_j]\!]_d)$ and take the disputed pair E as output.

4.3 Computation of Multiplication Gates

As outlined in Section 2.2, we rely on party elimination framework to achieve perfect security. At a high level, the whole circuit is uniformly divided into n segments, which will be computed sequentially. Parties first evaluate the segment and then verify the computation. If the verification fails, parties identify a disputed pair, eliminate them, and re-compute the current segment. They repeat this procedure until success and move to the next segment. In each segment, the computation mainly contains computing multiplication gates and handling network routing. We demonstrate how to evaluate multiplication gates and verify the computation of multiplication gates in the following, and defer the part for handling network routing in Section 4.4.

Evaluation of Multiplication Gates. For a group of k multiplication gates, all parties follow Beaver multiplication protocol by consuming a packed Beaver triple $([\![\boldsymbol{a}]\!]_{t+k-1}, [\![\boldsymbol{b}]\!]_{t+k-1}, [\![\boldsymbol{c}]\!]_{t+k-1})$. To perform degree reduction, a double sharing $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\boldsymbol{r}]\!]_{t+2k-2})$ is required, which can be picked from a double sharing over bivariate polynomials $([\![\boldsymbol{R}]\!]_{t+k-1}, [\![\boldsymbol{R}]\!]_{t+2k-2})$. Moreover, to prevent so-called double dipping attack [GBO+23,DEN24], first pointed out by [GLS19], due to cross-layer computation without verification, we follow a solution proposed in [SY24]. In this approach, before the computation, all parties first prepare

degree-$(n'-1)$ packed Shamir sharings of the form $[\boldsymbol{o}]_{n'-1}$ with secret $\boldsymbol{o} = \boldsymbol{0} \in \mathbb{F}^k$, then add these high-degree packed Shamir sharings as masks to protect privacy of honest parties against a corrupted P_{king}. After verifying P_{king} always distributed valid degree-$(k-1)$ packed Shamir sharings, all parties reveal the masks to let P_{king} verify the consistency of the packed sharings it received.

We present the protocol for evaluating multiplication gates in Mult. The communication cost of Mult is $\mathsf{P2P}(O(n)) + O(1) \times \mathsf{BC}(O(\log n + \log |\mathbb{F}|))$.

Protocol 3: Mult

All parties hold $[\boldsymbol{x}]_{t+k-1}$ and $[\boldsymbol{y}]_{t+k-1}$, and aim to compute $[\boldsymbol{x} * \boldsymbol{y}]_{t+k-1}$. In preprocessing phase, all parties have prepared

- a packed Beaver triple $(\llbracket \boldsymbol{a} \rrbracket_{t+k-1}, \llbracket \boldsymbol{b} \rrbracket_{t+k-1}, \llbracket \boldsymbol{c} \rrbracket_{t+k-1})$ with $\boldsymbol{a} * \boldsymbol{b} = \boldsymbol{c}$
- and a pair of random packed Shamir sharings $(\llbracket \boldsymbol{r} \rrbracket_{t+k-1}, \llbracket \boldsymbol{r} \rrbracket_{t+2k-2})$.

All parties hold a degree-$(n'-1)$ packed sharing $[\boldsymbol{o}]_{n'-1}$ with $\boldsymbol{o} = \boldsymbol{0} \in \mathbb{F}^k$.

1. All parties locally compute $[\boldsymbol{x} + \boldsymbol{a}]_{t+k-1} \leftarrow [\boldsymbol{x}]_{t+k-1} + \llbracket \boldsymbol{a} \rrbracket_{t+k-1}$ and $[\boldsymbol{y} + \boldsymbol{b}]_{t+k-1} \leftarrow [\boldsymbol{y}]_{t+k-1} + \llbracket \boldsymbol{b} \rrbracket_{t+k-1}$, and send their shares to P_{king}.
2. P_{king} checks whether each of the sharings $[\boldsymbol{x} + \boldsymbol{a}]_{t+k-1}, [\boldsymbol{y} + \boldsymbol{b}]_{t+k-1}$ is a valid degree-$(t+k-1)$ packed Shamir sharing.
   - If not, say $[\boldsymbol{u}]_{t+k-1}$ is inconsistent, where $\boldsymbol{u} \in \{\boldsymbol{x} + \boldsymbol{a}, \boldsymbol{y} + \boldsymbol{b}\}$, P_{king} broadcasts $(i, \texttt{Inconsistent})$ to indicate the i-th sharing $[\boldsymbol{u}]_{t+k-1}$ is inconsistent for some $i \in \{1, 2\}$. Then all parties invoke Analyze-Sharing$(P_{\text{king}}, [\boldsymbol{u}]_{t+k-1})$ to locate a disputed pair E.
3. P_{king} reconstructs $\boldsymbol{x} + \boldsymbol{a}$, $\boldsymbol{y} + \boldsymbol{b}$ and distributes $[\boldsymbol{x} + \boldsymbol{a}]_{k-1}, [\boldsymbol{y} + \boldsymbol{b}]_{k-1}$.
4. All parties locally compute

$$\begin{aligned}[\boldsymbol{z} + \boldsymbol{r}]_{t+2k-2} \leftarrow & [\boldsymbol{x} + \boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y} + \boldsymbol{b}]_{k-1} - [\boldsymbol{x} + \boldsymbol{a}]_{k-1} \cdot \llbracket \boldsymbol{b} \rrbracket_{t+k-1} \\ & - \llbracket \boldsymbol{a} \rrbracket_{t+k-1} \cdot [\boldsymbol{y} + \boldsymbol{b}]_{k-1} + \llbracket \boldsymbol{c} \rrbracket_{t+k-1} + \llbracket \boldsymbol{r} \rrbracket_{t+2k-2}.\end{aligned}$$

   All parties then locally compute $[\boldsymbol{z} + \boldsymbol{r}]_{n'-1} \leftarrow [\boldsymbol{z} + \boldsymbol{r}]_{t+2k-2} + [\boldsymbol{o}]_{n'-1}$, and send their shares to P_{king}.
5. P_{king} reconstructs $\boldsymbol{z} + \boldsymbol{r}$ and distributes $[\boldsymbol{z} + \boldsymbol{r}]_{k-1}$.
6. All parties locally compute $[\boldsymbol{z}]_{t+k-1} \leftarrow [\boldsymbol{z} + \boldsymbol{r}]_{k-1} - \llbracket \boldsymbol{r} \rrbracket_{t+k-1}$.

Verification of Computing Multiplication Gates. We follow the approach in [SY24] to verify the computation of Mult. Concretely, all parties first verify whether P_{king} distributes consistent degree-$(k-1)$ packed Shamir sharings in Step 3 and Step 5. After making sure P_{king} indeed distributed consistent degree-$(k-1)$ packed Shamir sharings and thus did not mount a double dipping attack, it is safe to reveal the degree-$(t+2k-2)$ packed sharing $[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$ to P_{king}, where the whole sharing is fully determined by shares of honest parties. In case of inconsistency, P_{king} invokes Analyze-Sharing to identify a disputed pair.

As P_{king} has already checked the consistency of the received packed Shamir sharings, it can be effectively viewed as that P_{king} distributes a pair of sharings $([\boldsymbol{u}]_{t+2k-2}, [\boldsymbol{u}]_{k-1})$ to all parties. We make use of the batched verification technique in [BH08] to verify whether P_{king} distributes degree-$(k-1)$ packed Shamir sharings with correct secrets.

The protocol for verifying the computation of multiplication gates appears in Vrfy-Mult. The cost of Vrfy-Mult to verify N invocations of Mult is $\mathsf{P2P}(O(N \cdot n + n^2)) + O(1) \times \mathsf{BA}(1) + O(1) \times \mathsf{BC}(O(\log N + \log n + \log |\mathbb{F}|))$.

Protocol 4: VRFY-MULT(P_{king}, N)

All parties aim to verify the correctness of N invocations of MULT which computes the multiplication of $[\boldsymbol{x}]_{t+k-1}$ and $[\boldsymbol{y}]_{t+k-1}$.

1. All parties invoke $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_9, 3N)$ to check whether P_{king} distributes valid degree-$(k-1)$ packed Shamir sharings $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}, [\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ in each invocation of MULT.
   - In case of failure, all parties take the disputed pair E as output.
2. For each invocation of MULT, all parties send their shares of $[\boldsymbol{o}]_{n'-1}$ to P_{king} used in Step 4 of MULT. P_{king} checks whether the received shares form a valid degree-$(n'-1)$ packed sharing with secret $\mathbf{0} \in \mathbb{F}^k$.
   - If not, P_{king} broadcasts `Inconsistent`. All parties send their transcripts produced when invoking ZERO-SHARING to P_{king}.
     - P_{king} either broadcasts $(P_i, \texttt{Corrupt})$ to indicate P_i does not honestly follow the protocol,
     - or broadcasts (ℓ, v, v', P_i, P_j) to indicate P_i and P_j disagree on the ℓ-th message exchanged between them, where P_i should have sent v to P_j while P_j claims to have received $v' \neq v$. Then P_i, P_j broadcast their messages sent to P_{king}.

     In the first case, all parties take $E \leftarrow \{P_{\text{king}}, P_i\}$ as the disputed pair. In the second case, all parties figure out two parties $E \leftarrow \{P_{i'}, P_{j'}\}$ broadcasting different values and output E.
   - If this check passes, P_{king} subtracts the degree-$(n'-1)$ packed Shamir sharings of $\mathbf{0}$ from the sharings it received in MULT. I.e., $[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2} \leftarrow [\boldsymbol{z}+\boldsymbol{r}]_{n'-1} - [\boldsymbol{o}]_{n'-1}$.
3. For each invocation of MULT, P_{king} checks whether the computed $[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$ is a valid degree-$(t+2k-2)$ packed Shamir sharing.
   - If not, P_{king} broadcasts `Inconsistent`. All parties invoke ANALYZE-SHARING($P_{\text{king}}, [\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$) to locate a disputed pair.
4. All parties invoke $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_{10}, 3N)$ with their shares of $([\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{x}+\boldsymbol{a}]_{t+k-1})$, $([\boldsymbol{y}+\boldsymbol{b}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1})$ and $([\boldsymbol{z}+\boldsymbol{r}]_{k-1}, [\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2})$ used in all invocations to check whether P_{king} distributes degree-$(k-1)$ packed Shamir sharings with correct secrets in each invocation of MULT.

4.4 Towards General Circuits

Handling Network Routing. When computing general circuits, the parties need to compute input packed sharings for each group of gates. However, the secrets may not be in a correct order or even lie in different packed sharings, which is referred to as *network routing* in the works [GPS21,GPS22]. The authors in [GPS21,GPS22] show that the problem of network routing can be reduced to two computation tasks, performing sharing transformation and collecting secrets. The task of sharing transformation requires all parties who hold their shares of $[\boldsymbol{x}]_{t+k-1}$ to compute $[\boldsymbol{L}(\boldsymbol{x})]_{t+k-1}$, where $\boldsymbol{L} : \mathbb{F}^k \to \mathbb{F}^k$ is a linear transformation. In the task of secret collection, all parties compute a packed sharing $[\boldsymbol{y}]_{t+k-1}$ from k packed sharings $[\boldsymbol{x}_1]_{t+k-1}, \ldots, [\boldsymbol{x}_k]_{t+k-1}$, where $\boldsymbol{y} = (y_1, \ldots, y_k)$ and $y_\ell = x_{\ell,\ell}$ for $\ell \in [k]$.

As for the first task, to act the linear transformation $\boldsymbol{L}$ on the secret $\boldsymbol{x}$, the parties first prepare a random linear mask of the form $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\boldsymbol{L}(\boldsymbol{r})]\!]_{t+k-1})$ and follow the *mask-and-open* recipe to compute $[\boldsymbol{L}(\boldsymbol{x})]_{t+k-1}$ from $[\boldsymbol{x}]_{t+k-1}$. To prepare random linear mask for k different linear transformations $\boldsymbol{L}_1, \ldots, \boldsymbol{L}_k$, we make use of an idea in [SY24] that if all k secrets of $\boldsymbol{x}$ are stored at the same position, then parties can locally apply the linear transformation. We observe that packed Shamir sharings over bivariate polynomials naturally satisfy this demand. Specifically, if all parties hold the sharing $[\![\boldsymbol{R}]\!]_{t+k-1}$, they can compute packed Shamir sharings for all row vectors, $[\![\boldsymbol{r}_{1,*}]\!]_{t+k-1}, \ldots, [\![\boldsymbol{r}_{k,*}]\!]_{t+k-1}$, from their column shares. Accordingly, they can compute packed Shamir sharings for all column vectors, $[\![\boldsymbol{r}_{*,1}]\!]_{t+k-1}, \ldots, [\![\boldsymbol{r}_{*,k}]\!]_{t+k-1}$, from their row shares. Suppose they want to perform the linear transformation $\boldsymbol{L}_\ell$ on the ℓ-th row vector $\boldsymbol{r}_{\ell,*}$. Notice that all secrets of $\boldsymbol{r}_{\ell,*}$ are stored at the ℓ-th position in the sharings of column vectors $[\![\boldsymbol{r}_{*,1}]\!]_{t+k-1}, \ldots, [\![\boldsymbol{r}_{*,k}]\!]_{t+k-1}$, which means the parties can locally compute linear combinations of these secrets. In doing this, the parties can obtain

their row shares of the sharing $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})]\!]_{t+k-1}$, where the ℓ-th row of $\vec{\boldsymbol{L}}(\boldsymbol{R}) \in \mathbb{F}^{k\times k}$ is $\boldsymbol{L}_\ell(\boldsymbol{r}_\ell)$ for $\ell \in [k]$. Each party further computes its column shares by having other parties reconstructing the column polynomial to it. As for the second task, parties use unit vectors to select corresponding secrets, which is followed with a degree reduction.

We present the protocols for preparing random linear masks for sharing transformation, performing sharing transformation, and selecting secrets in Rand-Linear-Mask, Sharing-Trans and select, respectively. The communication cost of Rand-Linear-Mask to prepare N random linear masks is $\mathsf{P2P}(O(\frac{N\cdot n^2}{k} + n^2)) + O(1)\times\mathsf{BC}(O(\log n + \log|\mathbb{F}|))$ plus one invocation of $\mathcal{F}_{\text{batched-vrfy}}(\Sigma_8, N/k)$, amounting to $\mathsf{P2P}(O(\frac{N\cdot n^2}{k} + n^3)) + O(1)\times\mathsf{BA}(1) + O(1)\times\mathsf{BC}(O(\log N + \log n + \log|\mathbb{F}|))$. The cost of Sharing-Trans is $\mathsf{P2P}(O(n)) + O(1)\times\mathsf{BC}(O(\log n + \log|\mathbb{F}|))$. The cost of select is $\mathsf{P2P}(O(n)) + O(1)\times\mathsf{BC}(O(\log n + \log|\mathbb{F}|))$.

Protocol 5: Rand-Linear-Mask(N)

All parties aim to prepare N pairs of random linear masks, each in the form of $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\boldsymbol{L}(\boldsymbol{r})]\!]_{t+k-1})$, where $\boldsymbol{L}: \mathbb{F}^k \to \mathbb{F}^k$ is a linear transformation, which can be represented by a $k\times k$ matrix $(L_{ij})_{ij}$. In the preprocessing phase, all parties have prepared

- $\frac{t+3k-1}{k} \cdot \lceil\frac{N}{k}\rceil$ random sharings of the form $[\![\boldsymbol{R}]\!]_{t+k-1}$ and
- $\frac{t+2k-1}{k} \cdot \lceil\frac{N}{k}\rceil$ random sharings of the form $[\![\boldsymbol{O}]\!]_{t+2k-2}$ with the secret $\boldsymbol{O} = 0^{k\times k}$.

1: **Computing linear transformations.** For every group of k linear transformations $\vec{\boldsymbol{L}} = (\boldsymbol{L}_1, \ldots, \boldsymbol{L}_k)$ mapping from $\mathbb{F}^k$ to $\mathbb{F}^k$, all parties aim to prepare random linear masks of the form $([\![\boldsymbol{R}]\!]_{t+k-1}, [\![\vec{\boldsymbol{L}}(\boldsymbol{R})]\!]_{t+k-1})$, where the ℓ-th row of $\boldsymbol{R} \in \mathbb{F}^{k\times k}$ is $\boldsymbol{r}_\ell$ and the ℓ-th row of $\vec{\boldsymbol{L}}(\boldsymbol{R}) \in \mathbb{F}^{k\times k}$ is $\boldsymbol{L}_\ell(\boldsymbol{r}_\ell)$ for $\ell \in [k]$. They first pick the following unused random sharings
   - two random degree-$(t+k-1, t+k-1)$ packed Shamir sharing $[\![\boldsymbol{R}]\!]_{t+k-1}, [\![\boldsymbol{W}]\!]_{t+k-1}$,
   - $t+k-1$ random degree-$(t+2k-2)$ packed Shamir sharings $\{[\![\boldsymbol{v}_\ell]\!]_{t+2k-2}\}_{\ell\in[t+k-1]}$,
   - and a degree-$(t+2k-2, t+2k-2)$ packed Shamir sharing $[\![\boldsymbol{O}]\!]_{t+2k-2}$ with secret being all zeros.

   1. Performing linear transformations: For each $\ell \in [k]$, all parties locally compute $[\boldsymbol{u}_{*,\ell}]_{t+2k-2} \leftarrow [\boldsymbol{L}_{*,\ell 1}]_{k-1} \cdot [\![\boldsymbol{r}_{*,1}]\!]_{t+k-1} + \cdots + [\boldsymbol{L}_{*,\ell k}]_{k-1} \cdot [\![\boldsymbol{r}_{*,k}]\!]_{t+k-1}$, where $\boldsymbol{r}_{*,\ell} = (r_{1,\ell}, \ldots, r_{k,\ell})$ for $\ell \in [k]$.
   2. Computing row polynomials $f_i(x)$: Each active party P_i locally computes its row share of $\vec{\boldsymbol{L}}(\boldsymbol{R})$, denoted by $f_i(x)$, by interpolating $t+2k-1$ points such that
      - $f_i(\beta_\ell) = [\boldsymbol{u}_{*,\ell}]^i_{t+2k-2}$ for each $\ell \in [k]$ and
      - $f_i(\alpha_\ell) = [\![\boldsymbol{v}_\ell]\!]^i_{t+2k-2}$ for each $\ell \in [t+k-1]$,

      where $[\boldsymbol{x}]^i_{t+2k-2}$ denotes the i-th share of the sharing $[\boldsymbol{x}]_{t+2k-2}$.
   3. Computing column polynomials $g_i(y)$:
      For each active party P_i,
      a. Each active party P_j computes $f_j(\alpha_i)$ and sends it to P_i.
      b. On receiving values from other parties, P_i checks whether they lie on a degree-$(t+2k-2)$ polynomial.
         - If the check fails, P_i broadcasts $(P_i, \texttt{Complain})$. Then all parties locate a disputed pair E as follows.
           A. All parties send their shares of $[\boldsymbol{u}_{*,1}]_{t+2k-2}, \ldots, [\boldsymbol{u}_{*,k}]_{t+2k-2}$ and $[\![\boldsymbol{v}_1]\!]_{t+2k-2}, \ldots, [\![\boldsymbol{v}_{t+k-1}]\!]_{t+2k-2}$ to P_i.
           B. P_i checks whether each active party P_j correctly computes $f_j(\alpha_i)$ from $[\boldsymbol{u}_{*,1}]^i_{t+2k-2}, \ldots, [\boldsymbol{u}_{*,k}]^i_{t+2k-2}$ and $[\![\boldsymbol{v}_1]\!]^i_{t+2k-2}, \ldots, [\![\boldsymbol{v}_{t+k-1}]\!]^i_{t+2k-2}$.
              - If not, P_i broadcasts $(P_j, \texttt{Incorrect})$. All parties take the disputed pair $\{P_i, P_j\}$ as output.
           C. At least one of $[\boldsymbol{u}_{*,1}]_{t+2k-2}, \ldots, [\boldsymbol{u}_{*,k}]_{t+2k-2}$, $[\![\boldsymbol{v}_1]\!]_{t+2k-2}, \ldots, [\![\boldsymbol{v}_{t+k-1}]\!]_{t+2k-2}$ is inconsistent, say $[\boldsymbol{u}]_{t+2k-2}$. P_i broadcasts j to indicate the j-th sharing $[\boldsymbol{u}]_{t+2k-2}$ is inconsistent. Then P_i and other parties invoke Analyze-Sharing$(P_i, [\boldsymbol{u}]_{t+2k-2})$ to locate a disputed pair E and take E as output.

– If the check passes, P_i interpolates to compute its column polynomial $g_i(y)$ such that $g_i(\alpha_j) = f_j(\alpha_i)$ for $j \in \mathcal{P}_{\texttt{active}}$.

All parties obtain their shares of $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})]\!]_{t+2k-2}$.

4. **Degree reduction.**
   a. All parties locally compute $[\![\vec{\boldsymbol{L}}(\boldsymbol{R}) + \boldsymbol{W}]\!]_{t+2k-2} \leftarrow [\![\vec{\boldsymbol{L}}(\boldsymbol{R})]\!]_{t+2k-2} + [\![\boldsymbol{W}]\!]_{t+k-1} + [\![\boldsymbol{O}]\!]_{t+2k-2}$ and send their shares to P_{king}.
   b. P_{king} together with other active parties invoke Check-One$(P_{\text{king}}, [\![\vec{\boldsymbol{L}}(\boldsymbol{R}) + \boldsymbol{W}]\!]_{t+2k-2})$ to check whether the receiving sharing is a valid degree-$(t+2k-2, t+2k-2)$ packed Shamir sharing.
      – In case of failure, all parties take the disputed pair E as output.
   c. P_{king} reconstructs the secret $\vec{\boldsymbol{L}}(\boldsymbol{R}) + \boldsymbol{W}$ and distributes $[\![\vec{\boldsymbol{L}}(\boldsymbol{R}) + \boldsymbol{W}]\!]_{k-1}$ to other active parties.
   d. All parties invoke Check-All$(P_{\text{king}}, [\![\vec{\boldsymbol{L}}(\boldsymbol{R}) + \boldsymbol{W}]\!]_{k-1})$ to check whether P_{king} distributes a valid degree-$(k-1, k-1)$ packed Shamir sharing.
      – In case of failure, all parties take the disputed pair E as output.
   e. All parties locally compute $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})]\!]_{t+k-1} \leftarrow [\![\vec{\boldsymbol{L}}(\boldsymbol{R}) + \boldsymbol{W}]\!]_{k-1} - [\![\boldsymbol{W}]\!]_{t+k-1}$.

2: **Verification.** All parties invoke $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_8, N/k)$ with their pairs of sharings in the form of $([\![\vec{\boldsymbol{L}}(\boldsymbol{R}) + \boldsymbol{W}]\!]_{k-1}, [\![\vec{\boldsymbol{L}}(\boldsymbol{R}) + \boldsymbol{W}]\!]_{t+2k-2})$ to check whether each pair of sharings corresponds to the same secrets.
   – In case of failure, all parties take the disputed pair E as output.

Protocol 6: Sharing-Trans

All parties hold $[\boldsymbol{x}]_{t+k-1}$ and aim to compute $[\boldsymbol{L}(\boldsymbol{x})]_{t+k-1}$ by consuming a pair of random linear mask $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\boldsymbol{L}(\boldsymbol{r})]\!]_{t+k-1})$.

1. All parties locally compute $[\boldsymbol{x} + \boldsymbol{r}]_{t+k-1} \leftarrow [\boldsymbol{x}]_{t+k-1} + [\![\boldsymbol{r}]\!]_{t+k-1}$ and send their shares to P_{king}.
2. P_{king} checks whether the receiving sharing $[\boldsymbol{x}+\boldsymbol{r}]_{t+k-1}$ is a valid degree-$(t+k-1)$ packed Shamir sharing.
   – If not, P_{king} broadcasts `Inconsistent`. All active parties invoke Analyze-Sharing$(P_{\text{king}}, [\boldsymbol{x} + \boldsymbol{r}]_{t+k-1})$ to locate a disputed pair E. Then all parties take E as output.
3. P_{king} reconstructs the secret $\boldsymbol{x}+\boldsymbol{r}$ and distributes $[\boldsymbol{x}+\boldsymbol{r}]_{k-1}, [\boldsymbol{L}(\boldsymbol{x}+\boldsymbol{r})]_{k-1}$ to other active parties.
4. All parties locally compute $[\boldsymbol{L}(\boldsymbol{x})]_{t+k-1} \leftarrow [\boldsymbol{L}(\boldsymbol{x} + \boldsymbol{r})]_{k-1} - [\![\boldsymbol{L}(\boldsymbol{r})]\!]_{t+k-1}$.

Protocol 7: SELECT

All parties hold $[\boldsymbol{x}_1]_{t+k-1}, [\boldsymbol{x}_2]_{t+k-1}, \ldots, [\boldsymbol{x}_k]_{t+k-1}$ and aim to compute $[\boldsymbol{y}]_{t+k-1}$, where $\boldsymbol{y} = (y_1, y_2, \ldots, y_k)$ such that $y_\ell = x_{\ell,\ell}$ for $\ell \in [k]$. In the preprocessing phase, all parties have prepared a pair of random packed Shamir sharings $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\boldsymbol{r}]\!]_{t+2k-2})$.

1. **Collecting secrets.** All parties locally compute $[\boldsymbol{y}]_{t+2k-2} \leftarrow [\boldsymbol{e}_1]_{k-1} \cdot [\boldsymbol{x}_1]_{t+k-1} + \cdots + [\boldsymbol{e}_k]_{k-1} \cdot [\boldsymbol{x}_k]_{t+k-1}$.
2. **Degree reduction.**
   (a) All parties locally compute $[\boldsymbol{y} + \boldsymbol{r}]_{t+2k-2} \leftarrow [\boldsymbol{y}]_{t+2k-2} + [\![\boldsymbol{r}]\!]_{t+2k-2}$ and send their shares to P_{king}.
   (b) P_{king} checks whether the received sharing $[\boldsymbol{y} + \boldsymbol{r}]_{t+2k-2}$ is a valid degree-$(t + 2k - 2)$ packed Shamir sharing.

- If not, P_{king} broadcasts `Inconsistent`. Then P_{king} and other active parties invoke ANALYZE-SHARING$(P_{\text{king}}, [\boldsymbol{y}+\boldsymbol{r}]_{t+2k-2})$ to locate a disputed pair E. Then all parties take E as output.

(c) P_{king} reconstructs the secret $\boldsymbol{y}+\boldsymbol{r}$ and distributes $[\boldsymbol{y}+\boldsymbol{r}]_{k-1}$ to all active parties.

(d) All parties locally compute $[\boldsymbol{y}]_{t+k-1} \leftarrow [\boldsymbol{y}+\boldsymbol{r}]_{k-1} - [\![\boldsymbol{r}]\!]_{t+k-1}$.

Verification of Sharing Transformation. When performing sharing transformation, parties need to verify whether P_{king} honestly applies the linear transformations on the secrets. More specifically, for each $([\boldsymbol{u}]_{k-1}, [\boldsymbol{L}_\ell(\boldsymbol{u}_\ell)]_{k-1})$ of k pairs of sharings, parties verify whether the secret $\boldsymbol{L}_\ell(\boldsymbol{u}_\ell)$ of second sharing corresponds to that of the secret $\boldsymbol{u}_\ell$ of first sharing after applying the linear transformation $\boldsymbol{L}_\ell$.

We present the protocol for verifying sharing transformation in VRFY-SHARING-TRANS. The communication cost of VRFY-SHARING-TRANS to verify N invocations of SHARING-TRANS is $\mathsf{P2P}(O(\frac{N\cdot n^2}{k}+N\cdot n+n^2)) + O(1)\times\mathsf{BA}(1) + O(1)\times\mathsf{BC}(O(\log N+\log n+\log|\mathbb{F}|))$.

Protocol 8: VRFY-SHARING-TRANS(P_{king}, N)

All parties aim to verify whether P_{king} correctly performs linear transformations in all N invocations of SHARING-TRANS. I.e., supposing P_{king} distributes $([\boldsymbol{u}]_{k-1}, [\boldsymbol{L}(\boldsymbol{u})]_{k-1})$, all parties check whether two sharings are valid degree-$(k-1)$ packed Shamir sharings and the secret of the second sharing corresponds to that of the first sharing after being acted with the linear transformation $\boldsymbol{L}$.

1: For every group of k pairs of sharings distributed by P_{king} $([\boldsymbol{u}_1]_{k-1}, [\boldsymbol{L}_1(\boldsymbol{u}_1)]_{k-1}), \ldots, ([\boldsymbol{u}_k]_{k-1}, [\boldsymbol{L}_k(\boldsymbol{u}_k)]_{k-1})$, all parties verify whether the secrets of $[\boldsymbol{u}_\ell]_{k-1}, [\boldsymbol{L}_\ell(\boldsymbol{u}_\ell)]_{k-1}$ are correctly computed for each $\ell\in[k]$ as follows.

1. **Transpose.** Each active party P_i locally computes its column share of $[\![\boldsymbol{U}]\!]_{k-1}$, denoted by $g_i(x)$, by interpolating k points such that $g_i(\beta_\ell) = [\boldsymbol{u}_\ell]^i_{k-1}$ for each $\ell\in[k]$. Then all parties compute their row shares of $[\![\boldsymbol{U}]\!]_{k-1}$ as below.
   For each active party P_i,
   a. Each active party P_j computes $g_j(\alpha_i)$ and sends it to P_i.
   b. On receiving values from other active parties, P_i uses error correcting property of degree-$(k-1)$ Shamir sharing to compute the correct version of the degree-$(k-1)$ polynomial $f_i(x)$ such that $f_i(\alpha_j) = g_j(\alpha_i)$ for $j\in\mathcal{P}_{\texttt{active}}$.
   All parties obtain their shares of $[\boldsymbol{u}_{*,\ell}]_{k-1}$ for each $\ell\in[k]$, where $\boldsymbol{u}_{*,\ell} = (u_{1,\ell}, u_{2,\ell}, \ldots, u_{k,\ell})$.
2. **Performing linear transformations.** For each $\ell\in[k]$, all parties locally compute $[\boldsymbol{v}_{*,\ell}]_{2k-2} \leftarrow [\boldsymbol{L}_{*,\ell 1}]_{k-1}\cdot[\boldsymbol{u}_{*,1}]_{k-1} + \cdots + [\boldsymbol{L}_{*,\ell k}]_{k-1}\cdot[\boldsymbol{u}_{*,k}]_{k-1}$. Notice that $\boldsymbol{v}_{*,\ell} = (\boldsymbol{L}_\ell(\boldsymbol{u}_1), \ldots, \boldsymbol{L}_\ell(\boldsymbol{u}_k))$ for each $\ell\in[k]$.
3. **Transpose.** Each active party P_i locally computes its row share of $[\![\boldsymbol{L}(\boldsymbol{U})]\!]_{2k-2}$, denoted by $f'_i(x)$, by interpolating k points such that $f'_i(\beta_\ell) = [\boldsymbol{v}_{*,\ell}]^i_{2k-2}$ for each $\ell\in[k]$. Then all parties compute their column shares of $[\![\boldsymbol{L}(\boldsymbol{U})]\!]_{k-1}$ as below.
   For each active party P_i,
   a. Each active party P_j computes $f'_j(\alpha_i)$ and sends it to P_i.
   b. On receiving values from other active parties, P_i uses error correcting property of degree-$(2k-2)$ Shamir sharing to compute the correct version of the degree-$(2k-2)$ polynomial $g'_i(y)$ such that $g'_i(\alpha_j) = f'_j(\alpha_i)$ for $j\in\mathcal{P}_{\texttt{active}}$.
   All parties obtain their shares of $[\boldsymbol{v}_\ell]_{k-1}$ for $\ell\in[k]$, where $\boldsymbol{v}_\ell = \boldsymbol{L}_\ell(\boldsymbol{u}_\ell)$.
4. For each $\ell\in[k]$, all parties locally compute $[\boldsymbol{o}_\ell]_{k-1} \leftarrow [\boldsymbol{v}_\ell]_{k-1} - [\boldsymbol{L}_\ell(\boldsymbol{u}_\ell)]_{k-1}$.

2: **Verification.** All parties invoke $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_{11}, N)$ with their shares obtained in Step 1.d to check whether each sharing is a valid degree-$(k-1)$ packed Shamir sharing with secret $\boldsymbol{0}\in\mathbb{F}^k$.
- In case of failure, all parties take E as output.

4.5 Summary for Segment Computation

Before summing up the computation for evaluating a segment, we notice that for each k outputs $\boldsymbol{z}$ of the current segment, all parties hold a degree-$(t+k-1)$ packed Shamir sharing $[\boldsymbol{z}]_{t+k-1}$, which can be decomposed as $[\boldsymbol{z}]_{t+k-1} = [\boldsymbol{z}^{(0)}]_{k-1} + [\![\boldsymbol{z}^{(1)}]\!]_{t+k-1}$, where $[\boldsymbol{z}^{(0)}]_{k-1}$ is a degree-$(k-1)$ packed Shamir sharing distributed by P_{king} and $[\![\boldsymbol{z}^{(1)}]\!]_{t+k-1}$ is a linear combination of row vectors of secret matrices shared via degree-$(t+k-1, t+k-1)$ packed Shamir sharings. Even though P_{king} might be eliminated in the future segment, parties can still recover the whole sharing $[\boldsymbol{z}^{(0)}]_{k-1}$ by utilizing the error-correcting property of degree-$(k-1)$ Shamir sharings.

We summarize the protocol for computing a segment containing addition gates and multiplication gates in Eval, which invokes Zero-Sharing to prepare degree-$(n'-1)$ packed zero-sharings used for evaluating multiplication gates. The communication cost of Zero-Sharing to prepare $n'-t$ random degree-$(n'-1)$ packed Shamir sharings with secret $\mathbf{0}$ is $\mathsf{P2P}(O(n^2))$. The communication cost of Eval to handle N multiplication gates and N_f fan-out gates is $\mathsf{P2P}(O(\frac{(N+N_f)\cdot n^2}{k^2} + n^3)) + O(1) \times \mathsf{BA}(1) + O(1) \times \mathsf{BC}(O(\log N + \log N_f + \log n + \log |\mathbb{F}|))$.

Protocol 9: Zero-Sharing

All parties prepare $n'-t$ random degree-$(n'-1)$ packed Shamir sharing with secret $\mathbf{0} \in \mathbb{F}^k$.

1. All parties agree on a Vandermonde matrix $\boldsymbol{M}^\top$ of size $n' \times (n'-t)$.
2. Each active party P_i randomly samples a random degree-$(n'-1)$ packed Shamir sharing $[\boldsymbol{o}^{(i)}]_{n'-1}$ with secret $\boldsymbol{o}^{(i)} = \mathbf{0} \in \mathbb{F}^k$ and distributes the shares to other parties.
3. All parties locally compute

$$([\boldsymbol{o}_1]_{n'-1}, \ldots, [\boldsymbol{o}_{n'-t}]_{n'-1})^\top = \boldsymbol{M} \cdot ([\boldsymbol{o}^{(1)}]_{n'-1}, \ldots, [\boldsymbol{o}^{(n')}]_{n'-1})^\top$$

and take the sharings $[\boldsymbol{o}_1]_{n'-1}, \ldots, [\boldsymbol{o}_{n'-t}]_{n'-1}$ as output.

Protocol 10: Eval

All parties evaluate the segment containing N multiplication gates and N_f fan-out gates. All parties agree on an active party $P_{\text{king}} \in \mathcal{P}_{\texttt{active}}$.
In the preprocessing phase, all parties have prepared

- $\lceil \frac{N}{k} \rceil$ packed Beaver triples of the form $([\![\boldsymbol{a}]\!]_{t+k-1}, [\![\boldsymbol{b}]\!]_{t+k-1}, [\![\boldsymbol{c}]\!]_{t+k-1})$,
- $\frac{N\cdot(2t+9k)}{k^3} + \frac{N_f\cdot(t+3k)}{k^3} + k + 2$ random sharings of the form $[\![\boldsymbol{R}]\!]_{t+k-1}$,
- and $\frac{N\cdot(2t+7k)}{k^3} + \frac{N_f\cdot(t+2k)}{k^3} + k + 1$ random sharings of the form $[\![\boldsymbol{O}]\!]_{t+2k-2}$ with the secret $\boldsymbol{O} = 0^{k\times k}$.

1: **Preparing random sharings.**
   1. All parties invoke $\lceil \frac{N}{(n'-t)\cdot k} \rceil$ times of Zero-Sharing to prepare $\lceil \frac{N}{k} \rceil$ random degree-$(n'-1)$ packed Shamir sharings in the form of $[\boldsymbol{o}]_{n'-1}$ with secret $\boldsymbol{o} = \mathbf{0} \in \mathbb{F}^k$.
   2. All parties invoke Rand-Linear-Mask$(2\lceil \frac{N}{k} \rceil + \lceil \frac{N_f}{k} \rceil)$ to prepare pairs of random sharings, each in the form of $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\boldsymbol{L}(\boldsymbol{r})]\!]_{t+k-1})$.

2: **Computation phase.** All parties compute the segment layer by layer.
   1. Permute input sharings from previous layers: For each input sharing $[\boldsymbol{x}]_{t+k-1}$, all parties first invoke select and then invoke Sharing-Trans with a pair of random sharings in the form of $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\boldsymbol{L}(\boldsymbol{r})]\!]_{t+k-1})$ to obtain the input sharing.
   2. For each group of k gates, let $[\boldsymbol{x}]_{t+k-1}, [\boldsymbol{y}]_{t+k-1}$ be the input sharings.
      - If they are addition gates, all parties locally compute $[\boldsymbol{z}]_{t+k-1} \leftarrow [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{y}]_{t+k-1}$, where $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y}$.

- If they are multiplication gates, all parties invoke MULT to compute $[\boldsymbol{z}]_{t+k-1}$, where $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$.

3. Handling fan-out gates and achieving non-collision property for the next layer: All parties invoke SHARING-TRANS to obtain the output sharings of the current layer.

3: **Verification phase.** All parties verify the computation of the segment.

1. All parties invoke VRFY-MULT$(P_{\text{king}}, \lceil \frac{N}{k} \rceil)$ to verify the correctness of all invocations of MULT.
2. All parties invoke $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_9, 6\lceil \frac{N}{k} \rceil + 2\lceil \frac{N_f}{k} \rceil)$ with their shares received in Step 3 of SHARING-TRANS and Step 2.c of SELECT to check whether P_{king} always distributes valid degree-$(k-1)$ packed Shamir sharings.
3. All parties invoke VRFY-SHARING-TRANS$(P_{\text{king}}, 2\lceil \frac{N}{k} \rceil + \lceil \frac{N_f}{k} \rceil)$ to verify the correctness of all invocations of SHARING-TRANS.
4. All parties invoke $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_{10}, 2\lceil \frac{N}{k} \rceil)$ with their shares in the form of $([\boldsymbol{y} + \boldsymbol{r}]_{k-1}, [\boldsymbol{y} + \boldsymbol{r}]_{t+2k-2})$ to check whether P_{king} always distributes degree-$(k-1)$ packed Shamir sharings with correct secrets in SELECT.
   - If one of the checks fails, all parties take the disputed pair E as output.
   - Otherwise, all parties accept the computation.

4.6 Main Protocol

We first deal the input phase and output phase, and then summarize our construction.

Input Phase. For input gates, when a party P_d aims to distribute its group $\boldsymbol{X} = (x_{ij})_{ij} \in \mathbb{F}^{k \times k}$ of k^2 inputs to all parties, it randomly samples a degree-$(t+k-1, t+k-1)$ packed Shamir sharing with secret $\boldsymbol{X}$ and distributes them to all parties. To protect against a corrupted dealer, all parties will cross check to ensure the dealer distributes a valid degree-$(t+k-1, t+k-1)$ packed Shamir sharing. To guarantee that an honest dealer can eventually distribute its inputs successfully, the dealer assigns zero polynomials to parties that are in conflict with it. As a result, each corrupted party is unable to disrupt the sharing procedure twice, ensuring the success within at most t trails. To avoid the communication blowup due to the failure attempts and retries, we uniformly divide the segment that contains all input gates belonging to P_d into n sub-segments. In doing this, the communication cost at most doubles in the worst case.

We formally present the protocol in INPUT, which invokes SHARE-INPUT to let the dealer share its inputs. The communication cost of INPUT to handle all N input gates belonging to party P_d and N_f fan-out gates is $\mathsf{P2P}(O(\frac{(N+N_f)\cdot n^2}{k^2} + n^3)) + O(1) \times \mathsf{BA}(1) + O(1) \times \mathsf{BC}(O(\log N_f + \log n + \log |\mathbb{F}|))$.

Protocol 11: SHARE-INPUT(P_d, m)

All parties maintain public variables $\mathsf{Corrupt}_d$ and $\mathsf{Conflicts}_\mathsf{d}$.

1. **Dealing shares.** Let $\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots, \boldsymbol{X}^{(m)}$ denote the inputs of P_d, where each $\boldsymbol{X}^{(\ell)}$ denotes k^2 input gates belonging to P_d. For each $\ell \in [m]$, P_d samples a random degree-$(t+k-1, t+k-1)$ packed Shamir sharing $[\![\boldsymbol{X}^{(\ell)}]\!]_{t+k-1}$ shared via a degree-$(t+k-1, t+k-1)$ bivariate polynomial $S^{(\ell)}(x, y)$ such that the shares held by parties in $\mathsf{Conflicts}_d \cap \mathcal{P}_{\mathtt{active}}$ are set to be zero polynomials, i.e., $f_j^{(\ell)}(x) \equiv g_j^{(\ell)}(y) \equiv 0$ for every $P_j \in \mathsf{Conflicts}_d \cap \mathcal{P}_{\mathtt{active}}$, where $f_j^{(\ell)}(x) = S^{(\ell)}(x, \alpha_j)$ and $g_j^{(\ell)}(y) = S^{(\ell)}(\alpha_j, y)$. Then P_d distributes $[\![\boldsymbol{X}^{(\ell)}]\!]_{t+k-1}$ to other parties.
2. **Consistency check.**
   For each $\ell \in [m]$,
   1. All active parties invoke CHECK-ALL$(P_d, [\![\boldsymbol{X}^{(\ell)}]\!]_{t+k-1})$ to check whether P_d distributes a valid degree-$(t+k-1, t+k-1)$ packed Shamir sharing.
      - In case of failure, all parties obtain a disputed pair E and go to Step 3.

2. Each active party P_i checks whether $f_i^{(\ell)}(\alpha_j) = g_i^{(\ell)}(\alpha_j) = 0$ for every $P_j \in \mathsf{Conflicts} \cap \mathcal{P}_{\mathtt{active}}$.
   - In case of failure, P_i broadcasts $(P_i, \mathtt{Complain})$. All parties set $E \leftarrow \{P_d, P_i\}$ and go to Step 3.

   If all checks pass, go to Step 4.
3. **Update.** All parties update $\mathsf{Conflicts}_d$ and $\mathsf{Corrupt}_d$ as follows.
   - If $E = \{P_d, P_i\}$, all parties set $\mathsf{Conflicts}_d \leftarrow \mathsf{Conflicts}_d \cup \{P_i\}$. If $|\mathsf{Conflicts}_d| \le t$, go to Step 1; if $|\mathsf{Conflicts}_d| > t$, set $\mathsf{Corrupt}_d \leftarrow 1$ and go to Step 4.
   - If $E = \{P_d\}$, all parties set $\mathsf{Corrupt}_d \leftarrow 1$ and go to Step 4.
   - If $P_d \notin E$, all parties take E as output.
4. **Output.** If $\mathsf{Corrupt}_d = 1$, i.e., P_d has been identified as a corrupted party, all parties set $[\![\boldsymbol{X}^{(\ell)}]\!]_{t+k-1} \leftarrow [\![\boldsymbol{0}]\!]_0$ for each $\ell \in [m]$. All parties take $[\![\boldsymbol{X}^{(1)}]\!]_{t+k-1}, \ldots, [\![\boldsymbol{X}^{(m)}]\!]_{t+k-1}$ as outputs.

Protocol 12: Input(P_d)

All parties handle the segment containing N input gates and N_f fan-out gates, where all inputs belong to P_d. All parties agree on an active party $P_{\text{king}} \in \mathcal{P}_{\mathtt{active}}$.
In the preprocessing phase, all parties have prepared

- $\frac{N_f \cdot (t+3k)}{k^3} + k + 2$ random sharings of the form $[\![\boldsymbol{R}]\!]_{t+k-1}$
- and $\frac{N_f \cdot (t+2k)}{k^3} + k + 1$ random sharings of the form $[\![\boldsymbol{O}]\!]_{t+2k-2}$ with secret $\boldsymbol{O} = 0^{k \times k}$.

1. **Preparing random linear masks.** All parties invoke Rand-Linear-Mask$(\lceil \frac{N_f}{k} \rceil)$ to prepare pairs of random sharings, each in the form of $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\boldsymbol{L}(\boldsymbol{r})]\!]_{t+k-1})$ to obtain the input sharing.
2. Input secret-sharing phase: All parties invoke Share-Input$(P_d, \lceil \frac{N}{k^2} \rceil)$ to let P_d distribute its inputs.
   - In case of failure, all parties take the disputed pair E as output.
3. Handling fan-out gates and achieving non-collision property for the next layer: All parties invoke Sharing-Trans to obtain the output sharings of the current layer.
4. **Verification.**
   (a) All parties invoke $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_9, 2\lceil \frac{N_f}{k} \rceil)$ with their shares received in Step 3 of Sharing-Trans to check whether P_{king} always distributes valid degree-$(k-1)$ packed Shamir sharings.
   (b) All parties invoke Vrfy-Sharing-Trans$(P_{\text{king}}, \lceil \frac{N_f}{k} \rceil)$ to verify the correctness of all invocations of Sharing-Trans.
   - If one of the checks fails, all parties take the disputed pair E as output.
   - Otherwise, all parties accept the computation.

Output Phase. After evaluating the whole circuit, when handling output gates, to ensure a success of reconstruction, all parties depack each degree-$(t+k-1)$ packed Shamir sharing $[\![\boldsymbol{z}]\!]_{t+k-1}$ corresponding to a group of k output wires to obtain degree-t Shamir sharings of each circuit output. To this end, all parties make use of a tuple of random sharings in the form of $(\{[r_j|j]_t\}_{j\in[k]}, [\![\boldsymbol{r}]\!]_{t+k-1})$ prepared in the preprocessing phase, where $\boldsymbol{r} = (r_j)_j \in \mathbb{F}^k$, containing a degree-$(t+k-1)$ packed Shamir sharing $[\![\boldsymbol{r}]\!]_{t+k-1}$ and degree-t Shamir sharings of each component $[r_j|j]_t$ with secret stored at the j-th position. Then all parties follow the *mask-open-unmask* template and add extra verification to ensure the correctness.

We present the protocol for handling output gates in Output. The communication cost of Output to handle N output gates is $\mathsf{P2P}(O(\frac{N \cdot n^2}{k^2} + n^3)) + O(1) \times \mathsf{BA}(1) + O(1) \times \mathsf{BC}(O(\log N + \log n + \log |\mathbb{F}|))$.

Protocol 13: OUTPUT

All parties handle the segment containing N output gates. All parties agree on an active party $P_{\text{king}} \in \mathcal{P}_{\texttt{active}}$.
In the preprocessing phase, all parties have prepared

- $\lceil \frac{N}{k^2} \rceil$ tuples of random sharings of the form $(\{[r_{ij}|j]_t\}_{i,j\in[k]}, [\![\boldsymbol{R}]\!]_{t+k-1})$ with $\boldsymbol{R} = (r_{ij})_{ij}$,
- $\frac{N\cdot(t+4k)}{k^3} + k + 2$ random sharings of the form $[\![\boldsymbol{R}]\!]_{t+k-1}$,
- and $\frac{N\cdot(t+3k)}{k^3} + k + 1$ random sharings of the form $[\![\boldsymbol{O}]\!]_{t+2k-2}$ with the secret $\boldsymbol{O} = 0^{k\times k}$.

1. **Preparing random sharings**. All parties invoke RAND-LINEAR-MASK($\lceil \frac{N}{k} \rceil$) to prepare pairs of random sharings, each in the form of $([\![\boldsymbol{u}]\!]_{t+k-1}, [\![\boldsymbol{L}(\boldsymbol{u})]\!]_{t+k-1})$.
2. **Permuting input sharings from previous layers.** For each input sharing $[\boldsymbol{x}]_{t+k-1}$, all parties first invoke SELECT with a pair of random sharings of the form $([\![\boldsymbol{u}]\!]_{t+k-1}, [\![\boldsymbol{u}]\!]_{t+2k-2})$ and then invoke SHARING-TRANS with a pair of random sharings of the form $([\![\boldsymbol{u}]\!]_{t+k-1}, [\![\boldsymbol{L}(\boldsymbol{u})]\!]_{t+k-1})$ to obtain the input sharing.
3. **Depack.**
   For every group $\boldsymbol{z} = (z_j)_j$ of k output gates, all parties prepare degree-t Shamir sharings of the outputs as follows.
   (a) All parties pick one unused random sharing $(\{[r_j|j]_t\}_{j\in[k]}, [\![\boldsymbol{r}]\!]_{t+k-1})$ prepared in the preprocessing phase, where $\boldsymbol{r} = (r_1, r_2, \ldots, r_k)$.
   (b) All parties locally compute $[\boldsymbol{z}+\boldsymbol{r}]_{t+k-1} \leftarrow [\boldsymbol{z}]_{t+k-1} + [\![\boldsymbol{r}]\!]_{t+k-1}$ and send their shares to P_{king}.
   (c) P_{king} checks whether the received shares of $[\boldsymbol{z}+\boldsymbol{r}]_{t+k-1}$ form a valid degree-$(t+k-1)$ packed Shamir sharing.
   – In case of failure, P_{king} broadcasts `Inconsistent`. Then P_{king} and other active parties invoke ANALYZE-SHARING$(P_{\text{king}}, [\boldsymbol{z}+\boldsymbol{r}]_{t+k-1})$ to locate a disputed pair E. All parties take E as output.
   (d) P_{king} reconstructs the secret $\boldsymbol{z}+\boldsymbol{r}$ and distributes $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ to other parties.
   (e) For $j \in [k]$, all parties locally compute $[z_j|j]_t \leftarrow [\boldsymbol{z}+\boldsymbol{r}]_{k-1} - [r_j|j]_t$.
4. **Verification.** All parties verify the computation as follows.
   (a) All parties invoke $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_9, 3\lceil \frac{N}{k} \rceil)$ with their shares received in Step 3 of SHARING-TRANS, Step 2.c of SELECT, and Step 3.d to check whether P_{king} always distributes valid degree-$(k-1)$ packed Shamir sharings.
   (b) All parties invoke VRFY-SHARING-TRANS$(P_{\text{king}}, \lceil \frac{N}{k} \rceil)$ to verify the correctness of all invocations of SHARING-TRANS.
   (c) All parties invoke $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_{10}, 2\lceil \frac{N}{k} \rceil)$ with their shares obtained in Step 3.d of the form $([\boldsymbol{z}+\boldsymbol{r}]_{k-1}, [\boldsymbol{z}+\boldsymbol{r}]_{t+k-1})$ and their shares obtained in Step 2.c of SELECT of the form $([\boldsymbol{y}+\boldsymbol{r}]_{k-1}, [\boldsymbol{y}+\boldsymbol{r}]_{t+k-1})$ to verify whether the first sharing of each pair has the same secret as that of the second sharing.
   – In case of failure, all parties take the disputed pair E as output.
   – Otherwise, all parties accept the computation.

Summary for Main Protocol. Throughput the protocol, we follow the party elimination framework. After evaluating a segment, if a failure occurs, a disputed pair will be identified. All parties then eliminate this pair from the set of active parties and re-evaluate the current segment until it succeeds. After degree-t Shamir sharings of all circuit outputs have been prepared, all parties proceed to reconstruct the secrets to a designated party, where the reconstruction always succeeds due to error-correcting property of degree-t Shamir sharings. As mentioned in Section 2.2, if, *before reconstruction*, $n/3$ parties have been eliminated, all parties will restart the computation by invoking a perfectly secure protocol in the client-server model against an adversary corrupting at most a $1/4$ fraction of servers, where all n parties would act as clients and all n' active parties would act as servers. We use $\mathcal{F}^*_{\text{main-client-server}}$ to characterize this functionality. We emphasize

that since no parties have received outputs before reconstruction, we do not need to ensure the same inputs from corrupted parties in this fallback procedure.

Assume there are $t' < n'/4$ active corrupted parties among all n' active parties. Such a perfectly secure protocol with $1/4$ corruption threshold for general circuits that achieves overall $O(|C|)$ communication can be constructed by adapting our current protocol. The idea is to utilize packed Shamir sharings. Suppose k' is the packing parameter. In particular, we conduct the following two modifications. First, we make use of the approach in [BH08] to prepare degree-$(t'+k'-1)$ packed Beaver triples, which costs $O(1)$ elements of communication per secret. To this end, we set the packing parameter k' to satisfy $t'+k'-1 \leq (n'-1)/3$. Secondly, instead of generating all required preprocessing data in advance, we prepare packed Beaver triples over bivariate polynomials as well as other preprocessing data used for the current segment just before computing this segment. In doing this, if an error occurs in this segment, parties can open their view generated when preparing the preprocessing data to identify a new disputed pair containing at least one corrupted party. After eliminating this pair of parties, they re-compute this segment until success. Notice that in the worst case, there are at least $2t'+1$ active honest parties, which ensures a degree-$(t'+2k-2)$ packed Shamir sharing is fully determined by shares of active honest parties. Choosing $k' = \lfloor (n'+11)/12 \rfloor$ satisfies our requirements.

Putting things altogether, we present the whole protocol in MAIN. We conclude its security property in Lemma 1 and show the framework of our construction in Figure 1.

The communication complexity of MAIN is $\mathsf{P2P}(O(C_I + C_A + C_M + D \cdot n + C_O \cdot n + n^4)) + O(n) \times \mathsf{BA}(1) + O(n) \times \mathsf{BC}(O(\log|C| + \log D + \log n + \log|\mathbb{F}|))$ plus $O(n)$ invocations of $\mathcal{F}_{\text{packed-triple}}(\lceil \frac{2C_M}{n \cdot k^2} \rceil, k)$, and one invocation of $\mathcal{F}^*_{\text{main-client-server}}$ in case all parties restart the whole computation, where C_I, C_A, C_M, C_O and D denote the number of input gates, addition gates, multiplication gates, output gates, and circuit depth, respectively, and $|C| = C_I + C_A + C_M + C_O$. Assuming $|C| \geq n^2 \cdot n$, we have $\log|C| \cdot n^2 \leq |C|$, which implies if we use a perfect Byzantine agreement with quadratic cost to instantiate both consensus and broadcast or use an expected constant-round broadcast protocol with amortized linear cost in [AC24], the total cost is $\mathsf{P2P}(O(|C| + D \cdot n + C_O \cdot n + n^4))$ elements plus packed Beaver triples over bivariate polynomials of size $O(|C|)$.

We conclude our construction of perfectly secure MPC in the preprocessing model in Theorem 6.

Protocol 14: MAIN

Let C_I, C_A, C_M and C_O be the number of input gates, addition gates, multiplication gates, and output gates, respectively. All parties initialize $\mathcal{P}_{\texttt{active}} \leftarrow \{1, \ldots, n\}$.

1: **Preprocessing phase.** All parties divide the whole required preprocessing data of
- $2C_M/k^2$ packed Beaver triples over bivariate polynomials,
- $\frac{32C_A+34C_M+16C_O}{k^2}$ random sharings of the form $[\![\boldsymbol{R}]\!]_{t+k-1}$,
- $\frac{28C_A+30C_M+14C_O}{k^2}$ random sharings of the form $[\![\boldsymbol{O}]\!]_{t+2k-2}$ with secret $\boldsymbol{O} = 0^{k\times k}$,
- and $2C_O/k^2$ tuples of random sharings of the form $(\{[r_{ij}|j]_t\}_{i,j\in[k]}, [\![\boldsymbol{R}]\!]_{t+k-1})$ with $\boldsymbol{R} = (r_{ij})_{ij} \in \mathbb{F}^{k\times k}$

into n segments. Initialize $i \leftarrow 1$.

(a) All parties invoke $\mathcal{F}_{\text{packed-triple}}(\lceil \frac{2C_M}{n\cdot k^2} \rceil, k)$, $\mathcal{F}_{\text{rand-sharing}}(\Sigma_5, \lceil \frac{32C_A+34C_M+16C_O}{n\cdot k^2} \rceil)$, $\mathcal{F}_{\text{rand-sharing}}(\Sigma_2, \lceil \frac{28C_A+30C_M+14C_O}{n\cdot k^2} \rceil)$, and $\mathcal{F}_{\text{rand-sharing}}(\Sigma_3, \lceil \frac{2C_O}{n\cdot k^2} \rceil)$ to prepare random secret sharings.

(b) In case of failure, output a new disputed pair E, and update $\mathcal{P}_{\texttt{active}} \leftarrow \mathcal{P}_{\texttt{active}} \setminus E$.
- If $|\mathcal{P}_{\texttt{active}}| \geq 2t$, go to Step 1.a to re-evaluate the current segment.
- If $|\mathcal{P}_{\texttt{active}}| < 2t$, go to Step 3 to restart the computation.

In case of success,
- if $i < n$, set $i \leftarrow i+1$, and go to Step 1.a to handle the next segment,
- if $i = n$, proceed to the online phase.

2: **Online phase.** Initialize $\mathsf{Corrupt}_s \leftarrow 0$ and $\mathsf{Conflicts}_s \leftarrow \emptyset$ for each $s \in [n]$. All parties invoke Algorithm 1 to divide the circuit C into $S = \Omega(n)$ segments, denoted by an array $\mathsf{Seg}[0 : S-1]$, initialize $i \leftarrow 0$, and handle all the segments sequentially as follows.

(a) Depending on the type of $\mathsf{Seg}[i]$, there are 3 cases.
- If $\mathsf{Seg}[i]$ is an input segment and the inputs belong to P_d, all parties invoke $\textsc{Input}(P_d)$.
- If $\mathsf{Seg}[i]$ is a computation segment, all parties invoke $\textsc{Eval}$.
- If $\mathsf{Seg}[i]$ is an output segment, all parties invoke $\textsc{Output}$.

(b) In case of failure, output a new disputed pair E, and update $\mathcal{P}_{\mathtt{active}} \leftarrow \mathcal{P}_{\mathtt{active}} \setminus E$.
- If $|\mathcal{P}_{\mathtt{active}}| \geq 2t$, go to Step 2.a to re-evaluate the current segment.
- If $|\mathcal{P}_{\mathtt{active}}| < 2t$, go to Step 3 to restart the computation.

In case of success,
- if $i < S-1$, set $i \leftarrow i+1$, and go to Step 2.a to handle the next segment,
- if $i = S-1$, go to Step 2.c to reconstruct the outputs.

(c) **Reconstruction.** For each output wire o, supposing it belongs to P_d and the secret is stored at the j-th position, all parties reconstruct their shares of $[o|j]_t$ to P_d. Then all parties learn their outputs.

3: **Restart.** All parties invoke $\mathcal{F}^*_{\text{main-client-server}}$, provide their inputs, and receive their outputs.

Functionality 2: $\mathcal{F}^*_{\text{main-client-server}}$

1. $\mathcal{F}^*_{\text{main-client-server}}$[a] receives the input from all clients. Let x denote the input.
2. $\mathcal{F}^*_{\text{main-client-server}}$ computes $C(x)$ and distributes the output to all clients.

[a] We write $\mathcal{F}^*_{\text{main-client-server}}$ to emphasize that the functionality $\mathcal{F}^*_{\text{main-client-server}}$ can be invoked only by an adversary corrupting at most a 1/4 fraction of the parties.

Functionality 3: $\mathcal{F}_{\text{main}}$

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

Theorem 6. *Let n be the number of parties and $k = n/12$, $t = (n-1)/3$. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2n$. For an arithmetic circuit C over $\mathbb{F}$, there exists a perfectly secure MPC protocol in the circuit-preprocessing model which computes C in the presence of a fully malicious adversary controlling at most t parties. The cost is $\mathsf{P2P}(O(|C|+D\cdot n+C_O\cdot n+n^4))$ elements plus $O(n)$ invocations of $\mathcal{F}_{\textit{packed-triple}}(\lceil \frac{2C_M}{n\cdot k^2} \rceil, k)$, where C_O, C_M and D denote the number of output gates, the number of multiplication gates, and circuit depth, respectively.*

Lemma 1. *Let $k = n/12$ and $t = (n-1)/3$. $\textsc{main}$ computes $\mathcal{F}_{\textit{main}}$ with perfect security in the $\mathcal{F}_{\textit{packed-triple}}$, $\mathcal{F}_{\textit{batched-vrfy}}$, $\mathcal{F}_{\textit{rand-sharing}}$, $\mathcal{F}^*_{\textit{main-client-server}}$-hybrid model against a fully malicious adversary who controls at most t corrupted parties.*

Proof (Proof of Lemma 1). We describe the strategy of the simulator Sim in Sim. Let $\mathcal{C}$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties. Let $\mathcal{C}_{\mathtt{active}}$ be a variable recording the set of currently active corrupted parties, i.e., $\mathcal{C}_{\mathtt{active}} = \mathcal{P}_{\mathtt{active}} \cap \mathcal{C}$. If $|\mathcal{C}| < t$, during the protocol execution, $\mathcal{S}$ will additionally generate the view of a set $\mathcal{H}'$ of $t - |\mathcal{C}|$ honest parties with smallest identities without using honest parties' inputs. Sim always records shares of parties in $\mathcal{C} \cup \mathcal{H}'$.

Algorithm 1: Segment Division

Input: A circuit C of circuit depth Depth with $|C|/k$ groups of gates, $g_0, \ldots, g_{|C|/k-1}$ labeled following the circuit topology. For a group g of k gates, denote the set of wires copied from the output wires of g by g_{out}.

Output: $S, \mathsf{Seg}[0 : \mathsf{S} - 1]$

Initialize $i \leftarrow 0, \mathsf{Seg}[i] \leftarrow \emptyset, \mathsf{SegSize} = 0, \mathsf{SegDepth} = 0$;

for $j = 0$ *to* $|C|/k - 1$ **do**

```
// Check the requirements of the segment and keep adding groups of gates until the segment
   is full or all gates have already been included.
```

if $\mathsf{SegSize} + k \leq |C|/n$ *and* $\mathsf{SegDepth} + \mathbf{1}_{g_j \textit{ belongs to a new layer}} \leq \mathsf{Depth}/n^2$ **then**

$\mathsf{Seg}[i] \leftarrow \mathsf{Seg}[i] \cup \{g_j\}$;

$\mathsf{SegSize} \leftarrow \mathsf{SegSize} + k$;

$\mathsf{SegDepth} \leftarrow \mathsf{SegDepth} + \mathbf{1}_{g_j \text{belongs to a new layer}}$;

if $j \neq |C|/k - 1$ **then**

Continue ;

```
// After determining the segment, handle fan-out gates of the segment.
```

$\mathsf{fanout\text{-}rest} \leftarrow \emptyset$;

for *each* g *in* $\mathsf{Seg}[i]$ **do**

Consider all wires in g_{out} ;

```
// Handle the fan-out gates that will be used in the current segment.
```

Collect a set F of all the wires that will be used as input wires in $\mathsf{Seg}[i]$, add the first $k \cdot \lceil \frac{|F|}{k} \rceil - |F|$ wires in $g_{out} \setminus F$ to F in order to make the size of F a multiple of k, and assign each k wires in F a fan-out gate which will be included in $\mathsf{Seg}[i]$;

For the rest wires in g_{out}, assign each k of them a fan-out gate which will be temporarily included in $\mathsf{fanout\text{-}rest}$;

```
// Handle the rest of the fan-out gates.
```

For all fan-out gates in $\mathsf{fanout\text{-}rest}$, uniformly divide them into $m = \lceil |\mathsf{fanout\text{-}rest}| / \lfloor \frac{|C|}{n} \rfloor \rceil$ segments, $\mathsf{Seg}[i+1], \ldots, \mathsf{Seg}[i+m]$, with each segment containing at most $\lfloor \frac{|C|}{n} \rfloor$ fan-out gates ;

```
// Start the next segment division.
```

$i \leftarrow i + m + 1, \mathsf{Seg}[i] \leftarrow \emptyset, \mathsf{SegSize} = 0, \mathsf{SegDepth} = 0$;

$S = i$;

Output $S, \mathsf{Seg}[0 : S - 1]$;

Simulator 1: Sim for MAIN

Sim invokes the adversary with sampled internal randomness.

1. **Preprocessing phase**. Sim simulates computation of each segment.
   - Sim emulates $\mathcal{F}_{\text{packed-triple}}$. Sim receives from the adversary shares of corrupted parties $\{f^z_{\ell,i}(x), g^z_{\ell,i}(y)\}_{\ell\in[N],z\in\{a,b,c\}}$ and an instruction, and records them.
     - If Sim receives `accept`, Sim additionally samples random polynomials as shares of parties in $\mathcal{C}\cup\mathcal{H}'\setminus\mathcal{C}_{\texttt{active}}$, and sends back the shares of corrupted parties to the adversary on behalf of $\mathcal{F}_{\text{packed-triple}}$.
     - If Sim receives `reject` and a new disputed pair E, Sim outputs E to the adversary on behalf of $\mathcal{F}_{\text{packed-triple}}$.
   - For $\Sigma\in\{\Sigma_5,\Sigma_2,\Sigma_3\}$, Sim emulates $\mathcal{F}_{\text{rand-sharing}}(\Sigma)$. Sim receives from the adversary shares of corrupted parties and an instruction, and records them.
     - If Sim receives `accept`, Sim additionally samples random shares as shares of parties in $\mathcal{C}\cup\mathcal{H}'\setminus\mathcal{C}_{\texttt{active}}$, and sends back the shares of corrupted parties to the adversary on behalf of $\mathcal{F}_{\text{rand-sharing}}$.
     - If Sim receives `reject` and a disputed pair E, Sim outputs E to the adversary on behalf of $\mathcal{F}_{\text{rand-sharing}}$.
2. **Online phase.** Sim simulates computation of each segment. In case of success, before starting simulating computation of next segment, Sim records shares of the output wires in this segment held by t parties in $\mathcal{C}\cup\mathcal{H}'$ for the future simulation.
   - For an input segment with inputs belonging to P_d, Sim invokes $\mathsf{Sim}_{\text{input}}$.
   - For a computation segment, Sim invokes $\mathsf{Sim}_{\text{eval}}$.
   - For an output segment, Sim invokes $\mathsf{Sim}_{\text{output}}$.
   - **Reconstruction.** Sim first invokes the ideal functionality $\mathcal{F}_{\text{main}}$ and provides the inputs of corrupted parties, while honest parties provide their inputs. Then Sim receives the outputs of corrupted parties from $\mathcal{F}_{\text{main}}$. For each output wire o belonging to a corrupted party, supposing the secret is stored at the j-th position, Sim first recovers the whole sharing $[o|j]_t$ using the secret o and t shares of parties in $\mathcal{C}\cup\mathcal{H}'$, and sends the shares of active honest parties to the adversary on behalf of the honest parties.
3. **Restart.** Sim emulates $\mathcal{F}^*_{\text{main-client-server}}$ and receives inputs of corrupted parties. Then Sim invokes the ideal functionality $\mathcal{F}_{\text{main}}$ and passes the inputs of corrupted parties, while honest parties provide their inputs. Then Sim receives outputs of corrupted parties and sends them to the adversary on behalf of $\mathcal{F}^*_{\text{main-client-server}}$.
4. Finally, Sim outputs what the adversary outputs.

Simulator 2: $\mathsf{Sim}_{\text{input}}$ for INPUT

1. In Step 1, Sim invokes $\mathsf{Sim}_{\text{rand-linear-mask}}$.
2. In Step 2, Sim simulates SHARE-INPUT(P_d) as follows.
   - If P_d is honest,
     (a) In Step 1, for each $\ell\in[m]$, Sim samples the shares of $[\![\boldsymbol{X}^{(\ell)}]\!]_{t+k-1}$ held by parties in $\mathcal{C}\cup\mathcal{H}'$, denoted by $(f_i^{(\ell)}(x), g_i^{(\ell)}(y))_{i\in\mathcal{C}\cup\mathcal{H}'}$ as follows. Sim randomly samples $2t$ degree-$(t+k-1)$ polynomials $f_i^{(\ell)}(x), g_i^{(\ell)}(y)$ such that
     - $f_i^{(\ell)}(\alpha_j) = g_i^{(\ell)}(j) = 0$ for each $j\in\mathsf{Conflicts}_d\cap\mathcal{P}_{\texttt{active}}$,
     - $f_i^{(\ell)}(\alpha_j) = g_j^{(\ell)}(\alpha_i)$ for all $i,j\in\mathcal{C}\cup\mathcal{H}'$.

     Then Sim sends shares of active corrupted parties to the adversary on behalf of honest P_d.

(b) In Step 2.1, Sim uses the shares of corrupted parties sampled above to simulate the behaviors of honest parties in $\textsc{Check-All}(P_d, [\![\boldsymbol{X}^{(\ell)}]\!]_{t+k-1})$.
(c) In Step 2.2, if P_i is honest, Sim doesn't complain. If P_i is corrupted, Sim follows the protocol honestly.
(d) In Step 3 and 4, Sim honestly follows the protocol.

– If P_d is corrupted, Sim honestly follows the protocol. If the check passes, Sim learns the inputs of P_d using shares of active honest parties.

3. In Step 3, Sim invokes $\mathsf{Sim}_{\text{sharing-trans}}$.
4. In Step 4.a, Sim emulates $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_9)$ by checking whether honest parties' shares of $[\boldsymbol{x}+\boldsymbol{r}]_{k-1}$ and $[\boldsymbol{L}(\boldsymbol{x}+\boldsymbol{r})]_{k-1}$ form valid degree-$(k-1)$ packed Shamir sharings. If the check passes, Sim waits for an instruction from the adversary,
   – if Sim receives `reject` and a disputed pair E, Sim outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$;
   – if Sim receives `accept`, Sim outputs `accept` on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.

   If the check fails, Sim receives a disputed pair E from the adversary, and outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.
5. In Step 4.b, Sim invokes $\mathsf{Sim}_{\text{vrfy-sharing-trans}}$.

Simulator 3: $\mathsf{Sim}_{\text{eval}}$ for $\textsc{Eval}$

1. In Step 1.1, Sim simulates each invocation of $\textsc{Zero-Sharing}$. In Step 2 of $\textsc{Zero-Sharing}$, for an active honest party P_i, Sim samples random elements as shares of active corrupted parties of $[\boldsymbol{o}^{(i)}]_{n'-1}$ and sends shares of active corrupted parties to the adversary on behalf of honest P_i. For an active corrupted party P_i, Sim receives shares of active honest parties' shares of $[\boldsymbol{o}^{(i)}]_{n'-1}$.
2. In Step 1.2, Sim invokes $\mathsf{Sim}_{\text{rand-linear-mask}}$.
3. In Step 2.1, Sim invokes $\mathsf{Sim}_{\text{select}}$ and $\mathsf{Sim}_{\text{sharing-trans}}$.
4. In Step 2.2, for addition gates, Sim computes shares of $[\boldsymbol{x}+\boldsymbol{y}]_{t+k-1}$ held by parties in $\mathcal{C} \cup \mathcal{H}'$ by $[\boldsymbol{x}+\boldsymbol{y}]_{t+k-1} \leftarrow [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{y}]_{t+k-1}$. For multiplication gates, Sim invokes $\mathsf{Sim}_{\text{mult}}$.
5. In Step 2.3, Sim invokes $\mathsf{Sim}_{\text{sharing-trans}}$.
6. In Step 3.1, Sim invokes $\mathsf{Sim}_{\text{vrfy-mult}}$.
7. In Step 3.2, Sim emulates $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_9)$ by checking whether the honest parties' shares of $[\boldsymbol{x}+\boldsymbol{r}]_{k-1}$, $[\boldsymbol{L}(\boldsymbol{x}+\boldsymbol{r})]_{k-1}$ are valid degree-$(k-1)$ packed Shamir sharings. If the check passes, Sim waits for an instruction from the adversary.
   – If Sim receives `reject` and a disputed pair E, Sim outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.
   – Otherwise, Sim outputs `accept` on behalf of $\mathcal{F}_{\text{batched-vrfy}}$. Sim reconstructs the secrets $\overline{\boldsymbol{x}+\boldsymbol{r}}$, $\overline{\boldsymbol{L}(\boldsymbol{x}+\boldsymbol{r})}$.

   If the check fails, Sim receives a disputed pair E from the adversary and outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.
8. In Step 3.3, Sim invokes $\mathsf{Sim}_{\text{vrfy-sharing-trans}}$.
9. In Step 3.4, Sim emulates $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_{10})$ by checking whether the secret of $[\boldsymbol{y}+\boldsymbol{r}]_{k-1}$ correspond to that of $[\boldsymbol{y}+\boldsymbol{r}]_{t+2k-2}$. If the check passes, Sim waits for an instruction from the adversary.
   – If Sim receives `reject` and a disputed pair E, Sim outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.
   – Otherwise, Sim outputs `accept` on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.

   If the check fails, Sim receives a disputed pair from the adversary and outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.

Simulator 4: $\mathsf{Sim}_{\text{rand-linear-mask}}$ for RAND-LINEAR-MASK

1. For every group of k linear transformations $\vec{\boldsymbol{L}} = (\boldsymbol{L}_1, \ldots, \boldsymbol{L}_k)$:
   (a) In Step 1.3.a,
   - for each corrupted P_i, Sim simulates the value $f_j(\alpha_i)$ for each active honest party P_j as follows. Sim computes the value $g_i(\alpha_\ell)$ held by each party P_ℓ in $\mathcal{C} \cup \mathcal{H}'$, which is a linear combination of its shares of $\{[\boldsymbol{u}_{*,\ell}]_{t+2k-2}\}_{\ell\in[k]}$ and $\{[\![\boldsymbol{v}_\ell]\!]_{t+2k-2}\}_{\ell\in[t+k-1]}$. Then Sim samples a random degree-$(t+2k-2)$ polynomial as g_i according to the above computed values. Then for each active honest party P_j, Sim computes $f_j(\alpha_i) \leftarrow g_i(\alpha_j)$ and sends it to the adversary on behalf of honest P_j.
   - for each honest P_i, Sim receives corrupted parties' values $\{f_j(\alpha_i)\}_{j\in\mathcal{C}_{\texttt{active}}}$ from the adversary.

   (b) In Step 1.3.b, for each honest P_i, Sim checks whether checks whether the adversary sends the correct value. If not, Sim broadcasts $(P_i, \texttt{Complain})$ on behalf of honest P_i. If P_i broadcasts $(P_i, \texttt{Complain})$, the simulation goes as follows.
   - In Step A, Sim first samples all honest parties' shares of $\{[\![\boldsymbol{r}_{*,\ell}]\!]_{t+k-1}\}_{\ell\in[k]}$ based on shares of parties in $\mathcal{C}\cup\mathcal{H}'$, then computes honest parties' shares of $\{[\boldsymbol{u}_{*,\ell}]_{t+2k-2}\}_{\ell\in[k]}$ and $\{[\![\boldsymbol{v}_\ell]\!]_{t+2k-2}\}_{\ell\in[t+k-1]}$, and sends them to the adversary on behalf of honest parties.
   - In Step B, if P_i is honest, Sim follows the protocol honestly.
   - In Step C, if P_i is honest, Sim follows the protocol honestly. If P_i broadcasts j to indicate the j-th sharing, denoted by $[\boldsymbol{u}]_{t+2k-2}$, is inconsistent, Sim invokes $\mathsf{Sim}_{\text{analyze-sharing}}(P_i, [\boldsymbol{u}]_{t+2k-2})$.

   Otherwise, Sim learns shares of $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})]\!]_{t+2k-2}$ held by parties in $\mathcal{C} \cup \mathcal{H}'$ and continues the simulation.
   (c) In Step 1.4.a, Sim computes shares of $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{t+2k-2}$ held by parties in $\mathcal{C}\cup\mathcal{H}'$ and randomly samples the whole sharing $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{t+2k-2}$ based on the above computed shares. Then Sim sends shares of active honest parties to P_{king} on behalf of honest parties.
   (d) In Step 1.4.b, if P_{king} is honest, Sim simulates P_{king} honestly. Then Sim also simulates CHECK-ONE$(P_{\text{king}}, [\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{t+2k-2})$ honestly.
   (e) In Step 1.4.c, if P_{king} is honest, Sim simulates P_{king} honestly. Sim receives honest parties' shares of $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{k-1}$ from P_{king}.
   (f) In Step 1.4.d, Sim simulates CHECK-ALL$(P_{\text{king}}, [\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{k-1})$ honestly.
2. In Step 2, Sim emulates $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_8)$ by checking whether the secret of $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{k-1}$ corresponds to that of $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{t+2k-2}$. If the check passes, Sim waits for an instruction from the adversary,
   - if Sim receives `reject` and a disputed pair E, Sim outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$;
   - if Sim receives `accept`, Sim outputs accept on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.

   If the check fails, Sim receives a disputed pair E from the adversary and outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.

Simulator 5: $\mathsf{Sim}_{\text{analyze-sharing}}(P_r, [\boldsymbol{u}]_{t+2k-2})$ for ANALYZE-SHARING

Suppose $[\boldsymbol{u}]_{t+2k-2} = [\boldsymbol{d}_0]_{2k-2} + [\![\boldsymbol{u}_0]\!]_{t+2k-2} + \sum_{\ell=1}^{s}[\boldsymbol{d}_\ell]_{k-1} \cdot [\![\boldsymbol{u}_\ell]\!]_{t+k-1}$. Sim has learnt active honest parties' shares of $[\boldsymbol{d}_0]_{2k-2}, [\boldsymbol{d}_1]_{k-1}, \ldots, [\boldsymbol{d}_s]_{k-1}$. If P_r is honest, Sim simulates P_r honestly.

1. In Step 3, Sim sends honest parties' shares of $[\boldsymbol{d}_0]_{2k-2}, [\boldsymbol{d}_1]_{k-1}, \ldots, [\boldsymbol{d}_s]_{k-1}$ to the adversary on behalf of honest parties.
2. In Step 4.a,
   - if P_r broadcasts `Inconsistent`, Sim randomly samples honest parties' shares of $[\![\boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{v}_s]\!]_{t+k-1}$ based on shares of parties in $\mathcal{C} \cup \mathcal{H}'$ and sends them

to P_r on behalf of honest parties. In Step 4.b, Sim honestly follows the protocol. In Step 4.c, if P_r broadcasts $(j, \texttt{Inconsistent})$, Sim invokes $\mathsf{Sim}_{\text{decompose-sharing}}(P_r, [\![\boldsymbol{v}_j]\!]_d)$.

– if P_r broadcasts `Consistent`, Sim randomly samples honest parties' shares of $[\![\boldsymbol{u}_0 + \boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{u}_1 + \boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{u}_s + \boldsymbol{v}_s]\!]_{t+k-1}$ based on shares of parties in $\mathcal{C} \cup \mathcal{H}'$ and sends them to P_r on behalf of honest parties. In Step 4.b, Sim honestly follows the protocol. In Step 4.c, if P_r broadcasts $(j, \texttt{Inconsistent})$, Sim invokes $\mathsf{Sim}_{\text{decompose-sharing}}(P_r, [\![\boldsymbol{u}_j + \boldsymbol{v}_j]\!]_d)$.

Simulator 6: $\mathsf{Sim}_{\text{decompose-sharing}}(P_r, [\![\boldsymbol{u}]\!]_d)$ for DECOMPOSE-SHARING

Suppose $[\![\boldsymbol{u}]\!]_d = [\![\boldsymbol{U}^{(1)}_{1,*}]\!]_d + [\![\boldsymbol{U}^{(2)}_{2,*}]\!]_d + \cdots + [\![\boldsymbol{U}^{(k)}_{k,*}]\!]_d$. If P_r is honest, Sim simulates P_r honestly.

1. In Step 3, Sim first computes corrupted parties' shares of $[\![\boldsymbol{w}]\!]_d$ and randomly samples a degree-d packed Shamir sharing as $[\![\boldsymbol{w}]\!]_d$ based on shares of corrupted parties. Then Sim broadcasts honest parties' shares of $[\![\boldsymbol{w}]\!]_d$ on behalf of honest parties.
2. In Step 4, if P_r is honest, Sim simulates P_r honestly.
   – If P_r broadcasts `Inconsistent`, Sim randomly samples $[\![\boldsymbol{W}^{(1)}]\!]_d, [\![\boldsymbol{W}^{(2)}]\!]_d, \ldots, [\![\boldsymbol{W}^{(k)}]\!]_d$ based on shares of parties in $\mathcal{C} \cup \mathcal{H}'$ and satisfying that $\sum_{\ell=1}^{k} [\![\boldsymbol{W}^{(1)}_{1,*}]\!]_d = [\![\boldsymbol{w}]\!]_d$. Then Sim simulates CHECK-ONE$(P_r, [\![\boldsymbol{W}^{(\ell)}]\!]_d)$ honestly for each $\ell \in [k]$.
   – If P_r broadcasts `Consistent`, Sim randomly samples $[\![\boldsymbol{U}^{(1)} + \boldsymbol{W}^{(1)}]\!]_d, [\![\boldsymbol{U}^{(2)} + \boldsymbol{W}^{(2)}]\!]_d, \ldots, [\![\boldsymbol{U}^{(k)} + \boldsymbol{W}^{(k)}]\!]_d$ based on shares of parties in $\mathcal{C} \cup \mathcal{H}'$ and satisfying that $\sum_{\ell=1}^{k} [\![\boldsymbol{U}^{(\ell)}_{\ell,*} + \boldsymbol{W}^{(\ell)}_{\ell,*}]\!]_d = [\![\boldsymbol{u} + \boldsymbol{w}]\!]_d$. Then Sim simulates CHECK-ONE$(P_r, [\![\boldsymbol{U}^{(\ell)} + \boldsymbol{W}^{(\ell)}]\!]_d)$ honestly for each $\ell \in [k]$.

Simulator 7: $\mathsf{Sim}_{\text{sharing-trans}}$ for SHARING-TRANS

Sim has learnt shares $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\boldsymbol{L}(\boldsymbol{r})]\!]_{t+k-1})$ held by parties in $\mathcal{C} \cup \mathcal{H}'$.

1. In Step 1, Sim simulates active honest parties' shares of $[\boldsymbol{x}+\boldsymbol{r}]_{t+k-1}$ as follows. Suppose $[\boldsymbol{x}]_{t+k-1} = [\boldsymbol{x}']_{k-1} + [\![\boldsymbol{x}'']\!]_{t+k-1}$. Sim first computes shares of $[\boldsymbol{x}'' + \boldsymbol{r}]_{t+k-1}$ held by parties in $\mathcal{C} \cup \mathcal{H}'$, and samples a random degree-$(t+k-1)$ packed Shamir sharing as $[\boldsymbol{x}''+\boldsymbol{r}]_{t+k-1}$ based on the computed shares above. Then Sim computes active honest parties' shares of $[\boldsymbol{x}+\boldsymbol{r}]_{t+k-1}$ by $[\boldsymbol{x}+\boldsymbol{r}]_{t+k-1} \leftarrow [\boldsymbol{x}']_{k-1} + [\boldsymbol{x}+\boldsymbol{r}]_{t+k-1}$ and sends them to P_{king} on behalf of honest parties.
2. In Step 2, if P_{king} is honest, Sim honestly follows the protocol. If P_{king} broadcasts `Inconsistent`, Sim invokes $\mathsf{Sim}_{\text{analyze-sharing}}(P_{\text{king}}, [\boldsymbol{x}+\boldsymbol{r}]_{t+k-1})$.
3. In Step 3, if P_{king} is honest, Sim honestly follows the protocol. Sim receives from P_{king} active honest parties' shares of $[\boldsymbol{x}+\boldsymbol{r}]_{k-1}$, $[\boldsymbol{L}(\boldsymbol{x}+\boldsymbol{r})]_{k-1}$.

Simulator 8: $\mathsf{Sim}_{\text{vrfy-sharing-trans}}$ for VRFY-SHARING-TRANS

Sim learnt active honest parties' shares of $([\boldsymbol{u}_1]_{k-1}, [\boldsymbol{L}_1(\boldsymbol{u}_1)]_{k-1}), \ldots, ([\boldsymbol{u}_k]_{k-1}, [\boldsymbol{L}_k(\boldsymbol{u}_k)]_{k-1})$.

1. In Step 1, 2, 3 and 4, Sim simulates behaviors of honest parties honestly. Then Sim learns honest parties' shares of $[\boldsymbol{o}_\ell]_{k-1}$ for $\ell \in [k]$.
2. In Step 5, Sim emulates $\mathcal{F}_{\text{batched-vrfy}}(P_r, \Sigma_{11}, k)$ by checking whether honest parties' shares of $[\boldsymbol{o}_\ell]_{k-1}$ form valid degree-$(k-1)$ packed Shamir sharings with secret $\boldsymbol{o}_\ell = \boldsymbol{0}$ for each $\ell \in [k]$. If the check passes, Sim waits for an instruction from the adversary.
   – If Sim receives `reject` and a disputed pair E, Sim outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.
   – Otherwise, Sim outputs `accept` on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.

   If the check fails, Sim receives a disputed pair E from the adversary, and outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.

Simulator 9: $\mathsf{Sim}_{\text{select}}$ for SELECT

Sim has learnt $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\boldsymbol{r}]\!]_{t+2k-2})$ held by parties in $\mathcal{C} \cup \mathcal{H}'$.

1. In Step 2.a, Sim simulates active honest parties' shares of $[\boldsymbol{y}+\boldsymbol{r}]_{t+2k-2}$ as follows. For $\ell \in [k]$, suppose $[\boldsymbol{x}_\ell]_{t+k-1} = [\boldsymbol{x}'_\ell]_{k-1} + [\![\boldsymbol{x}''_\ell]\!]_{t+k-1}$. Let $[\boldsymbol{z}]_{2k-2} = \sum_{\ell=1}^{k} [\boldsymbol{e}_\ell]_{k-1} \cdot [\boldsymbol{x}'_\ell]_{k-1}$ and $[\boldsymbol{w}]_{t+2k-2} = \sum_{\ell=1}^{k} [\boldsymbol{e}_\ell]_{k-1} \cdot [\![\boldsymbol{x}''_\ell]\!]_{t+k-1} + [\![\boldsymbol{r}]\!]_{t+2k-2}$. Sim first computes active honest parties' shares of $[\boldsymbol{z}]_{2k-2}$. Then Sim computes shares of $[\boldsymbol{w}]_{t+2k-2}$ held by parties in $\mathcal{C} \cup \mathcal{H}'$, samples a random degree-$(t+2k-2)$ packed Shamir sharing as $[\boldsymbol{w}]_{t+2k-2}$ based on the computed shares above, computes active honest parties' shares of $[\boldsymbol{y}+\boldsymbol{r}]_{t+2k-2}$ by $[\boldsymbol{y}+\boldsymbol{r}]_{t+2k-2} \leftarrow [\boldsymbol{z}]_{2k-2} + [\boldsymbol{w}]_{t+2k-2}$, and sends them to P_{king} on behalf of honest parties.
2. In Step 2.b, if P_{king} is honest, Sim simulates P_{king} honestly. If P_{king} broadcasts `Inconsistent`, Sim invokes $\mathsf{Sim}_{\text{analyze-sharing}}(P_{\text{king}}, [\boldsymbol{y}+\boldsymbol{r}]_{t+2k-2})$.
3. In Step 2.c, if P_{king} is honest, Sim simulates P_{king} honestly. Sim receives active honest parties' shares of $[\boldsymbol{y}+\boldsymbol{r}]_{k-1}$ from P_{king}.

Simulator 10: $\mathsf{Sim}_{\text{mult}}$ for MULT

Sim has learnt shares of $([\![\boldsymbol{a}]\!]_{t+k-1}, [\![\boldsymbol{b}]\!]_{t+k-1}, [\![\boldsymbol{c}]\!]_{t+k-1})$ and $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\boldsymbol{r}]\!]_{t+2k-2})$ held by parties in $\mathcal{C} \cup \mathcal{H}'$.

1. In Step 1, Sim simulates honest parties' shares of $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$ as follows. Suppose $[\boldsymbol{x}]_{t+k-1} = [\boldsymbol{x}']_{k-1} + [\![\boldsymbol{x}'']\!]_{t+k-1}$. Sim has learnt active honest parties' shares of $[\boldsymbol{x}']_{k-1}$ and shares of $[\![\boldsymbol{x}''+\boldsymbol{a}]\!]_{t+k-1}$ held by parties in $\mathcal{C} \cup \mathcal{H}'$. Then Sim samples a random degree-$(t+k-1)$ packed Shamir sharing as $[\![\boldsymbol{x}''+\boldsymbol{a}]\!]_{t+k-1}$ based on the computed shares above and computes active parties' shares of $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}$ by $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1} \leftarrow [\boldsymbol{x}']_{k-1} + [\![\boldsymbol{x}''+\boldsymbol{a}]\!]_{t+k-1}$. Similarly, $\mathcal{S}$ simulates active honest parties' shares of $[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$. Sim sends active honest parties' shares of $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$ to P_{king} on behalf of honest parties.
2. In Step 2, if P_{king} is honest, Sim simulates P_{king} honestly. If P_{king} broadcasts $(i, \texttt{Inconsistent})$ to indicate the i-th sharing, denoted by $[\boldsymbol{u}]_{t+k-1}$, is inconsistent, Sim invokes $\mathsf{Sim}_{\text{analyze-sharing}}(P_{\text{king}}, [\boldsymbol{u}]_{t+k-1})$.
3. In Step 3, if P_{king} is honest, Sim simulates P_{king} honestly. Sim receives active honest parties' shares of $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ from P_{king}.
4. In Step 4, Sim samples random elements as active honest parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{n'-1}$ and sends them to P_{king} on behalf of honest parties.
5. In Step 5, if P_{king} is honest, Sim simulates P_{king} honestly. Sim receives active honest parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$.

Simulator 11: $\mathsf{Sim}_{\text{vrfy-mult}}$ for VRFY-MULT

1. When simulating each invocation of MULT, Sim has learnt honest parties' shares of $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$, and $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$. In Step 1, Sim emulates $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_9)$ by checking whether honest parties' shares of $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ and $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ form valid degree-$(k-1)$ packed Shamir sharings. If the check passes, Sim waits for an instruction from the adversary.
   - If Sim receives `reject` and a disputed pair E, Sim outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.
   - Otherwise, Sim outputs `accept` on behalf of $\mathcal{F}_{\text{batched-vrfy}}$. Sim recovers the whole sharings $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}$, $[\boldsymbol{y}+\boldsymbol{b}]_{k-1}$, $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ using the shares of honest parties and then reconstructs the secrets $\overline{\boldsymbol{x}+\boldsymbol{a}}$, $\overline{\boldsymbol{y}+\boldsymbol{b}}$, $\overline{\boldsymbol{z}+\boldsymbol{r}}$.

   If the check fails, Sim receives a disputed pair E from the adversary, and outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.

2. In Step 2, Sim simulates honest parties' shares of $[\boldsymbol{o}]_{n'-1}$ as follows. Sim first defaults[a] corrupted parties' shares of $[\boldsymbol{o}]_{n'-1}$, computes corrupted parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$, computes corrupted parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{n'-1}$ by $[\boldsymbol{z}+\boldsymbol{r}]_{n'-1} \leftarrow [\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}+[\boldsymbol{o}]_{n'-1}$, and learns the whole sharing of $[\boldsymbol{z}+\boldsymbol{r}]_{n'-1}$. Then Sim reconstructs the secret $\boldsymbol{z}+\boldsymbol{r}$ using the sharing $[\boldsymbol{z}+\boldsymbol{r}]_{n'-1}$ and samples honest parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$ based on the shares of parties in $\mathcal{C}\cup\mathcal{H}'$ and the secret $\boldsymbol{z}+\boldsymbol{r}$. Then Sim computes honest parties' shares of $[\boldsymbol{o}]_{n'-1}$ by $[\boldsymbol{o}]_{n'-1} \leftarrow [\boldsymbol{z}+\boldsymbol{r}]_{n'-1} - [\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$ and sends active honest parties' shares of $[\boldsymbol{o}]_{n'-1}$ to P_{king} on behalf of honest parties. If P_{king} is honest, Sim honestly follows the protocol.
   - If P_{king} broadcasts `Inconsistent`, Sim simulates honest parties' shares of $([\boldsymbol{o}^{(i)}]_{n'-1})_{i\in\mathcal{H}\cap\mathcal{P}_{\texttt{active}}}$ as follows. Let $\mathcal{H}''$ denote the set of active honest parties with $n'-t$ smallest identities. For each active honest party P_i in $\mathcal{H}\cap\mathcal{P}_{\texttt{active}}\setminus\mathcal{H}''$, Sim samples a random degree-$(n'-1)$ packed Shamir sharing as $[\boldsymbol{o}^{(i)}]_{n'-1}$ based on shares of active corrupted parties. Then Sim computes $([\boldsymbol{o}^{(i)}]_{n'-1})^\top_{i\in\mathcal{H}''} \leftarrow \boldsymbol{M}^{-1}_{\mathcal{H}''}\cdot(([\boldsymbol{o}_i]_{n'-1})^\top_{i\in[n'-t]} - \boldsymbol{M}_{\overline{\mathcal{H}''}}\cdot([\boldsymbol{o}^{(i)}]_{n'-1})^\top_{i\in\overline{\mathcal{H}''}})$, where $\overline{\mathcal{H}''} = \mathcal{P}_{\texttt{active}}\setminus\mathcal{H}''$. Then Sim sends honest parties' transcripts produced in Zero-Sharing to P_{king} on behalf of honest parties.
3. In Step 3, if P_{king} is honest, Sim honestly follows the protocol. If P_{king} broadcasts `Inconsistent`, Sim invokes $\mathsf{Sim}_{\text{analyze-sharing}}(P_{\text{king}}, [\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2})$.
4. In Step 4, Sim emulates $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_{10})$ by checking whether the secrets of $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}$, $[\boldsymbol{y}+\boldsymbol{b}]_{k-1}$, and $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$, which are $\overline{\boldsymbol{x}+\boldsymbol{a}}$, $\overline{\boldsymbol{y}+\boldsymbol{b}}$, and $\overline{\boldsymbol{z}+\boldsymbol{r}}$, correspond to that of $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}$, $[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$, and $[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$ accordingly. If the check passes, Sim waits for an instruction from the adversary.
   - If Sim receives `reject` and a disputed pair E, Sim outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.
   - Otherwise, Sim outputs `accept` on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.

   If the check fails, Sim receives a disputed pair from the adversary and outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.

[a] When simulating Zero-Sharing, for each honest party P_i, Sim has sampled shares of corrupted parties of $[\boldsymbol{o}^{(i)}]_{n'-1}$. For each corrupted party P_i, Sim has learnt honest parties' shares of $[\boldsymbol{o}^{(i)}]_{n'-1}$. Then Sim defaults all but one corrupted parties' shares of $[\boldsymbol{o}^{(i)}]_{n'-1}$ to be 0 and sets the last share to satisfy that all shares form a degree-$(n'-1)$ packed Shamir sharing with secret $\boldsymbol{o}^{(i)} = \boldsymbol{0}$. Then Sim computes corrupted parties' shares of $([\boldsymbol{o}_i]_{n'-1})_{i\in[n'-t']}$, which are linear combinations of their shares of $([\boldsymbol{o}^{(i)}]_{n'-1})_{i\in\mathcal{P}_{\texttt{active}}}$.

Simulator 12: $\mathsf{Sim}_{\text{output}}$ for Output

Sim learnt shares of $(\{[r_j|j]_t\}_{j\in[k]}, [\![\boldsymbol{r}]\!]_{t+k-1})$ held by parties in $\mathcal{C}\cup\mathcal{H}'$.

1. In Step 1, Sim invokes $\mathsf{Sim}_{\text{rand-linear-mask}}$.
2. In Step 2, for each input sharing $[\boldsymbol{x}]_{t+k-1}$, Sim invokes $\mathsf{Sim}_{\text{select}}$ and $\mathsf{Sim}_{\text{sharing-trans}}$.
3. In Step 3.b, Sim simulates active honest parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{t+k-1}$ as follows. Suppose $[\boldsymbol{z}]_{t+k-1} = [\boldsymbol{z}']_{k-1} + [\![\boldsymbol{z}'']\!]_{t+k-1}$. Sim first computes shares of $[\![\boldsymbol{z}''+\boldsymbol{r}]\!]_{t+k-1}$ held by parties in $\mathcal{C}\cup\mathcal{H}'$, and samples a random degree-$(t+k-1)$ packed Shamir sharing as $[\![\boldsymbol{z}''+\boldsymbol{r}]\!]_{t+k-1}$ based on the shares computed above. Then Sim computes active honest parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{t+k-1}$ by $[\boldsymbol{z}+\boldsymbol{r}]_{t+k-1} \leftarrow [\boldsymbol{z}']_{k-1} + [\![\boldsymbol{z}''+\boldsymbol{r}]\!]_{t+k-1}$ and sends them to P_{king} on behalf of honest parties.
4. In Step 3.c, if P_{king} is honest, Sim honestly follows the protocol. If P_{king} broadcasts `Inconsistent`, Sim invokes $\mathsf{Sim}_{\text{analyze-sharing}}(P_{\text{king}}, [\boldsymbol{z}+\boldsymbol{r}]_{t+k-1})$.
5. In Step 3.d, if P_{king} is honest, Sim honestly follows the protocol. Sim receives honest parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ from P_{king}.
6. In Step 4.a, Sim emulates $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_9)$ by checking whether honest parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ form valid degree-$(k-1)$ packed Shamir sharings. If the check passes, Sim waits for an instruction from the adversary.

- If Sim receives `reject` and a disputed pair E, Sim outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.
- Otherwise, Sim outputs `accept` on behalf of $\mathcal{F}_{\text{batched-vrfy}}$. Sim recovers the whole sharings $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ using the shares of honest parties.

If the check fails, Sim receives a disputed pair E from the adversary, and outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.

7. In Step 4.b, Sim invokes $\mathsf{Sim}_{\text{vrfy-sharing-trans}}$.
8. In Step 4.c, Sim emulates $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_{10})$ by checking whether the secret of $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ corresponds to that of $[\boldsymbol{z}+\boldsymbol{r}]_{t+k-1}$. If the check passes, Sim waits for an instruction from the adversary.
   - If Sim receives `reject` and a disputed pair E, Sim outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.
   - Otherwise, Sim outputs `accept` on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.

   If the check fails, Sim receives a disputed pair E from the adversary, and outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$.
9. If all checks pass, Sim computes shares of $[z_j|j]_t$ held by parties in $\mathcal{C}\cup\mathcal{H}'$ as $[z_j|j]_t \leftarrow [\boldsymbol{z}+\boldsymbol{r}]_{k-1} - [r_j|j]_t$ for $j \in [k]$.

Hybrid Argument. Now we show that Sim perfectly simulates the behaviors of honest parties. Consider the following hybrids.

$\mathbf{Hybrid}_0$: The execution in the real world.

$\mathbf{Hybrid}_1$: In this hybrid, Sim simulates the **Preprocessing phase** as in Step 1 of Sim. Compared with $\mathbf{Hybrid}_0$, the only change is that, instead of invoking the hybrid functionalities $\mathcal{F}_{\text{packed-triple}}$, $\mathcal{F}_{\text{rand-sharing}}(\Sigma_5)$, $\mathcal{F}_{\text{rand-sharing}}(\Sigma_2)$, and $\mathcal{F}_{\text{rand-sharing}}(\Sigma_3)$, Sim emulates them by itself. Notice that the emulation doesn't utilize inputs of honest parties. Then Sim generates shares of honest parties in $\mathcal{P}_{\texttt{active}} \cap \mathcal{H} \setminus \mathcal{H}'$. Hence, the distribution of $\mathbf{Hybrid}_1$ is identical to that of $\mathbf{Hybrid}_0$.

$\mathbf{Hybrid}_2$: In this hybrid, we focus on simulation of Input.

$\mathbf{Hybrid}_{2,1}$: In this hybrid, we focus on simulation of Rand-Linear-Mask.

$\mathbf{Hybrid}_{2,1,1}$: In this hybrid, in Step 1.3.a, for each active corrupted party P_i and active honest party P_j, we change the computation of $f_j(\alpha_i)$. Sim first computes $g_i(\alpha_\ell)$ for $\ell \in \mathcal{C}\cup\mathcal{H}'$, samples a random degree-$(t+2k-2)$ polynomial as g_i under the constraint that $g_i(\alpha_\ell)$ for every $\ell \in \mathcal{C}\cup\mathcal{H}'$. For each active honest party P_j, Sim computes $\{[\![\boldsymbol{v}_\ell]\!]^j_{t+2k-2}\}_{\ell\in[t+k-1]}$ as linear combinations of $\{[\![\boldsymbol{r}_{*,\ell}]\!]^j_{t+k-1}\}_{\ell\in[k]}$ and $\{g_i(\alpha_j)\}_{i\in\mathcal{C}_{\texttt{active}}}$. This does not change the output distribution since there is a one-to-one correspondence between $\{[\![\boldsymbol{v}_\ell]\!]_{t+2k-2}\}_{\ell\in[t+k-1]}$ and $\{g_i\}_{i\in\mathcal{C}_{\texttt{active}}}$ given $\{[\![\boldsymbol{r}_{*,\ell}]\!]_{t+k-1}\}_{\ell\in[k]}$. Hence, the distribution of $\mathbf{Hybrid}_{2,1,1}$ is identical to that of $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_{2,1,2}$: In this hybrid, in Step 4.a, we change the computation of $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{t+2k-2}$. Sim first computes shares of $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{t+2k-2}$ held by parties in $\mathcal{C}\cup\mathcal{H}'$ and samples a random degree-$(t+2k-2, t+2k-2)$ packed Shamir sharing as $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{t+2k-2}$ based on computed shares above. Then Sim computes the sharing $[\![\boldsymbol{W}]\!]_{t+2k-2}$ by $[\![\boldsymbol{W}]\!]_{t+2k-2} \leftarrow [\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{t+2k-2} - [\![\vec{\boldsymbol{L}}(\boldsymbol{R})]\!]_{t+2k-2}$. This change does not change the distribution due to the randomness of $[\![\boldsymbol{W}]\!]_{t+2k-2}$. Hence, the distribution of $\mathbf{Hybrid}_{2,1,2}$ is identical to that of $\mathbf{Hybrid}_{2,1,1}$.

$\mathbf{Hybrid}_{2,1,3}$: In this hybrid, in Step 2, we change the simulation of $\mathcal{F}_{\text{batched-vrfy}}$. Instead of invoking $\mathcal{F}_{\text{batched-vrfy}}$, Sim emulates itself by checking whether the secret of $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{k-1}$ corresponds to that of $[\![\vec{\boldsymbol{L}}(\boldsymbol{R})+\boldsymbol{W}]\!]_{t+2k-2}$. This does not change the output distribution. Hence, the distribution of $\mathbf{Hybrid}_{2,1,3}$ is identical to that of $\mathbf{Hybrid}_{2,1,2}$.

$\mathbf{Hybrid}_{2,2}$: In this hybrid, we focus on simulation of Share-Input(P_d).

$\mathbf{Hybrid}_{2,2,1}$: In this hybrid, in Step 1, if P_d is honest, we change the computation of $[\![\boldsymbol{X}^{(\ell)}]\!]_{t+k-1}$ for each $\ell \in [m]$. Sim first samples random shares as shares of $[\![\boldsymbol{X}^{(\ell)}]\!]_{t+k-1}$ held by parties in $\mathcal{C}\cup\mathcal{H}'$ and then generates the whole sharing based on the sampled shares and the secret. The distribution of $\mathbf{Hybrid}_{2,2,1}$ is identical to that of $\mathbf{Hybrid}_{2,1,3}$.

$\mathbf{Hybrid}_{2,2,2}$: In this hybrid, in Step 2.1, if P_d is honest, for each $\ell \in [m]$, we change the simulation of CHECK-ALL$(P_d, [\![\boldsymbol{X}^{(\ell)}]\!]_{t+k-1})$. Sim uses the sampled shares held by parties in $\mathcal{C} \cup \mathcal{H}'$ to simulate the behaviors of active honest parties and delay the sampling of all shares of honest parties until the end of INPUT. Since P_d is honest, the distribution of $\mathbf{Hybrid}_{2,2,2}$ is identical to that of $\mathbf{Hybrid}_{2,2,1}$.

$\mathbf{Hybrid}_{2,3}$: In this hybrid, we focus on simulation of SHARING-TRANS.

$\mathbf{Hybrid}_{2,3,1}$: In this hybrid, in Step 1, we change the computation of $[\boldsymbol{x}+\boldsymbol{r}]_{t+k-1}$. Let $[\boldsymbol{x}+\boldsymbol{r}]_{t+k-1} = [\boldsymbol{x}']_{k-1} + [\![\boldsymbol{x}'']\!]_{t+k-1}$. Sim first computes shares of $[\![\boldsymbol{x}''+\boldsymbol{r}]\!]_{t+k-1}$ held by parties in $\mathcal{C} \cup \mathcal{H}'$, samples a random degree-$(t+k-1)$ packed Shamir sharing as $[\![\boldsymbol{x}''+\boldsymbol{r}]\!]_{t+k-1}$ based on the computed shares above. Then Sim computes active honest parties' shares of $[\boldsymbol{x}+\boldsymbol{r}]_{t+k-1}$ by $[\boldsymbol{x}+\boldsymbol{r}]_{t+k-1} \leftarrow [\boldsymbol{x}']_{k-1} + [\![\boldsymbol{x}''+\boldsymbol{r}]\!]_{t+k-1}$ and their shares of $[\![\boldsymbol{r}]\!]_{t+k-1}$ by $[\![\boldsymbol{r}]\!]_{t+k-1} \leftarrow [\![\boldsymbol{x}''+\boldsymbol{r}]\!]_{t+k-1} - [\![\boldsymbol{x}'']\!]_{t+k-1}$. Due to the randomness of $[\![\boldsymbol{r}]\!]_{t+k-1}$, the distribution of $\mathbf{Hybrid}_{2,3,1}$ is identical to that of $\mathbf{Hybrid}_{2,2,2}$.

$\mathbf{Hybrid}_{2,3,2}$: In this hybrid, we focus on simulation of ANALYZE-SHARING$(P_r, [\boldsymbol{u}]_{t+2k-2})$.

$\mathbf{Hybrid}_{2,3,2,1}$: In this hybrid, in Step 4.a, if P_r broadcasts `Consistent`, we change the computation of $[\![\boldsymbol{u}_0+\boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{u}_1+\boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{u}_s+\boldsymbol{v}_s]\!]_{t+k-1}$. Sim first computes shares of $[\![\boldsymbol{u}_0+\boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{u}_1+\boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{u}_s+\boldsymbol{v}_s]\!]_{t+k-1}$ held by parties in $\mathcal{C} \cup \mathcal{H}'$, randomly samples shares of all honest parties based on the computed shares above, and computes $[\![\boldsymbol{v}_0]\!]_{t+2k-2} \leftarrow [\![\boldsymbol{u}_0+\boldsymbol{v}_0]\!]_{t+2k-2} - [\![\boldsymbol{u}_0]\!]_{t+2k-2}$ and $[\![\boldsymbol{v}_\ell]\!]_{t+k-1} \leftarrow [\![\boldsymbol{u}_\ell+\boldsymbol{v}_\ell]\!]_{t+k-1} - [\![\boldsymbol{u}_\ell]\!]_{t+k-1}$ for $\ell \in [s]$. Due to the randomness of $[\![\boldsymbol{v}_0]\!]_{t+2k-2}, [\![\boldsymbol{v}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{v}_s]\!]_{t+k-1}$, the distribution of $\mathbf{Hybrid}_{2,3,2,1}$ is identical to that of $\mathbf{Hybrid}_{2,3,1}$.

$\mathbf{Hybrid}_{2,3,2,2}$: In this hybrid, in Step 4.a, if P_r broadcasts `Consistent`, we focus on the simulation of DECOMPOSE-SHARING$(P_r, [\![\boldsymbol{u}]\!]_d)$.

$\mathbf{Hybrid}_{2,3,2,2,1}$: In this hybrid, in Step 4, if P_r broadcasts `Inconsistent`, we change the computation of $\{[\![\boldsymbol{W}^{(\ell)}]\!]_d\}_{\ell\in[k]}$. Sim first samples a random degree-d packed Shamir sharing as $[\![\boldsymbol{w}]\!]_d$ based on shares of $[\![\boldsymbol{w}]\!]_d$ held by parties in $\mathcal{C} \cup \mathcal{H}'$. Then Sim randomly samples $\{[\![\boldsymbol{W}^{(\ell)}]\!]_d\}_{\ell\in[k]}$ based on shares of $[\![\boldsymbol{w}]\!]_d$ held by parties in $\mathcal{C} \cup \mathcal{H}'$ and under the constraint $\sum_{\ell\in[k]} [\![\boldsymbol{W}^{(\ell)}_{\ell,*}]\!]_d = [\![\boldsymbol{w}]\!]_d$. The distribution of $\mathbf{Hybrid}_{2,3,2,2,1}$ is identical to that of $\mathbf{Hybrid}_{2,3,2,1}$.

$\mathbf{Hybrid}_{2,3,2,2,2}$: In this hybrid, in Step 4, if P_r broadcasts `Consistent`, we change the computation of $\{[\![\boldsymbol{U}^{(\ell)}+\boldsymbol{W}^{(\ell)}]\!]_d\}_{\ell\in[k]}$. Sim first randomly samples a degree-d packed Shamir sharing as $[\![\boldsymbol{u}+\boldsymbol{w}]\!]_d$ based on shares of parties in $\mathcal{C} \cup \mathcal{H}'$. Then Sim randomly samples $\{[\![\boldsymbol{U}^{(\ell)}+\boldsymbol{W}^{(\ell)}]\!]_d\}_{\ell\in[k]}$ based on shares of $[\![\boldsymbol{u}+\boldsymbol{w}]\!]_d$ held by parties in $\mathcal{C} \cup \mathcal{H}'$ and under the constraint $\sum_{\ell\in[k]} [\![\boldsymbol{U}^{(\ell)}_{\ell,*}+\boldsymbol{W}^{(\ell)}_{\ell,*}]\!]_d = [\![\boldsymbol{u}+\boldsymbol{w}]\!]_d$. For $\ell \in [k]$, Sim computes $[\![\boldsymbol{W}^{(\ell)}]\!]_d \leftarrow [\![\boldsymbol{U}^{(\ell)}+\boldsymbol{W}^{(\ell)}]\!]_d - [\![\boldsymbol{U}^{(\ell)}]\!]_d$. Due to the randomness of $\{[\![\boldsymbol{W}^{(\ell)}]\!]_d\}_{\ell\in[k]}$, the distribution of $\mathbf{Hybrid}_{2,3,2,2,2}$ is identical to that of $\mathbf{Hybrid}_{2,3,2,2,1}$.

$\mathbf{Hybrid}_{2,4}$: In this hybrid, in Step 4.a, instead of invoking $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_9)$, Sim emulates by itself by checking whether shares of active honest parties of $[\boldsymbol{x}+\boldsymbol{r}]_{k-1}$ and $[\boldsymbol{L}(\boldsymbol{x}+\boldsymbol{r})]_{k-1}$ form valid degree-$(k-1)$ packed Shamir sharings. The distribution of $\mathbf{Hybrid}_{2,4}$ is identical to that of $\mathbf{Hybrid}_{2,3,2,2,2}$.

$\mathbf{Hybrid}_{2,5}$: In this hybrid, we focus on simulation of VRFY-SHARING-TRANS.

$\mathbf{Hybrid}_{2,5,1}$: In this hybrid, in Step 5, instead of invoking $\mathcal{F}_{\text{batched-vrfy}}(P_r, \Sigma_{11})$, Sim emulates by itself by checking whether active honest parties' shares of $[\boldsymbol{o}_\ell]_{k-1}$ form valid degree-$(k-1)$ packed Shamir sharing with secret $\boldsymbol{o}_\ell = \boldsymbol{0}$ for each $\ell \in [k]$. The distribution of $\mathbf{Hybrid}_{2,5,1}$ is identical to that of $\mathbf{Hybrid}_{2,4}$.

$\mathbf{Hybrid}_{2,6}$: In this hybrid, if all checks pass, we delay the sampling of whole input sharings distributed by honest parties until the end of INPUT. The distribution of $\mathbf{Hybrid}_{2,6}$ is identical to that of $\mathbf{Hybrid}_{2,5,1}$.

$\mathbf{Hybrif}_3$: In this hybrid, we focus on simulation of EVAL.

$\mathbf{Hybrid}_{3,1,1}$: In this hybrid, we focus on simulation of ZERO-SHARING. In Step 2 of ZERO-SHARING, for each active honest party P_i, Sim first samples random elements as shares of active corrupted parties of $[\boldsymbol{o}^{(i)}]_{n'-1}$ and then samples the whole sharing $[\boldsymbol{o}^{(i)}]_{n'-1}$. The distribution of $\mathbf{Hybrid}_{3,1,1}$ is identical to that of $\mathbf{Hybrid}_{2,6}$.

$\mathbf{Hybrid}_{3,1,2}$: In this hybrid, we change the simulation of RAND-LINEAR-MASK as in $\mathsf{Sim}_{\text{rand-linear-mask}}$ and delay the sampling of whole sharing $([\![\boldsymbol{R}]\!]_{t+k-1}, [\![\vec{\boldsymbol{L}}(\boldsymbol{R})]\!]_{t+k-1})$ until the end of RAND-LINEAR-MASK. The distribution of $\mathbf{Hybrid}_{3,1,2}$ is identical to that of $\mathbf{Hybrid}_{3,1,1}$ due to the same reasons as $\mathbf{Hybrid}_{2,1}$.

$\mathbf{Hybrid}_{3,2,1,1}$: In this hybrid, we focus on simulation of SELECT.

$\mathbf{Hybrid}_{3,2,1,1,1}$: In this hybrid, in Step 2.a, we change the computation of $[\boldsymbol{y}+\boldsymbol{r}]_{t+2k-2}$. For $\ell \in [k]$, suppose $[\boldsymbol{x}_\ell]_{t+k-1} = [\boldsymbol{x}'_\ell]_{k-1} + [\![\boldsymbol{x}''_\ell]\!]_{t+k-1}$. Let $[\boldsymbol{z}]_{2k-2} = \sum_{\ell=1}^{k} [\boldsymbol{e}_\ell]_{k-1} \cdot [\boldsymbol{x}'_\ell]_{k-1}$ and $[\boldsymbol{w}]_{t+2k-2} = \sum_{\ell=1}^{k} [\boldsymbol{e}_\ell]_{k-1} \cdot [\![\boldsymbol{x}''_\ell]\!]_{t+k-1} + [\![\boldsymbol{r}]\!]_{t+2k-2}$. Sim first computes active honest parties' shares of $[\boldsymbol{z}]_{2k-2}$. Then Sim computes shares of $[\boldsymbol{w}]_{t+2k-2}$ held by parties in $\mathcal{C} \cup \mathcal{H}'$, samples a random degree-$(t+2k-2)$ packed Shamir sharing as $[\boldsymbol{w}]_{t+2k-2}$ based on the computed shares above, computes active honest parties' shares of $[\boldsymbol{y}+\boldsymbol{r}]_{t+2k-2}$ by $[\boldsymbol{y}+\boldsymbol{r}]_{t+2k-2} \leftarrow [\boldsymbol{z}]_{2k-2} + [\boldsymbol{w}]_{t+2k-2}$. Eventually, Sim computes $[\![\boldsymbol{r}]\!]_{t+2k-2} \leftarrow [\boldsymbol{w}]_{t+2k-2} - \sum_{\ell=1}^{k} [\boldsymbol{e}_\ell]_{k-1} \cdot [\![\boldsymbol{x}_\ell]\!]_{t+k-1}$. Due to the randomness of $[\![\boldsymbol{r}]\!]_{t+k-1}$, the distribution of $\mathbf{Hybrid}_{3,2,1,1,1}$ is identical to that of $\mathbf{Hybrid}_{3,1,2}$.

$\mathbf{Hybrid}_{3,2,1,1,2}$: In this hybrid, in Step 2.b, if P_{king} broadcasts $\texttt{Inconsistent}$, we change the simulation of $\textsc{Analyze-Sharing}(P_{\text{king}}, [\boldsymbol{y}+\boldsymbol{r}]_{t+2k-2})$ as described in $\mathsf{Sim}_{\text{analyze-sharing}}$ and compute honest parties' shares of $[\![\boldsymbol{x}''_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{x}''_k]\!]_{t+k-1}$ and $[\![\boldsymbol{r}]\!]_{t+2k-2}$ until the end of $\textsc{Analyze-Sharing}$. The distribution of $\mathbf{Hybrid}_{3,2,1,1,2}$ is identical to that of $\mathbf{Hybrid}_{3,2,1,1,1}$ due to the same reasons as $\mathbf{Hybrid}_{2,3,2}$.

$\mathbf{Hybrid}_{3,2,1,1,3}$: In this hybrid, we compute honest parties' shares of $[\![\boldsymbol{x}''_1]\!]_{t+k-1}$, $\ldots$, $[\![\boldsymbol{x}''_k]\!]_{t+k-1}$ and $[\![\boldsymbol{r}]\!]_{t+2k-2}$ until the end of $\textsc{select}$. The distribution of $\mathbf{Hybrid}_{3,2,1,1,3}$ is identical to that of $\mathbf{Hybrid}_{3,2,1,1,2}$.

$\mathbf{Hybrid}_{3,2,1,2}$: In this hybrid, we change the simulation of $\textsc{Sharing-Trans}$ as $\mathsf{Sim}_{\text{sharing-trans}}$ and compute honest parties' shares of $[\boldsymbol{x}]_{t+k-1}$ and $[\![\boldsymbol{r}]\!]_{t+k-1}$ until the end of $\textsc{Sharing-Trans}$. The distribution of $\mathbf{Hybrid}_{3,2,1,2}$ is identical to that of $\mathbf{Hybrid}_{3,2,1,1,3}$ due to the same reasons as $\mathbf{Hybrid}_{2,3}$.

$\mathbf{Hybrid}_{3,2,2}$: In this hybrid, we focus on simulation of $\textsc{Mult}$.

$\mathbf{Hybrid}_{3,2,2,1}$: In this hybrid, we change the computation of $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$. Suppose $[\boldsymbol{x}]_{t+k-1} = [\boldsymbol{x}']_{k-1} + [\![\boldsymbol{x}'']\!]_{t+k-1}$. Sim has learnt active honest parties' shares of $[\boldsymbol{x}']_{k-1}$ and shares of $[\![\boldsymbol{x}''+\boldsymbol{a}]\!]_{t+k-1}$ held by parties in $\mathcal{C} \cup \mathcal{H}'$. Then Sim samples a random degree-$(t+k-1)$ packed Shamir sharing as $[\![\boldsymbol{x}''+\boldsymbol{a}]\!]_{t+k-1}$ based on the computed shares above and computes active parties' shares of $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}$ by $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1} \leftarrow [\boldsymbol{x}']_{k-1} + [\![\boldsymbol{x}''+\boldsymbol{a}]\!]_{t+k-1}$. Then Sim computes $[\![\boldsymbol{a}]\!]_{t+k-1} \leftarrow [\![\boldsymbol{x}''+\boldsymbol{a}]\!]_{t+k-1} - [\![\boldsymbol{x}'']\!]_{t+k-1}$. Similarly, Sim change the computation of $[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$ as described above. Due to the randomness of $[\![\boldsymbol{a}]\!]_{t+k-1}$ and $[\![\boldsymbol{b}]\!]_{t+k-1}$, the distribution of $\mathbf{Hybrid}_{3,2,2,1}$ is identical to that of $\mathbf{Hybrid}_{3,2,1,2}$.

$\mathbf{Hybrid}_{3,2,2,2}$: In this hybrid, in Step 2, if P_{king} broadcasts $(i, \texttt{Inconsistent})$, we change the simulation of $\textsc{Analyze-Sharing}(P_{\text{king}}, [\boldsymbol{u}]_{t+k-1})$ as described in $\mathsf{Sim}_{\text{analyze-sharing}}$ and compute honest parties' shares of $[\![\boldsymbol{x}'']\!]_{t+k-1}$, $[\![\boldsymbol{a}]\!]_{t+k-1}$, $[\![\boldsymbol{y}'']\!]_{t+k-1}$ and $[\![\boldsymbol{b}]\!]_{t+k-1}$ until the end of $\textsc{Analyze-Sharing}$. The distribution of $\mathbf{Hybrid}_{3,2,2,2}$ is identical to that of $\mathbf{Hybrid}_{3,2,2,1}$ due to the same reasons as $\mathbf{Hybrid}_{2,3,2}$.

$\mathbf{Hybrid}_{3,2,2,3}$: In this hybrid, in Step 4, we change the computation of $[\boldsymbol{z}+\boldsymbol{r}]_{n'-1}$. Sim first samples random elements as active honest parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{n'-1}$. Then Sim defaults corrupted parties' shares of $[\boldsymbol{o}]_{n'-1}$, computes their shares of $[\boldsymbol{z}+\boldsymbol{r}]_{n'-1}$ by $[\boldsymbol{z}+\boldsymbol{r}]_{n'-1} \leftarrow [\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2} + [\boldsymbol{o}]_{n'-1}$, and reconstructs the secret $\boldsymbol{z}+\boldsymbol{r}$.

Sim samples honest parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$ based on shares of parties in $\mathcal{C} \cup \mathcal{H}'$ and the secret $\boldsymbol{z}+\boldsymbol{r}$, and computes honest parties' shares of $[\![\boldsymbol{r}]\!]_{t+2k-2}$ by $[\![\boldsymbol{r}]\!]_{t+2k-2} \leftarrow [\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2} - [\boldsymbol{z}]_{t+2k-2}$.

Then Sim computes honest parties' shares of $\{[\boldsymbol{o}_i]_{n'-1}\}_{i\in[n'-t]}$ produced in $\textsc{Zero-Sharing}$ as follows. Let $\mathcal{H}''$ denote the set of active honest parties with $n'-t$ smallest identities. For each active honest party P_i in $\mathcal{H} \cap \mathcal{P}_{\texttt{active}} \setminus \mathcal{H}''$, Sim samples a random degree-$(n'-1)$ packed Shamir sharing as $[\boldsymbol{o}^{(i)}]_{n'-1}$ based on shares of active corrupted parties which have been already sampled. Then Sim computes $([\boldsymbol{o}^{(i)}]_{n'-1})^{\top}_{i\in\mathcal{H}''} \leftarrow \boldsymbol{M}^{-1}_{\mathcal{H}''} \cdot (([\boldsymbol{o}_i]_{n'-1})^{\top}_{i\in[n'-t]} - \boldsymbol{M}_{\overline{\mathcal{H}''}} \cdot ([\boldsymbol{o}^{(i)}]_{n'-1})^{\top}_{i\in\overline{\mathcal{H}''}})$, where $\overline{\mathcal{H}''} = \mathcal{P}_{\texttt{active}} \setminus \mathcal{H}''$. Due to the randomness of honest parties' shares of $\{[\boldsymbol{o}^{(i)}]_{n'-1}\}_{i\in\mathcal{H}''}$ and $[\![\boldsymbol{r}]\!]_{t+2k-2}$, the distribution of $\mathbf{Hybrid}_{3,2,2,3}$ is identical to that of $\mathbf{Hybrid}_{3,2,2,2}$.

$\mathbf{Hybrid}_{3,2,3}$: In this hybrid, we change the simulation of $\textsc{Sharing-Trans}$ as described in $\mathsf{Sim}_{\text{sharing-trans}}$ and compute honest parties' shares of $[\boldsymbol{x}]_{t+k-1}$ and $[\![\boldsymbol{r}]\!]_{t+k-1}$ until the end of $\textsc{Sharing-Trans}$. The distribution of $\mathbf{Hybrid}_{3,2,3}$ is identical to that of $\mathbf{Hybrid}_{3,2,2,3}$ due to the same reasons as $\mathbf{Hybrid}_{2,3}$.

$\mathbf{Hybrid}_{3,3,1}$: In this hybrid, we focus on the simulation of $\textsc{Vrfy-Mult}$.

$\mathbf{Hybrid}_{3,3,1,1}$: In this hybrid, in Step 1, instead of invoking $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_9)$, Sim emulates by itself by checking whether honest parties' shares of $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ and $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ form valid degree-$(k-1)$ packed Shamir sharings. The distribution of $\mathbf{Hybrid}_{3,3,1,1}$ is identical to that of $\mathbf{Hybrid}_{3,2,3}$.

$\mathbf{Hybrid}_{3,3,1,2}$: In this hybrid, in Step 3, if P_{king} broadcasts $\texttt{Inconsistent}$, we change the simulation of $\textsc{Analyze-Sharing}(P_{\text{king}}, [\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2})$ as described in $\mathsf{Sim}_{\text{analyze-sharing}}$ and compute honest parties' shares

of $[\![\boldsymbol{r}]\!]_{t+2k-2}$ until the end of ANALYZE-SHARING. Due to the same reasons as $\mathbf{Hybrid}_{2,3,2}$, the distribution of $\mathbf{Hybrid}_{3,3,1,2}$ is identical to that of $\mathbf{Hybrid}_{3,3,1,1}$.

$\mathbf{Hybrid}_{3,3,1,3}$: In this hybrid, in Step 4, instead of invoking $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_{10})$, Sim emulates by itself by checking whether the secrets of $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}$, $[\boldsymbol{y}+\boldsymbol{b}]_{k-1}$, and $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$, which are $\overline{\boldsymbol{x}+\boldsymbol{a}}$, $\overline{\boldsymbol{y}+\boldsymbol{b}}$, and $\overline{\boldsymbol{z}+\boldsymbol{r}}$, correspond to that of $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}$, $[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$, and $[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$ accordingly. The distribution of $\mathbf{Hybrid}_{3,3,1,3}$ is identical to that of $\mathbf{Hybrid}_{3,3,1,2}$.

$\mathbf{Hybrid}_{3,3,2}$: In this hybrid, in Step 3.2 of EVAL, instead of invoking $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_9)$, Sim emulates by itself by checking whether the honest parties' shares of $[\boldsymbol{x}+\boldsymbol{r}]_{k-1}$, $[\boldsymbol{L}(\boldsymbol{x}+\boldsymbol{r})]_{k-1}$ are valid degree-$(k-1)$ packed Shamir sharings. The distribution of $\mathbf{Hybrid}_{3,3,2}$ is identical to that of $\mathbf{Hybrid}_{3,3,1,3}$.

$\mathbf{Hybrid}_{3,3,3}$: In this hybrid, in Step 3.3, we change the simulation of VRFY-SHARING-TRANS as described in $\mathsf{Sim}_{\text{vrfy-sharing-trans}}$. Due to the same reasons as $\mathbf{Hybrid}_{2,5}$, the distribution of $\mathbf{Hybrid}_{3,3,3}$ is identical to that of $\mathbf{Hybrid}_{3,3,2}$.

$\mathbf{Hybrid}_{3,3,4}$: In this hybrid, in Step 3.4 of EVAL, instead of invoking $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_{10})$, Sim emulates by itself by checking whether the secret of $[\boldsymbol{y}+\boldsymbol{r}]_{k-1}$ correspond to that of $[\boldsymbol{y}+\boldsymbol{r}]_{t+2k-2}$. The distribution of $\mathbf{Hybrid}_{3,3,4}$ is identical to that of $\mathbf{Hybrid}_{3,3,3}$.

$\mathbf{Hybrid}_{3,4}$: In this hybrid, we focus on simulation of OUTPUT.

$\mathbf{Hybrid}_{3,4,1}$: In this hybrid, we change the simulation of RAND-LINEAR-MASK as in $\mathsf{Sim}_{\text{rand-linear-mask}}$ and delay the sampling of whole sharing $([\![\boldsymbol{R}]\!]_{t+k-1}, [\![\vec{\boldsymbol{L}}(\boldsymbol{R})]\!]_{t+k-1})$ until the end of RAND-LINEAR-MASK. The distribution of $\mathbf{Hybrid}_{3,4,1}$ is identical to that of $\mathbf{Hybrid}_{3,3,4}$ due to the same reasons as $\mathbf{Hybrid}_{2,1}$.

$\mathbf{Hybrid}_{3,4,2,1}$: In this hybrid, in Step 2, we change the simulation of SELECT as in $\mathsf{Sim}_{\text{select}}$ and compute honest parties' shares of $[\![\boldsymbol{x}'_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{x}'_k]\!]_{t+k-1}$ and $[\![\boldsymbol{r}]\!]_{t+2k-2}$ until the end of SELECT. The distribution of $\mathbf{Hybrid}_{3,4,2,1}$ is identical to that of $\mathbf{Hybrid}_{3,4,1}$ due to the same reasons as $\mathbf{Hybrid}_{3,2,1,1}$.

$\mathbf{Hybrid}_{3,4,2,2}$: In this hybrid, in Step 2, we change the simulation of SHARING-TRANS as $\mathsf{Sim}_{\text{sharing-trans}}$ and compute honest parties' shares of $[\boldsymbol{x}]_{t+k-1}$ and $[\![\boldsymbol{r}]\!]_{t+k-1}$ until the end of SHARING-TRANS. The distribution of $\mathbf{Hybrid}_{3,4,2,2}$ is identical to that of $\mathbf{Hybrid}_{3,4,2,1}$ due to the same reasons as $\mathbf{Hybrid}_{2,3}$.

$\mathbf{Hybrid}_{3,4,3}$: In this hybrid, in Step 3.b, we change the computation of $[\boldsymbol{z}+\boldsymbol{r}]_{t+k-1}$. Suppose $[\boldsymbol{z}]_{t+k-1} = [\boldsymbol{z}']_{k-1} + [\![\boldsymbol{z}'']\!]_{t+k-1}$. Sim first computes shares of $[\![\boldsymbol{z}''+\boldsymbol{r}]\!]_{t+k-1}$ held by parties in $\mathcal{C} \cup \mathcal{H}'$, and samples a random degree-$(t+k-1)$ packed Shamir sharing as $[\![\boldsymbol{z}''+\boldsymbol{r}]\!]_{t+k-1}$ based on the shares computed above. Then Sim computes active honest parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{t+k-1}$ by $[\boldsymbol{z}+\boldsymbol{r}]_{t+k-1} \leftarrow [\boldsymbol{z}']_{k-1} + [\![\boldsymbol{z}''+\boldsymbol{r}]\!]_{t+k-1}$. Then Sim computes honest parties' shares of $[\![\boldsymbol{r}]\!]_{t+k-1}$ by $[\![\boldsymbol{r}]\!]_{t+k-1} \leftarrow [\boldsymbol{z}+\boldsymbol{r}]_{t+k-1} - [\boldsymbol{z}]_{t+k-1}$. Due to the randomness of $[\![\boldsymbol{r}]\!]_{t+k-1}$, the distribution of $\mathbf{Hybrid}_{3,4,3}$ is identical to that of $\mathbf{Hybrid}_{3,4,2,2}$.

$\mathbf{Hybrid}_{3,4,4}$: In this hybrid, in Step 3.d, if P_{king} broadcasts `Inconsistent`, Sim simulates ANALYZE-SHARING$(P_{\text{king}}, [\boldsymbol{z}+\boldsymbol{r}]_{t+k-1})$ as described in $\mathsf{Sim}_{\text{analyze-sharing}}$ and computes honest parties' shares of $[\![\boldsymbol{r}]\!]_{t+k-1}$ until the end of ANALYZE-SHARING. Due to the same reasons as $\mathbf{Hybrid}_{2,3,2}$, the distribution of $\mathbf{Hybrid}_{3,4,4}$ is identical to that of $\mathbf{Hybrid}_{3,4,3}$.

$\mathbf{Hybrid}_{3,4,5}$: In this hybrid, in Step 4.a, instead of invoking $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_9)$, Sim emulates by itself by checking whether honest parties' shares of $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ form valid degree-$(k-1)$ packed Shamir sharings. The distribution of $\mathbf{Hybrid}_{3,4,5}$ is identical to that of $\mathbf{Hybrid}_{3,4,4}$.

$\mathbf{Hybrid}_{3,4,6}$: In this hybrid, in Step 4.b, we change the simulation of VRFY-SHARING-TRANS as described in $\mathsf{Sim}_{\text{vrfy-sharing-trans}}$. Due to the same reasons as $\mathbf{Hybrid}_{2,5}$, the distribution of $\mathbf{Hybrid}_{3,4,6}$ is identical to that of $\mathbf{Hybrid}_{3,4,5}$.

$\mathbf{Hybrid}_{3,4,7}$: In this hybrid, in Step 4.c, instead of invoking $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_{10})$, Sim emulates by itself by checking whether the secret of $[\boldsymbol{y}+\boldsymbol{r}]_{k-1}$ correspond to that of $[\boldsymbol{y}+\boldsymbol{r}]_{t+2k-2}$. The distribution of $\mathbf{Hybrid}_{3,4,7}$ is identical to that of $\mathbf{Hybrid}_{3,4,6}$.

$\mathbf{Hybrid}_{3,5}$: In this hybrid, we focus on simulation of **Reconstruction**. Sim first invokes the ideal functionality $\mathcal{F}_{\text{main}}$. Sim provides inputs of corrupted parties while honest parties provide their inputs. Then Sim receives outputs of corrupted parties. For each output wire o belonging to a corrupted party, supposing the secret is stored at the j-th position, Sim first computes t shares of $[o|j]_t$ held by parties in $\mathcal{C} \cup \mathcal{H}'$ and recovers the whole sharing $[o|j]_t$ based on the secret o and t shares. Since all checks have been passed, by the correctness of the protocol, the distribution of $\mathbf{Hybrid}_{3,5}$ is identical to that of $\mathbf{Hybrid}_{3,4,7}$.

Hybrid$_4$: In this hybrid, we focus on simulation of **Restart**. Instead of invoking $\mathcal{F}^*_{\text{main-client-server}}$, Sim emulates by itself as follows. Sim receives inputs of corrupted parties. Then Sim invokes the ideal functionality $\mathcal{F}_{\text{main}}$ and passes the inputs of corrupted parties, while honest parties provide their inputs. Then Sim receives outputs of corrupted parties and sends them to the adversary on behalf of $\mathcal{F}^*_{\text{main-client-server}}$. By the correctness of $\mathcal{F}^*_{\text{main-client-server}}$, the distribution of **Hybrid**$_4$ is identical to that of **Hybrid**$_{3,5}$.

Hybrid$_5$: In this hybrid, we do not compute shares held by honest parties in $\mathcal{H} \setminus \mathcal{H}'$ and do not use honest parties' inputs. The distribution of **Hybrid**$_5$ is identical to that of **Hybrid**$_4$.

Since **Hybrid**$_5$ already corresponds to the ideal world, we prove that Sim perfectly simulates behaviors of honest parties. □

5 Lower Bounds

Recall that we have already reduced the task of constructing perfectly secure MPC protocols to preparing error-free packed Beaver triples over bivariate polynomials. However, in this section, we will establish a lower bound on communication complexity of preparing such kind of error-free packed Beaver triples.

To achieve this, we first prove a communication lower bound for verifying correctness of packed Beaver triples with perfect security with abort in Section 5.2. Next, we observe that assuming error-free packed Beaver triples, all parties can efficiently verify the correctness of packed Beaver triples, which implies the functionality of preparing error-free packed Beaver triples (even for perfect security with abort) cannot be instantiated with low communication. We defer the detailed description for this reduction to Section 5.3.

Before diving into the study of communication lower bounds, we first introduce several useful notions in the context of protocol execution below.

Notations. Suppose there are n parties, denoted by $P_1, P_2, \ldots, P_n$, and an n-party protocol Π. Note that each execution of Π is fully determined by a tuple consisting of all parties' inputs and internal randomness, denoted by $d := (I_1, I_2, \ldots, I_n; r_1, r_2, \ldots, r_n)$, where I_i and r_i represent P_i's input and internal randomness, respectively. We use $\texttt{exec} := \texttt{exec}(\Pi, d)$ to denote the execution defined by running Π on d. Before the execution, each party P_i has its input $\texttt{input}^{P_i}(\Pi, d) = I_i$ and samples its internal randomness $\texttt{rand}^{P_i}(\Pi, d) = r_i$. At each round of the protocol, for each party, its current view (including its input, internal randomness, and all messages sent or received so far) determines the messages it sends in this round. In the final round of the protocol, its view determines its protocol output.

After the execution, each party P_i obtains its view $\texttt{view}^{P_i}(\Pi, d)$, which contains its input $\texttt{input}^{P_i}(\Pi, d)$, its internal randomness $\texttt{rand}^{P_i}(\Pi, d)$, its transcript $\texttt{transcript}^{P_i}(\Pi, d)$, and its output $\texttt{output}^{P_i}(\Pi, d)$. We use $\texttt{view}^{P_i}(\Pi, I_1, \ldots, I_n)$ to denote the distribution of the view of P_i generated by running Π on $(I_1, \ldots, I_n)$ with parties' internal randomness sampled uniformly at random. Given the view of party P $\texttt{view}^P$, we say a tuple of inputs and internal randomness d satisfies $\texttt{view}^P$ if $\texttt{view}^P(\Pi, d) = \texttt{view}^P$, which is denoted by $d \vdash \texttt{view}^P$. We say a view $\texttt{view}$ is plausible for P_i if there exists a tuple d of inputs and internal randomness from all parties satisfying $\texttt{view}$, i.e., $\texttt{view}^{P_i}(\Pi, d) = \texttt{view}$. We assume each party has its internal randomness of finite size, which ensures the underlying probability space has a finite sample space.

5.1 Warm-up: Impossibility Result of Computing AND of Two Bits

The pioneer work [BGW88] shows a strong negative result that information-theoretic MPC cannot exist without honest majority, even when only semi-honest security is considered. In particular, it can be shown that there dose not exist 2-party protocols securely realizing the functionality of computing AND of two bits $f_{\text{AND}} : (x_A, x_B) \mapsto (x_A \cdot x_B, x_A \cdot x_B)$ against one semi-honest corruption. We extend the proof for this 2-party negative result to 3-party case by additionally considering a malicious adversary. We formally present the statement in Lemma 2.

Lemma 2. *There does not exist a 3-party protocol computing $f_{\text{AND}} : (\perp, x_2, x_3) \mapsto (\perp, x_2 \cdot x_3, x_2 \cdot x_3)$ with perfect security with abort against any fully malicious adversary controlling at most one corrupted party.*

Proof (Proof of Lemma 2). Denote three parties by P_1, P_2 and P_3. For the sake of contradiction, suppose there exists a 3-party protocol π computing $f_{\mathtt{AND}}$ with perfect security with abort against a fully malicious adversary controlling one corrupted party. Denote an honest execution where P_i uses x_i as input and r_i as internal randomness for $i \in \{1,2,3\}$ by $\mathtt{exec}(\pi, \bot, x_2, x_3; r_1, r_2, r_3)$. We first consider an honest execution $\mathtt{exec}_0 := \mathtt{exec}(\pi, \bot, 0, 0; r_1, r_2, r_3)$.

By perfect privacy against P_2, there exists an honest execution $\mathtt{exec}_1 := \mathtt{exec}(\pi, \bot, 0, 1; r'_1, r_2, r'_3)$ such that P_2 has the same view as that produced by $\mathtt{exec}_0$. To see why this is the case, by perfect privacy of π, there exists a simulator $\mathcal{S}_2$ which takes as input the input and output of P_2, and outputs the view of P_2 such that for any $x_2, x_3 \in \{0,1\}$, $\mathcal{S}_2(x_2, x_2 \cdot x_3) \equiv \mathtt{view}^{P_3}(\pi, \bot, x_2, x_3)$. Then by property of $f_{\mathtt{AND}}$, $\mathtt{view}^{P_2}(\pi, \bot, 0, 0) \equiv \mathcal{S}_2(0, 0 \cdot 0) \equiv \mathcal{S}_2(0, 0 \cdot 1) \equiv \mathtt{view}^{P_2}(\pi, \bot, 0, 1)$. This implies there exists a choice of internal randomness of P_1 and P_3, r'_1, r'_3, such that $\mathtt{exec}_0$ and $\mathtt{exec}_1 := \mathtt{exec}(\pi, \bot, 0, 1; r'_1, r_2, r'_3)$ generate identical views for P_2. Similarly, there exists an honest execution $\mathtt{exec}_2 := \mathtt{exec}(\pi, \bot, 1, 0; r''_1, r''_2, r_3)$ such that P_3 has the same view as that produced by $\mathtt{exec}_0$.

Now we claim that existence of such three executions $\mathtt{exec}_0, \mathtt{exec}_1$ and $\mathtt{exec}_2$ contradicts with perfect correctness of π. To show this, we construct a malicious execution, denoted by $\mathtt{exec}_3$, as follows.

- P_2 uses 1 as input and r''_2 as internal randomness and honestly follows the protocol.
- P_3 uses 1 as input and r'_3 as internal randomness and honestly follows the protocol.
- Malicious P_1 first emulates $\mathtt{exec}_1$ and $\mathtt{exec}_2$ to compute the messages it would send to P_2 and P_3 in both executions. In the actual execution, P_1 behaves inconsistently by sending to P_2 the messages it would have sent in $\mathtt{exec}_2$, while sending to P_3 the messages it would have sent in $\mathtt{exec}_1$.

To argue the outputs of P_2 and P_3 in $\mathtt{exec}_3$, we prove the following claim.

> *Claim.* The transcripts between P_2 and P_1 in executions $\mathtt{exec}_3$ and $\mathtt{exec}_2$ are identical. The transcripts between P_3 and P_1 in executions $\mathtt{exec}_3$ and $\mathtt{exec}_1$ are identical. The transcripts between P_2 and P_3 are identical in four executions $\mathtt{exec}_0, \mathtt{exec}_1, \mathtt{exec}_2$ and $\mathtt{exec}_3$.
>
> *Proof.* We prove this statement by induction. First, by construction of $\mathtt{exec}_1$ and $\mathtt{exec}_2$, the transcripts between P_2 and P_3 in $\mathtt{exec}_0, \mathtt{exc}_1$ and $\mathtt{exec}_2$ are identical.
> Consider the first message in $\mathtt{exec}_3$.
> - If it is from P_1 to P_2, by construction of $\mathtt{exec}_3$, this message is identical to that of $\mathtt{exec}_2$. Similar, if it is from P_1 to P_3, by construction of $\mathtt{exec}_3$, this messages is identical to that of $\mathtt{exec}_1$.
> - If it is from P_2 (to either P_1 or P_3), since P_2 uses the same input 1 and internal randomness r''_2 in $\mathtt{exec}_3$ and $\mathtt{exec}_2$, this message is identical to that of $\mathtt{exec}_2$.
> - If it is from P_3 (to either P_1 or P_2), since P_3 uses the same input 1 and internal randomness r'_3 in $\mathtt{exec}_3$ and $\mathtt{exec}_1$, this message is identical to that of $\mathtt{exec}_1$.
>
> Now assume the first $T-1$ messages already satisfy the statement. We prove this holds for the T-th message.
> - If it is from P_1 (to either P_2 or P_3), by construction of $\mathtt{exec}_3$, it satisfies the statement.
> - If it is sent from P_2 (to either P_1 or P_3), by induction hypothesis, the messages previously sent or received by P_2 are identical in $\mathtt{exec}_3$ and $\mathtt{exec}_2$. Hence, the current views for P_2 (including its input 1, internal randomness r''_2, and messages sent or received itself) are identical in $\mathtt{exec}_3$ and $\mathtt{exec}_2$, and this message sent from P_2 is identical in $\mathtt{exec}_3$ and $\mathtt{exec}_2$. Moreover, if this message is from P_2 to P_3, it is identical in all four executions.
> - If it is sent from P_3 (to either P_1 or P_2), by induction hypothesis, the messages previously sent or received by P_3 are identical in $\mathtt{exec}_3$ and $\mathtt{exec}_1$. Hence, the current views for P_3 (including its input 1, internal randomness r'_3, and messages sent or received itself) are identical in $\mathtt{exec}_3$ and $\mathtt{exec}_1$, and this message sent from P_3 is identical in $\mathtt{exec}_3$ and $\mathtt{exec}_1$. Moreover, if this message is from P_3 to P_2, it is identical in all four executions.
>
> By induction, the statement holds for all messages in $\mathtt{exec}_3$. □

According to the claim above, when comparing $\mathtt{exec}_3$ and $\mathtt{exec}_2$, P_2 has the same views in both executions, and thus P_2 would output 0 in both executions. However, by perfect correctness of π, both P_2 and P_3 should output either $1 \cdot 1 = 1$ or $\bot$, which leads to a contradiction. □

Implication for n-party setting. When there are n parties and $t \geq n/3$, we show Lemma 2 implies there does not exist an n-party protocol computing the AND of two bits held by two different parties (P_A and P_B) with perfect security with abort against any fully malicious adversary controlling at most t corrupted parties. In fact, for the sake of contradiction, if there exists an n-party protocol computing the AND function with perfect security with abort, then let P_2 emulate t parties containing P_A, let P_3 emulate t parties containing P_B, and let P_1 emulate the rest of the parties. In doing this, we obtain a 3-party protocol computing $f_{\texttt{AND}}$ with perfect security with abort against one corrupted party, which contradicts with Lemma 2. This negative result suggests $n > 3t$ is necessary (and sufficient) for computing the AND function with perfect security with abort.

Remark 1. We note that even assuming a broadcast channel, Lemma 2 (and its implication for n-party setting) holds. This is because the broadcast messages are in both P_2's view and P_3's view. Thus, they are identical in all three executions $\texttt{exec}_0, \texttt{exec}_1, \texttt{exec}_2$. Then in $\texttt{exec}_3$, P_1 simply broadcasts the same messages as those in $\texttt{exec}_0, \texttt{exec}_1, \texttt{exec}_2$. We can show that the same argument works for $\texttt{exec}_3$ and both P_2, P_3 would output 0 in $\texttt{exec}_3$ which breaks the perfect correctness.

5.2 Communication Complexity of Verifying Packed Triples

As mentioned in Section 2.3, we generalize the 3-party impossibility result by introducing a fourth party, P_0, which is involved in privately computing the AND of two bits. If P_0 does not communicate with any other party, the setting is identical to the case of 3-party computation. More generally and intuitively, if P_0 communicates only a few bits, thus failing to provide sufficient information to P_2 and P_3, then this leads to a contradiction with perfect security. This contradiction arises by considering an adversary that mounts a similar attack to that described in the proof of Lemma 2 in 3-party setting.

To derive a communication lower bound for preparing error-free packed Beaver triples, we consider the task of verifying correctness of packed triples and formally characterize this functionality as $\mathcal{F}_{\text{vrfy-triple-abort}}(N, k)$, where $k(\leq (t+2)/2)$ secrets are packed into each of N packed triples. We emphasize that *perfect security with abort* is a weaker notion than perfect security in the sense that it allows an adversary to abort the functionality. In addition, we further weaken the functionality by revealing honest parties' inputs to the adversary when the functionality decides to abort. Building on the above intuition, we prove a communication lower bound that in order to verify N packed Beaver triples with perfect security with abort, each party must communicate $\Omega(N \cdot k)$ field elements in expectation, leading to a total communication of $\Omega(N \cdot n^2)$ elements if $k = \Omega(n)$. This result is formally stated as Lemma 3.

Functionality 4: $\mathcal{F}_{\text{vrfy-triple-abort}}(N, k)$

1. $\mathcal{F}_{\text{vrfy-triple-abort}}$ receives the set of corrupted parties $\mathcal{C}$.
2. $\mathcal{F}_{\text{vrfy-triple-abort}}$ receives from all honest parties their shares of $\{[\boldsymbol{a}_\ell]_{t+k-1}, [\boldsymbol{b}_\ell]_{t+k-1}, [\boldsymbol{c}_\ell]_{t+k-1}\}_{\ell \in [N]}$, reconstructs the secrets to learn $\{\boldsymbol{a}_\ell, \boldsymbol{b}_\ell, \boldsymbol{c}_\ell\}_{\ell \in [N]}$, where $\boldsymbol{a}_\ell, \boldsymbol{b}_\ell, \boldsymbol{c}_\ell \in \mathbb{F}^k$, and sends shares of the corrupted parties to the adversary.
3. $\mathcal{F}_{\text{vrfy-triple-abort}}$ checks if $\boldsymbol{a}_\ell * \boldsymbol{b}_\ell = \boldsymbol{c}_\ell$ holds for each $\ell \in [N]$.
   - If all checks pass, $\mathcal{F}_{\text{vrfy-triple-abort}}$ waits for an instruction from the adversary:
     - Upon receiving `reject`, output 0 to all parties and reveals $\{[\boldsymbol{a}_\ell]_{t+k-1}, [\boldsymbol{b}_\ell]_{t+k-1}, [\boldsymbol{c}_\ell]_{t+k-1}\}_{\ell \in [N]}$ to the adversary.
     - Upon receiving `accept`, output 1 to all parties.
   - Otherwise, $\mathcal{F}_{\text{vrfy-triple-abort}}$ outputs 0 to all parties and reveals $\{[\boldsymbol{a}_\ell]_{t+k-1}, [\boldsymbol{b}_\ell]_{t+k-1}, [\boldsymbol{c}_\ell]_{t+k-1}\}_{\ell \in [N]}$ to the adversary.

Lemma 3. *Let $n = 3t + 1$ and $k \in \mathbb{N}$ such that $1 \leq k \leq t + 1$. There exist constants $c_1, c_2 > 0$ such that for all $N > c_1$ and $k > c_2$, any protocol computing $\mathcal{F}_{\text{vrfy-triple-abort}}(N, k)$ with perfect security against any fully malicious adversary controlling at most t corrupted parties satisfies that each party has expected communication of at least $\frac{N \cdot k}{8}$ elements, leading to total communication of at least $\frac{N \cdot k \cdot n}{16}$ elements, when $\mathcal{F}_{\text{vrfy-triple-abort}}$ outputs 1. This lower bound implies an amortized cost of $\Omega(n)$ elements per secret.*

Proof (Proof of Lemma 3). Denote $n = 3t + 1$ parties by $P_0, P_{1,1}, \ldots, P_{1,t}, P_{2,1}, \ldots, P_{2,t}, P_{3,1}, \ldots, P_{3,t}$. Additionally, denote $\{P_{1,i}\}_{i\in[t]}$, $\{P_{2,i}\}_{i\in[t]}$ and $\{P_{3,i}\}_{i\in[t]}$ by P_1, P_2 and P_3, respectively. Suppose π is an n-party protocol computing $\mathcal{F}_{\text{vrfy-triple-abort}}$ with perfect security against any fully malicious adversary controlling at most t corrupted parties. Since all honest parties have the same output in $\mathcal{F}_{\text{vrfy-triple-abort}}$, we use $\texttt{output}(\pi, d)$ to denote the output of an honest party in execution (π, d). We will consider the set $\mathcal{D}$ containing all tuples of inputs and internal randomness such that an honest execution defined by this tuple outputs 1, i.e., $\mathcal{D} := \{d \mid \texttt{output}(\pi, d) = 1\}$. We assume each party has its internal randomness of finite size, which implies the cardinality of $\mathcal{D}$ is finite. We model $\mathcal{D}$ as the sample space, and assume a uniform distribution over $\mathcal{D}$. For any event $\mathcal{E} \subseteq \mathcal{D}$, we denote its probability by $\Pr_{d\leftarrow\mathcal{D}}[\mathcal{E}]$, or simply $\Pr_d[\mathcal{E}]$.

Without loss of generality, we will focus on the communication complexity of party P_0. We denote the length of its transcript by a random variable $C_0 : \mathcal{D} \to \mathbb{N}$. We set $c_1 = \lceil \frac{24}{\log F} \rceil$ and $c_2 = 24$. For the sake of contradiction, suppose there exist $N > c_1$ and $k > c_2$ such that the communication of P_0 in π satisfies that

$$\mathbb{E}_{d\leftarrow\mathcal{D}}[C_0(d)] < \frac{N \cdot k}{8}.$$

Extending the idea of Lemma 2, the proof is divided into three steps.

Step 1. Define three honest executions satisfying delicate requirements.
Step 2. Construct a (dishonest) execution with an incorrect output by composing three honest executions chosen in last step, which leads to a contradiction with perfect correctness of π.
Step 3. Prove existence of such three honest executions based on perfect privacy of π and the assumption that P_0 only has limited amount of communication.

The formal proof works as follows.

Step 1. Define three honest executions. We start with sampling an honest execution $\texttt{exec}_0$ with output 1 by sampling $d = (I_0, I_1, I_2, I_3; r_0, r_1, r_2, r_3)$ from $\mathcal{D}$ uniformly at random. Suppose $(\boldsymbol{a}^{(0)}, \boldsymbol{b}^{(0)}, \boldsymbol{c}^{(0)})$ with $\boldsymbol{a}^{(0)}, \boldsymbol{b}^{(0)} \in \mathbb{F}^{N\cdot k}$ and $\boldsymbol{c}^{(0)} = \boldsymbol{a}^{(0)} * \boldsymbol{b}^{(0)}$ are the secrets of the N packed triples defined by d. Let $\mathcal{D}_1(d) \subset \mathcal{D}$ be the set of all tuples of the form $d' = (I'_0, I'_1, I'_2, I'_3; r'_0, r'_1, r'_2, r'_3)$ satisfying the following conditions.

- $\texttt{view}^{P_2}(\pi, d) = \texttt{view}^{P_2}(\pi, d')$, which implies $I'_2 = I_2, r'_2 = r_2$;
- $\texttt{input}^{P_0}(\pi, d) = \texttt{input}^{P_0}(\pi, d')$ and $\texttt{transcript}^{P_0}(\pi, d) = \texttt{transcript}^{P_0}(\pi, d')$, implying $I'_0 = I_0$;
- Letting $(\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)})$ denote the secrets of the N packed triples defined by d', we require that $\boldsymbol{c}^{(1)} = \boldsymbol{a}^{(1)} * \boldsymbol{b}^{(1)}$ and $\boldsymbol{b}^{(1)} = \boldsymbol{b}^{(0)}$. We denote this constraint compactly by writing $d' \vdash \boldsymbol{b}^{(0)}$.

Then we define $\texttt{event}$ 1 to be an event that the number of possible distinct secrets of the first sharing among the executions in the set $\mathcal{D}_1(d)$ is strictly greater than $F^{\frac{N\cdot k}{2}}$, i.e., $|\{\boldsymbol{a}^{(1)} \in \mathbb{F}^{N\cdot k} \mid \exists\, d' \in \mathcal{D}_1(d) \text{ s.t. } d' \vdash \boldsymbol{a}^{(1)}\}| > F^{\frac{N\cdot k}{2}}$. Similarly, let $\mathcal{D}_2(d) \subset \mathcal{D}$ be the set of all tuples of the form $d'' = (I''_0, I''_1, I''_2, I''_3; r''_0, r''_1, r''_2, r''_3) \in \mathcal{D}$ satisfying the following conditions.

- $\texttt{view}^{P_3}(\pi, d) = \texttt{view}^{P_3}(\pi, d'')$, which implies $I''_3 = I_3, r''_3 = r_3$;
- $\texttt{input}^{P_0}(\pi, d) = \texttt{input}^{P_0}(\pi, d'')$ and $\texttt{transcript}^{P_0}(\pi, d) = \texttt{transcript}^{P_0}(\pi, d'')$, implying $I''_0 = I_0$;
- Letting $(\boldsymbol{a}^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{c}^{(2)})$ denote the secrets of the N packed triples defined by d'', we require that $\boldsymbol{c}^{(2)} = \boldsymbol{a}^{(2)} * \boldsymbol{b}^{(2)}$ and $\boldsymbol{a}^{(2)} = \boldsymbol{a}^{(0)}$. We denote this constraint compactly by writing $d'' \vdash \boldsymbol{a}^{(0)}$.

Then we define $\texttt{event}$ 2 to be an event that the number of possible distinct secrets of the second sharing among the executions in the set $\mathcal{D}_2(d)$ is strictly greater than $F^{\frac{N\cdot k}{2}}$, i.e., $|\{\boldsymbol{b}^{(2)} \in \mathbb{F}^{N\cdot k} \mid \exists\, d'' \in \mathcal{D}_2 \text{ s.t. } d'' \vdash \boldsymbol{b}^{(2)}\}| > F^{\frac{N\cdot k}{2}}$.

Step 2. Construct a dishonest execution. If $\texttt{event}$ 1 and $\texttt{event}$ 2 happen simultaneously, we claim this breaks the perfect correctness of π. In fact, we first pick $d' = (I'_0, I'_1, I'_2, I'_3; r'_0, r'_1, r'_2, r'_3)$ and $d'' = (I''_0, I''_1, I''_2, I''_3; r''_0, r''_1, r''_2, r''_3)$ as tuples of inputs and internal randomness satisfying the requirements of $\texttt{event}$ 1 and $\texttt{event}$ 2, respectively. I.e., $d' \in \mathcal{D}_1, d'' \in \mathcal{D}_2$. Let $\texttt{exec}_1 := \texttt{exec}(\pi, d')$ and $\texttt{exec}_2 := \texttt{exec}(\pi, d'')$. Then we construct a dishonest execution denoted by $\texttt{exec}_3(d', d'')$ as follows and illustrate this construction in Figure 2.

- P_0 takes I_0 as its input, uses r_0 (or equivalently, r'_0, r''_0) as its internal randomness, and honestly follows the protocol.
- P_2 takes I''_2 as its input, uses r''_2 as its internal randomness, and honestly follows the protocol.
- P_3 takes I'_3 as its input, uses r'_3 as its internal randomness, and honestly follows the protocol.
- An adversary controlling parties $\{P_{1,i}\}_{i\in[t]}$ launches a similar attack as in Lemma 2. In particular, P_1 first emulates $\mathtt{exec}_0$, $\mathtt{exec}_1$, and $\mathtt{exec}_2$ to compute the messages it would send to P_0, P_2, P_3. Then malicious P_1 behaves inconsistently across parties. When interacting with P_0, P_1 uses the messages in $\mathtt{exec}_0$ (or equivalently, $\mathtt{exec}_1$, $\mathtt{exec}_2$); when interacting with P_2, P_1 uses the messages in $\mathtt{exec}_3$; when interacting with P_3, P_1 uses the messages in $\mathtt{exec}_1$.

We show that honest party will obtain an incorrect output in $\mathtt{exec}_3(d', d'')$. On the one hand, to argue the output of P_0, P_2 and P_3 in $\mathtt{exec}_3$, we prove the following proposition.

Proposition 1. *The transcripts between P_2 and P_1 in executions $\mathtt{exec}_3$ and $\mathtt{exec}_2$ are identical. The transcripts between P_3 and P_1 in executions $\mathtt{exec}_3$ and $\mathtt{exec}_1$ are identical. The transcripts between P_2 and P_3 and transcripts between P_0 and each party in $\{P_1, P_2, P_3\}$ are identical in four executions $\mathtt{exec}_0$, $\mathtt{exec}_1$, $\mathtt{exec}_2$ and $\mathtt{exec}_3$.*

According to this proposition, when comparing $\mathtt{exec}_3$ and $\mathtt{exec}_2$, P_2 has the same views in both executions and thus P_2 would output 1 in both executions.

On the other hand, we observe that there exist $d' \in \mathcal{D}_1$ and $d'' \in \mathcal{D}_2$ such that the triples corresponding to the input in $\mathtt{exec}_3(d', d'')$ do not correctly satisfy the multiplication relations. First, by linearity of packed Shamir sharings, we have the following proposition.

Proposition 2. *Let $(\boldsymbol{a}^{(3)}, \boldsymbol{b}^{(3)}, \boldsymbol{c}^{(3)})$ be the secrets of N packed triples determined by $2t+1$ parties' input (I_0, I''_2, I'_3) in $\mathtt{exec}_3(d', d'')$. Then it holds that*

$$(\boldsymbol{a}^{(3)}, \boldsymbol{b}^{(3)}, \boldsymbol{c}^{(3)}) = (\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}) + (\boldsymbol{a}^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{c}^{(2)}) - (\boldsymbol{a}^{(0)}, \boldsymbol{b}^{(0)}, \boldsymbol{c}^{(0)}).$$

*Then $(\boldsymbol{a}^{(3)}, \boldsymbol{b}^{(3)}, \boldsymbol{c}^{(3)}) = (\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(2)}, \boldsymbol{a}^{(0)} * \boldsymbol{b}^{(2)} + \boldsymbol{a}^{(1)} * \boldsymbol{b}^{(0)} - \boldsymbol{a}^{(0)} * \boldsymbol{b}^{(0)})$, which implies that the input of $\mathtt{exec}_3(d', d'')$ **doesn't** satisfy the multiplication relation if there exists $\ell \in [N \cdot k]$ such that $a^{(1)}_\ell \neq a^{(0)}_\ell$ and $b^{(2)}_\ell \neq b^{(0)}_\ell$.*

Since $\mathtt{event}$ 1 happens, we have $|\{\boldsymbol{a}^{(1)} \in \mathbb{F}^{N\cdot k} \mid \exists\ d' \in \mathcal{D}_1(d) \text{ s.t. } d' \vdash \boldsymbol{a}^{(1)}\}| > F^{\frac{N\cdot k}{2}}$, which implies that the executions in $\mathcal{D}_1(d)$ correspond to more than $F^{\frac{N\cdot k}{2}}$ distinct tuples of secrets. Hence, there exist at least $\frac{N\cdot k}{2}+1$ positions such that, for each of these positions, there exists an execution in $\mathcal{D}_1(d)$ whose secret of the first sharing $\boldsymbol{a}^{(1)}$ differs from $\boldsymbol{a}^{(0)}$ at that position, i.e., $|\{\ell \mid \exists\ d' \in \mathcal{D}_1(d),\ \text{s.t. } d' \vdash \boldsymbol{a}^{(1)}, a^{(1)}_\ell \neq a^{(0)}_\ell\}| \geq \frac{N\cdot k}{2} + 1$. Similarly, since $\mathtt{event}$ 2 happens, we have $|\{\boldsymbol{b}^{(2)} \in \mathbb{F}^{N\cdot k} \mid \exists\ d'' \in \mathcal{D}_2(d) \text{ s.t. } d'' \vdash \boldsymbol{b}^{(2)}\}| > F^{\frac{N\cdot k}{2}}$, and therefore $|\{\ell \mid \exists\ d'' \in \mathcal{D}_2(d),\ \text{s.t. } d'' \vdash \boldsymbol{b}^{(2)}, b^{(2)}_\ell \neq b^{(0)}_\ell\}| \geq \frac{N\cdot k}{2} + 1$. Thus, by $\frac{N\cdot k}{2} + 1 + \frac{N\cdot k}{2} + 1 > N \cdot k$, there exist $d' \in \mathcal{D}_1(d)$ and $d'' \in \mathcal{D}_2(d)$ such that $\exists\ \ell \in [N \cdot k]$ s.t. $a^{(1)}_\ell \neq a^{(0)}_\ell$ and $b^{(2)}_\ell \neq b^{(0)}_\ell$, where $(\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)})$ and $(\boldsymbol{a}^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{c}^{(2)})$ are the secrets of triples corresponding to d' and d'', respectively. Then by our observation above, the triples in $\mathtt{exec}_3(d', d'')$ do not satisfy the multiplication relation. Hence, the perfect correctness of π is expected to make sure all honest parties take 0 as output no matter how an adversary deviates from the protocol. Thus, we figure out an execution $\mathtt{exec}_3(d', d'')$ in which the adversary controlling P_1 manages to let the protocol execution to end up with an incorrect output, resulting in a contradiction with perfect correctness of π.

Step 3. Existence of three honest executions. In order to complete the proof, we are targeted to prove that $\Pr_{d\leftarrow\mathcal{D}}[\mathtt{event}\ 1 \wedge \mathtt{event}\ 2] > 0$. By union bound, we have that

$$\Pr_{d\leftarrow\mathcal{D}}[\mathtt{event}\ 1 \wedge \mathtt{event}\ 2] \geq 1 - \Pr_{d\leftarrow\mathcal{D}}[\overline{\mathtt{event}\ 1}] - \Pr_{d\leftarrow\mathcal{D}}[\overline{\mathtt{event}\ 2}].$$

Then it suffices to prove $\Pr_{d\leftarrow\mathcal{D}}[\overline{\mathtt{event}\ 1}] < \frac{1}{2}$ and $\Pr_{d\leftarrow\mathcal{D}}[\overline{\mathtt{event}\ 2}] < \frac{1}{2}$. We compute $\Pr_{d\leftarrow\mathcal{D}}[\overline{\mathtt{event}\ 1}]$ and the case of $\mathtt{event}$ 2 is similar. Before the computation, we first show that given P_2's view and part of the

input, there exists a sufficiently large number of candidate executions, making use of perfect privacy against a corrupt P_2. We formally state this as the following proposition. Looking ahead, assuming P_0 only has a limited amount of communication in expectation, we are likely to find a large subset of these executions that generate identical views for P_0 (excluding its internal randomness), denoted by $\texttt{input}^{P_0}||\texttt{transcript}^{P_0}$.

Proposition 3. *Let* $\texttt{view}^{P_2}$ *be a plausible view of* P_2 *with* 1 *as output, i.e., there exists a tuple* $d = (I_0, I_1, I_2, I_3;\ r_0, r_1, r_2, r_3) \in \mathcal{D}$ *such that* $\texttt{view}^{P_2}(\pi, d) = \texttt{view}^{P_2}$ *and* $\texttt{output}(\pi, d) = 1$. *Then for every* $\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}} \in \mathbb{F}^{N\cdot k}$, *there exists a tuple* $\bar{d} = (\bar{I_0}, \bar{I_1}, I_2, \bar{I_3}; \bar{r_0}, \bar{r_1}, r_2, \bar{r_3}) \in \mathcal{D}$ *of all parties' inputs and internal randomness such that* $\texttt{view}^{P_2}(\pi, \bar{d}) = \texttt{view}^{P_2}$ *and* $(\bar{I_0}, \bar{I_1}, I_2, \bar{I_3})$ *forms degree-*$(t+k-1)$ *packed Shamir sharings of* $(\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}, \bar{\boldsymbol{a}} * \bar{\boldsymbol{b}})$.

Let $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a}*\boldsymbol{b})$ *be the tuple of the secrets defined by* d. *Let* $\mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}} \subset \mathcal{D}$ *be the set of all honest executions satisfying 1) the view of* P_2 *is* $\texttt{view}^{P_2}$ *and 2) the secret of the second sharing is* $\boldsymbol{b}$. *Then the number of possible distinct secrets of the first sharing among the executions in the set* $\mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}}$ *is* $F^{N\cdot k}$, *i.e.,*

$$|\{\bar{\boldsymbol{a}} \in \mathbb{F}^{N\cdot k} \mid \exists\ \bar{d} \in \mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}} \text{ s.t. } \bar{d} \vdash \bar{\boldsymbol{a}}\}| = F^{N\cdot k}.$$

Next, we show that, given the view of P_2 $\texttt{view}^{P_2}$, and a fixed value of the second sharing's secret $\boldsymbol{b}$, the executions consistent with $\texttt{view}^{P_2}$ and $\boldsymbol{b}$ are uniformly distributed over all possible values of the first sharing's secret $\bar{\boldsymbol{a}}$. This is formally stated in the following proposition. Consequently, if there exist sufficiently many executions all consistent with the view of P_2 $\texttt{view}^{P_2}$, the second sharing's secret $\boldsymbol{b}$, and P_0's input together with its transcript $\texttt{input}^{P_0}||\texttt{transcipt}^{P_0}$, it implies the existence of a large number of distinct possible values for $\bar{\boldsymbol{a}}$, which in turn implies $\texttt{event}$ 1 occurs.

Proposition 4. *Given a view of* P_2 $\texttt{view}^{P_2}$ *and a secret of the second sharing* $\boldsymbol{b}$, *we define the set* $\mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}}$ *as in the previous claim. For any two vectors of secrets* $\bar{\boldsymbol{a}}_1, \bar{\boldsymbol{a}}_2 \in \mathbb{F}^{N\cdot k}$, *the number of executions in* $\mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}}$ *with the first sharing's secret being* $\bar{\boldsymbol{a}}_1$ *is equal to the number of executions with the first sharing's secret being* $\bar{\boldsymbol{a}}_2$, *i.e.,*

$$|\{d \in \mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}} \mid d \vdash \bar{\boldsymbol{a}}_1\}| = |\{d \in \mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}} \mid d \vdash \bar{\boldsymbol{a}}_2\}|.$$

We defer the deduction of upper bounding $\Pr_{d \leftarrow \mathcal{D}}[\overline{\texttt{event 1}}] < 1/2$ and proofs of four propositions above to Section B.

According to previous analysis, we prove the statement. □

5.3 Communication Complexity of Preparing Packed Triples over Bivariate Polynomials

We are now ready to derive a communication lower bound for preparing error-free packed Beaver triples over bivariate polynomials with perfect security with abort whose functionality is characterized as $\mathcal{F}_{\text{packed-triple-abort}}$. As a side-note, for protocols preparing preprocessing data, there is no gap between security with abort and security with identifiable abort. As if one has a protocol realizing security with abort, then in the event of an abort, after all parties open their views during the protocol execution, a disputed pair is expected to be identified, achieving security with identifiable abort. This indicates that a protocol that securely computes $\mathcal{F}_{\text{packed-triple-abort}}$ would imply the existence of a protocol that securely computes $\mathcal{F}_{\text{packed-triple}}$.

To establish a lower bound for $\mathcal{F}_{\text{packed-triple-abort}}$, we employ the technique of sacrificing to construct an efficient protocol that verifies the correctness of packed triples assuming error-free packed triples. In particular, using the batch-wise verification technique in [BH08], we can efficiently verify whether all (honest) parties jointly hold consistent linear secret sharings. Towards verifying the correctness of a packed Beaver triple $([\boldsymbol{a}]_{t+k-1}, [\boldsymbol{b}]_{t+k-1}, [\boldsymbol{c}]_{t+k-1})$, all parties first prepare an error-free packed Beaver triple $([\tilde{\boldsymbol{a}}]_{t+k-1}, [\tilde{\boldsymbol{b}}]_{t+k-1}, [\tilde{\boldsymbol{c}}]_{t+k-1})$ with a guarantee that $\tilde{\boldsymbol{a}} * \tilde{\boldsymbol{b}} = \tilde{\boldsymbol{c}}$, make use of it to compute their shares of $[\boldsymbol{a}*\boldsymbol{b} - \boldsymbol{c}]_{t+2k-2}$ following Beaver multiplication template. After verifying all parties honestly perform the computation of $[\boldsymbol{a} * \boldsymbol{b} - \boldsymbol{c}]_{t+2k-2}$, which corresponds to a task of verifying consistency of linear secret sharings, to check the correctness of $([\boldsymbol{a}]_{t+k-1}, [\boldsymbol{b}]_{t+k-1}, [\boldsymbol{c}]_{t+k-1})$, they only need to verify whether $[\boldsymbol{a} * \boldsymbol{b} - \boldsymbol{c}]_{t+2k-2}$ is a valid degree-$(t + 2k - 2)$ packed Shamir sharing with secret $\boldsymbol{0}$, which again corresponds to a task of verifying consistency of linear secret sharings. Notice that, when this verification fails, the adversary will additionally

learn the information regarding $\boldsymbol{a} * \boldsymbol{b} - \boldsymbol{c}$. However, this suffices to achieve our goal as we are only targeted for perfect security with abort, where privacy is not guaranteed in the event of an abort.

This effectively reduces the task of verifying packed triples to that of preparing error-free packed triples. As a result, a communication lower bound of verifying packed triples would imply that of preparing error-free packed triples. The communication lower bound is stated in Theorem 7, and the protocol for verifying packed triples is presented in Vrfy-Triple-Abort, with its security property concluded in Lemma 4.

Functionality 5: $\mathcal{F}_{\text{packed-triple-abort}}(N, k)$

1. $\mathcal{F}_{\text{packed-triple-abort}}$ receives a set $\mathcal{C}$ of corrupted parties' identities.
2. $\mathcal{F}_{\text{packed-triple-abort}}$ receives from the adversary shares of corrupted parties $\{f^a_{\ell,i}(x), g^a_{\ell,i}(y), f^b_{\ell,i}(x), g^b_{\ell,i}(y), f^c_{\ell,i}(x), g^c_{\ell,i}(y)\}_{\ell\in[N],i\in\mathcal{C}}$ with a promise that
   - $f^z_{\ell,i}(x), g^z_{\ell,i}(y)$ are degree-$(t+k-1)$ polynomials for $\ell \in [N], i \in \mathcal{C}, z \in \{a,b,c\}$,
   - $f^z_{\ell,i}(\alpha_j) = g^z_{\ell,j}(\alpha_i)$ for every $\ell \in [N], i, j \in \mathcal{C}, z \in \{a,b,c\}$.
3. For $\ell \in [N]$, $\mathcal{F}_{\text{packed-triple-abort}}$ samples a packed triple over bivariate polynomials $(\llbracket \boldsymbol{A}_\ell \rrbracket_{t+k-1}, \llbracket \boldsymbol{B}_\ell \rrbracket_{t+k-1}, \llbracket \boldsymbol{C}_\ell \rrbracket_{t+k-1})$, where $\boldsymbol{C}_\ell = \boldsymbol{A}_\ell * \boldsymbol{B}_\ell$, $\boldsymbol{A}_\ell = (A_{\ell,ij})_{ij}$, $\boldsymbol{B}_\ell = (B_{\ell,ij})_{ij}$, and $\boldsymbol{C}_\ell = (C_{\ell,ij})_{ij}$, according to shares of corrupted parties. Concretely, $\mathcal{F}_{\text{packed-triple-abort}}$ samples three degree-$(t+k-1, t+k-1)$ bivariate polynomials $S^a_\ell(x,y)$, $S^b_\ell(x,y)$, $S^c_\ell(x,y)$ such that
   - $S^a_\ell(\beta_i, \beta_j) = A_{\ell,ij}$, $S^b_\ell(\beta_i, \beta_j) = B_{\ell,ij}$, and $S^c_\ell(\beta_i, \beta_j) = C_{\ell,ij}$ for $x', y' \in [k]$,
   - $S^z_\ell(x, \alpha_i) = f^z_{\ell,i}(x), S^z_\ell(\alpha_i, y) = g^z_{\ell,i}(y)$ for $i \in \mathcal{C}, z \in \{a,b,c\}$.
4. $\mathcal{F}_{\text{packed-triple-abort}}$ waits for an instruction from the adversary.
   - Upon receiving `accept`, for each $\ell \in [N], i \in [n], z \in \{a,b,c\}$, compute $f^z_{\ell,i}(x) := S^z_\ell(x, \alpha_i), g^z_{\ell,i}(y) := S^z_\ell(\alpha_i, y)$ and send $\{f^z_{\ell,i}(x), g^z_{\ell,i}(y)\}_{\ell\in[N],z\in\{a,b,c\}}$ to P_i as its shares of $\{\llbracket \boldsymbol{A}_\ell \rrbracket_{t+k-1}, \llbracket \boldsymbol{B}_\ell \rrbracket_{t+k-1}, \llbracket \boldsymbol{C}_\ell \rrbracket_{t+k-1}\}_{\ell\in[N]}$.
   - Upon receiving `reject`, output `abort` to all parties.

Theorem 7. *Let $n = 3t+1$, $k \in \mathbb{N}$ such that $1 \le k \le \frac{t+2}{2}$, and $N > n$. There exists a constant $c > 0$ such that for all $k > c$, any protocol computing $\mathcal{F}_{\text{packed-triple-abort}}(N, k)$ with perfect security against a fully malicious adversary controlling at most t corrupted parties communicates at least $\frac{N \cdot k^2 \cdot n}{32}$ field elements in expectation, when $\mathcal{F}_{\text{packed-triple-abort}}$ doesn't abort. This lower bound implies an amortized cost of $\Omega(n)$ elements per secret.*

Protocol 15: Vrfy-Triple-Abort(N, k)

All parties hold their shares of $\{[\boldsymbol{a}_\ell]_{t+k-1}, [\boldsymbol{b}_\ell]_{t+k-1}, [\boldsymbol{c}_\ell]_{t+k-1}\}_{\ell\in[N]}$ and aim to check whether $\boldsymbol{c}_\ell = \boldsymbol{a}_\ell * \boldsymbol{b}_\ell$ for each $\ell \in [N]$.

1: **Preparing packed triples.** All parties invoke $\mathcal{F}_{\text{packed-triple-abort}}(N, k)$ to obtain N packed triples over bivariate polynomials and transform them into packed triples over univariate polynomials, which are denoted by $\{[\tilde{\boldsymbol{a}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{b}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{c}}_\ell]_{t+k-1}\}_{\ell\in[N]}$.[a]
   - If $\mathcal{F}_{\text{packed-triple-abort}}$ outputs `abort`, all parties take 0 as output and abort. Otherwise, all parties proceed.

2: **Preparing random masks.** All parties invoke $\mathcal{F}_{\text{rand-sharing}}(\Sigma_1, N)$ to prepare N random degree-$(t+2k-2)$ packed Shamir sharings $\{[\boldsymbol{o}_\ell]_{t+2k-2}\}_{\ell\in[N]}$ with each $\boldsymbol{o}_\ell = \boldsymbol{0} \in \mathbb{F}^k$.

3: **Sacrifice.** For each $\ell \in [N]$, all parties sacrifice a packed triple $([\tilde{\boldsymbol{a}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{b}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{c}}_\ell]_{t+k-1})$ to check the correctness of $([\boldsymbol{a}_\ell]_{t+k-1}, [\boldsymbol{b}_\ell]_{t+k-1}, [\boldsymbol{c}_\ell]_{t+k-1})$ as follows.

1. All parties locally compute $[\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{t+k-1} \leftarrow [\boldsymbol{a}_\ell]_{t+k-1} + [\tilde{\boldsymbol{a}}_\ell]_{t+k-1}$, $[\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{t+k-1} \leftarrow [\boldsymbol{b}_\ell]_{t+k-1} + [\tilde{\boldsymbol{b}}_\ell]_{t+k-1}$ and send them to P_{king}.
2. P_{king} reconstructs the secrets $\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell, \boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell$ and distributes $[\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{k-1}, [\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{k-1}$ to all parties.

3. All parties locally compute

$$\begin{aligned}[\boldsymbol{\theta}_\ell]_{t+2k-2} \leftarrow & [\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{k-1} \cdot [\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{k-1} - [\tilde{\boldsymbol{a}}_\ell]_{t+k-1} \cdot [\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{k-1} \\ & - [\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{k-1} \cdot [\tilde{\boldsymbol{b}}_\ell]_{t+k-1} + [\tilde{\boldsymbol{c}}_\ell]_{t+k-1} \\ & - [\boldsymbol{c}_\ell]_{t+k-1} + [\boldsymbol{o}_\ell]_{t+2k-2}.\end{aligned}$$

4: **Verification.** All parties first invoke $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_7, 2N)$ with their shares of $\{([\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{t+k-1}, [\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{k-1})\}_{\ell\in[N]}$ and $\{([\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{t+k-1}, [\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{k-1})\}_{\ell\in[N]}$ to check whether P_{king} honestly follows the protocol.

- If $\mathcal{F}_{\text{vrfy-linear-abort}}$ outputs 0, all parties take 0 as output.
- Otherwise, all parties proceed as follows.

All parties then invoke $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_1, N)$ with their shares of $\{[\boldsymbol{\theta}_\ell]_{t+2k-2}\}_{\ell\in[N]}$ to check whether they hold valid degree-$(t+2k-2)$ packed Shamir sharings with secret $\mathbf{0} \in \mathbb{F}^k$.

- If $\mathcal{F}_{\text{vrfy-linear-abort}}$ output 0, all parties take 0 as output.
- Otherwise, all parties take 1 as output.

[a] Suppose all parties obtain their shares of $[\![\boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{B}]\!]_{t+k-1}, [\![\boldsymbol{C}]\!]_{t+k-1}$ by invoking $\mathcal{F}_{\text{packed-triple-abort}}$, and each party P_i obtains its shares as $(f_i^a(x), g_i^a(y), f_i^b(x), g_i^b(y), f_i^c(x), g_i^c(y))$, which are degree-$(t+k-1)$ polynomials. Suppose $\boldsymbol{A} = (\boldsymbol{a}_j)_{j\in[k]}$, where $\boldsymbol{a}_j \in \mathbb{F}^k$ is the j-th row of $\boldsymbol{A}$. Then for each $j \in [k]$, P_i can locally compute its share of $[\boldsymbol{a}_j]_{t+k-1}$ as $g_i^a(\beta_j)$. Similarly, P_i can also locally transform its shares of $[\![\boldsymbol{B}]\!]_{t+k-1}, [\![\boldsymbol{C}]\!]_{t+k-1}$ into its shares of packed Shamir sharings over univariate polynomials.

Lemma 4. *Let $t \le (n-1)/3$, $k \in \mathbb{N}$ such that $1 \le k \le \frac{t+2}{2}$ and $N > n$. $\Pi_{\text{VRFY-TRIPLE-ABORT}}(N,k)$ computes $\mathcal{F}_{\text{vrfy-triple-abort}}(N,k)$ with perfect security in $\mathcal{F}_{\text{packed-triple-abort}}$, $\mathcal{F}_{\text{rand-sharing}}$, $\mathcal{F}_{\text{vrfy-linear-abort}}$-hybrid model against any fully malicious adversary controlling t corrupted parties. The communication of $\Pi_{\text{VRFY-TRIPLE-ABORT}}(N,k)$ is $O(N \cdot n)$ field elements plus one invocation of $\mathcal{F}_{\text{packed-triple-abort}}(\lceil \frac{N}{k} \rceil, k)$.*

Proof (Proof of Lemma 4). We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties. The core of the simulation is to ensure privacy for honest parties is protected when verification succeeds, making use of the randomness of error-free packed Beaver triples and degree-$(t+2k-2)$ packed Shamir sharings with secret $\mathbf{0}$.

The simulator $\mathcal{S}$ internally invokes the adversary with randomness sampled by itself and works as follows. First, $\mathcal{S}$ and all honest parties invoke $\mathcal{F}_{\text{vrfy-triple-abort}}$ with honest parties providing their shares of $\{([\boldsymbol{a}_\ell]_{t+k-1}, [\boldsymbol{b}_\ell]_{t+k-1}, [\boldsymbol{c}_\ell]_{t+k-1})\}_{\ell\in[N]}$. Then $\mathcal{S}$ receives corrupted parties' shares of $\{([\boldsymbol{a}_\ell]_{t+k-1}, [\boldsymbol{b}_\ell]_{t+k-1}, [\boldsymbol{c}_\ell]_{t+k-1})\}_{\ell\in[N]}$ from $\mathcal{F}_{\text{vrfy-triple-abort}}$ and records them for the simulation below.

1. In Step 1, $\mathcal{S}$ emulates $\mathcal{F}_{\text{packed-triple-abort}}$ by receiving from the adversary corrupted parties' shares of $\{([\tilde{\boldsymbol{a}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{b}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{c}}_\ell]_{t+k-1})\}_{\ell\in[N]}$ and an instruction.
   - If $\mathcal{S}$ receives `reject`, $\mathcal{S}$ outputs `abort` on behalf of $\mathcal{F}_{\text{packed-triple-abort}}$. Then $\mathcal{S}$ sends `reject` to $\mathcal{F}_{\text{vrfy-triple-abort}}$ and go to the last step of the simulation.
   - If $\mathcal{S}$ receives `accept`, $\mathcal{S}$ sends back corrupted parties' shares to the adversary on behalf of $\mathcal{F}_{\text{packed-triple-abort}}$, and proceeds the simulation.
2. In Step 2, $\mathcal{S}$ emulates $\mathcal{F}_{\text{rand-sharing}}(\Sigma_1)$ by receiving corrupted parties' shares of $\{[\boldsymbol{o}_\ell]_{t+2k-2}\}_{\ell\in[N]}$ from the adversary and sending them back to the adversary.

   Simulation of Sacrifice.
3. In Step 3.1, for each $\ell \in [N]$, $\mathcal{S}$ computes corrupted parties' shares of $[\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{t+k-1}$ and $[\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{t+k-1}$ as $[\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{t+k-1} \leftarrow [\boldsymbol{a}_\ell]_{t+k-1} + [\tilde{\boldsymbol{a}}_\ell]_{t+k-1}$, $[\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{t+k-1} \leftarrow [\boldsymbol{b}_\ell]_{t+k-1} + [\tilde{\boldsymbol{b}}_\ell]_{t+k-1}$, samples random degree-$(t+k-1)$ packed Shamir sharings as $[\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{t+k-1}$ and $[\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{t+k-1}$ based on corrupted parties' shares, and sends the shares of honest parties to P_{king} on behalf of honest parties.
4. In Step 3.2, if P_{king} is honest, $\mathcal{S}$ follows the protocol honestly. On receiving honest parties' shares of $\overline{[\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{k-1}}, \overline{[\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{k-1}}$ from P_{king}, $\mathcal{S}$ checks as follows.

- If one of them does not form a valid degree-$(k-1)$ packed Shamir sharing, $\mathcal{S}$ latter will output 0 when $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_7)$ is invoked.
- If they both form valid degree-$(k-1)$ packed Shamir sharings, $\mathcal{S}$ recovers the whole sharings and checks whether the secrets of $[\overline{\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell}]_{k-1}$ and $[\overline{\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell}]_{k-1}$ are the same as that of $[\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{t+k-1}$ and $[\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{t+k-1}$, respectively. If not, $\mathcal{S}$ latter will output 0 when $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_7)$ is invoked.

Simulation of verification and output.

5. In Step 4, when emulating $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_7)$, $\mathcal{S}$ sends honest parties' shares of $\{[\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{t+k-1}, [\overline{\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell}]_{k-1}\}_{\ell\in[N]}$ and $\{[\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{t+k-1}, [\overline{\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell}]_{k-1}\}_{\ell\in[N]}$ computed in Step 3 of the simulation to the adversary on behalf of $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_7)$.
   - If all checks pass in the last step, i.e., $\overline{\tilde{\boldsymbol{a}}_\ell + \boldsymbol{a}_\ell} = \tilde{\boldsymbol{a}}_\ell + \boldsymbol{a}_\ell$ and $\overline{\tilde{\boldsymbol{b}}_\ell + \boldsymbol{b}_\ell} = \tilde{\boldsymbol{b}}_\ell + \boldsymbol{b}_\ell$ for each $\ell \in [N]$, then S receives an instruction from the adversary.
     - If $\mathcal{S}$ receives `reject` from the adversary, $\mathcal{S}$ outputs 0 to the adversary on behalf of $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_7)$. Then $\mathcal{S}$ sends `reject` to $\mathcal{F}_{\text{vrfy-triple-abort}}$ and go to the last step of the simulation.
     - If $\mathcal{S}$ receives `accept` from the adversary, $\mathcal{S}$ outputs 1 to the adversary, and proceeds to next step.
   - If one of checks in the last step fails, $\mathcal{S}$ outputs 0 to the adversary on behalf of $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_7)$. Then $\mathcal{S}$ sends `reject` to $\mathcal{F}_{\text{vrfy-triple-abort}}$ and go to the last step of the simulation.
6. When emulating $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_1)$,
   - if $\mathcal{F}_{\text{vrfy-triple-abort}}$ asks for an instruction, which means all checks in $\mathcal{F}_{\text{vrfy-triple-abort}}$ pass, i.e., $\boldsymbol{\theta}_\ell = \mathbf{0}$ for each $\ell \in [N]$, the emulation of $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_1)$ is as follows. $\mathcal{S}$ first computes corrupted parties' shares of $[\boldsymbol{\theta}_\ell]_{t+2k-2}$ as $[\boldsymbol{\theta}_\ell]_{t+2k-2} \leftarrow [\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{k-1} \cdot [\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{k-1} - [\tilde{\boldsymbol{a}}_\ell]_{t+k-1} \cdot [\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{k-1} - [\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{k-1} \cdot [\tilde{\boldsymbol{b}}_\ell]_{t+k-1} + [\tilde{\boldsymbol{c}}_\ell]_{t+k-1} - [\boldsymbol{c}_\ell]_{t+k-1} + [\boldsymbol{o}_\ell]_{t+2k-2}$ for each $\ell \in [N]$. Based on the secrets $\boldsymbol{\theta}_\ell = \mathbf{0}$ and the shares of corrupted parties, $\mathcal{S}$ randomly samples N degree-$(t+2k-2)$ packed Shamir sharings $\{[\boldsymbol{\theta}_\ell]_{t+2k-2}\}_{\ell\in[N]}$. Then $\mathcal{S}$ sends honest parties' shares of $\{[\boldsymbol{\theta}_\ell]_{t+2k-2}\}_{\ell\in[N]}$ to the adversary on behalf of $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_1)$.
     - If $\mathcal{S}$ receives `reject` from the adversary, $\mathcal{S}$ outputs 0 to the adversary on behalf of $\mathcal{F}_{\text{vrfy-linear-abort}}$, sends `reject` to $\mathcal{F}_{\text{vrfy-triple-abort}}$, and go to the last step of the simulation.
     - If $\mathcal{S}$ receives `accept` from the adversary, $\mathcal{S}$ outputs 1 to the adversary on behalf of $\mathcal{F}_{\text{vrfy-linear-abort}}$, and sends `accept` to $\mathcal{F}_{\text{vrfy-triple-abort}}$.
   - If the check in $\mathcal{F}_{\text{vrfy-triple-abort}}$ fails, i.e., there exists $\ell \in [N]$ such that $\boldsymbol{a}_\ell * \boldsymbol{b}_\ell \neq \boldsymbol{c}_\ell$, $\mathcal{S}$ receives from $\mathcal{F}_{\text{vrfy-triple-abort}}$ all parties' shares of $\{([\boldsymbol{a}_\ell]_{t+k-1}, [\boldsymbol{b}_\ell]_{t+k-1}, [\boldsymbol{c}_\ell]_{t+k-1})\}_{\ell\in[N]}$. For each $\ell \in [N]$, in order to sample honest parties' shares of $[\boldsymbol{\theta}_\ell]_{t+2k-2}$,
     (a) $\mathcal{S}$ computes $\boldsymbol{a}_\ell, \boldsymbol{b}_\ell, \boldsymbol{c}_\ell$ and $\boldsymbol{\theta}_\ell \leftarrow \boldsymbol{a}_\ell * \boldsymbol{b}_\ell - \boldsymbol{c}_\ell$.
     (b) $\mathcal{S}$ samples a random degree-$(t+2k-2)$ packed Shamir sharing as $[\boldsymbol{\theta}_\ell]_{t+2k-2}$ according to the secret $\boldsymbol{\theta}_\ell$ and shares of corrupted parties.

     $\mathcal{S}$ reveals to the adversary honest parties' shares of $\{[\boldsymbol{\theta}_\ell]_{t+2k-2}\}_{\ell\in[N]}$ and outputs 0 to the adversary on behalf of $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_1)$.
7. $\mathcal{S}$ outputs what the adversary outputs and honest parties output what they receive from $\mathcal{F}_{\text{vrfy-triple-abort}}$.

We now show that the distribution of the output in the ideal world is identical to that in the real world using the following hybrid arguments.

$\mathbf{Hyb}_0$: In this hybrid, we consider the execution in the real world.

$\mathbf{Hyb}_1$: In this hybrid, instead of invoking $\mathcal{F}_{\text{packed-triple-abort}}$, $\mathcal{S}$ emulates the interaction between $\mathcal{F}_{\text{packed-triple-abort}}$ and the adversary itself by receiving the shares of corrupted parties and an instruction from the adversary. If $\mathcal{S}$ receives `abort`, $\mathcal{S}$ samples the whole sharings and sends them to the adversary. Otherwise, $\mathcal{S}$ sends corrupted parties' shares back to the adversary and delays the sampling of honest parties' shares. $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same distributions.

$\mathbf{Hyb}_2$: In this hybrid, instead of invoking $\mathcal{F}_{\text{rand-sharing}}(\Sigma_1)$, $\mathcal{S}$ emulates the interaction between $\mathcal{F}_{\text{rand-sharing}}$ and the adversary itself by receiving the shares of corrupted parties and sending them back to the adversary. $\mathcal{S}$ delays the sampling of honest parties' shares of $\{[\boldsymbol{o}_\ell]_{t+2k-2}\}_{\ell\in[N]}$. $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ have the same distributions.

$\mathbf{Hyb}_3$: In this hybrid, we change the way of generating honest parties' shares of $[\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{t+k-1}$ and $[\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{t+k-1}$. We first compute corrupted parties' shares of $[\boldsymbol{a}_\ell + \tilde{\boldsymbol{a}}_\ell]_{t+k-1}$ and $[\boldsymbol{b}_\ell + \tilde{\boldsymbol{b}}_\ell]_{t+k-1}$, sample random

degree-$(t+k-1)$ packed Shamir sharings as $[\boldsymbol{a}_\ell+\tilde{\boldsymbol{a}}_\ell]_{t+k-1}$ and $[\boldsymbol{b}_\ell+\tilde{\boldsymbol{b}}_\ell]_{t+k-1}$ based on shares of corrupted parties. Based on the secrets of $[\boldsymbol{a}_\ell+\tilde{\boldsymbol{a}}_\ell]_{t+k-1}, [\boldsymbol{b}_\ell+\tilde{\boldsymbol{b}}_\ell]_{t+k-1}$, we compute $\tilde{\boldsymbol{a}}_\ell, \tilde{\boldsymbol{b}}_\ell$ as $\tilde{\boldsymbol{a}}_\ell \leftarrow (\boldsymbol{a}_\ell+\tilde{\boldsymbol{a}}_\ell)-\boldsymbol{a}_\ell, \tilde{\boldsymbol{b}}_\ell \leftarrow (\boldsymbol{b}_\ell+\tilde{\boldsymbol{b}}_\ell)-\boldsymbol{b}_\ell$ and $\tilde{\boldsymbol{c}}_\ell \leftarrow \tilde{\boldsymbol{a}}_\ell * \tilde{\boldsymbol{b}}_\ell$. Then based on corrupted parties' shares of $[\tilde{\boldsymbol{a}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{b}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{c}}_\ell]_{t+k-1}$ and the computed secrets, we can compute honest parties' shares of $[\tilde{\boldsymbol{a}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{b}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{c}}_\ell]_{t+k-1}$ and hence their shares of packed triples over bivariate polynomials. Due to the randomness of $[\tilde{\boldsymbol{a}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{b}}_\ell]_{t+k-1}$, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ have the same distributions.

$\mathbf{Hyb}_4$: In this hybrid, instead of invoking $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_7)$ to check the consistency of pairs of packed Shamir sharings $\{([\boldsymbol{a}_\ell+\tilde{\boldsymbol{a}}_\ell]_{t+k-1}, [\overline{\boldsymbol{a}_\ell+\tilde{\boldsymbol{a}}_\ell}]_{k-1})\}_{\ell\in[N]}$, $\{([\boldsymbol{b}_\ell+\tilde{\boldsymbol{b}}_\ell]_{t+k-1}, [\overline{\boldsymbol{b}_\ell+\tilde{\boldsymbol{b}}_\ell}]_{k-1})\}_{\ell\in[N]}$, $\mathcal{S}$ emulates the behavior of $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_7)$ as described in the Step 5 of the simulation. By functionality of $\mathcal{F}_{\text{vrfy-linear-abort}}$, $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ have the same distributions.

$\mathbf{Hyb}_5$: In this hybrid, instead of invoking $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_1)$ to check all parties' shares of $\{[\boldsymbol{\theta}_\ell]_{t+2k-2}\}_{\ell\in[N]}$, $\mathcal{S}$ emulates $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_1)$ itself, which is feasible since $\mathcal{S}$ knows honest parties' shares of $\{[\tilde{\boldsymbol{a}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{b}}_\ell]_{t+k-1}, [\tilde{\boldsymbol{c}}_\ell]_{t+k-1}, [\boldsymbol{c}_\ell]_{t+k-1}, [\boldsymbol{o}_\ell]_{t+2k-2}\}_{\ell\in[N]}$ and thus can compute honest parties' shares of $\{[\boldsymbol{\theta}_\ell]_{t+2k-2}\}_{\ell\in[N]}$. $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ have the same distributions.

$\mathbf{Hyb}_6$: In this hybrid, we change the emulation of $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_1)$ as follows. $\mathcal{S}$ first learns the secrets $\{\boldsymbol{a}_\ell, \boldsymbol{b}_\ell, \boldsymbol{c}_\ell\}_{\ell\in[N]}$. For every $\ell\in[N]$, $\mathcal{S}$ checks whether $\boldsymbol{a}_\ell * \boldsymbol{b}_\ell = \boldsymbol{c}_\ell$.

- If this holds for every $\ell\in[N]$, $\mathcal{S}$ changes the way of computing honest parties' shares of $[\boldsymbol{\theta}_\ell]_{t+2k-2}$ by first computing corrupted parties' shares of $[\boldsymbol{\theta}_\ell]_{t+2k-2}$ and randomly sampling a degree-$(t+2k-2)$ packed Shamir sharing as $[\boldsymbol{\theta}_\ell]_{t+2k-2}$ based on shares of corrupted parties and the secret $\boldsymbol{\theta}_\ell = (\boldsymbol{a}_\ell+\tilde{\boldsymbol{a}}_\ell) * (\boldsymbol{b}_\ell+\tilde{\boldsymbol{b}}_\ell) - \tilde{\boldsymbol{a}}_\ell * (\boldsymbol{b}_\ell+\tilde{\boldsymbol{b}}_\ell) - (\boldsymbol{a}_\ell+\tilde{\boldsymbol{a}}_\ell) * \tilde{\boldsymbol{b}}_\ell + \tilde{\boldsymbol{c}}_\ell - \boldsymbol{c}_\ell = \boldsymbol{a}_\ell * \boldsymbol{b}_\ell - \boldsymbol{c}_\ell = \boldsymbol{0}$.
- If one of the checks fails, $\mathcal{S}$ changes the way of computing honest parties' shares of $[\boldsymbol{\theta}_\ell]_{t+2k-2}$ by first computing corrupted parties' shares of $[\boldsymbol{\theta}_\ell]_{t+2k-2}$ and randomly sampling a degree-$(t+2k-2)$ packed Shamir sharing as $[\boldsymbol{\theta}_\ell]_{t+2k-2}$ based on shares of corrupted parties and the secret $\boldsymbol{\theta}_\ell = \boldsymbol{a}_\ell * \boldsymbol{b}_\ell - \boldsymbol{c}_\ell$.

Then $\mathcal{S}$ computes honest parties' shares of $[\boldsymbol{a}_\ell * \boldsymbol{b}_\ell - \boldsymbol{c}_\ell]_{t+2k-2}$ as $[\boldsymbol{a}_\ell * \boldsymbol{b}_\ell - \boldsymbol{c}_\ell]_{t+2k-2} \leftarrow [\boldsymbol{a}_\ell+\tilde{\boldsymbol{a}}_\ell]_{k-1} \cdot [\boldsymbol{b}_\ell+\tilde{\boldsymbol{b}}_\ell]_{k-1} - [\tilde{\boldsymbol{a}}_\ell]_{t+k-1} \cdot [\boldsymbol{b}_\ell+\tilde{\boldsymbol{b}}_\ell]_{k-1} - [\boldsymbol{a}_\ell+\tilde{\boldsymbol{a}}_\ell]_{k-1} \cdot [\tilde{\boldsymbol{b}}_\ell]_{t+k-1} + [\tilde{\boldsymbol{c}}_\ell]_{t+k-1} - [\boldsymbol{c}_\ell]_{t+k-1}$ and computes their shares of $[\boldsymbol{o}_\ell]_{t+2k-2}$ as $[\boldsymbol{o}_\ell]_{t+2k-2} \leftarrow [\boldsymbol{\theta}_\ell]_{t+2k-2} - [\boldsymbol{a}_\ell * \boldsymbol{b}_\ell - \boldsymbol{c}_\ell]_{t+2k-2}$. $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ have the same distributions due to the randomness of $[\boldsymbol{o}_\ell]_{t+2k-2}$ and the fact that $\tilde{\boldsymbol{a}}_\ell * \tilde{\boldsymbol{b}}_\ell = \tilde{\boldsymbol{c}}_\ell$ for each $\ell\in[N]$.

$\mathbf{Hyb}_7$: In this hybrid, we do not compute honest parties' shares of $[\boldsymbol{o}_\ell]_{t+2k-2}$, $[\tilde{\boldsymbol{a}}_\ell]_{t+k-1}$, $[\tilde{\boldsymbol{b}}_\ell]_{t+k-1}$, and $[\tilde{\boldsymbol{c}}_\ell]_{t+k-1}$, which does not affect the output distribution as they are not used in $\mathbf{Hyb}_6$. Furthermore, we let $\mathcal{F}_{\text{vrfy-triple-abort}}$ take the role of $\mathcal{S}$ to check whether the secrets of each packed triple $([\boldsymbol{a}_\ell]_{t+k-1}, [\boldsymbol{b}_\ell]_{t+k-1}, [\boldsymbol{c}_\ell]_{t+k-1})$ satisfy multiplication relations. $\mathcal{S}$ simulates honest parties' outputs as follows. If one of the following events happens, $\mathcal{S}$ sends `reject` to $\mathcal{F}_{\text{vrfy-triple-abort}}$.

- $\mathcal{S}$ outputs `abort` on behalf of $\mathcal{F}_{\text{packed-triple-abort}}$.
- $\mathcal{S}$ outputs 0 on behalf of $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_7)$.
- $\mathcal{S}$ receives `reject` from the adversary when emulating $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_1)$.

Otherwise, $\mathcal{S}$ sends `accept` to $\mathcal{F}_{\text{vrfy-triple-abort}}$. In doing this, $\mathcal{S}$ does not use honest parties' inputs for simulation. Since $\mathcal{F}_{\text{vrfy-triple-abort}}$ does the same as $\mathcal{S}$ in $\mathbf{Hyb}_6$, $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ have the same distributions.

Since $\mathbf{Hyb}_7$ corresponds to the ideal world, VRFY-TRIPLE-ABORT securely computes $\mathcal{F}_{\text{vrfy-triple-abort}}$ in the $\mathcal{F}_{\text{packed-triple-abort}}$, $\mathcal{F}_{\text{rand-sharing}}$, $\mathcal{F}_{\text{vrfy-linear-abort}}$-hybrid model. □

Proof (Proof of Theorem 7). Notice that the communication cost of VRFY-TRIPLE-ABORT is $4N \cdot n$ field elements, one invocation of $\mathcal{F}_{\text{packed-triple-abort}}(\lceil \frac{N}{k} \rceil, k)$ to prepare N degree-$(t+k-1)$ error-free packed Beaver triples, one invocation of $\mathcal{F}_{\text{rand-sharing}}(\Sigma_1, N)$ to prepare N random packed Shamir sharings in the form of $[\boldsymbol{0}]_{t+2k-2}$, one invocation of $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_7, 2N)$ to verify consistency of $2N$ linear secret sharings in the form of $([\boldsymbol{u}]_{t+k-1}, [\boldsymbol{u}]_{k-1})$, one invocation of $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma_1, N)$ to verify consistency of N linear secret sharings in the form of $[\boldsymbol{0}]_{t+2k-2}$. This amounts to $O(N \cdot n)$ elements plus one invocation of $\mathcal{F}_{\text{packed-triple-abort}}(\lceil \frac{N}{k} \rceil, k)$. By Lemma 4, plus instantiations of $\mathcal{F}_{\text{vrfy-linear-abort}}$ and $\mathcal{F}_{\text{rand-sharing}}$ sketched in Section A.2, the task of computing $\mathcal{F}_{\text{vrfy-triple-abort}}$ is reduced to computing $\mathcal{F}_{\text{packed-triple-abort}}$. Suppose $c' > 0$ is a constant such that the total concrete communication cost of VRFY-TRIPLE-ABORT(N, k) is no more than $c' \cdot N \cdot n$ elements.

Let π be any protocol computing $\mathcal{F}_{\text{packed-triple-abort}}(\lceil \frac{N}{k} \rceil, k)$ achieving perfect security against any adversary corrupting at most t parties. We set $c = \max\{32 \cdot c', 24\}$, where we need $c \geq 24$ to ensure the lower bound in Lemma 3 holds. For the sake of contradiction, suppose there exists $k \in \mathbb{N}$ with $k > c$ such that the communication of π is less than $\frac{\frac{N}{k} \cdot k^2 \cdot n}{32}$ field elements. By composing π, VRFY-TRIPLE-ABORT(N, k), and protocols instantiating $\mathcal{F}_{\text{vrfy-linear-abort}}$ and $\mathcal{F}_{\text{rand-sharing}}$, one can derive a protocol for computing $\mathcal{F}_{\text{vrfy-triple-abort}}(N, k)$ achieving perfect security in the plain model with communication of less than $\frac{\frac{N}{k} \cdot k^2 \cdot n}{32} + c' \cdot N \cdot n < \frac{N \cdot k \cdot n}{16}$ elements if $k > 32 \cdot c'$. This contradicts the lower bound established in Lemma 3, which requires all parties to communicate at least $\frac{N \cdot k \cdot n}{16}$ elements. Thus, we complete the proof of Theorem 7. □

More generally, when considering the setting of sub-optimal corruption threshold, where at most $t = \frac{n-s}{3}$ corrupted parties are controlled by an adversary for some $s > 1$, we obtain a communication lower bound stated as Corollary 1. To show this, we follow the framework in [DLN19] and prove that to prepare N packed Beaver triples over bivariate polynomials, every group of s parties are expected to communicate at least $\Omega(N \cdot k^2)$ elements based on Lemma 3.

Corollary 1 (Sub-optimal corruption). *Let $n = 3t + s$, $k \in \mathbb{N}$ such that $1 \leq k \leq \frac{t+2}{2}$, and $N > n$. There exists a constant $c > 0$ such that for all $k > c$, any protocol computing $\mathcal{F}_{\text{packed-triple-abort}}(N, k)$ with perfect security against a fully malicious adversary controlling at most t corrupted parties communicates at least $\frac{N \cdot k^2 \cdot n/s}{32}$ field elements, when $\mathcal{F}_{\text{packed-triple-abort}}$ doesn't abort. This lower bound implies an amortized cost of $\Omega(n/s)$ elements per secret.*

Proof (Proof of Corollary 1). Before starting the proof, we notice in the proof of Lemma 3, the privacy against P_0 is not utilized to derive the lower bound. Specifically, we show that for all $N > \lceil \frac{24}{\log F} \rceil$ and $k > 24$, any protocol computing $\mathcal{F}_{\text{vrfy-triple-abort}}(N, k)$ with perfect security against any adversary controlling up to t parties in P_1, P_2, P_3 requires P_0 to communicate at least $\frac{N \cdot k}{8}$ elements. Our proof of this corollary will rely on this statement.

Suppose π is a $(3t + s)$-party protocol computing $\mathcal{F}_{\text{packed-triple-abort}}(\lceil \frac{N}{k} \rceil, k)$ with perfect security against any adversary corrupting at most t parties. We begin by dividing n positions of π into n/s disjoint groups of s positions, and consider an arbitrary group P of s positions. Recall $c' > 0$ is a constant such that the concrete communication cost in total of VRFY-TRIPLE-ABORT(N, k) is no more than $c' \cdot N \cdot n$ elements. We set $c = \max\{32 \cdot c', 24\}$. For the sake of contradiction, assume that there exists $k \in \mathbb{N}$ with $k > c$ such that the communication of π for s positions in P is less than $\frac{\frac{N}{k} \cdot k^2}{16} = \frac{N \cdot k}{16}$ elements.

Now we obtain a $(3t + 1)$-party protocol π' computing $\mathcal{F}_{\text{vrfy-triple-abort}}(N, k)$ as follows.

- All $3t + 1$ parties invoke the $(3t + s)$-party protocol π with P_0 emulating s positions in P to prepare $\lceil \frac{N}{k} \rceil$ degree-$(t + k - 1, t + k - 1)$ packed Beaver triples over bivariate polynomials. After that, P_0 obtains s parties' shares of packed triples and chooses one of them as its shares of packed triples.
- All $3t + 1$ parties then follow VRFY-TRIPLE-ABORT(N, k) to verify the correctness of packed triples.

On the one hand, if P_0 is honest, the resulting $(3t + 1)$-party protocol π' computes $\mathcal{F}_{\text{vrfy-triple-abort}}(N, k)$ with perfect security against any adversary corrupting up to t parties in $\{P_1, P_2, P_3\}$. On the other hand, if we avoid P_0 to play the role of P_{king} in VRFY-TRIPLE-ABORT, the communication of π' for P_0 is less than $\frac{N \cdot k}{16} + 2 \cdot c' \cdot N < \frac{N \cdot k}{8}$ elements if $k > 32 \cdot c'$. This contradicts the previously established lower bound, which requires P_0 to communicate at least $\frac{N \cdot k}{8}$ elements in any such protocol.

Therefore, every group of s positions of π should communicate at least $\frac{N \cdot k}{16}$ elements, leading to an overall communication of at least $\frac{N \cdot k \cdot n/s}{32}$ elements to instantiate $\mathcal{F}_{\text{packed-triple-abort}}(\lceil \frac{N}{k} \rceil, k)$. □

5.4 Communication Complexity of Inner Product Functions

In this part, we extend the approach used to prove a communication lower bound for verifying packed Beaver triples to establish a lower bound for computing the inner product functions. With a notice that computing

the inner product of two length-N vectors can be represented as a circuit of size $O(N)$, we prove that any protocol computing this with perfect security with abort communicates $\Omega(N \cdot n)$ elements.

Intuitively, the task of computing the inner product functions is similar to that of verifying whether triples satisfy the multiplication relations in spirit as the views of parties will contain their inputs and outputs, no matter the outputs are from computation or to be verified, in both cases. This analogue allows us to derive a contradiction by simultaneously appealing to three constraints: perfect privacy, perfect correctness, and the assumption that parties are limited to only have a small amount of communication, additionally exploring non-linearity of inner product functions. We formally conclude the communication lower bound for inner product functions in Theorem 8.

Theorem 8. *Let $n = 3t+1$ and $P_0, P_{1,1}, \ldots, P_{1,t}, P_{2,1}, \ldots, P_{2,t}, P_{3,1}, \ldots, P_{3,t}$ be n parties. The inner product function $f_{IP,N}$ is defined as follows. It first receives two vectors of N elements $\boldsymbol{x} \in \mathbb{F}^N$ and $\boldsymbol{y} \in \mathbb{F}^N$ from $P_{2,1}$ and $P_{3,1}$, respectively, then computes the inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$, and outputs z to $P_{2,1}$ and w to $P_{3,1}$, where z and w are sampled uniformly at random subject to $z + w = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$.*

There exists a constant $c > 0$ such that for all $N > c$, any protocol computing $f_{IP,N}$ with perfect security with abort against a fully malicious adversary controlling at most t corrupted parties satisfies that each party (other than $P_{2,1}, P_{3,1}$) has expected communication of at least $\frac{N}{9}$ elements, leading to total communication of at least $\frac{N \cdot (n-2)}{18}$ elements, when the functionality $f_{IP,N}$ doesn't abort.

Proof (Proof of Theorem 8). Suppose π is an n-party protocol computing $f_{\mathrm{IP},N}$ with perfect security with abort against any fully malicious adversary controlling at most t corrupted parties. We use $\mathcal{D}$ to denote the set of all honest executions that output correctly, i.e., $\mathcal{D} := \{d = (\boldsymbol{x}, \boldsymbol{y}; r_0, r_1, r_2, r_3) \mid (z, w) \leftarrow \mathtt{output}(\pi, d) \text{ s.t. } z + w = \langle \boldsymbol{x}, \boldsymbol{y} \rangle\}$. We assume each party has its internal randomness of finite size, which implies the set $\mathcal{D}$ has finite cardinality. We model $\mathcal{D}$ as the sample space, and assume a uniform distribution over $\mathcal{D}$. For any event $\mathcal{E} \subseteq \mathcal{D}$, we denote its probability by $\Pr_{d \leftarrow \mathcal{D}}[\mathcal{E}]$, or simply $\Pr_d[\mathcal{E}]$.

We consider communication complexity of $n-2$ parties who don't receive outputs. Without loss of generality, we focus on the communication complexity of P_0 and denote by a random variable $C_0 : \mathcal{D} \to \mathbb{N}$ the length of its transcript. We set $c = \lceil \frac{24}{\log F} \rceil$. For the sake of contradiction, suppose there exists N with $N > c$ such that the communication of π for P_0 is such that

$$\mathbb{E}_{d \leftarrow \mathcal{D}}[C_0] < \frac{N}{9}.$$

Following the proof framework in Lemma 3, recall the proof is divided into three steps.

Step 1. Define three honest executions satisfying several delicate requirements.
Step 2. Construct a dishonest execution with an incorrect output by composing three honest executions chosen in last step, which leads to a contradiction with perfect correctness of π.
Step 3. Prove existence of such three honest executions based on perfect privacy of π and the assumption that P_0 only communicated a few elements in expectation.

Step 1. Defining three honest executions. We start with sampling an honest execution $\mathtt{exec}_0$ by sampling $d = (\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}; r_0, r_1, r_2, r_3)$ from $\mathcal{D}$ uniformly at random. Suppose $(\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}, z^{(0)}, w^{(0)})$ is the tuple of inputs and outputs corresponding to d. Let $\mathcal{D}_1(d) \subset \mathcal{D}$ be the set of all tuples of the form $d' = (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}; r'_0, r'_1, r'_2, r'_3)$ satisfying the conditions.

- $\mathtt{view}^{P_2}(\pi, d) = \mathtt{view}^{P_2}(\pi, d')$, which implies $\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)}$, $r'_2 = r_2$, and $z^{(1)} = z^{(0)}$;
- $\mathtt{transcript}^{P_0}(\pi, d) = \mathtt{transcript}^{P_0}(\pi, d')$.

Then we define $\mathtt{event}$ 1 to an event that the number of possible distinct inputs of $P_{3,1}$ among the executions in the set $\mathcal{D}_1(d)$ is strictly greater than $F^{\frac{N}{2}} + 1$, i.e., $|\{\bar{\boldsymbol{y}} \in \mathbb{F}^N \mid \exists\ d' \in \mathcal{D}_1(d), d' \vdash \bar{\boldsymbol{y}}\}| > F^{\frac{N}{2}} + 1$.

Similarly, let $\mathcal{D}_2(d) \subseteq \mathcal{D}$ be the set of all tuples of the form $d'' = (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}; r''_0, r''_1, r''_2, r''_3)$ satisfying the following conditions.

- $\mathtt{view}^{P_3}(\pi, d) = \mathtt{view}^{P_3}(\pi, d'')$, which implies $\boldsymbol{y}^{(2)} = \boldsymbol{y}^{(0)}$, $r''_3 = r_3$, and $w^{(2)} = w^{(0)}$;

– $\texttt{transcript}^{P_0}(\pi, d) = \texttt{transcript}^{P_0}(\pi, d'')$.

We define $\texttt{event}$ 2 to be an event that the number of possible distinct inputs of $P_{2,1}$ among the executions in the set $\mathcal{D}_2(d)$ is strictly greater than $F^{\frac{N}{2}}+1$, i.e., $\{\bar{\boldsymbol{x}} \in \mathbb{F}^N \mid \exists\ d'' \in \mathcal{D}_2(d), d'' \vdash \bar{\boldsymbol{x}}\}| > F^{\frac{N}{2}}+1$. Latter, we write $\mathcal{D}_1$ and $\mathcal{D}_2$ short for $\mathcal{D}_1(d)$ and $\mathcal{D}_2(d)$, respectively.

Step 2. Constructing a dishonest execution. If $\texttt{event}$ 1 and $\texttt{event}$ 2 happen simultaneously, we claim this breaks down perfect correctness of π. In fact, we first pick $d' = (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}; r'_0, r'_1, r'_2, r'_3)$ and $d'' = (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}; r''_0, r''_1, r''_2, r''_3)$ as tuples of inputs and internal randomness satisfying two requirements of $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. I.e., $d' \in \mathcal{D}_1, d'' \in \mathcal{D}_2$. Let $\texttt{exec}_1 := \texttt{exec}(\pi, d')$ and $\texttt{exec}_2 := \texttt{exec}(\pi, d'')$. Then we construct a dishonest execution denoted by $\texttt{exec}_3(d', d'')$ as follows.

– P_0 uses r_0 (or equivalently, r'_0, r''_0) as its internal randomness, and honestly follows the protocol.
– P_2 takes $\boldsymbol{x}^{(2)}$ as its input, uses r''_2 as its internal randomness, and honestly follows the protocol.
– P_3 takes $\boldsymbol{y}^{(1)}$ as its input, uses r'_3 as its internal randomness, and honestly follows the protocol.
– An adversary controlling parties $\{P_{1,i}\}_{i\in[t]}$ launches a similar attack as in Lemma 2 and Lemma 3. In particular, P_1 first emulates $\texttt{exec}_0$, $\texttt{exec}_1$ and $\texttt{exec}_2$ to compute the messages it would send to P_0, P_2, P_3. Then malicious P_1 behaves inconsistently across parties. When interacting with P_0, P_1 uses the messages in $\texttt{exec}_0$ (or equivalently, $\texttt{exec}_1$, $\texttt{exec}_2$); when interacting with P_2, P_1 uses the messages in $\texttt{exec}_3$; when interacting with P_3, P_1 uses the messages in $\texttt{exec}_1$.

We show that if both $\texttt{event}$ 1 and $\texttt{event}$ 2 happen, there exist $d' \in \mathcal{D}_1, d'' \in \mathcal{D}_2$ such that $P_{2,1}, P_{3,1}$ will obtain an incorrect output in $\texttt{exec}_3(d', d'')$. Let $z^{(3)}$ and $w^{(3)}$ be the outputs of $P_{2,1}$ and $P_{3,1}$, respectively. On the one hand, to argue the output of $P_{2,1}$ and $P_{3,1}$ in $\texttt{exec}_3(d', d'')$, we rely on the following claim whose proof is analogous to its counterpart in Lemma 3, and is therefore omitted.

> *Claim.* The transcripts between P_2 and P_1 in executions $\texttt{exec}_3$ and $\texttt{exec}_2$ are identical. The transcripts between P_3 and P_1 in executions $\texttt{exec}_3$ and $\texttt{exec}_1$ are identical. The transcripts between P_2 and P_3 and transcripts between P_0 and each party in $\{P_1, P_2, P_3\}$ are identical in four executions $\texttt{exec}_0, \texttt{exec}_1, \texttt{exec}_2$ and $\texttt{exec}_3$.

According to this claim, when comparing $\texttt{exec}_3$ and $\texttt{exec}_2$, P_2 has the same views in both executions and thus P_2 would output $z^{(2)}$ in both executions, i.e., $z^{(3)} = z^{(2)}$. Similarly, we have P_3 would output $w^{(1)}$ in both $\texttt{exec}_3$ and $\texttt{exec}_1$, i.e., $w^{(3)} = w^{(1)}$.

On the other hand, we observe that there exist $d' \in \mathcal{D}_1$ and $d'' \in \mathcal{D}_2$ such that the inputs and outputs $(\boldsymbol{x}^{(3)}, \boldsymbol{y}^{(3)}, z^{(3)}, w^{(3)})$ in $\texttt{exec}_3$ does not satisfy $\langle \boldsymbol{x}^{(3)}, \boldsymbol{y}^{(3)} \rangle = z^{(3)} + w^{(3)}$. First, we have the following observation due to non-linearity of inner product functions.

> *Claim.* The tuple of inputs and outputs in $\texttt{exec}_3(d', d'')$ is $(\boldsymbol{x}^{(3)}, \boldsymbol{y}^{(3)}, z^{(3)}, w^{(3)}) = (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(1)}, z^{(2)}, w^{(1)})$. The output in $\texttt{exec}_3$ are **incorrect** with respect to input if $\langle \boldsymbol{x}^{(2)} - \boldsymbol{x}^{(0)}, \boldsymbol{y}^{(1)} - \boldsymbol{y}^{(0)} \rangle \neq 0$.
>
> *Proof.* Since $\texttt{exec}_0, \texttt{exec}_1$ and $\texttt{exec}_2$ have correct outputs with respect to their inputs, we have that $\langle \boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)} \rangle = z^{(0)} + w^{(0)}$, $\langle \boldsymbol{x}^{(0)}, \boldsymbol{y}^{(1)} \rangle = z^{(0)} + w^{(1)}$, and $\langle \boldsymbol{x}^{(2)}, \boldsymbol{y}^{(0)} \rangle = z^{(2)} + w^{(0)}$. Then we get
>
> $$\begin{aligned}
> &\langle \boldsymbol{x}^{(2)}, \boldsymbol{y}^{(1)} \rangle - (z^{(2)} + w^{(1)}) \\
> =&\langle \boldsymbol{x}^{(2)}, \boldsymbol{y}^{(1)} \rangle - (z^{(0)} + w^{(1)}) - (z^{(2)} + w^{(0)}) + (z^{(0)} + w^{(0)}) \\
> =&\langle \boldsymbol{x}^{(2)}, \boldsymbol{y}^{(1)} \rangle - \langle \boldsymbol{x}^{(0)}, \boldsymbol{y}^{(1)} \rangle - \langle \boldsymbol{x}^{(2)}, \boldsymbol{y}^{(0)} \rangle + \langle \boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)} \rangle \\
> =&\langle \boldsymbol{x}^{(2)} - \boldsymbol{x}^{(0)}, \boldsymbol{y}^{(1)} - \boldsymbol{y}^{(0)} \rangle,
> \end{aligned}$$
>
> which implies if $\langle \boldsymbol{x}^{(2)} - \boldsymbol{x}^{(0)}, \boldsymbol{y}^{(1)} - \boldsymbol{y}^{(0)} \rangle \neq 0$, then the output in $\texttt{exec}_3(d', d'')$ is incorrect. □

Let $V_1 = \{\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(0)} \mid \exists\ d' \in \mathcal{D}_1 \text{ s.t } d \vdash \boldsymbol{y}^{(1)}\}$ and $V_2 = \{\boldsymbol{x}^{(2)} - \boldsymbol{x}^{(0)} \mid \exists\ d'' \in \mathcal{D}_2 \text{ s.t } d \vdash \boldsymbol{x}^{(2)}\}$. Since both $\texttt{event}$ 1 and $\texttt{event}$ 2 happen, then we have $|V_1| > F^{\frac{N}{2}}$ and $|V_2| > F^{\frac{N}{2}}$. We claim there exist $\boldsymbol{v}_1 \in V_1, \boldsymbol{v}_2 \in V_2$ such that $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle \neq 0$.

For the sake of contradiction, suppose for every $\boldsymbol{v}_1 \in V_1$ and $\boldsymbol{v}_2 \in V_2$, $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle = 0$. First, we span V_1 and V_2 to linear subspaces of $\mathbb{F}^N$, i.e., $V_1' = \mathsf{span}(V_1)$, $V_2' = \mathsf{span}(V_2)$. It is clear that for any $\boldsymbol{v}_1 \in V_1'$ and $\boldsymbol{v}_2 \in V_2'$, we have $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle = 0$ and thus $V_1' \perp V_2'$, which implies that $\mathsf{dim}V_1' + \mathsf{dim}V_2' \leq N$. However, by observing that $|V_1'| \geq |V_1| > F^{\frac{N}{2}}, |V_2'| \geq |V_2| > F^{\frac{N}{2}}$, we have that $\mathsf{dim}V_1' > \frac{N}{2}, \mathsf{dim}V_2' > \frac{N}{2}$ and thus $\mathsf{dim}V_1' + \mathsf{dim}V_2' > N$, which leads to a contradiction.

Then according to the claim above, it is guaranteed that there exist $d' \in \mathcal{D}_1, d'' \in \mathcal{D}_2$ such that the outputs in $\mathtt{exec}_3(d', d'')$ are incorrect with respect to its inputs, which contradicts with perfect correctness of π.

Step 3. Existence of three honest executions. Following Lemma 3, it suffices to prove $\Pr_{d \leftarrow \mathcal{D}}[\overline{\mathtt{event}\ 1}] < \frac{1}{2}$. The intuition for upper bounding $\Pr_{d \leftarrow \mathcal{D}}[\overline{\mathtt{event}\ 1}]$ is as follows.

- Given the view of P_2 $\mathtt{view}^{P_2}$, we first show that for any input of $P_{3,1}$ $\bar{\boldsymbol{y}}$, there is an execution consistent with $\mathtt{view}^{P_2}$ and $\bar{\boldsymbol{y}}$, based on perfect privacy against corrupted P_2. Traversing all $\bar{\boldsymbol{y}}$ leads a set of sufficiently many executions consistent with $\mathtt{view}^{P_2}$. This is formally stated as the first claim below.
- Next, assuming P_0 only has a limited amount of communication in expectation, for some transcript of P_0 $\mathtt{transcript}^{P_0}$, there are sufficiently many executions consistent with both $\mathtt{view}^{P_2}$ and $\mathtt{transcript}^{P_0}$.
- Finally, by privacy against P_2, we have that, given the view of P_2 $\mathtt{view}^{P_2}$, the executions consistent with $\mathtt{view}^{P_2}$ are uniformly distributed over all possible values for the input of $P_{3,1}$ $\bar{\boldsymbol{y}}$. This is formally stated as the second claim below, whose proof is analogous to its counterpart in Lemma 3, and therefore is omitted.
- Consequently, there is a large number of executions consistent with both $\mathtt{view}^{P_2}$ and $\mathtt{transcript}^{P_0}$ covers a large number of distinct possible values for $\bar{\boldsymbol{y}}$, which in turn implies $\mathtt{event}\ 1$ happens.

Claim. Let $\mathtt{view}^{P_2}$ be the view of P_2 generated by an execution with correct outputs, i.e., there exists a tuple $d = (\boldsymbol{x}, \boldsymbol{y}; r_0, r_1, r_2, r_3) \in \mathcal{D}$ such that $\mathtt{view}^{P_2}(\pi, d) = \mathtt{view}^{P_2}$. Then for every $\bar{\boldsymbol{y}} \in \mathbb{F}^N$, there exists a tuple $\bar{d} = (\boldsymbol{x}, \bar{\boldsymbol{y}}; \bar{r_0}, \bar{r_1}, r_2, \bar{r_3}) \in \mathcal{D}$ of all parties' inputs and internal randomness such that $\mathtt{view}^{P_2}(\pi, \bar{d}) = \mathtt{view}^{P_2}$.
Let $\mathcal{D}_{\mathtt{view}^{P_2}} \subset \mathcal{D}$ be the set of all honest executions satisfying the view of P_2 is $\mathtt{view}^{P_2}$. Then the number of possible distinct inputs of P_2 among the executions in the set $\mathcal{D}_{\mathtt{view}^{P_2}}$ is F^N, i.e.,

$$|\{\bar{\boldsymbol{y}} \in \mathbb{F}^N \mid \exists\ \bar{d} \in \mathcal{D}_{\mathtt{view}^{P_2}} \text{ s.t. } \bar{d} \vdash \bar{\boldsymbol{y}}\}| = F^N.$$

Proof. By perfect privacy against a (semi-honest) corrupted P_2, there exists a simulator $\mathcal{S}_2$ which takes as input the input and output of P_2, and outputs the view of P_2 such that for any input $\bar{\boldsymbol{y}}$ of $P_{3,1}$, it holds that

$$\mathcal{S}_2(\boldsymbol{x}, f^2_{\mathrm{IP},N}(\boldsymbol{x}, \bar{\boldsymbol{y}})) \equiv \mathtt{view}^{P_2}(\pi, \boldsymbol{x}, \bar{\boldsymbol{y}}),$$

where $f^2_{\mathrm{IP},N}(\boldsymbol{x}, \boldsymbol{y})$ denotes the output distribution of $P_{2,1}$, which is a uniform distribution over $\mathbb{F}$ regardless of values of $(\boldsymbol{x}, \boldsymbol{y})$. Hence, for every $\bar{\boldsymbol{y}}$, by property of $f^2_{\mathrm{IP},N}$, we have

$$\begin{aligned}
\mathtt{view}^{P_2}(\pi, \boldsymbol{x}, \bar{\boldsymbol{y}}) &\equiv \mathcal{S}_2(\boldsymbol{x}, f^2_{\mathrm{IP},N}(\boldsymbol{x}, \bar{\boldsymbol{y}})) \\
&\equiv \mathcal{S}_2(\boldsymbol{x}, f^2_{\mathrm{IP},N}(\boldsymbol{x}, \boldsymbol{y})) \\
&\equiv \mathtt{view}^{P_2}(\pi, \boldsymbol{x}, \boldsymbol{y}).
\end{aligned}$$

Then there exists a choice of a tuple of internal randomness from P_0, P_1 and P_3, $(\bar{r_0}, \bar{r_1}, \bar{r_3})$, such that $\mathtt{view}^{P_2}(\boldsymbol{x}, \bar{\boldsymbol{y}}; \bar{r_0}, \bar{r_1}, r_2, \bar{r_3}) = \mathtt{view}^{P_2}(\pi, d)$.
By the reasoning above, for any $\bar{\boldsymbol{y}} \in \mathbb{F}^N$, $\bar{\boldsymbol{y}}$ can lead to a satisfied execution in $\mathcal{D}_{\mathtt{view}^{P_2}}$. Since there are F^N possibilities of $\bar{\boldsymbol{y}}$, we conclude the number of possible distinct inputs of $P_{3,1}$ among the executions in $\mathcal{D}_{\mathtt{view}^{P_2}}$ is F^N. □

Claim. Given a view of P_2 $\mathtt{view}^{P_2}$, we define the set $\mathcal{D}_{\mathtt{view}^{P_2}}$ as in the previous claim. For any two vectors $\bar{\boldsymbol{y}}_1, \bar{\boldsymbol{y}}_2 \in \mathbb{F}^N$, the number of executions in $\mathcal{D}_{\mathtt{view}^{P_2}}$ with $P_{3,1}$'s input being $\bar{\boldsymbol{y}}_1$ is equal to the number of executions with $P_{3,1}$'s input being $\bar{\boldsymbol{y}}_2$, i.e.,

$$|\{d \in \mathcal{D}_{\mathtt{view}^{P_2}} \mid d \vdash \bar{\boldsymbol{y}}_1\}| = |\{d \in \mathcal{D}_{\mathtt{view}^{P_2}} \mid d \vdash \bar{\boldsymbol{y}}_2\}|.$$

The computation of $\Pr_{d\leftarrow\mathcal{D}}[\overline{\texttt{event 1}}]$ is also similar to that in Lemma 3. We present it here for completeness. Let E_1 be the event $(C_0(d) < 3\cdot\mathbb{E}[C_0])$. For simplicity of notation, we abuse the notation E_1 to also denote the set of all executions in which $C_0(d) < 3\cdot\mathbb{E}[C_0]$, i.e., $E_1 = \{d\in\mathcal{D} \mid C_0(d) < 3\cdot\mathbb{E}[C_0]\}$.

$$
\begin{aligned}
&\Pr_{d\leftarrow\mathcal{D}}[\overline{\texttt{event 1}}]\\
=&\sum_{\texttt{view}^{P_2}} \Pr_{d\leftarrow\mathcal{D}}[\overline{\texttt{event 1}}\wedge d\vdash\texttt{view}^{P_2}]\\
=&\sum_{\texttt{view}^{P_2}} \Pr_{d\leftarrow\mathcal{D}}[\overline{\texttt{event 1}}\wedge E_1\wedge d\vdash\texttt{view}^{P_2}] + \Pr_{d\leftarrow\mathcal{D}}[\overline{\texttt{event 1}}\wedge \overline{E_1}\wedge d\vdash\texttt{view}^{P_2}]\\
=&\sum_{\texttt{view}^{P_2}} \Pr_{d\leftarrow\mathcal{D}}[\overline{\texttt{event 1}}\mid E_1\wedge d\vdash\texttt{view}^{P_2}]\cdot\Pr_{d\leftarrow\mathcal{D}}[E_1\wedge d\vdash\texttt{view}^{P_2}]\\
&+\Pr_{d\leftarrow\mathcal{D}}[\overline{\texttt{event 1}}\mid \overline{E_1}\wedge d\vdash\texttt{view}^{P_2}]\cdot\Pr_{d\leftarrow\mathcal{D}}[\overline{E_1}\wedge d\vdash\texttt{view}^{P_2}]\\
=&\sum_{\texttt{view}^{P_2}} \Pr_{d\leftarrow\mathcal{D}}[\overline{\texttt{event 1}}\mid E_1\wedge d\vdash\texttt{view}^{P_2}]\cdot\Pr_{d\leftarrow\mathcal{D}}[E_1\wedge d\vdash\texttt{view}^{P_2}]\\
&+\Pr_{d\leftarrow\mathcal{D}}[\overline{\texttt{event 1}}\mid \overline{E_1}\wedge d\vdash\texttt{view}^{P_2}]\cdot(\Pr_{d\leftarrow\mathcal{D}}[d\vdash\texttt{view}^{P_2}] - \Pr_{d\leftarrow\mathcal{D}}[E_1\wedge d\vdash\texttt{view}^{P_2}])
\end{aligned}
$$

$$
\begin{aligned}
<&\sum_{\texttt{view}^{P_2}} \frac{1}{6}\cdot\frac{1}{\Pr_{d\leftarrow\mathcal{D}}[E_1\mid d\vdash\texttt{view}^{P_2}]}\cdot\Pr_{d\leftarrow\mathcal{D}}[E_1\wedge d\vdash\texttt{view}^{P_2}]\\
&+1\cdot(\Pr_{d\leftarrow\mathcal{D}}[d\vdash\texttt{view}^{P_2}] - \Pr_{d\leftarrow\mathcal{D}}[E_1\wedge d\vdash\texttt{view}^{P_2}])\\
=&\sum_{\texttt{view}^{P_2}} \frac{1}{6}\cdot\Pr_{d\leftarrow\mathcal{D}}[d\vdash\texttt{view}^{P_2}] + \Pr_{d\leftarrow\mathcal{D}}[d\vdash\texttt{view}^{P_2}] - \Pr_{d\leftarrow\mathcal{D}}[E_1\wedge d\vdash\texttt{view}^{P_2}]\\
=&1+\frac{1}{6}-\Pr_{d\leftarrow\mathcal{D}}[E_1]\\
\le&1+\frac{1}{6}-(1-\frac{1}{3})\\
=&\frac{1}{2}.
\end{aligned}
$$

The deduction above takes advantage of the following two inequalities:

1. $\Pr_{d\leftarrow\mathcal{D}}[\overline{\texttt{event 1}}\mid E_1\wedge d\vdash\texttt{view}^{P_2}] < \frac{1}{6}\cdot\frac{1}{\Pr_{d\leftarrow\mathcal{D}}[E_1\mid d\vdash\texttt{view}^{P_2}]}$. We prove this as below.

   To compute the probability, we first introduce some notations. For simplicity, we write $\texttt{t}^{P_0}$ short for $\texttt{transcript}^{P_0}$. Define $\mathcal{D}_{\texttt{view}^{P_2}}\subset\mathcal{D}$ to the set of all executions in which the view of P_2 is $\texttt{view}^{P_2}$, i.e., $\mathcal{D}_{\texttt{view}^{P_2}} = \{d\in\mathcal{D}\mid d\vdash\texttt{view}^{P_2}\}$. Define $\texttt{Bad}_{\texttt{view}^{P_2},\texttt{t}^{P_0}}\subset\mathcal{D}$ to be the set of all executions in which

   - the view of P_2 is $\texttt{view}^{P_2}$,
   - the transcript of P_0 is $\texttt{t}^{P_0}$,
   - the possibilities of distinct values of the input of $P_{3,1}$ is no more than $F^{\frac{N}{2}}$.

Notice that

$$
\begin{aligned}
&\Pr_{d\leftarrow\mathcal{D}}[\overline{\texttt{event 1}} \mid E_1 \wedge d \vdash \texttt{view}^{P_2}] \\
=&\frac{\sum_{\texttt{t}^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\texttt{t}^{P_0}}|}{|E_1 \cap \mathcal{D}_{\texttt{view}^{P_2}}|} && (1)\\
\leq&\frac{|\{\texttt{t}^{P_0}(\pi,d) \mid d \in \mathcal{D}_{\texttt{view}^{P_2}}\}| \cdot \max_{\texttt{t}^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\texttt{t}^{P_0}}|}{|E_1 \cap \mathcal{D}_{\texttt{view}^{P_2}}|} && (2)\\
\leq&\frac{F^{3\cdot\mathbb{E}[C_0]} \cdot \max_{\texttt{t}^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\texttt{t}^{P_0}}|}{|E_1 \cap \mathcal{D}_{\texttt{view}^{P_2}}|} && (3)\\
=&\frac{F^{3\cdot\mathbb{E}[C_0]} \cdot \max_{\texttt{t}^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\texttt{t}^{P_0}}|}{|\mathcal{D}_{\texttt{view}^{P_2}}|} \cdot \frac{|\mathcal{D}_{\texttt{view}^{P_2}}|}{|E_1 \cap \mathcal{D}_{\texttt{view}^{P_2}}|} \\
\leq&\frac{F^{3\cdot\mathbb{E}[C_0]} \cdot (F^{\frac{N}{2}}+1)}{F^N} \cdot \frac{|\mathcal{D}_{\texttt{view}^{P_2}}|}{|E_1 \cap \mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}|} && (4)\\
<&\frac{1}{6} \cdot \frac{|\mathcal{D}_{\texttt{view}^{P_2}}|}{|E_1 \cap \mathcal{D}_{\texttt{view}^{P_2}}|} && (5)\\
=&\frac{1}{6} \cdot \frac{1}{\Pr_{d\leftarrow\mathcal{D}}[E_1 \mid d \vdash \texttt{view}^{P_2}]},
\end{aligned}
$$

where the step-by-step reasoning is given as follows. To show Equality 1, we group all executions $E_1 \cap \mathcal{D}_{\texttt{view}^{P_2}}$ according to P_0's transcript. Then $\texttt{event}$ 1 doesn't happen if and only if the sampled execution d lies in the set $\texttt{Bad}_{\texttt{view}^{P_2},\texttt{t}^{P_0}}$ for some $\texttt{t}^{P_0}$.
Inequality 2 holds because

$$
\begin{aligned}
&\sum_{\texttt{t}^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\texttt{t}^{P_0}}| \\
\leq&\sum_{\texttt{t}^{P_0}} \max_{\texttt{t}^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\texttt{t}^{P_0}}| \\
=&|\{\texttt{t}^{P_0}(\pi,d) \mid d \in \mathcal{D}_{\texttt{view}^{P_2}}\}| \cdot \max_{\texttt{t}^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\texttt{t}^{P_0}}|
\end{aligned}
$$

As for Inequality 3, because for any execution in E_1, the length of P_0's transcript is at most $3\cdot\mathbb{E}[C_0]$, resulting in at most $F^{3\cdot\mathbb{E}[C_0]}$ possibilities.
Regarding Inequality 4, by two claims above and the definition of $\texttt{Bad}_{\texttt{view}^{P_2},\texttt{t}^{P_0}}$, we have

$$
\frac{\max_{\texttt{t}^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\texttt{t}^{P_0}}|}{|\mathcal{D}_{\texttt{view}^{P_2}}|} \leq \frac{(F^{\frac{N}{2}}+1)\cdot|\{d \in \mathcal{D}_{\texttt{view}^{P_2}} \mid d \vdash \bar{\boldsymbol{y}}\}|}{F^N \cdot |\{d \in \mathcal{D}_{\texttt{view}^{P_2}} \mid d \vdash \bar{\boldsymbol{y}}\}|} = \frac{F^{\frac{N}{2}}+1}{F^N},
$$

for any $\bar{\boldsymbol{y}} \in \mathbb{F}^N$ representing the input of $P_{3,1}$.
Finally, regarding Inequality 5, by $N > \lceil \frac{24}{\log F} \rceil$ and the assumption that $\mathbb{E}_d[C_0] < \frac{N}{9}$, we have that

$$
\frac{F^{3\cdot\mathbb{E}[C_0]} \cdot (F^{\frac{N}{2}}+1)}{F^N} < \frac{2\cdot F^{\frac{N}{3}}}{F^{\frac{N}{2}}} = \frac{2}{F^{\frac{N}{6}}} < \frac{1}{6}.
$$

2. $\Pr_{d\leftarrow\mathcal{D}}[E_1] \geq 1 - \frac{1}{3}$. This can be proved in the same way as in Lemma 3.

By previous analysis, we prove the statement. □

Notice in the proof of Theorem 8, the privacy against P_0 is not utilized to derive the lower bound. Specifically, we show that for all $N > \lceil \frac{24}{\log F} \rceil$, any protocol computing $f_{\mathrm{IP},N}$ with perfect security with abort against any adversary controlling up to t parties in P_1, P_2, P_3 requires P_0 to communicate at least $\frac{N}{9}$ elements.

Based on this, we generalize Theorem 8 to the sub-optimal corruption setting and to the preprocessing setting as follows.

Following a similar reasoning as Corollary 1, we get a communication lower bound for inner product functions when corruption threshold is $\frac{n-s}{3}$, for some $s > 1$. In particular, we show any protocol that computes $f_{\text{IP},N}$ with perfect security with abort communicates at least $\Omega(N \cdot n/s)$ elements in expectation, which is formally stated as Corollary 2.

Corollary 2 (Sub-optimal corruption). *Let $n = 3t + s$. There exists a constant $c > 0$ such that for all $N > c$, any protocol computing the inner product of two length-N vectors $f_{IP,N}$ with perfect security with abort against a fully malicious adversary controlling at most t corrupted parties communicates st least $\frac{N\cdot(n-2)}{18\cdot s}$ elements in expectation, when the functionality $f_{IP,N}$ doesn't abort.*

Proof (Proof of Corollary 2). Suppose π is a $(3t + s)$-party protocol computing $f_{\text{IP},N}$ with perfect security against with abort any adversary corrupting at most t parties. We begin by dividing the $n - 2$ positions in π who don't have inputs into $(n - 2)/s$ disjoint groups of s parties and consider an arbitrary group P of s parties. We set $c = \lceil \frac{24}{\log F} \rceil$. For the sake of contradiction, assume that there exists N with $N > c$ such that the communication of π for parties in P is less than $\frac{N}{9}$ elements.

Now letting P_0 emulate s positions in P, this would yield an $(3t + 1)$-party protocol π' for $f_{\text{IP},N}$. On the one hand, if P_0 is honest, π' computes $f_{\text{IP},N}$ with perfect security with abort against any adversary corrupting up to t parties in $\{P_1, P_2, P_3\}$. On the other hand, the communication of π' for P_0 is less than $\frac{N}{9}$ elements. This contradicts the previously established lower bound, which requires P_0 to communicate at least $\frac{N}{9}$ elements in any such protocol. Therefore, every group of s parties should communicate at least $\frac{N}{9}$ elements, leading to an overall communication of at least $\frac{N\cdot(n-2)}{18\cdot s}$ elements to securely compute $f_{\text{IP},N}$. □

We further extend the lower bound to the preprocessing setting. In particular, we show that any protocol that computes $f_{\text{IP},N}$ with perfect security with abort in the preprocessing model either communicates $\Omega(N\cdot n)$ elements, or requires preprocessing data of size at least $\Omega(N)$, which is formally stated as Corollary 3.

Corollary 3 (Preprocessing data size). *Let $n = 3t + 1$. There exists a constant $c > 0$ such that for all $N > c$, any protocol computing $f_{IP,N}$ with perfect security with abort in the prerocessing model against at most t corrupted parties either communicates at least $\frac{N\cdot(n-2)}{36}$ elements in expectation, or requires preprocessing data of size at least $\frac{N}{18}$, when the functionality $f_{IP,N}$ doesn't abort.*

Proof (Proof of Corollary 3). Suppose there is a protocol π computing $f_{\text{IP},N}$ in the preprocessing model with perfect security with abort against any fully malicious adversary controlling at most t corrupted parties. We set $c = \lceil \frac{24}{\log F} \rceil$. For the sake of contradiction, assume that there exists N with $N > c$ such tha 1) a party who doesn't have input, say P_0, communicates less than $\frac{N}{18}$ elements in expectation in π and 2) the size of preprocessing data in π is less than $\frac{N}{18}$.

Now we derive a protocol π' in the plain model by letting P_0 first distribute the preprocessing data to all parties and then letting all parties run the protocol π to compute $f_{\text{IP},N}$. On the one hand, if P_0 is an honest party, then π' is secure against any adversary corrupting up to t parties in P_1, P_2, P_3. On the other hand, by assumption, the expected communication cost for P_0 in π' is less than $\frac{N}{18} + \frac{N}{18} = \frac{N}{9}$ elements. This contradicts the previously established lower bound, which requires P_0 to communicate at least $\frac{N}{9}$ elements in any such protocol. Hence, either the communication of π is at least $\frac{N\cdot(n-2)}{36}$ elements, or the preprocessing data of π is of size at least $\frac{N}{18}$, completing the proof of the corollary. □

6 Preparing Packed Beaver Triples

Recall we build a perfectly secure MPC protocol for general circuits with $O(C)$ communication in the preprocessing model. The preprocessing data is of size $O(C)$ and contains $O(C/k^2)$ packed Beaver triples over bivariate polynomials. However, we prove preparing $O(C/k^2)$ packed Beaver triples even with perfect

security with abort requires communicating $\Omega(C/k^2 \cdot k^2 \cdot n) = \Omega(C \cdot n)$ elements, amounting to $\Omega(n)$ elements per triple. To show the tightness of this lower bound, we present a protocol to prepare N packed Beaver triples over bivariate polynomials with $O(N \cdot n^3)$ elements, amounting to $O(n)$ elements per secret if $k = \Omega(n)$ and therefore matching the lower bound.

To prepare a degree-$(t+k-1, t+k-1)$ packed Beaver triple $(\llbracket \boldsymbol{A} \rrbracket_{t+k-1}, \llbracket \boldsymbol{B} \rrbracket_{t+k-1}, \llbracket \boldsymbol{C} \rrbracket_{t+k-1})$ over bivariate polynomials, notice that the sharing process of secret sharing over bivariate polynomials is a linear circuit taking input as the secrets to be shared and the randomness. Based on this observation, all parties first prepare k^2 degree-t Beaver triples and random degree-t Shamir sharings, whose secrets will play the role of $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ and the randomness of the sharing process, respectively, and fully determine the whole bivariate polynomials. Then they can compute degree-t sharings of each evaluation point of the bivariate polynomials and send to each party P_i their shares of the points that are supposed to be held by P_i, who then can reconstruct the secrets and compute its shares of $(\llbracket \boldsymbol{A} \rrbracket_{t+k-1}, \llbracket \boldsymbol{B} \rrbracket_{t+k-1}, \llbracket \boldsymbol{C} \rrbracket_{t+k-1})$.

We present the protocol in Packed-Triple(N, k). The security of Packed-Triple(N, k) follows directly from the error correcting property of degree-t Shamir sharings and is therefore omitted. We conclude its security property as Lemma 5.

The communication cost to prepare N degree-$(t+k-1, t+k-1)$ packed Beaver triple over bivariate polynomials is $3N \cdot n^2 \cdot (t+k)$ elements plus one invocation of $\mathcal{F}_{\text{triple}}(N \cdot k^2)$ and one invocations of $\mathcal{F}_{\text{rand-sharing}}(\Sigma_4, 3N \cdot ((t+k)^2 - k^2))$, amounting to $\mathsf{P2P}(O(N \cdot n^3)) + O(1) \times \mathsf{BA}(1) + O(1) \times \mathsf{BC}(O(\log N + \log n + \log |\mathbb{F}|))$, and therefore $O(n)$ elements per secret.

Lemma 5. *Let $t = (n-1)/3$ and $k \leq t+1$. Packed-Triple(N, k) computes $\mathcal{F}_{\text{packed-triple}}(N, k)$ with perfect security in the $\mathcal{F}_{\text{rand-sharing}}$, $\mathcal{F}_{\text{triple}}$-hybrid model against a fully malicious adversary who controls at most t corrupted parties.*

Protocol 16: Packed-Triple(N, k)

For every $\ell \in [N]$, all parties prepare a packed Beaver triple over bivariate polynomials $(\llbracket \boldsymbol{A}_\ell \rrbracket_{t+k-1}, \llbracket \boldsymbol{B}_\ell \rrbracket_{t+k-1}, \llbracket \boldsymbol{C}_\ell \rrbracket_{t+k-1})$ as follows:

1: **Preparing triples.** All parties invoke $\mathcal{F}_{\text{triple}}(k^2)$ to obtain k^2 degree-t Beaver triples $\{[a_{ij}]_t, [b_{ij}]_t, [c_{ij}]_t\}_{i,j \in [k]}$. They set three matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}^{k \times k}$ of secrets as $\boldsymbol{A} \leftarrow (a_{ij})_{ij}$, $\boldsymbol{B} \leftarrow (b_{ij})_{ij}$, and $\boldsymbol{C} \leftarrow (c_{ij})_{ij}$.

- In case of failure, output a new disputed pair E.

2: **Preparing random sharings.** All parties invoke $\mathcal{F}_{\text{rand-sharing}}(\Sigma_4, 3N \cdot ((t+k)^2 - k^2))$ to obtain $3N \cdot ((t+k)^2 - k^2)$ random degree-t Shamir sharings.

- In case of failure, output a new disputed pair E.

3: **Computing bivariate polynomials.** For each $\boldsymbol{U} = (u_{ij})_{ij} \in \{\boldsymbol{A}_\ell, \boldsymbol{B}_\ell, \boldsymbol{C}_\ell\}_{\ell \in [N]}$, all parties compute a degree-$(t+k-1, t+k-1)$ packed Shamir sharing $\llbracket \boldsymbol{U} \rrbracket_{t+k-1}$ as follows.

1. All parties pick $(t+k)^2 - k^2$ unused random degree-t Shamir sharings $\{[r_{ij}]_t\}_{(i,j) \in [t+k] \times [t+k], (i,j) \notin [k] \times [k]}$.
2. All parties set a degree-$(t+k-1, t+k-1)$ bivariate $S(s, y)$ such that

$$\forall\ (i,j) \in [k] \times [k], [S(\beta_i, \beta_j)]_t = [u_{ij}]_t,$$
$$\forall\ (i,j) \in [t+k] \times [t+k] \setminus [k] \times [k], [S(\alpha_i, \alpha_j)]_t = [r_{ij}]_t.$$

3. For each $i \in [n]$, all parties compute the shares of P_i as follows.
   (a) For every $j \in [t+k]$, all parties compute their shares of $[S(\alpha_j, \alpha_i)]_t$ and $[S(\alpha_i, \alpha_j)]_t$. They then send these shares to P_i.
   (b) On receiving messages from all parties, each P_i reconstructs the shares to learn $\{S(\alpha_j, \alpha_i)\}_{j \in [t+k]}$ and $\{S(\alpha_i, \alpha_j)\}_{j \in [t+k]}$, computes two polynomials $f_i(x), g_i(y)$ as its shares of $\llbracket \boldsymbol{U} \rrbracket_{t+k-1}$, where $f_i(x) := S(x, \alpha_i), g_i(y) := S(\alpha_i, y)$.

7 Packed Beaver Triples from 3-Party Beaver Triples

Recall we construct a perfectly secure MPC protocol with $O(C)$ communication assuming preprocessing data of packed Beaver triples over bivariate polynomials of size $O(C)$. However, we establish a communication lower bound that preparing such kind of packed Beaver triples requires $\Omega(n)$ communication per secret, failing to achieve overall $O(C)$ communication in the plain model. To account for this, we present a protocol preparing such kind of packed Beaver triples with communication of $O(n)$ elements per secret. In this section, we follow the idea in [SY25] which focuses on honest majority setting and reduces preparing packed Beaver triples of size $O(C)$ to $O(C)$ pairwise random oblivious linear evaluation (OLE) correlations, to reduce packed Beaver triples over bivariate polynomials of size $O(C)$ to $O(C)$ 3-party Beaver triples over bivariate polynomials.

The reason we resort to 3-party Beaver triples is due to the insufficiency of pairwise correlated randomness. In particular, We conjecture that pairwise correlated randomness may be insufficient to prepare n-party packed Beaver triples over bivariate polynomials. Intuitively, corrupted parties may change their preprocessing data and proceed honestly. If the correlation still holds, though only happen with negligible probability, the protocol succeeds; however, with pairwise correlations, corrupted parties do not know whether it does apriori. The fact that the protocol succeeds would leak partial information about honest parties' preprocessing data to the adversary, violating perfect secrecy. Instead, we use 3-party Beaver triples over bivariate polynomials. It guarantees that corrupted parties will learn themselves whether the correlation holds with the modified preprocessing data, avoiding the above selective-failure attack.

To prepare n party packed Beaver triples from 3-party Beaver triples, following [GLS24,SY25], we make use of packed triple extraction, an extension of triple extraction proposed in [CP17], to prepare packed Beaver triples. Specifically, each party first distributes packed Beaver triples to all parties. Then all parties use these initial $N = 3n$ packed Beaver triples with $n - 1 \sim \frac{N}{3}$ of them are known to the adversary to establish three polynomials $\boldsymbol{F}, \boldsymbol{G}$ and $\boldsymbol{H} := \boldsymbol{F} * \boldsymbol{G}$, where $\boldsymbol{F}, \boldsymbol{G}$ are of degree $\frac{N-1}{2}$ and $\boldsymbol{H}$ is of degree $N - 1$. So far all parties hold their shares of $\frac{N+1}{2}$ packed triples. Considering the correctness, among the initial N packed triples, a number of $2n + 1 \sim \frac{2N}{3}$ packed triples distributed by honest parties are guaranteed to satisfy the multiplication relations. To ensure the correctness of the extracted packed triples, we need honest parties to additionally check a number of $n - 1 \sim \frac{N}{3}$ packed triples, which would imply N packed triples distributed or checked by honest parties are correct, and therefore three polynomials $\boldsymbol{F}, \boldsymbol{G}$ and $\boldsymbol{H}$ are correctly established. Regarding the privacy, we count the number of extracted random packed triples. Since $n - 1 \sim \frac{N}{3}$ packed triples are checked by honest parties, $\frac{n-1}{2} \sim \frac{N}{6}$ packed triples are checked by corrupted parties and thus have been revealed to the adversary, leading to a number of $\frac{n-1}{2} + n - 1 \sim \frac{N}{2}$ packed triples known to the adversary. Hence, only a number of $\frac{3n+1}{2} - \frac{3n-3}{2} = O(1)$ random packed triples can be extracted, leading to $O(n)$ elements per secret and failing to achieve $O(1)$ elements per secret.

To resolve this issue, we follow [SY25] to use the idea of party virtualization, which is first proposed in [Bra87]. Based on this idea, [SY25] manages to boost the corruption threshold outside the virtual parties to satisfy a strong honest majority which allows packed triple extraction to extract a constant fraction of random packed triples on the one hand, and emulates each virtual party who is expected to distribute packed triples to all parties using pairwise random OLE correlations on the other hand. Adapting to our setting of perfect security, we group each tuple of three parties into a virtual party and rely on three-party degree-$(1, 1)$ Beaver triples over bivariate polynomials. Such Beaver triples are shared among three parties via degree-$(1, 1)$ bivariate polynomials, enabling them to emulate a virtual party that distributes packed triples.

In doing this, for a virtual party containing at least two honest parties, the secrets of the Beaver triples are fully determined by shares of honest parties. Relying on this redundancy, checking the consistency of three sharings distributed by three parties inside this virtual party ensures packed triples distributed by this virtual party satisfy the multiplication relations with secrets unknown to the adversary. Consequently, if consistency checks pass, a $1 - (\frac{1}{3})^3 - 3 \cdot (\frac{1}{3})^2 \cdot \frac{2}{3} = \frac{20}{27}$ fraction of original packed triples correctly satisfies the multiplication relations and is unknown to the adversary. Considering the correctness, to make sure three polynomials $\boldsymbol{F}, \boldsymbol{G}$ and $\boldsymbol{H}$ are correctly computed, all parties perform the procedure of triple verification where honest parties check a $1 - \frac{20}{27} = \frac{7}{27}$ fraction of packed triples. Regarding the privacy, since a $\frac{7}{27}$ fraction

of packed triples is checked by honest parties, a $\frac{7}{27} \cdot \frac{1}{2} = \frac{3.5}{27}$ fraction is also checked by corrupted parties and loses their privacy. Eventually, all parties manage to extract a $\frac{1}{2} - \frac{7}{27} - \frac{3.5}{27} = \frac{1}{9}$ fraction of random packed Beaver triples, leading to $O(1)$ elements plus $O(1)$ three-party Beaver triples per secret.

Moreover, to achieve security with identifiable abort, all parties are expected to identify a disputed pair when checks fail. When the consistency check fails and a virtual party distributing inconsistent sharings is located, three parties in this virtual party open their degree-$(1,1)$ sharings, and a disputed pair or a corrupted party can be identified by relying on the cross checking property of secret sharings via bivariate polynomials. When the triple verification fails and a virtual party distributing a packed triple not satisfying the multiplication relation is located, by previous analysis, this virtual party can only contain at most one honest party. Hence, any two parties of the three could be viewed as a disputed pair.

We first introduce our notations regarding three-party Beaver triples below and describe the functionality preparing three-party degree-$(1,1)$ Beaver triples as $\mathcal{F}_{\text{3-triple}}$. We show the protocol for preparing packed Beaver triples over bivariate polynomials assuming three-party Beaver triples in Packed-Triple under the party elimination framework and conclude its security property in Lemma 6.

The communication cost for Packed-Triple to prepare a number of $\frac{(n'^3)}{9}$ degree-$(t+k-1, t+k-1)$ packed Beaver triples is $\mathsf{P2P}(O(n'^5)) + O(1) \times \mathsf{BA}(1) + O(1) \times \mathsf{BC}(O(\log n' + \log |\mathbb{F}|))$ plus $k^2 \cdot (n')^3$ invocations of $\mathcal{F}_{\text{3-triple}}$, and thus $O(1)$ elements plus $O(1)$ three-party Beaver triples per secret.

Notations. We will use $[\![s]\!]_1$ to denote a Shamir sharing of $x \in \mathbb{F}$ shared via a degree-$(1,1)$ bivariate polynomial $S(x,y)$ satisfying $S(0,0) = s$ among three parties P_{ℓ_1}, P_{ℓ_2} and P_{ℓ_3}, where each party P_{ℓ_v} takes two degree-1 polynomials $f_v^S(x) := S(x,v), g_v^S(y) := S(v,y)$ as its share of $[\![s]\!]_1$. Since degree-$(1,1)$ Shamir sharing is only used in the three-party setting (and not in the n-party setting), this notation is unambiguous.

Let $\boldsymbol{U} \in \mathbb{F}[X]^{k \times k}$ be a $k \times k$ matrix of degree-1 univariate polynomials. For $v \in \{1,2,3\}$, let $\boldsymbol{U}(v) \in \mathbb{F}^{k \times k}$ denote the matrix obtained by evaluating each polynomial in $\boldsymbol{U}$ at $x = v$.

Functionality 6: $\mathcal{F}_{\text{3-triple}}$

$\mathcal{F}_{\text{3-triple}}$ receives three parties' identities (ℓ_1, ℓ_2, ℓ_3), where $\ell_1, \ell_2, \ell_3 \in \mathcal{P}_{\texttt{active}}$, and receives a set $\mathcal{C}$ of identities of corrupted parties. According to $|\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\}|$, there are four cases:

1. If $\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\} = \emptyset$, $\mathcal{F}_{\text{3-triple}}$ samples a random degree-$(1,1)$ three-party triple $([\![A]\!]_1, [\![B]\!]_1, [\![C]\!]_1)$ such that $C = A \cdot B$, and distributes it to three parties.
2. If $\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\} = \{\ell_u\}$, supposing $\{\ell_v, \ell_w\} = \{\ell_1, \ell_2, \ell_3\} \setminus \{\ell_u\}$, $\mathcal{F}_{\text{3-triple}}$ receives $\{f_u^U(x), g_u^U(y)\}_{U \in \{A,B,C\}}$ [a] from the adversary, randomly samples a degree-$(1,1)$ three-party triple $([\![A]\!]_1, [\![B]\!]_1, [\![C]\!]_1)$ such that $U(x,u) = f_u^U(x), U(u,y) = g_u^U(y)$ for every $U \in \{A,B,C\}$, and distributes it to three parties.
3. If $\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\} = \{\ell_u, \ell_v\}$, supposing $\{\ell_w\} = \{\ell_1, \ell_2, \ell_3\} \setminus \{\ell_u, \ell_v\}$, $\mathcal{F}_{\text{3-triple}}$ receives $\{f_u^U(x), g_u^U(y), f_v^U(x), g_v^U(y)\}_{U \in \{A,B,C\}}$ from the adversary, computes a degree-$(1,1)$ three-party triple $([\![A]\!]_1, [\![B]\!]_1, [\![C]\!]_1)$ such that $U(x,u) = f_u^U(x), U(u,y) = g_u^U(y), U(x,v) = f_v^U(x), U(v,x) = g_v^U(y)$ for every $U \in \{A,B,C\}$, and distributes it to three parties.
4. If $\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\} = \{\ell_1, \ell_2, \ell_3\}$, $\mathcal{F}_{\text{3-triple}}$ receives $\{f_v^U(x), g_v^U(y)\}_{v \in [3], U \in \{A,B,C\}}$ from the adversary, computes a degree-$(1,1)$ three-party triple $([\![A]\!]_1, [\![B]\!]_1, [\![C]\!]_1)$ such that $U(x,v) = f_v^U(x), U(v,y) = g_v^U(y)$ for $U \in \{A,B,C\}, v \in [3]$, and distributes it to three parties.

[a] We assume, by default, that the adversary provides the functionality with shares of corrupted parties that are consistent to degree-$(1,1)$ Shamir sharings of a Beaver triple. In particular, 1) the shares are degree-1 polynomials for Cases 2, 3, and 4; 2) the polynomials agree on all common points held by two corrupted parties for Cases 3 and 4; 3) three secrets satisfy the multiplication relation for Case 4.

Protocol 17: PACKED-TRIPLE(k) in $\mathcal{F}_{\text{3-triple}}$-hybrid model

Let $N = (n')^3$. All parties aim to prepare $\frac{N}{9}$ random packed Beaver triples over bivariate polynomials. For simplicity of notations, let $\mathcal{P}_{\texttt{active}} = \{P_1, \ldots, P_{n'}\}$. All parties agree on a designated party P_{king}.

0: **Preparing random sharings.**

1. All parties invoke $\mathcal{F}_{\text{rand-sharing}}(\Sigma_2, \frac{N-1}{2})$ and $\mathcal{F}_{\text{rand-sharing}}(\Sigma_5, \frac{N-1}{2})$ to prepare $\frac{N-1}{2}$ pairs of random double sharings in the form of $(\llbracket \boldsymbol{R} \rrbracket_{t+k-1}, \llbracket \boldsymbol{R} \rrbracket_{t+2k-2})$, where $\llbracket \boldsymbol{R} \rrbracket_{t+2k-2}$ is computed as $\llbracket \boldsymbol{R} \rrbracket_{t+2k-2} \leftarrow \llbracket \boldsymbol{R} \rrbracket_{t+k-1} + \llbracket \boldsymbol{O} \rrbracket_{t+2k-2}$.

1: **Packed triple distribution.**

1. For $\ell \in [N]$, supposing $\ell = (\ell_1 - 1) \cdot (n')^2 + (\ell_2 - 1) \cdot n' + \ell_3$ for $\ell_1, \ell_2, \ell_3 \in \mathcal{P}_{\texttt{active}}$, three parties $P_{\ell_1}, P_{\ell_2}, P_{\ell_3}$ distribute a degree-$(t+k-1, t+k-1)$ packed Beaver triple to all parties.
   (a) Three parties $P_{\ell_1}, P_{\ell_2}, P_{\ell_3}$ invoke k^2 times of $\mathcal{F}_{\text{3-triple}}$ to obtain k^2 three-party degree-$(1,1)$ Beaver triples $\{\llbracket A_{\ell,ij} \rrbracket_1, \llbracket B_{\ell,ij} \rrbracket_1, \llbracket C_{\ell,ij} \rrbracket_1\}_{i,j \in [k]}$, where each P_{ℓ_v} holds two degree-1 polynomials $U_{\ell,ij}(x, v), U_{\ell,ij}(v, x)$ as its shares of $\llbracket U_{\ell,ij} \rrbracket_1$ for $U \in \{A, B, C\}, i, j \in [k]$.
   (b) For $v \in \{1, 2, 3\}$, P_{ℓ_v} samples three random degree-$(t+k-1, t+k-1)$ packed Shamir sharings $(\llbracket \boldsymbol{A}_\ell(v) \rrbracket_{t+k-1}, \llbracket \boldsymbol{B}_\ell(v) \rrbracket_{t+k-1}, \llbracket \boldsymbol{C}_\ell(v) \rrbracket_{t+k-1})$ such that $\boldsymbol{U}_\ell(v) = (U_{\ell,ij}(v, 0))_{ij} \in \mathbb{F}^{k \times k}$, and distributes them to all parties.
   For $\boldsymbol{U} \in \{\boldsymbol{A}_\ell(v), \boldsymbol{B}_\ell(v), \boldsymbol{C}_\ell(v)\}$, all parties invoke CHECK-ALL$(P_{\ell_v}, \llbracket \boldsymbol{U} \rrbracket_{t+k-1})$ to check whether P_{ℓ_v} distributes a valid degree-$(t+k-1, t+k-1)$ packed Shamir sharing.
   – If the check fails, all parties take the disputed pair E as output.
   (c) For $\boldsymbol{U} \in \{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}\}$, all parties locally compute

$$\llbracket \boldsymbol{U}_\ell \rrbracket_{t+k-1} \leftarrow \lambda_{1,1}(0) \cdot \llbracket \boldsymbol{U}_\ell(1) \rrbracket_{t+k-1} + \lambda_{1,2}(0) \cdot \llbracket \boldsymbol{U}_\ell(2) \rrbracket_{t+k-1},$$

   where $\lambda_{1,i}$ denotes the Lagrange coefficient for degree-1 polynomials at evaluation point $x = i$, i.e., $\lambda_{1,i}(\alpha) = \prod_{j \in \{1,2\}, j \neq i} \frac{(\alpha - j)}{(i - j)}$ for $i \in \{1, 2\}$.

2. **Checking consistency.** For $\ell \in [N]$ and $\boldsymbol{U} \in \{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}\}$, all parties locally compute

$$\begin{aligned} \llbracket \boldsymbol{\theta}_\ell^U \rrbracket_{t+k-1} \leftarrow & \lambda_{1,1}(3) \cdot \llbracket \boldsymbol{U}_\ell(1) \rrbracket_{t+k-1} + \lambda_{1,2}(3) \cdot \llbracket \boldsymbol{U}_\ell(2) \rrbracket_{t+k-1} \\ & - \llbracket \boldsymbol{U}_\ell(3) \rrbracket_{t+k-1}. \end{aligned}$$

   All parties invoke BATCHED-VRFY$(3N)$ with their shares of $\{\llbracket \boldsymbol{\theta}_\ell^U \rrbracket_{t+k-1}\}_{U \in \{A,B,C\}, \ell \in [N]}$ and their degree-$(1,1)$ Shamir shares of $\{\llbracket U_{\ell,ij} \rrbracket_1\}_{U \in \{A,B,C\}, \ell \in [N], i,j \in [k]}$ to check whether each $\llbracket \boldsymbol{\theta}_\ell^U \rrbracket_{t+k-1}$ is a degree-$(t+k-1, t+k-1)$ packed Shamir sharing with secret $\boldsymbol{0} \in \mathbb{F}^{k \times k}$.
   – If the check fails, all parties take the disputed pair E as output.

2: **Packed triple extraction.**

1. All parties set two $k \times k$ matrices of degree-$\frac{N-1}{2}$ polynomials $\boldsymbol{F}, \boldsymbol{G}$ such that

$$\forall\ \ell \in [\frac{N+1}{2}], \llbracket \boldsymbol{F}(\ell) \rrbracket_{t+k-1} = \llbracket \boldsymbol{A}_\ell \rrbracket_{t+k-1}, \llbracket \boldsymbol{G}(\ell) \rrbracket_{t+k-1} = \llbracket \boldsymbol{B}_\ell \rrbracket_{t+k-1}.$$

2. For every $\ell \in \{\frac{N+3}{2}, \ldots, N\}$, all parties invoke MULT with their shares of $\llbracket \boldsymbol{F}(\ell) \rrbracket_{t+k-1}, \llbracket \boldsymbol{G}(\ell) \rrbracket_{t+k-1}$, a packed triple $(\llbracket \boldsymbol{A}_\ell \rrbracket_{t+k-1}, \llbracket \boldsymbol{B}_\ell \rrbracket_{t+k-1}, \llbracket \boldsymbol{C}_\ell \rrbracket_{t+k-1})$, an unused double sharing $(\llbracket \boldsymbol{R} \rrbracket_{t+k-1}, \llbracket \boldsymbol{R} \rrbracket_{t+2k-2})$, to compute their shares of $\llbracket \boldsymbol{F}(\ell) * \boldsymbol{G}(\ell) \rrbracket_{t+k-1}$.
3. **Multiplication verification.** Suppose that in last step, all parties obtain their shares of $\{\llbracket \boldsymbol{V}_\ell \rrbracket_{k-1}, \llbracket \boldsymbol{V}_\ell \rrbracket_{t+k-1}\}_{\ell \in [N-1]}$ and $\{\llbracket \boldsymbol{W}_\ell \rrbracket_{k-1}, \llbracket \boldsymbol{W}_\ell \rrbracket_{t+2k-2}\}_{\ell \in [\frac{N-1}{2}]}$. All parties invoke $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_8, \frac{3(N-1)}{2})$ with these shares to check whether each pair of sharings corresponds to the same secrets.
   – If the check fails, all parties take the disputed pair E as output.

4. All parties set a $k \times k$ matrix of degree-$(N-1)$ polynomials $\boldsymbol{H}$ such that

$$\forall \ell \in [\frac{N+1}{2}], [\![\boldsymbol{H}(\ell)]\!]_{t+k-1} = [\![\boldsymbol{C}_\ell]\!]_{t+k-1},$$

$$\forall \ell \in \{\frac{N+3}{2}, \ldots, N\}, [\![\boldsymbol{H}(\ell)]\!]_{t+k-1} = [\![\boldsymbol{F}(\ell) * \boldsymbol{G}(\ell)]\!]_{t+k-1}.$$

3: **Triple verification.**

1. For $\ell \in [\frac{10.5N}{27}]$, all parties compute their shares of $([\![\boldsymbol{F}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{H}(N+\ell)]\!]_{t+k-1})$ and send them to $P_i := P_{(\ell \mod n')}$, who checks whether the receiving shares form a degree-$(t+k-1, t+k-1)$ packed Beaver triple over bivariate polynomial.
   (a) P_i invokes Check-One$(P_i, [\![\boldsymbol{U}]\!]_{t+k-1})$ for every $\boldsymbol{U} \in \{\boldsymbol{F}(N+\ell), \boldsymbol{G}(N+\ell), \boldsymbol{H}(N+\ell)\}$ to check whether $[\![\boldsymbol{U}]\!]_{t+k-1}$ is a valid degree-$(t+k-1, t+k-1)$ packed Shamir sharing.
      – If the check fails, all parties take the disputed pair E as output.
   (b) P_i reconstructs the secrets $(\boldsymbol{F}(N+\ell), \boldsymbol{G}(N+\ell), \boldsymbol{H}(N+\ell))$ and checks whether $\boldsymbol{F}(N+\ell) * \boldsymbol{G}(N+\ell) = \boldsymbol{H}(N+\ell)$.
      – If the check fails, P_i broadcasts $(\texttt{Incorrect}, \ell, P_i)$.
      – Otherwise, P_i accept the sharings.
2. If every active party accepts, all parties take as output their shares of $\{([\![\boldsymbol{F}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{H}(N+\ell)]\!]_{t+k-1})\}_{\ell \in \{\frac{10.5N}{27}+1, \ldots, \frac{N+1}{2}\}}$.
   – Otherwise, the party broadcasting $\texttt{Incorrect}$ with the smallest identity, say P_i, plays the role of a referee to locate a disputed pair. All parties invoke Fault-Localize-Triple(P_i).

Protocol 18: Batched-Vrfy(N)

All parties aim to check whether their shares of $\{[\![\boldsymbol{\theta}_\ell]\!]_{t+k-1}\}_{\ell \in [N]}$ form N degree-$(t+k-1, t+k-1)$ packed Shamir sharings with secret $\mathbf{0} \in \mathbb{F}^{k \times k}$. In case that faults occur, all parties locate a disputed pair relying on degree-$(1,1)$ Shamir sharings.
All parties agree on an $n' \times (n'-t')$ Vandermonde matrix $\boldsymbol{M}$.
Verification.

1: All parties invoke $\mathcal{F}_{\text{rand-sharing}}(\Sigma_6, N)$ to prepare N random sharings in the form of $[\![\boldsymbol{O}]\!]_{t+k-1}$.
2: For each group of $n'-t'$ sharings $\{[\![\boldsymbol{\theta}_\ell]\!]_{t+k-1}\}_{\ell \in [n'-t']}$ to be checked, all parties pick $n'-t'$ unused random sharings $\{[\![\boldsymbol{O}_\ell]\!]_{t+k-1}\}_{\ell \in [n'-t']}$ and verify the correctness of $\{[\![\boldsymbol{\theta}_\ell]\!]_{t+k-1}\}_{\ell \in [n'-t']}$ as follows.

1. For $\ell \in [n'-t']$, all parties locally compute $[\![\boldsymbol{\epsilon}_\ell]\!]_{t+k-1} \leftarrow [\![\boldsymbol{\theta}_\ell]\!]_{t+k-1} + [\![\boldsymbol{O}_\ell]\!]_{t+k-1}$.
2. All parties locally compute

$$([\![\boldsymbol{\epsilon}^{(1)}]\!]_{t+k-1}, \ldots, [\![\boldsymbol{\epsilon}^{(n')}]\!]_{t+k-1})^\top$$
$$\leftarrow \boldsymbol{M}^\top \cdot ([\![\boldsymbol{\epsilon}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{\epsilon}_{n'-t'}]\!]_{t+k-1})^\top.$$

3. For $s \in \mathcal{P}_{\texttt{active}}$, all parties send their shares of $[\![\boldsymbol{\epsilon}^{(s)}]\!]_{t+k-1}$ to P_s. Then P_i checks the correctness of $[\![\boldsymbol{\epsilon}^{(s)}]\!]_{t+k-1}$ as follows.
   (a) P_s invokes Check-One$(P_s, [\![\boldsymbol{\epsilon}^{(s)}]\!]_{t+k-1})$ with all parties to check whether the received shares form degree-$(t+k-1, t+k-1)$ packed Shamir sharings.
      – If the check fails, all parties take the disputed pair E as output.
   (b) P_s reconstructs the secrets $\boldsymbol{\epsilon}^{(s)}$ and checks whether $\boldsymbol{\epsilon}^{(s)} = \mathbf{0}$.
      – If the check fails, P_s broadcasts $(\texttt{Incorrect}, P_s)$.
      – Otherwise, P_s accepts the sharing.
4. If every active party accepts, all parties accept this group of sharings and continue checking the next group. Otherwise, the party broadcasting $\texttt{Incorrect}$ with the smallest identity, say P_s, plays the role of a referee to locate a disputed pair as follows.

Fault localization.

Locating an incorrect sharing.

4. All parties send their shares of $\{[\![\boldsymbol{\theta}_\ell]\!]_{t+k-1}, [\![\boldsymbol{O}_\ell]\!]_{t+k-1}\}_{\ell\in[n'-t']}$ to P_s.
5. For $\ell \in [n'-t']$, P_s invokes Check-One$(P_s, [\![\boldsymbol{O}_\ell]\!]_{t+k-1})$ to check whether $[\![\boldsymbol{O}_\ell]\!]_{t+k-1}$ is a degree-$(t+k-1, t+k-1)$ packed Shamir sharing.
   - If the check fails, all parties take the disputed pair E as output.
6. P_s checks whether the shares sent by each party $P_{s'}$ are consistent with the shares sent in Step 2.3.
   - If not, P_s broadcasts $(P_{s'}, \texttt{Corrupt})$. All parties take the disputed pair $E \leftarrow \{P_s, P_{s'}\}$ as output.
7. For $\ell \in [n'-t']$, P_s checks the correctness of $[\![\boldsymbol{\theta}_\ell]\!]_{t+k-1}$ as follows.
   (a) P_s invokes Check-One$(P_s, [\![\boldsymbol{\theta}_\ell]\!]_{t+k-1})$ to check whether $[\![\boldsymbol{\theta}_\ell]\!]_{t+k-1}$ is a degree-$(t+k-1, t+k-1)$ packed Shamir sharing.
      - If the check fails, all parties take the disputed pair E as output.
   (b) P_s reconstructs the secret $\boldsymbol{\theta}_\ell$ and checks whether $\boldsymbol{\theta}_\ell = \mathbf{0} \in \mathbb{F}^{k\times k}$.
      - If the check fails, P_s broadcasts $(\ell, i, j, \texttt{Incorrect})$ to indicate $\theta_{\ell,ij} \neq 0$, where $\boldsymbol{\theta}_\ell = (\theta_{\ell,ij})_{ij}$. Then a disputed pair will be identified as follows.

Identifying a disputed pair.

8. Letting $\ell = (\ell_1 - 1)\cdot(n')^2 + (\ell_2 - 1)\cdot n' + \ell_3$, each P_{ℓ_v} broadcasts its shares of $[\![U_{\ell,ij}]\!]_1$, i.e., two degree-1 polynomials $U_{\ell,ij}(x, v), U_{\ell,ij}(v, x)$, and the sharing $[\![\boldsymbol{U}_\ell(v)]\!]_{t+k-1}$. All parties locate a disputed pair as follows.
   (a) Each party $P_{s'}$ checks whether the values broadcast by P_{ℓ_v} are different from values received in Step 1.1.b of Packed-Triple.
      - If the check fails, $P_{s'}$ broadcasts $(\texttt{Inconsistent}, P_{s'}, P_{\ell_v})$. All parties set $E \leftarrow \{P_{s'}, P_{\ell_v}\}$ and take E as output.
   (b) All parties check whether shares of $[\![U_{\ell,ij}]\!]_1$ broadcast by P_{ℓ_v} are consistent with degree-$(1,1)$ Shamir sharing, i.e., $U_{\ell,ij}(x, v), U_{\ell,ij}(v, y)$ are of degree-1 and they agree on the common point.
      - If the check fails, all parties set $E \leftarrow \{P_{\ell_v}\}$, and take E as output.
   (c) All parties check whether the sharing $[\![\boldsymbol{U}_\ell(v)]\!]_{t+k-1}$ broadcast by P_{ℓ_v} is a degree-$(t+k-1, t+k-1)$ packed Shamir sharing and the secret $U_{\ell,ij}(v) = U_{\ell,ij}(v, 0)$.
      - If the check fails, all parties set $E \leftarrow \{P_{\ell_v}\}$, and take E as output.
   (d) All parties check whether two parties $P_{\ell_v}, P_{\ell_{v'}}$ broadcast the same values for common points $U_{\ell,ij}(v, v'), U_{\ell,ij}(v', v)$ which are jointly held by them.
      - If the check fails, all parties set $E \subset \{P_{\ell_v}, P_{\ell_{v'}}\}$, and take E as output.

   If none of the cases above happens, all parties set $E \leftarrow \{P_s\}$ and take E as output.

Protocol 19: Mult

All parties hold their shares of $[\![\boldsymbol{X}]\!]_{t+k-1}$ and $[\![\boldsymbol{Y}]\!]_{t+k-1}$, and aim to compute their shares of $[\![\boldsymbol{X} * \boldsymbol{Y}]\!]_{t+k-1}$ by consuming

- a degree-$(t+k-1.t+k-1)$ packed Beaver triple $([\![\boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{B}]\!]_{t+k-1}, [\![\boldsymbol{C}]\!]_{t+k-1})$ and
- a pair of random packed Shamir sharings $([\![\boldsymbol{R}]\!]_{t+k-1}, [\![\boldsymbol{R}]\!]_{t+2k-2})$.[a]

1. All parties compute $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1} = [\![\boldsymbol{X}]\!]_{t+k-1} + [\![\boldsymbol{A}]\!]_{t+k-1}$ and $[\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{t+k-1} = [\![\boldsymbol{Y}]\!]_{t+k-1} + [\![\boldsymbol{B}]\!]_{t+k-1}$, and send their shares to P_{king}.
2. For $\boldsymbol{U} \in \{\boldsymbol{X}+\boldsymbol{A}, \boldsymbol{Y}+\boldsymbol{B}\}$, P_{king} and all parties invoke Check-One$(P_{\text{king}}, [\![\boldsymbol{U}]\!]_{t+k-1})$ to check whether all parties send correct shares to P_{king}.
   - If the check fails, all parties take the disputed pair E as output.

3. P_{king} reconstructs $\boldsymbol{X}+\boldsymbol{A}$ and $\boldsymbol{Y}+\boldsymbol{B}$, and distributes $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{k-1}, [\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{k-1}$ to all parties.
4. For $\boldsymbol{U} \in \{\boldsymbol{X}+\boldsymbol{A}, \boldsymbol{Y}+\boldsymbol{B}\}$, all parties invoke Check-All$(P_{\text{king}}, [\![\boldsymbol{U}]\!]_{k-1})$ to check whether P_{king} distributes consistent degree-$(k-1,k-1)$ packed Shamir sharings.
5. All parties locally compute

$$[\![\boldsymbol{Z}]\!]_{t+2k-2} \leftarrow [\![\boldsymbol{X}+\boldsymbol{A}]\!]_{k-1} \cdot [\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{k-1} - [\![\boldsymbol{X}+\boldsymbol{A}]\!]_{k-1} \cdot [\![\boldsymbol{B}]\!]_{t+k-1} - [\![\boldsymbol{A}]\!]_{t+k-1} \cdot [\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{k-1} + [\![\boldsymbol{C}]\!]_{t+k-1}.$$

   All parties then locally compute $[\![\boldsymbol{Z}+\boldsymbol{R}]\!]_{t+2k-2} \leftarrow [\![\boldsymbol{Z}]\!]_{t+2k-2} + [\![\boldsymbol{R}]\!]_{t+2k-2}$, and send their shares to P_{king}.
6. P_{king} and all parties invoke Check-One$(P_{\text{king}}, [\![\boldsymbol{Z}+\boldsymbol{R}]\!]_{t+2k-2})$ to check whether all parties send correct shares to P_{king}.
   - If the check fails, all parties take the disputed pair E as output.
7. P_{king} reconstructs $\boldsymbol{Z}+\boldsymbol{R}$ and distributes $[\![\boldsymbol{Z}+\boldsymbol{R}]\!]_{k-1}$ to all parties.
8. All parties invoke Check-All$(P_{\text{king}}, [\![\boldsymbol{Z}+\boldsymbol{R}]\!]_{k-1})$ to check whether P_{king} distributes consistent degree-$(k-1,k-1)$ packed Shamir sharings.
9. All parties locally compute $[\![\boldsymbol{Z}]\!]_{t+k-1} \leftarrow [\![\boldsymbol{Z}+\boldsymbol{R}]\!]_{k-1} - [\![\boldsymbol{R}]\!]_{t+k-1}$.
10. All parties additionally take as output their shares of $([\![\boldsymbol{X}+\boldsymbol{A}]\!]_{k-1}, [\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1})$, $([\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{k-1}, [\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{t+k-1})$, and $([\![\boldsymbol{Z}+\boldsymbol{R}]\!]_{k-1}, [\![\boldsymbol{Z}+\boldsymbol{R}]\!]_{t+2k-2})$, which will be batched and verified to check whether P_{king} distributes degree-$(k-1,k-1)$ packed Shamir sharings with correct secrets.

[a] Different from the situation in MPC in the preprocessing model, as all parties can verify whether P_{king} distributes packed Shamir sharings of degree-$(k-1,k-1)$ and doesn't mount a double dipping attack, they don't need to add degree-$(n'-1,n'-1)$ packed Shamir sharings with secret $\mathbf{0}$ to protect privacy of shares of honest parties.

Protocol 20: Fault-Localize-Triple(P_i)

Checking P_i.

1. P_i broadcasts the inconsistent packed triple it detected $([\![\boldsymbol{F}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{H}(N+\ell)]\!]_{t+k-1})$. All parties check P_i as follows.
   (a) Each party $P_{i'}$ checks whether P_i broadcasts the same shares as it sent to P_i.
      - If the check fails, $P_{i'}$ broadcasts $(\texttt{Incorrect}, P_{i'})$. All parties set $E \leftarrow \{P_i, P_{i'}\}$ and take E as output.
   (b) All parties check each of the three sharings broadcast by P_i is a degree-$(t+k-1,t+k-1)$ packed Shamir sharing.
   (c) All parties reconstruct the secrets $\boldsymbol{F}(N+\ell), \boldsymbol{G}(N+\ell), \boldsymbol{H}(N+\ell)$ and check whether they violate the multiplication relation, i.e., $\boldsymbol{F}(N+\ell) * \boldsymbol{G}(N+\ell) \neq \boldsymbol{H}(N+\ell)$.
      - If one of two checks above fails, all parties set $E \leftarrow \{P_i\}$, and take E as output.

Locating an incorrect packed triple.

2. All parties send to P_i their shares of $\{[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}, [\![\boldsymbol{C}_\ell]\!]_{t+k-1}\}_{\ell\in[N]}$.
3. For $\ell \in [N]$, P_i invokes Check-One$(P_i, [\![\boldsymbol{U}]\!]_{t+k-1})$ for every $\boldsymbol{U} \in \{\boldsymbol{A}_\ell, \boldsymbol{B}_\ell, \boldsymbol{C}_\ell\}$ to check whether the received shares form a degree-$(t+k-1,t+k-1)$ packed Shamir sharing.
   - If the check fails, all parties take the disputed pair E as output.
4. For $\ell \in [N]$, P_i reconstructs the secrets $(\boldsymbol{A}_\ell, \boldsymbol{B}_\ell, \boldsymbol{C}_\ell)$ and checks whether $\boldsymbol{A}_\ell * \boldsymbol{B}_\ell = \boldsymbol{C}_\ell$.
   - P_i broadcasts $(\ell, \texttt{Incorrect})$ and the received shares $([\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}, [\![\boldsymbol{C}_\ell]\!]_{t+k-1})$ to indicate the ℓ-th packed triple does not satisfy the multiplication relation.

Identifying a disputed pair.

5. All parties locate a disputed pair as follows.

(a) Each party $P_{i'}$ checks whether P_i broadcasts the same shares as it received from the dealer in Step 1.1.b of Packed-Triple.
  - If the check fails, $P_{i'}$ broadcasts $(\texttt{Incorrect}, P_{i'})$. All parties set $E \leftarrow \{P_i, P_{i'}\}$ and take E as output.

(b) All parties check whether each of the three sharings broadcast by P_i is a degree-$(t+k-1, t+k-1)$ packed Shamir sharing.
  - If the check fails, all parties set $E \leftarrow \{P_i\}$, and output E.

(c) All parties reconstructs the secrets $\boldsymbol{A}_\ell, \boldsymbol{B}_\ell, \boldsymbol{C}_\ell$.
  - If $\boldsymbol{A}_\ell * \boldsymbol{B}_\ell \neq \boldsymbol{C}_\ell$, letting $\{\ell_1, \ell_2, \ell_3\}$ be the identities of three parties distributing the ℓ-th packed triple, all parties set $E \leftarrow \{P_{\ell_1}, P_{\ell_2}\}$ and take E as output. In this case, $\{\ell_1, \ell_2, \ell_3\}$ contains at least two corrupted parties.
  - If $\boldsymbol{A}_\ell * \boldsymbol{B}_\ell = \boldsymbol{C}_\ell$, all parties set $E \leftarrow \{P_i\}$, and output E.

Lemma 6. *Let $t = (n-1)/3$ and $k \leq (t+2)/2$. Packed-Triple(k) computes $\mathcal{F}_{\text{packed-triple}}(\frac{N}{9}, k)$ with perfect security in the $\mathcal{F}_{\text{3-triple}}$, $\mathcal{F}_{\text{rand-sharing}}$, $\mathcal{F}_{\text{batched-vrfy}}$-hybrid model against a fully malicious adversary who controls at most t corrupted parties.*

Proof (Proof of Lemma 6). We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties. Let $\mathcal{C}_{\texttt{active}}$ denote the set of active corrupted parties and $t' = |\mathcal{C}_{\texttt{active}}|$. Let $\mathcal{H}_{\texttt{active}}$ denote the set of active honest parties. The core of the simulation is to delay the sampling of packed triples distributed by committees containing at least two honest parties to argue that the extracted $N/9$ packed triples are unknown to the adversary.

The simulator $\mathcal{S}$ internally invokes the adversary with randomness sampled by itself and works as follows.

1. In Step 0, $\mathcal{S}$ emulates $\mathcal{F}_{\text{rand-sharing}}(\Sigma_2, \frac{N-1}{2})$ and $\mathcal{F}_{\text{rand-sharing}}(\Sigma_5, \frac{N-1}{2})$, receives corrupted parties' shares of $[\![\boldsymbol{R}]\!]_{t+k-1}$ and $[\![\boldsymbol{O}]\!]_{t+2k-2}$ from the adversary, and sends them back to the adversary on behalf of $\mathcal{F}_{\text{rand-sharing}}$.

   Simulation of packed triple distribution.
2. In Step 1.1.a, for each $\ell \in [N]$ such that $\{\ell_1, \ell_2, \ell_3\} \cap \mathcal{C} \neq \emptyset$, for $i, j \in [k]$, $\mathcal{S}$ emulates $\mathcal{F}_{\text{3-triple}}$, receives corrupted parties' shares from the adversary, records them, and sends them back to the adversary on behalf of $\mathcal{F}_{\text{3-triple}}$. According to $|\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\}|$, there are three cases.
   - If $|\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\}| = 3$, $\mathcal{S}$ learns three values $U_{\ell,ij}(1,0), U_{\ell,ij}(2,0), U_{\ell,ij}(3,0)$.
   - If $\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\} = \{\ell_u, \ell_v\}$, supposing $\{\ell_w\} = \{\ell_1, \ell_2, \ell_3\} \backslash \{\ell_u, \ell_v\}$, $\mathcal{S}$ learns two values $U_{\ell,ij}(u,0), U_{\ell,ij}(v,0)$ and then computes the degree-1 polynomial $U_{\ell,ij}(x,0)$, and therefore $U_{\ell,ij}(w,0)$.
   - If $\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\} = \{\ell_u\}$, $\mathcal{S}$ learns $U_{\ell,ij}(u,0)$.

   This implies that when $|\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\}| \leq 1$, the Beaver triple is unknown to the adversary. When $|\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\}| \geq 2$, the whole sharings of the Beaver triple is known to the adversary.
3. In Step 1.1.b,
   - when a corrupted P_{ℓ_v} distributes three degree-$(t+k-1, t+k-1)$ packed Shamir sharings $([\![\boldsymbol{A}_\ell(v)]\!]_{t+k-1}$, $[\![\boldsymbol{B}_\ell(v)]\!]_{t+k-1}$, $[\![\boldsymbol{C}_\ell(v)]\!]_{t+k-1})$, $\mathcal{S}$ receives shares of honest parties from the adversary, and honestly simulates honest parties in Check-All$(P_{\ell_v}, [\![\boldsymbol{U}]\!]_{t+k-1})$ for $\boldsymbol{U} \in \{\boldsymbol{A}_\ell(v), \boldsymbol{B}_\ell(v), \boldsymbol{C}_\ell(v)\}$;
     - If a disputed pair E is output, $\mathcal{S}$ sends $\texttt{reject}$ together with E to $\mathcal{F}_{\text{packed-triple}}$.
     - If the check passes, $\mathcal{S}$ computes the whole sharing $(\widetilde{[\![\boldsymbol{A}_\ell(v)]\!]_{t+k-1}}, \widetilde{[\![\boldsymbol{B}_\ell(v)]\!]_{t+k-1}}, \widetilde{[\![\boldsymbol{C}_\ell(v)]\!]_{t+k-1}})$ according to shares of honest parties, and learns the secrets $\widetilde{\boldsymbol{A}_\ell(v)}, \widetilde{\boldsymbol{B}_\ell(v)}, \widetilde{\boldsymbol{C}_\ell(v)}$, where we write $\widetilde{\boldsymbol{U}}$ to distinguish it from the value $\boldsymbol{U}$ computed when emulating $\mathcal{F}_{\text{3-triple}}$.
   - when an honest P_{ℓ_v} distributes three degree-$(t+k-1, t+k-1)$ packed Shamir sharings $([\![\boldsymbol{A}_\ell(v)]\!]_{t+K-1}$, $[\![\boldsymbol{B}_\ell(v)]\!]_{t+k-1}, [\![\boldsymbol{C}_\ell(v)]\!]_{t+k-1})$, $\mathcal{S}$ samples random degree-$(t+k-1)$ polynomials $\{f_j^U(x), g_j^U(y)\}_{j \in \mathcal{C}_{\texttt{active}}}$ such that $f_j^U(\alpha_i) = g_i^U(\alpha_j)$ for $i, j \in \mathcal{C}_{\texttt{active}}$, and sends them to the adversary on behalf of honest P_{ℓ_v}. When simulating Check-All$(P_{\ell_v}, [\![\boldsymbol{U}]\!]_{t+k-1})$ for every $\boldsymbol{U} \in \{\boldsymbol{A}_\ell(v), \boldsymbol{B}_\ell(v), \boldsymbol{C}_\ell(v)\}$, for each active honest party P_i, $\mathcal{S}$ computes the values of P_i from sampled shares of corrupted parties, i.e., $f_i^U(\alpha_j) = g_j^U(\alpha_i), g_i^U(\alpha_j) = f_j^U(\alpha_i)$ for $j \in \mathcal{C}_{\texttt{active}}$. Then $\mathcal{S}$ simulates behaviors of honest parties using the values computed above in Check-All$(P_{\ell_v}, [\![\boldsymbol{U}]\!]_{t+k-1})$.

- If a disputed pair E is output, $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.

4. In Step 1.1.c, for each $\ell \in [N]$, for $\boldsymbol{U} \in \{\boldsymbol{A}_\ell, \boldsymbol{B}_\ell, \boldsymbol{C}_\ell\}$, $\mathcal{S}$ computes the additive error $\Delta\boldsymbol{U}(v) \in \mathbb{F}^{k\times k}$ on $[\![\boldsymbol{U}(v)]\!]_{t+k-1}$ for each $v \in [3]$ as follows.
   - For $\ell_v \in \mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\}$, recall $\mathcal{S}$ learns $\widetilde{\boldsymbol{U}(v)}$ when simulating Step 1.1.b and learns $\boldsymbol{U}(v)$ when emulating $\mathcal{F}_{\text{3-triple}}$. $\mathcal{S}$ computes $\Delta\boldsymbol{U}(v) \leftarrow \widetilde{\boldsymbol{U}(v)} - \boldsymbol{U}(v)$, where $\boldsymbol{U}(v) = (U_{\ell,ij}(v,0))_{i,j\in[k]}$.
   - For $\ell_v \in \mathcal{H} \cap \{\ell_1, \ell_2, \ell_3\}$, $\mathcal{S}$ sets $\Delta\boldsymbol{U}(v) = \boldsymbol{0}$.
5. In Step 1.2, $\mathcal{S}$ simulates Batched-Vrfy.
   (a) In Step 1, $\mathcal{S}$ emulates $\mathcal{F}_{\text{rand-sharing}}(\Sigma_6, N)$, receives shares of corrupted parties from the adversary, records them, and sends them back to the adversary on behalf of $\mathcal{F}_{\text{rand-sharing}}$.
   (b) In Step 2.1, for $\ell \in [n'-t']$, $\mathcal{S}$ computes corrupted parties' shares of $[\![\boldsymbol{\epsilon}_\ell]\!]_{t+k-1}$ as $[\![\boldsymbol{\epsilon}_\ell]\!]_{t+k-1} \leftarrow [\![\boldsymbol{\theta}_\ell]\!]_{t+k-1} + [\![\boldsymbol{O}_\ell]\!]_{t+k-1}$. $\mathcal{S}$ randomly samples a degree-$(t+k-1, t+k-1)$ packed Shamir sharing as $[\![\boldsymbol{\epsilon}_\ell]\!]_{t+k-1}$ based on shares of corrupted parties and the secret $\boldsymbol{\epsilon}_\ell = \lambda_{1,1}(3)\cdot\Delta\boldsymbol{U}_\ell(1) + \lambda_{1,2}(3)\cdot\Delta\boldsymbol{U}_\ell(2) - \Delta\boldsymbol{U}_\ell(3) \in \mathbb{F}^{k\times k}$.
   (c) In Step 2.2, $\mathcal{S}$ computes the whole sharings of $\{[\![\boldsymbol{\epsilon}^{(\ell)}]\!]_{t+k-1}\}_{\ell\in[n'-t']}$ by computing
   $$([\![\boldsymbol{\epsilon}^{(1)}]\!]_{t+k-1}, \ldots, [\![\boldsymbol{\epsilon}^{(n')}]\!]_{t+k-1})^\top \leftarrow \boldsymbol{M}^\top \cdot ([\![\boldsymbol{\epsilon}_1]\!]_{t+k-1}, \ldots, [\![\boldsymbol{\epsilon}_{n'-t'}]\!]_{t+k-1})^\top.$$
   (d) In Step 2.3, since $\mathcal{S}$ has honest parties' shares of $\{[\![\boldsymbol{\epsilon}^{(\ell)}]\!]_{t+k-1}\}_{\ell\in[N]}$, $\mathcal{S}$ faithfully follows the protocol.
   - In Step 2.3.a, if a disputed pair E is output, $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.

   (e) In Step 2.4, if some party P_s broadcasts $(\texttt{Incorrect}, P_s)$, $\mathcal{S}$ samples the honest parties' shares of $([\![\boldsymbol{\theta}_\ell]\!]_{t+k-1}, [\![\boldsymbol{O}_\ell]\!]_{t+k-1})$ based on shares of corrupted parties and the secrets $\boldsymbol{\theta}_\ell^U = \lambda_{1,1}(3)\cdot\Delta\boldsymbol{U}_\ell(1) + \lambda_{1,2}(3)\cdot\Delta\boldsymbol{U}_\ell(2) + \Delta\boldsymbol{U}_\ell(3)$ and $\boldsymbol{O}_\ell = \boldsymbol{0}$. Then $\mathcal{S}$ simulates the protocol honestly to identify a disputed pair E. Then $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.
   If all parties accept, $\mathcal{S}$ further computes as follows.
   - If $|\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\}| \geq 2$, $\mathcal{S}$ computes the whole sharings of the packed triple as $[\![\boldsymbol{A}_\ell]\!]_{t+k-1} \leftarrow \lambda_{1,1}(0)\cdot[\![\boldsymbol{A}_\ell(1)]\!] + \lambda_{1,2}(0)\cdot[\![\boldsymbol{A}_\ell(2)]\!]_{t+k-1}$, $[\![\boldsymbol{B}_\ell]\!]_{t+k-1} \leftarrow \lambda_{1,1}(0)\cdot[\![\boldsymbol{B}_\ell(1)]\!] + \lambda_{1,2}(0)\cdot[\![\boldsymbol{B}_\ell(2)]\!]_{t+k-1}$, $[\![\boldsymbol{C}_\ell]\!]_{t+k-1} \leftarrow \lambda_{1,1}(0)\cdot[\![\boldsymbol{C}_\ell(1)]\!] + \lambda_{1,2}(0)\cdot[\![\boldsymbol{C}_\ell(2)]\!]_{t+k-1}$, which implies the whole packed triple is known to the adversary.
     Define the set $\mathcal{E} \subset [N]$ to contain all such ℓs. Then $|\mathcal{E}| = \frac{7N}{27}$. Let $\mathcal{E}_1 = \mathcal{E} \cap [\frac{N+1}{2}]$ and $\mathcal{E}_2 = \mathcal{E} \setminus \mathcal{E}_1$, i.e., $\mathcal{E}_2 = \mathcal{E} \cap \{\frac{N+3}{2}, \ldots, N\}$.
   - If $|\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\}| \leq 1$, $\mathcal{S}$ computes corrupted parties' shares as $[\![\boldsymbol{A}_\ell]\!]_{t+k-1} \leftarrow \lambda_{1,1}(0)\cdot[\![\boldsymbol{A}_\ell(1)]\!] + \lambda_{1,2}(0)\cdot[\![\boldsymbol{A}_\ell(2)]\!]_{t+k-1}$, $[\![\boldsymbol{B}_\ell]\!]_{t+k-1} \leftarrow \lambda_{1,1}(0)\cdot[\![\boldsymbol{B}_\ell(1)]\!] + \lambda_{1,2}(0)\cdot[\![\boldsymbol{B}_\ell(2)]\!]_{t+k-1}$, $[\![\boldsymbol{C}_\ell]\!]_{t+k-1} \leftarrow \lambda_{1,1}(0)\cdot[\![\boldsymbol{C}_\ell(1)]\!] + \lambda_{1,2}(0)\cdot[\![\boldsymbol{C}_\ell(2)]\!]_{t+k-1}$, which implies the secrets of the packed triple is unknown to the adversary.
     Define the set $\mathcal{D} \subset [N]$ to contain all such ℓs. Then $|\mathcal{D}| = \frac{20N}{27}$. Let $\mathcal{D}_1 = \mathcal{D} \cap [\frac{N+1}{2}]$ and $\mathcal{D}_2 = \mathcal{D} \setminus \mathcal{D}_1$, i.e., $\mathcal{D}_2 = \mathcal{D} \cap \{\frac{N+3}{2}, \ldots, N\}$. Let $\mathcal{D}_{1,1} \subset \mathcal{D}_1$ be the subset in $\mathcal{D}_1$ with $|\mathcal{E}_2|$ smallest identities. Let $\mathcal{D}_{1,2} \subset \mathcal{D} \setminus \mathcal{D}_{1,1}$ be the subset of $\mathcal{D}_1 \setminus \mathcal{D}_{1,1}$ with $|\frac{3.5N}{27}|$ smallest identities. Finally, let $\mathcal{D}_{1,0} = \mathcal{D}_1 \setminus (\mathcal{D}_{1,1} \cup \mathcal{D}_{1,2})$.

Simulation of packed triple extraction.

6. In Step 2.1, for every $\ell \in \{\frac{N+3}{2}, \ldots, N\}$, $\mathcal{S}$ computes corrupted parties' shares of $[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}$ and $[\![\boldsymbol{G}(\ell)]\!]_{t+k-1}$ as linear combinations of their shares of $\{[\![\boldsymbol{F}(\ell')]\!]_{t+k-1}\}_{\ell'\in[\frac{N-1}{2}]}$ and $\{[\![\boldsymbol{G}(\ell')]\!]_{t+k-1}\}_{\ell'\in[\frac{N-1}{2}]}$, respectively.
7. In Step 2.2, for each $\ell \in \{\frac{N+3}{2}, \ldots, N\}$, $\mathcal{S}$ simulates the invocation of Mult according whether the ℓ-th packed triple is known by the adversary. Recall $\mathcal{S}$ learns corrupted parties' shares of $[\![\boldsymbol{X}]\!]_{t+k-1}, [\![\boldsymbol{Y}]\!]_{t+k-1}$ when simulating Step 2.1. Recall $\mathcal{S}$ learns corrupted parties' shares of $([\![\boldsymbol{R}]\!]_{t+k-1}, [\![\boldsymbol{R}]\!]_{t+2k-2})$ when emulating $\mathcal{F}_{\text{rand-sharing}}(\Sigma_2, \frac{N-1}{2})$ and $\mathcal{F}_{\text{rand-sharing}}(\Sigma_5, \frac{N-1}{2})$ in Step 0.
   - If the ℓ-th packed triple $([\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}, [\![\boldsymbol{C}_\ell]\!]_{t+k-1})$ is known to the adversary, i.e., $\ell \in \mathcal{E}_2$, $\mathcal{S}$ simulates as follows. Recall $\mathcal{S}$ learns the whole sharing of the packed triple $([\![\boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{B}]\!]_{t+k-1}, [\![\boldsymbol{C}]\!]_{t+k-1})$ when simulating Step 3 of Batched-Vrfy.
     (a) In Step 1, $\mathcal{S}$ randomly samples two degree-$(t+k-1, t+k-1)$ packed Shamir sharings as $[\![\boldsymbol{X}]\!]_{t+k-1}, [\![\boldsymbol{Y}]\!]_{t+k-1}$ based on shares of corrupted parties. $\mathcal{S}$ computes honest parties' shares of $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1}$ and $[\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{t+k-1}$ as $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1} \leftarrow [\![\boldsymbol{X}]\!]_{t+k-1} + [\![\boldsymbol{A}]\!]_{t+k-1}$, $[\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{t+k-1} \leftarrow [\![\boldsymbol{Y}]\!]_{t+k-1} + [\![\boldsymbol{B}]\!]_{t+k-1}$, and sends them P_{king} on behalf of honest parties.

(b) In Step 2, for $\boldsymbol{U} \in \{\boldsymbol{X}+\boldsymbol{A}, \boldsymbol{Y}+\boldsymbol{B}\}$, since $\mathcal{S}$ learns honest parties' shares of $[\![\boldsymbol{U}]\!]_{t+k-1}$, $\mathcal{S}$ honestly simulates behaviors of honest parties in Check-One$(P_{\text{king}}, [\![\boldsymbol{U}]\!]_{t+k-1})$.
- If a disputed pair E is output, $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.

(c) In Step 3, if P_{king} is honest, $\mathcal{S}$ follows the protocol honestly. $\mathcal{S}$ receives honest parties' shares of $[\![\overline{\boldsymbol{X}+\boldsymbol{A}}]\!]_{k-1}$ and $[\![\overline{\boldsymbol{Y}+\boldsymbol{B}}]\!]_{k-1}$ from P_{king}, where we write $\overline{\boldsymbol{U}}$, which is distributed by possibly corrupt P_{king}, to distinguish it from the value $\boldsymbol{U}$, which is computed by the simulator $\mathcal{S}$.

(d) In Step 4, for $\boldsymbol{U} \in \{\overline{\boldsymbol{X}+\boldsymbol{A}}, \overline{\boldsymbol{Y}+\boldsymbol{B}}\}$, since $\mathcal{S}$ learns honest parties' shares of $[\![\boldsymbol{U}]\!]_{k-1}$, $\mathcal{S}$ faithfully simulates the behaviors of honest parties in Check-All$(P_{\text{king}}, [\![\boldsymbol{U}]\!]_{k-1})$.
- If a disputed pair E is output, $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.

(e) In Step 5, $\mathcal{S}$ computes honest parties' shares of $[\![\overline{\boldsymbol{Z}}]\!]_{t+2k-2}$ as $[\![\overline{\boldsymbol{Z}}]\!]_{t+2k-2} \leftarrow [\![\overline{\boldsymbol{X}+\boldsymbol{A}}]\!]_{k-1} \cdot [\![\overline{\boldsymbol{Y}+\boldsymbol{B}}]\!]_{k-1} - [\![\overline{\boldsymbol{X}+\boldsymbol{A}}]\!]_{k-1} \cdot [\![\boldsymbol{B}]\!]_{t+k-1} - [\![\boldsymbol{A}]\!]_{t+k-1} \cdot [\![\overline{\boldsymbol{Y}+\boldsymbol{B}}]\!]_{k-1} + [\![\boldsymbol{C}]\!]_{t+k-1}$.
Then $\mathcal{S}$ randomly samples a pair of double sharings $([\![\boldsymbol{R}]\!]_{t+k-1}, [\![\boldsymbol{R}]\!]_{t+2k-2})$ based on shares of corrupted parties. $\mathcal{S}$ computes honest parties' shares of $[\![\overline{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{t+2k-2}$ as $[\![\overline{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{t+2k-2} \leftarrow [\![\overline{\boldsymbol{Z}}]\!]_{t+2k-2} + [\![\boldsymbol{R}]\!]_{t+2k-2}$, and sends them to P_{king} on behalf of honest parties.

(f) In Step 6, since $\mathcal{S}$ learns honest parties' shares of $[\![\overline{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{t+2k-2}$, $\mathcal{S}$ faithfully simulates behaviors of honest parties in Check-One$(P_{\text{king}}, [\![\overline{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{t+2k-2})$.
- If a disputed pair E is output, $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.

(g) In Step 7, if P_{king} is honest, $\mathcal{S}$ follows the protocol honestly. $\mathcal{S}$ receives honest parties' shares of $[\![\widetilde{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{k-1}$, where we write $\widetilde{\boldsymbol{U}}$, which is distributed by possibly corrupt P_{king}, to distinguish it from the value $\overline{U}$, which is computed by the simulator $\mathcal{S}$.

(h) In Step 8, since $\mathcal{S}$ learns honest parties' shares of $[\![\widetilde{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{k-1}$, $\mathcal{S}$ honestly simulates the behaviors of honest parties in Check-All$(P_{\text{king}}, [\![\widetilde{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{k-1})$.
- If a disputed pair E is output, $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.

$\mathcal{S}$ computes the whole sharing $[\![\widetilde{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{k-1}$ according to shares of honest parties.

(i) In Step 9, $\mathcal{S}$ computes the whole sharing $[\![\widetilde{\boldsymbol{Z}}]\!]_{t+k-1}$ as $[\![\widetilde{\boldsymbol{Z}}]\!]_{t+k-1} \leftarrow [\![\widetilde{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{k-1} - [\![\boldsymbol{R}]\!]_{t+k-1}$.

– If the ℓ-th packed triple $([\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}, [\![\boldsymbol{C}_\ell]\!]_{t+k-1})$ is unknown by the adversary, i.e., $\ell \in \mathcal{D}_2$, $\mathcal{S}$ simulates as follows. Recall $\mathcal{S}$ learns corrupted parties' shares of the packed triple $([\![\boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{B}]\!]_{t+k-1}$, $[\![\boldsymbol{C}]\!]_{t+k-1})$ when simulating Step 2.4 of Batched-Vrfy.

(a) In Step 1, $\mathcal{S}$ computes corrupted parties' shares of $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{t+k-1}$. $\mathcal{S}$ randomly samples two degree-$(t+k-1, t+k-1)$ packed Shamir sharings as $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{t+k-1}$ based on shares of corrupted parties. $\mathcal{S}$ sends honest parties' shares of $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{t+k-1}$ to P_{king} on behalf of honest parties.

(b) In Step 2, for $\boldsymbol{U} \in \{\boldsymbol{X}+\boldsymbol{A}, \boldsymbol{Y}+\boldsymbol{B}\}$, $\mathcal{S}$ simulates the behaviors of honest parties in Check-One$(P_{\text{king}}, [\![\boldsymbol{U}]\!]_{t+k-1})$ honestly.
- If a disputed pair E is output, $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.

(c) In Step 3, if P_{king} is honest, $\mathcal{S}$ follows the protocol honestly. $\mathcal{S}$ received honest parties' shares of $[\![\overline{\boldsymbol{X}+\boldsymbol{A}}]\!]_{k-1}$ and $[\![\overline{\boldsymbol{Y}+\boldsymbol{B}}]\!]_{k-1}$ from P_{king}.

(d) In Step 4, for $\boldsymbol{U} \in \{\overline{\boldsymbol{X}+\boldsymbol{A}}, \overline{\boldsymbol{Y}+\boldsymbol{B}}\}$, since $\mathcal{S}$ learns honest parties' shares of $[\![\boldsymbol{U}]\!]_{k-1}$, $\mathcal{S}$ faithfully simulates the behaviors of honest parties in Check-All$(P_{\text{king}}, [\![\boldsymbol{U}]\!]_{k-1})$.
- If a disputed pair E is output, $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.

$\mathcal{S}$ computes the whole sharings of $[\![\overline{\boldsymbol{X}+\boldsymbol{A}}]\!]_{k-1}, [\![\overline{\boldsymbol{Y}+\boldsymbol{B}}]\!]_{k-1}$ according to shares of honest parties.

(e) In Step 5, $\mathcal{S}$ computes corrupted parties' shares of $[\![\overline{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{t+2k-2} \leftarrow [\![\overline{\boldsymbol{X}+\boldsymbol{A}}]\!]_{k-1} \cdot [\![\overline{\boldsymbol{Y}+\boldsymbol{B}}]\!]_{k-1} - [\![\overline{\boldsymbol{X}+\boldsymbol{A}}]\!]_{k-1} \cdot [\![\boldsymbol{B}]\!]_{t+k-1} - [\![\boldsymbol{A}]\!]_{t+k-1} \cdot [\![\overline{\boldsymbol{Y}+\boldsymbol{B}}]\!]_{k-1} + [\![\boldsymbol{C}]\!]_{t+k-1} + [\![\boldsymbol{R}]\!]_{t+2k-2}$. $\mathcal{S}$ randomly samples a degree-$(t+2k-2, t+2k-2)$ packed Shamir sharing as $[\![\overline{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{t+2k-2}$ based on shares of corrupted parties, and sends shares of honest parties to P_{king} on behalf of honest parties.

(f) In Step 6, $\mathcal{S}$ faithfully simulates behaviors of honest parties in Check-One$(P_{\text{king}}, [\![\overline{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{t+2k-2})$.
- If a disputed pair E is output, $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.

(g) In Step 7, if P_{king} is honest, $\mathcal{S}$ follows the protocol honestly. $\mathcal{S}$ receives honest parties' shares of $[\![\widetilde{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{k-1}$ from P_{king}.

(h) In Step 8, since $\mathcal{S}$ learns honest parties' shares of $[\![\widetilde{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{k-1}$, $\mathcal{S}$ honestly simulates the behaviors of honest parties in Check-All(P_{king}, $[\![\widetilde{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{k-1}$).
- If a disputed pair E is output, $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.

$\mathcal{S}$ computes the whole sharing $[\![\widetilde{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{k-1}$ according to shares of honest parties.

(i) In Step 9, $\mathcal{S}$ computes corrupted parties' shares of $[\![\widetilde{\boldsymbol{Z}}]\!]_{t+k-1}$ as $[\![\widetilde{\boldsymbol{Z}}]\!]_{t+k-1} \leftarrow [\![\widetilde{\boldsymbol{Z}+\boldsymbol{R}}]\!]_{k-1} - [\![\boldsymbol{R}]\!]_{t+k-1}$.

8. In Step 2.3, recall $\mathcal{S}$ learns honest parties' shares of $\{[\![\overline{\boldsymbol{V}_\ell}]\!]_{k-1}, [\![\boldsymbol{V}_\ell]\!]_{t+k-1}\}_{\ell\in[N]}$ and $\{[\![\widetilde{\boldsymbol{U}_\ell}]\!]_{k-1}, [\![\overline{\boldsymbol{U}_\ell}]\!]_{t+2k-2}\}_{\ell\in[\frac{N-1}{2}]}$ when simulating Mult. $\mathcal{S}$ emulates $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma)$ as follows. $\mathcal{S}$ sends the honest parties' shares of $\{[\![\overline{\boldsymbol{V}_\ell}]\!]_{k-1}, [\![\boldsymbol{V}_\ell]\!]_{t+k-1}\}_{\ell\in[N]}$ and $\{[\![\widetilde{\boldsymbol{U}_\ell}]\!]_{k-1}, [\![\overline{\boldsymbol{U}_\ell}]\!]_{t+2k-2}\}_{\ell\in[\frac{N-1}{2}]}$ to the adversary on behalf of $\mathcal{F}_{\text{batched-vrfy}}$. Then $\mathcal{S}$ checks whether $\overline{\boldsymbol{V}_\ell} = \boldsymbol{V}_\ell$ for every $\ell \in [N-1]$ and $\widetilde{\boldsymbol{U}_\ell} = \overline{\boldsymbol{U}_\ell}$ for every $\ell \in [\frac{N-1}{2}]$.
   - If all checks pass, i.e., $\overline{\boldsymbol{V}_\ell} = \boldsymbol{V}_\ell$ for every $\ell \in [N-1]$ and $\widetilde{\boldsymbol{U}_\ell} = \overline{\boldsymbol{U}_\ell}$ for every $\ell \in [\frac{N-1}{2}]$, $\mathcal{S}$ waits for an instruction from the adversary.
     - If $\mathcal{S}$ receives `accept` from the adversary, $\mathcal{S}$ outputs `accept` to the adversary on behalf of $\mathcal{F}_{\text{batched-vrfy}}$, and continues the simulation.
       * In this case, $\mathcal{S}$ knows for every $\ell \in \{\frac{N+3}{2}, \ldots, N\}$, after invoking Mult, all parties hold their shares of $[\![\boldsymbol{F}(\ell) * \boldsymbol{G}(\ell) - \boldsymbol{A}_\ell * \boldsymbol{B}_\ell + \boldsymbol{C}_\ell]\!]_{t+k-1}$.
     - If $\mathcal{S}$ receives `reject` and a disputed pair $E \subset \mathcal{P}_{\texttt{active}}$ from the adversary, $\mathcal{S}$ outputs `abort` and E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$, and sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.
   - If one of the checks fails, $\mathcal{S}$ receives a disputed pair $E \subset \mathcal{P}_{\texttt{active}}$ from the adversary and outputs E on behalf of $\mathcal{F}_{\text{batched-vrfy}}$, and sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.
9. In Step 2.4, recall for every $\ell \in [\frac{N+1}{2}]$, $\mathcal{S}$ learns corrupted parties' shares of $[\![\boldsymbol{C}_\ell]\!]_{t+k-1}$ when simulating Batched-Vrfy, and for every $\ell \in \{\frac{N+3}{2}, \ldots, N\}$, $\mathcal{S}$ learns corrupted parties' shares of $[\![\boldsymbol{F}(\ell) * \boldsymbol{G}(\ell) - \boldsymbol{A}_\ell * \boldsymbol{B}_\ell + \boldsymbol{C}_\ell]\!]_{t+k-1}$ when simulating Mult. Then $\mathcal{S}$ learns corrupted parties' shares of $[\![\overline{\boldsymbol{H}}(\cdot)]\!]_{t+k-1}$, where $\overline{\boldsymbol{H}}$ is a $k \times k$ matrix of k^2 degree-$(N-1)$ polynomials defined as $\overline{\boldsymbol{H}}(\ell) = \boldsymbol{C}_\ell$ for every $\ell \in [\frac{N+1}{2}]$ and $\overline{\boldsymbol{H}}(\ell) = \boldsymbol{F}(\ell) * \boldsymbol{G}(\ell) - \boldsymbol{A}_\ell * \boldsymbol{B}_\ell + \boldsymbol{C}_\ell$ for every $\ell \in \{\frac{N+3}{2}, \ldots, N\}$.

Simulation of triple verification.

10. In Step 3.1, for $\ell \in [N]$ such that $|\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\}| \geq 2$, i.e., $\ell \in \mathcal{E}$, recall $\mathcal{S}$ learns $(\boldsymbol{A}_\ell, \boldsymbol{B}_\ell, \boldsymbol{C}_\ell)$ when simulating Step 2.4 of Batched-Vrfy. Then $\mathcal{S}$ checks whether $\boldsymbol{A}_\ell * \boldsymbol{B}_\ell = \boldsymbol{C}_\ell$ holds.
Denote the set of packed triples which are checked by corrupted parties by $\mathcal{E}_3$. Then $|\mathcal{E}_3| = \frac{3.5N}{27}$.
   - If one of the checks fails, $\mathcal{S}$ computes honest parties' shares of $[\![\boldsymbol{F}(\cdot)]\!]_{t+k-1}$, $[\![\boldsymbol{G}(\cdot)]\!]_{t+k-1}$, and $[\![\overline{\boldsymbol{H}}(\cdot)]\!]_{t+k-1}$ as follows.
     - For each $\ell \in \mathcal{D}_{1,0} \cup \mathcal{D}_{1,2}$, $\mathcal{S}$ randomly samples honest parties' shares of $[\![\boldsymbol{A}_\ell]\!]_{t+k-1}$ and $[\![\boldsymbol{B}_\ell]\!]_{t+k-1}$ based on shares of corrupted parties.
     - For $\ell \in \mathcal{D}_{1,1}$, $\mathcal{S}$ computes honest parties' shares of $[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}$, $[\![\boldsymbol{G}(\ell)]\!]_{t+k-1}$ as linear combinations of their shares of $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{E}_1\cup\mathcal{D}_{1,0}\cup\mathcal{D}_{1,2}\cup\mathcal{E}_2}$.
     - For each $\ell \in \mathcal{D}_1$, $\mathcal{S}$ randomly samples honest parties' shares of $[\![\boldsymbol{C}_\ell]\!]_{t+k-1}$ based on shares of corrupted parties and the constraint $\boldsymbol{C}_\ell = \boldsymbol{A}_\ell * \boldsymbol{B}_\ell$. Notice that $[\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1} = [\![\boldsymbol{C}_\ell]\!]_{t+k-1}$ for $\ell \in \mathcal{D}_1$.
     - For $\ell \in \mathcal{D}_2$, $\mathcal{S}$ computes honest parties' shares of $[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}$, $[\![\boldsymbol{G}(\ell)]\!]_{t+k-1}$ and their shares of $[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}$. $\mathcal{S}$ samples honest parties' shares of $[\![\boldsymbol{C}_\ell]\!]_{t+k-1}$ based on shares of corrupted parties and $\boldsymbol{C}_\ell = \boldsymbol{A}_\ell * \boldsymbol{B}_\ell$, and samples a pair of double sharings in the form of $([\![\boldsymbol{R}]\!]_{t+k-1}, [\![\boldsymbol{R}]\!]_{t+2k-2})$ based on shares of corrupted parties. In doing this, $\mathcal{S}$ can faithfully compute honest parties' shares of $[\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1}$ for $\ell \in \mathcal{D}_2$.
     - For each $\ell \in \mathcal{E}$, recall $\mathcal{S}$ learns honest parties' shares of $[\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1}$ when simulating Step 2.4 of Batched-Vrfy and Step 9 of Mult.

   $\mathcal{S}$ then faithfully follows the protocol to identify a disputed pair $E \subset \mathcal{P}_{\texttt{active}}$. Then $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.

– If all checks pass, $\mathcal{S}$ learns $\overline{\boldsymbol{H}} = \boldsymbol{F} * \boldsymbol{G}$. For $\ell \in [\frac{10.5N}{27}]$, $\mathcal{S}$ computes corrupted parties' shares of $([\![\boldsymbol{F}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{H}(N+\ell)]\!]_{t+k-1})$ as linear combinations of their shares of $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{H}(\ell)]\!]_{t+k-1}\}_{\ell\in[N]}$. Then $\mathcal{S}$ simulates according to whether $P_i := P_{(\ell \mod n')}$ is corrupted.
  - If P_i is corrupted, $\mathcal{S}$ samples honest parties' shares of $([\![\boldsymbol{F}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{H}(N+\ell)]\!]_{t+k-1})$ based on shares of corrupted parties and the constraint $\boldsymbol{H}(N+\ell) = \boldsymbol{F}(N+\ell) * \boldsymbol{G}(N+\ell)$. Then $\mathcal{S}$ sends shares of honest parties to the adversary on behalf of honest parties. Then in Step 3.1.a, $\mathcal{S}$ faithfully follows the protocol.
    * If a disputed pair E is output, $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.
  - If P_i is honest, $\mathcal{S}$ receives corrupted parties' shares of $([\![\boldsymbol{F}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{H}(N+\ell)]\!]_{t+k-1})$ from the adversary.
    In Step 3.1.a, for $\boldsymbol{U} \in \{\boldsymbol{F}(N+\ell), \boldsymbol{G}(N+\ell), \boldsymbol{H}(N+\ell)\}$, $\mathcal{S}$ computes the evaluation points joint held by an honest party and a corrupted party from corrupted parties' shares of $[\![\boldsymbol{U}]\!]_{t+k-1}$ computed previously. In particular, for every $j_1 \in \mathcal{C}_{\texttt{active}}$ and $j_2 \in \mathcal{H}_{\texttt{active}}$, $\mathcal{S}$ computes $f^U_{j_2}(\alpha_{j_1}) := g^U_{j_1}(\alpha_{j_2})$ and $g^U_{j_2}(\alpha_{j_1}) := f^U_{j_1}(\alpha_{j_2})$. $\mathcal{S}$ uses these values to simulate the behaviors of honest parties.
    * If a disputed pair E is output, $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.

    In Step 3.1.b, $\mathcal{S}$ broadcasts `accept` on behalf of honest P_i.

  In Step 3.2,
  - If all active parties broadcast `accept`, for every $\ell \in \{\frac{10.5N}{27}+1, \ldots, \frac{N+1}{2}\}$, $\mathcal{S}$ computes shares of corrupted parties of $([\![\boldsymbol{F}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{H}(N+\ell)]\!]_{t+k-1})$ as linear combinations of their shares of $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{H}(\ell)]\!]_{t+k-1}\}_{\ell\in[N]}$, sends them and `accept` to $\mathcal{F}_{\text{packed-triple}}$.
  - If party P_i broadcasts $(\texttt{Incorrect}, \ell, P_i)$, then $\mathcal{S}$ simulates Fault-Localize-Triple(P_i) as follows.
    * In Step 1, $\mathcal{S}$ simulates the behaviors of honest parties faithfully. Since the triple checked by P_i is correct, either P_i is identified as a corrupted party, or a disputed pair $\{P_i, P_{i'}\}$ is identified.

  $\mathcal{S}$ sends `reject` and the disputed pair E to $\mathcal{F}_{\text{packed-triple}}$.

11. $\mathcal{S}$ outputs what the adversary outputs and honest parties output their outputs received from $\mathcal{F}_{\text{packed-triple}}$.

Before showing the distribution of the output in the ideal world is identical to that in the real world, we first prove two claims regarding the effectiveness of **Consistency check** and **Triple verification** as follows.

Claim. If all parties accept in **Consistency check** when invoking Batched-Vrfy, then for each $\ell \in [N]$, the secrets of sharings $[\![\widetilde{\boldsymbol{U}_\ell(1)}]\!]_{t+k-1}$, $[\![\widetilde{\boldsymbol{U}_\ell(2)}]\!]_{t+k-1}$, $[\![\widetilde{\boldsymbol{U}_\ell(3)}]\!]_{t+k-1}$ distributed by P_{ℓ_1}, P_{ℓ_2}, P_{ℓ_3} (cf. Step 3 of the simulation), respectively, constitute a $k \times k$ matrix of degree-1 polynomials.

Proof. If all parties accept their sharings, then $n'-t'$ tuples of sharings $\{[\![\boldsymbol{\epsilon}^{(\ell)}]\!]_{t+k-1}\}_{\ell\in\mathcal{P}_{\texttt{active}}}$ checked by $n'-t'$ active honest parties are guaranteed to be correct, i.e., their secrets are $\boldsymbol{0} \in \mathbb{F}^{k\times k}$. By property of Vandermonde matrix, there is a one-to-one correspondence between $n'-t'$ sharings $\{[\![\boldsymbol{\epsilon}^{(\ell)}]\!]_{t+k-1}\}_{\ell\in\mathcal{P}_{\texttt{active}}}$ checked by active honest parties and original $n'-t'$ sharings $\{[\![\boldsymbol{\epsilon}_\ell]\!]_{t+k-1}\}_{\ell\in[n'-t']}$.

Furthermore, by functionality of $\mathcal{F}_{\text{rand-sharing}}$, all honest parties' shares of $\{[\![\boldsymbol{O}_\ell]\!]_{t+k-1}\}$ form degree-$(t+k-1, t+k-1)$ packed Shamir sharings with secret being $\boldsymbol{0} \in \mathbb{F}^{k\times k}$. Therefore, all honest parties' shares of $\{[\![\boldsymbol{\theta}_\ell]\!]_{t+k-1}\}_N$ form degree-$(t+k-1, t+k-1)$ packed Shamir sharings with secret being $\boldsymbol{0} \in \mathbb{F}^{k\times k}$. As $\boldsymbol{0} = \boldsymbol{\theta}_\ell = \lambda_{1,1}(3)\cdot\widetilde{\Delta\boldsymbol{U}_\ell(1)} + \lambda_{1,2}(3)\cdot\Delta\boldsymbol{U}_\ell(2) + \Delta\boldsymbol{U}_\ell(3)$, this in turn implies the secrets of sharings $[\![\widetilde{\boldsymbol{U}_\ell(1)}]\!]_{t+k-1}$, $[\![\widetilde{\boldsymbol{U}_\ell(2)}]\!]_{t+k-1}$, $[\![\widetilde{\boldsymbol{U}_\ell(3)}]\!]_{t+k-1}$ constitute a $k \times k$ matrix of degree-1 polynomials. □

Claim. If all parties accept in **Triple verification**, supposing they hold shares of $[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}$, $[\![\boldsymbol{G}(\ell)]\!]_{t+k-1}$, $[\![\boldsymbol{H}(\ell)]\!]_{t+k-1}$, then it holds that $\overline{\boldsymbol{H}}(\ell) = \boldsymbol{F}(\ell) * \boldsymbol{G}(\ell)$ for every $\ell \in [N]$, i.e., $\overline{\boldsymbol{H}} = \boldsymbol{F} * \boldsymbol{G}$.

Proof. By the claim above, if all parties accept the sharings in **Consistency check**, then for each $\boldsymbol{U} \in \{\boldsymbol{A}_\ell, \boldsymbol{B}_\ell, \boldsymbol{C}_\ell\}$, three sharings $[\![\widetilde{\boldsymbol{U}_\ell(1)}]\!]_{t+k-1}$, $[\![\widetilde{\boldsymbol{U}_\ell(2)}]\!]_{t+k-1}$, $[\![\widetilde{\boldsymbol{U}_\ell(3)}]\!]_{t+k-1}$ distributed by the ℓ-th committee constitute a $k \times k$ matrix of degree-1 polynomials. Hence, if they are distributed by a committee containing at least two honest parties, the secrets are fully determined by sharings distributed by honest parties, and therefore guaranteed to satisfy the multiplication relations. This implies there are a number of $(1 - (\frac{1}{3})^3 - 3 \cdot (\frac{1}{3})^2 \cdot \frac{2}{3}) \cdot N = \frac{20N}{27}$ packed triples are correct.

In **Triple verification**, there are a number of $\frac{10.5N}{27} \cdot \frac{2}{3} = \frac{7N}{27}$ triples checked by honest parties. Therefore, there are $\frac{20N}{27} + \frac{7N}{27} = N$ packed triples guaranteed to be correct, which means the $k \times k$ matrix of degree-$(N-1)$ polynomials $\boldsymbol{F} * \boldsymbol{G} - \boldsymbol{H}$ has at least N roots. Hence, $\boldsymbol{F} * \boldsymbol{G} - \boldsymbol{H} \equiv \boldsymbol{0}$. □

We now show that the distribution of the output in the ideal world is identical to that in the reak world using the following hybrid arguments.

$\mathbf{Hyb}_0$: In this hybrid, we consider the execution in the real word.

$\mathbf{Hyb}_1$: In this hybrid, in Step 0, instead of invoking $\mathcal{F}_{\text{rand-sharing}}(\Sigma_2, \frac{N-1}{2})$ and $\mathcal{F}_{\text{rand-sharing}}(\Sigma_5, \frac{N-1}{2})$, $\mathcal{S}$ emulates by itself and records shares of corrupted parties. $\mathcal{S}$ delays the sampling of honest parties' shares. $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have identical distributions.

$\mathbf{Hyb}_2$: In this hybrid, in Step 1.1.a, instead of invoking $\mathcal{F}_{\text{3-triple}}$, $\mathcal{S}$ emulates by itself and records shares of corrupted party. For $\ell \in [N]$ such that $|\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\}| \leq 1$, $\mathcal{S}$ delays the sampling of honest parties' shares. $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ have identical distributions.

$\mathbf{Hyb}_3$: In this hybrid, in Step 1.1.b, when an honest party distributes degree-$(t+k-1, t+k-1)$ packed Shamir sharings, $\mathcal{S}$ samples random shares subject to degree-$(t+k-1, t+k-1)$ packed Shamir shairngs as shares of corrupted parties. When invoking Check-All, $\mathcal{S}$ computes the evaluation points held by an honest party and a corrupted party using shares of corrupted parties. $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ have identical distributions.

$\mathbf{Hyb}_4$: In this hybrid, in Step 1.1.b, when a corrupted party P_{ℓ_v} distributes degree-$(t+k-1, t+k-1)$ packed Shamir sharing $[\![\widetilde{\boldsymbol{U}_\ell(v)}]\!]_{t+k-1}$. If all parties accept in Check-All, $\mathcal{S}$ reconstructs the secret $\widetilde{\boldsymbol{U}_\ell(v)}$, and additionally computes difference $\Delta\boldsymbol{U}_\ell(v) \leftarrow \Delta\boldsymbol{U}_\ell(v) - \boldsymbol{U}_\ell(v)$, where $\boldsymbol{U}_\ell(v)$ is computed using shares of P_{ℓ_v} received when $\mathcal{F}_{\text{3-triple}}$. $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ have identical distributions.

$\mathbf{Hyb}_5$: In this hybrid, when simulating Step 1 of Batched-Vrfy, instead of invoking $\mathcal{F}_{\text{rand-sharing}}(\Sigma_6, N)$, $\mathcal{S}$ emulates it by itself and records shares of corrupted parties. $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ have identical distributions.

$\mathbf{Hyb}_6$: In this hybrid, when simulating Step 2.1 of Batched-Vrfy, we change the computation of honest parties' shares of $[\![\boldsymbol{\epsilon}_\ell]\!]_{t+k-1}$ as follows. We first compute corrupted parties' shares of $[\![\boldsymbol{\epsilon}_\ell]\!]_{t+k-1}$ and the secrets as $\boldsymbol{\epsilon}_\ell = \lambda_{1,1}(3) \cdot \Delta\boldsymbol{U}_\ell(1) + \lambda_{1,2}(3) \cdot \Delta\boldsymbol{U}_\ell(2) - \Delta\boldsymbol{U}_\ell(3)$, then randomly samples a degree-$(t+k-1, t+k-1)$ packed Shamir sharing $[\![\boldsymbol{\theta}_\ell]\!]_{t+k-1}$ based on shares of corrupted parties and the secrets. Then we compute honest parties' shares of $[\![\boldsymbol{O}_\ell]\!]_{t+k-1}$ as $[\![\boldsymbol{O}_\ell]\!]_{t+k-1} \leftarrow [\![\boldsymbol{\theta}_\ell]\!]_{t+k-1} - [\![\boldsymbol{\epsilon}_\ell]\!]_{t+k-1}$. $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ have identical distributions due to the randomness of $[\![\boldsymbol{O}_\ell]\!]_{t+k-1}$.

$\mathbf{Hyb}_7$: In this hybrid, when simulating Batched-Vrfy, if all parties broadcast `accept`, we do not compute honest parties' shares of $[\![\boldsymbol{O}_\ell]\!]_{t+k-1}$, which is feasible since they not used in $\mathbf{Hyb}_6$. $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ have identical distributions.

$\mathbf{Hyb}_8$: In this hybrid, when simulating Batched-Vrfy, if all parties broadcast `accept`, we compute corrupted parties' shares of $[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}, [\![\boldsymbol{C}]\!]_{t+k-1}$ for each $\ell \in [N]$. Additionally, for each $\ell \in [N]$ such that $|\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\}| \geq 2$, we compute the whole sharings $[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}, [\![\boldsymbol{C}]\!]_{t+k-1}$. $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ have identical distributions.

$\mathbf{Hyb}_9$: In this hybrid, when simulating Mult, when all parties use a packed triple known to the adversary, we change the computation of honest parties' shares of $[\![\boldsymbol{X} + \boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{Y} + \boldsymbol{B}]\!]_{t+k-1}$ as follows. We first compute corrupted parties' shares of $[\![\boldsymbol{X}]\!]_{t+k-1}$, $[\![\boldsymbol{Y}]\!]_{t+k-1}$ and randomly sample two sharings as $[\![\boldsymbol{X}]\!]_{t+k-1}$, $[\![\boldsymbol{Y}]\!]_{t+k-1}$ based on shares of corrupted parties. Then we compute honest parties' shares of $\{[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}\}_{\ell \in \mathcal{D}_{1,1}}$ as linear combinations of their shares of $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}\}_{\ell \in \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{D}_{1,0} \cup \mathcal{D}_{1,2}}$, then sample honest parties' shares of $[\![\boldsymbol{C}_\ell]\!]_{t+k-1}$ based on shares of corrupted parties and the secret $\boldsymbol{C}_\ell = \boldsymbol{A}_\ell * \boldsymbol{B}_\ell$ for each $\ell \in \mathcal{D}_{1,1}$. $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ have identical distributions because there is a one-to-one correspondence between $\{[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}\}_{\ell \in \mathcal{D}_{1,1}}$ and $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}\}_{\ell \in \mathcal{E}_2}$ given values of $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}\}_{\ell \in \mathcal{E}_1 \cup \mathcal{D}_{1,0} \cup \mathcal{D}_{1,2}}$, and $\{[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}\}_{\ell \in \mathcal{D}_{1,1}}$ are random in $\mathbf{Hyb}_8$.

$\mathbf{Hyb}_{10}$: In this hybrid, when simulating Mult, when all parties use a packed triple unknown to the adversary, we change the computation of honest parties' shares of $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1}$ and $[\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{t+k-1}$ as follows. We first compute corrupted parties' shares of $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1}$ and $[\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{t+k-1}$, randomly sample two degree-$(t+k-1,t+k-1)$ packed Shamir sharings as $[\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1}$ and $[\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{t+k-1}$ based on shares of corrupted parties. Then we compute honest parties' shares of $[\![\boldsymbol{X}]\!]_{t+k-1}$ and $[\![\boldsymbol{Y}]\!]_{t+k-1}$ as $[\![\boldsymbol{X}]\!]_{t+k-1} \leftarrow [\![\boldsymbol{X}+\boldsymbol{A}]\!]_{t+k-1} - [\![\boldsymbol{A}]\!]_{t+k-1}$ and $[\![\boldsymbol{Y}]\!]_{t+k-1} \leftarrow [\![\boldsymbol{Y}+\boldsymbol{B}]\!]_{t+k-1} - [\![\boldsymbol{B}]\!]_{t+k-1}$, respectively. $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_{9}$ have identical distributions because of the randomness of $[\![\boldsymbol{A}]\!]_{t+k-1}, [\![\boldsymbol{B}]\!]_{t+k-1}$.

$\mathbf{Hyb}_{11}$: In this hybrid, instead of invoking $\mathcal{F}_{\text{batched-vrfy}}(P_{\text{king}}, \Sigma_8)$, $\mathcal{S}$ emulates itself as follows. $\mathcal{S}$ sends honest parties' shares of $\{[\![\overline{\boldsymbol{V}_\ell}]\!]_{k-1}, [\![\boldsymbol{V}_\ell]\!]_{t+k-1}\}_{\ell\in[N]}$ and $\{[\![\widetilde{\boldsymbol{U}_\ell}]\!]_{k-1}, [\![\overline{\boldsymbol{U}_\ell}]\!]_{t+2k-2}\}_{\ell\in\frac{N-1}{2}}$ to the adversary and checks whether $\overline{\boldsymbol{V}_\ell} = \boldsymbol{V}_\ell$ for every $\ell \in [N]$ and $\widetilde{\boldsymbol{U}_\ell} = \overline{\boldsymbol{U}_\ell}$ for every $\ell \in [\frac{N-1}{2}]$. $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ have identical distributions.

$\mathbf{Hyb}_{12}$: In this hybrid, when simulating Step 3.1, $\mathcal{S}$ additionally checks for $\ell \in [N]$ such that $|\mathcal{C} \cap \{\ell_1, \ell_2, \ell_3\}| \geq 2$, i.e., $\ell \in \mathcal{E}$, whether it holds that $\boldsymbol{A}_\ell * \boldsymbol{B}_\ell = \boldsymbol{C}_\ell$. If all checks pass, we change the computation of honest parties' shares of $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}, [\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{E}_3}$ as follows. For each $\ell \in \mathcal{E}_3$, we first compute corrupted parties' shares of $([\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}, [\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1})$, and randomly sample two degree-$(t+k-1, t+k-1)$ packed Shamir sharings as $[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}$ based on shares of corrupted parties. Then we compute $\boldsymbol{F}(\ell), \boldsymbol{G}(\ell)$ and $\overline{\boldsymbol{H}}(\ell) \leftarrow \boldsymbol{F}(\ell) * \boldsymbol{G}(\ell)$, and randomly sample a degree-$(t+k-1, t+k-1)$ packed Shamir sharing as $[\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1}$ based on shares of corrupted parties and the secret. Then we compute honest parties' shares of $\{[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,2}}$ as linear combinations of their shares of $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{E}_1\cup\mathcal{E}_2\cup\mathcal{E}_3\cup\mathcal{D}_{1,0}}$, then compute honest parties' shares of $\{[\![\boldsymbol{C}_\ell]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,2}}$ as linear combinations of their shares of $\{[\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{E}_1\cup\mathcal{E}_2\cup\mathcal{E}_3\cup\mathcal{D}_2\cup\mathcal{D}_{1,0}\cup\mathcal{D}_{1,1}}$. $\mathbf{Hyb}_{12}$ and $\mathbf{Hyb}_{11}$ have identical distributions because

- by the second claim above, if all checks pass, we have $\boldsymbol{F} * \boldsymbol{G} = \overline{\boldsymbol{H}}$;
- there is a one-to-one correspondence between $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{E}_3}$ and $\{[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,2}}$ given values in $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{E}_1\cup\mathcal{E}_2\cup\mathcal{D}_{1,0}}$, and $\{[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,2}}$ are random in $\mathbf{Hyb}_{11}$;
- there is a one-to-one correspondence between $\{[\![\boldsymbol{C}_\ell]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,2}}$ and $\{[\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{E}_3}$ given the values in $\{[\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{E}_1\cup\mathcal{E}_2\cup\mathcal{D}_2\cup\mathcal{D}_{1,0}\cup\mathcal{D}_{1,1}}$ and $\{[\![\boldsymbol{C}_\ell]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,2}}$ are random subject to shares of corrupted parties and the secrets $\boldsymbol{C}_\ell = \boldsymbol{A}_\ell * \boldsymbol{B}_\ell$ in $\mathbf{Hyb}_{11}$.

$\mathbf{Hyb}_{13}$: In this hybrid, we change the way of simulating honest parties' outputs and let honest parties receive outputs from $\mathcal{F}_{\text{packed-triple}}$.

- If one of checks in $\mathbf{Hyb}_{12}$ fails, a disputed pair E has been identified. $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.
- If all checks in $\mathbf{Hyb}_{12}$ pass,
  - if some party broadcasts `Incorrect` in **triple verification**, a disputed pair E has been identified. $\mathcal{S}$ sends `reject` together with E to $\mathcal{F}_{\text{packed-triple}}$.
  - if all parties broadcast `accept` in **triple verification**, for each $\ell \in \{\frac{10.5N}{27}+1, \ldots, \frac{N+1}{2}\}$, we compute corrupted parties' shares of $([\![\boldsymbol{F}(N+\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(N+\ell)]\!]_{t+k-1}, [\![\overline{\boldsymbol{H}}(N+\ell)]\!]_{t+k-1})$ and sends them to $\mathcal{F}_{\text{packed-triple}}$.

  We compute honest parties' shares of $\{[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,0}}$ as linear combinations of their shares of $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{E}_1\cup\mathcal{E}_2\cup\mathcal{E}_3\cup\{N+\frac{10.5N}{N}+1,\ldots,N+\frac{N+1}{2}\}}$. We then compute their shares of $\{[\![\boldsymbol{C}_\ell]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,0}}$ as linear combinations of their shares of $\{[\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{E}_1\cup\mathcal{E}_2\cup\mathcal{E}_3\cup\mathcal{D}_2\cup\mathcal{D}_{1,1}\cup\mathcal{D}_{1,2}}$ and $\{[\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1}\}_{\ell\in\{N+\frac{10.5N}{N}+1,\ldots,N+\frac{N+1}{2}\}}$.

$\mathbf{Hyb}_{13}$ and $\mathbf{Hyb}_{12}$ have identical distributions because

- there is a one-to-one correspondence between $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}\}_{\ell\in\{N+\frac{10.5N}{N}+1,\ldots,N+\frac{N+1}{2}\}}$ and $\{[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,0}}$ given values in $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{E}_1\cup\mathcal{E}_2\cup\mathcal{E}_3}$, and $\{[\![\boldsymbol{A}_\ell]\!]_{t+k-1}, [\![\boldsymbol{B}_\ell]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,0}}$ are random in $\mathbf{Hyb}_{12}$;

– there is a one-to-one correspondence between $\{[\![\boldsymbol{C}_\ell]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,0}}$ and $\{[\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1}\}_{\ell\in\{N+\frac{10.5N}{N}+1,\ldots,N+\frac{N+1}{2}\}}$ given values in $\{[\![\overline{\boldsymbol{H}}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{E}_1\cup\mathcal{E}_2\cup\mathcal{E}_3\cup\mathcal{D}_2\cup\mathcal{D}_{1,1}\cup\mathcal{D}_{1,2}}$, and $\{[\![\boldsymbol{C}_\ell]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,0}}$ are random subject to shares of corrupted parties and the secrets $\boldsymbol{C}_\ell = \boldsymbol{A}_\ell * \boldsymbol{B}_\ell$ in $\mathbf{Hyb}_{12}$.

$\mathbf{Hyb}_{14}$: In this hybrid, if all checks in $\mathbf{Hyb}_{12}$ pass and all parties broadcast `accept` in **triple verification**, we do not compute honest parties' shares of $\{[\![\boldsymbol{F}(\ell)]\!]_{t+k-1}, [\![\boldsymbol{G}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_{1,0}\cup\mathcal{D}_{1,1}\cup\mathcal{D}_{1,2}}$ and their shares of $\{[\![\boldsymbol{H}(\ell)]\!]_{t+k-1}\}_{\ell\in\mathcal{D}_2\cup\mathcal{D}_{1,0}\cup\mathcal{D}_{1,1}\cup\mathcal{D}_{1,2}}$, which is feasible as they are not used in the simulation. $\mathbf{Hyb}_{14}$ and $\mathbf{Hyb}_{13}$ have identical distributions.

Since $\mathbf{Hyb}_{14}$ corresponds to the ideal world, Packed-Triple securely computes $\mathcal{F}_{\text{packed-triple}}$ in the $\mathcal{F}_{\text{3-triple}}$, $\mathcal{F}_{\text{rand-sharing}}$-hybrid model. □

References

AAPP23. Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Detect, pack and batch: Perfectly-secure MPC with linear communication and constant expected time. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 251–281, Cham, 2023. Springer Nature Switzerland.

AAY21. Ittai Abraham, Gilad Asharov, and Avishay Yanai. Efficient perfectly secure computation with optimal resilience. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography*, pages 66–96, Cham, 2021. Springer International Publishing.

AC24. Gilad Asharov and Anirudh Chandramouli. Perfect (parallel) broadcast in constant expected rounds via statistical VSS. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 310–339, Cham, 2024. Springer Nature Switzerland.

BGP92. Piotr Berman, Juan A. Garay, and Kenneth J. Perry. *Bit optimal distributed consensus*, page 313–321. Plenum Press, USA, 1992.

BGW88. 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.

BH08. Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *Theory of Cryptography*, pages 213–230, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

Bra87. Gabriel Bracha. An O(log n) expected rounds randomized byzantine generals protocol. *J. ACM*, 34(4):910–920, oct 1987.

Can00. Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13:143–202, 2000.

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.

CP17. Ashish Choudhury and Arpita Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Transactions on Information Theory*, 63:428–468, 2017.

CW92. Brian A. Coan and Jennifer L. Welch. Modular construction of a byzantine agreement protocol with optimal message bit complexity. *Information and Computation*, 97(1):61–85, 1992.

DEN24. Anders Dalskov, Daniel Escudero, and Ariel Nof. Fully secure MPC and zk-FLIOP over rings: New constructions, improvements and extensions. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, pages 136–169, Cham, 2024. Springer Nature Switzerland.

DLN19. Ivan Damgård, Kasper Green Larsen, and Jesper Buus Nielsen. Communication lower bounds for statistically secure MPC, with or without preprocessing. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 61–84, Cham, 2019. Springer International Publishing.

DLS21. Ivan Bjerre Damgård, Boyang Li, and Nikolaj Ignatieff Schwartzbach. More Communication Lower Bounds for Information-Theoretic MPC. In Stefano Tessaro, editor, *2nd Conference on Information-Theoretic Cryptography (ITC 2021)*, volume 199 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

DNOR16. Ivan Damgård, Jesper Buus Nielsen, Rafail Ostrovsky, and Adi Rosén. Unconditionally secure computation with reduced interaction. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 420–447, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

DNPR16. Ivan Damgård, Jesper Buus Nielsen, Antigoni Polychroniadou, and Michael A. Raskin. On the communication required for unconditionally secure multiplication. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, Lecture Notes in Computer Science, pages 459–488. Springer, 2016.

DPPR26. Ivan Damgård, Shravani Patil, Arpita Patra, and Lawrence Roy. New upper and lower bounds for perfectly secure MPC. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026*, pages 366–396, Cham, 2026. Springer Nature Switzerland.

EGPS22. Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. TurboPack: Honest majority MPC with constant online communication. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 951–964, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

GBO+23. Mariana Gama, Emad Heydari Beni, Emmanuela Orsini, Nigel P. Smart, and Oliver Zajonc. MPC with delayed parties over star-like networks. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 172–203, Singapore, 2023. Springer Nature Singapore.

GLS19. Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional MPC with guaranteed output delivery. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 85–114, Cham, 2019. Springer International Publishing.

GLS24. Vipul Goyal, Chen-Da Liu-Zhang, and Yifan Song. Towards achieving asynchronous MPC with linear communication and optimal resilience. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, pages 170–206, Cham, 2024. Springer Nature Switzerland.

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. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 275–304, Cham, 2021. Springer International Publishing.

GPS22. Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 3–32, Cham, 2022. Springer Nature Switzerland.

HMP00. Martin Hirt, Ueli Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, pages 143–161, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

PP25. Shravani Patil and Arpita Patra. Perfectly-secure network-agnostic MPC with optimal resiliency. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '25, page 121–130, New York, NY, USA, 2025. Association for Computing Machinery.

SY24. Yifan Song and Xiaxi Ye. Perfectly-secure MPC with constant online communication complexity. In Elette Boyle and Mohammad Mahmoody, editors, *Theory of Cryptography*, pages 329–361, Cham, 2024. Springer Nature Switzerland.

SY25. Yifan Song and Xiaxi Ye. Honest majority MPC with $\tilde{O}(|C|)$ communication in Minicrypt. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025*, pages 453–483, Cham, 2025. Springer Nature Switzerland.

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.

A Additional Preliminary

A.1 Properties of Packed Shamir Sharings over Bivariate Polynomials

The following proposition regarding degree-(d, d) bivariate polynomials implies two useful properties of degree-(d, d) packed Shamir sharings: 1) the whole sharing is fully determined by $d + 1$ parties' shares when pairwise consistency is assumed and 2) any set of shares of $d+1-k$ parties gives no information about the secret, which are stated formally as two claims below.

Proposition 5. *For $d+1$ distinct field elements $\alpha_1, \ldots, \alpha_d, \alpha_{d+1}$, let $f_1(x)$, …, $f_{d+1}(x)$ be arbitrary degree-d polynomials. Then there exists a unique degree-(d, d) bivariate polynomial $S(x, y)$ such that for each $i \in [d+1]$, $S(x, \alpha_i) = f_i(x)$.*

Proof. Considering the existence, define a degree-(d,d) bivariate polynomial $S(x,y)$ as $S(x,y) = \sum_{i=1}^{d+1} f_i(x) \cdot \prod_{j\in[d],j\neq i} \frac{y-\alpha_j}{\alpha_i-\alpha_j}$. We claim that $S(x,\alpha_r) \equiv f_r(x)$ for every $r \in [d+1]$.

In fact, for every $r \in [d+1]$,

$$\begin{aligned} S(x,\alpha_r) &= \sum_{i\in[d+1]} f_i(x) \prod_{j\in[d+1],j\neq i} \frac{\alpha_r-\alpha_j}{\alpha_i-\alpha_j} \\ &= f_r(x) + \sum_{i\in[d+1],i\neq r} f_i(x) \prod_{j\in[d+1],j\neq i} \frac{\alpha_r-\alpha_j}{\alpha_i-\alpha_j} \\ &= f_r(x). \end{aligned}$$

Now we focus on the uniqueness of S. For the sake of contradiction, suppose there exists a degree-(d,d) bivariate polynomial $S' \not\equiv S$ such that $S'(x,\alpha_i) = f_i(x)$ for every $i \in [d+1]$. Let $R(x,y) = S(x,y) - S'(x,y)$. Since $R(x,y)$ is a degree-(d,d) bivariate polynomial, there exists degree-d polynomials $r_0(y)$, …, $r_d(y)$ such that $R(x,y) = \sum_{j=0}^{d} r_j(y) \cdot x^j$. Then for every $i \in [d+1]$, $R(x,\alpha_i) = S(x,\alpha_i) - S'(x,\alpha_i) = f_i(x) - f_i(x) = 0$, which implies $\sum_{j=0}^{d} r_j(\alpha_i) \cdot x^j = 0$. Hence, for every $i \in [d+1]$, $r_j(\alpha_i) = 0$ holds for every $j \in \{0,1,\ldots,d\}$. Since for every $j \in \{0,1,\ldots,d\}$, $r_j(\cdot)$ is a degree-d polynomial, we have $r_j(y) \equiv 0$. Therefore, $R(x,y) \equiv 0$, which leads to a contradiction. □

Claim (Pairwise consistency check). Let $\mathcal{I} \subset [n]$ such that $|\mathcal{I}| \geq d+1$. Let $\{f_i(x), g_i(y)\}_{i\in\mathcal{I}}$ be a set of pairs of degree-d polynomials. If for every $i,j \in \mathcal{I}$, it holds that $f_i(\alpha_j) = g_j(\alpha_i)$, then there exists a unique degree-(d,d) bivariate polynomial S such that $f_i(x) = S(x,\alpha_i)$ and $g_i(y) = S(\alpha_i,y)$ for every $i \in \mathcal{I}$.

Proof. Let $\mathcal{I}' \subset [n]$ such that $\mathcal{I} \subseteq \mathcal{I}'$ and $|\mathcal{I}'| = d+1$. Considering the existence, by Proposition 5, there exists a degree-(d,d) polynomial $S(x,y)$ such that $S(x,\alpha_i) = f_i(x)$ for every $i \in \mathcal{I}'$. Then we prove $S(x,y)$ satisfies the requirement. In fact, for every $j \in \mathcal{I}$,

$$\forall\ i \in \mathcal{I}', g_j(\alpha_i) = f_i(\alpha_j) = S(\alpha_j,\alpha_i),$$

which implies $g_j(y) \equiv S(\alpha_j,y)$. For every $j \in \mathcal{I}$,

$$\forall\ i \in \mathcal{I}', f_j(\alpha_i) = g_i(\alpha_j) = S(\alpha_i,\alpha_j),$$

which implies $f_j(x) \equiv S(x,\alpha_j)$. The uniqueness of S follows from the uniqueness in Proposition 5. □

Claim (Secrecy lemma). Let $\mathcal{I} \subset [n]$ such that $|\mathcal{I}| \leq d+1-k$. For any two $k \times k$ matrices of k^2 secrets $\boldsymbol{X}_1, \boldsymbol{X}_2 \in \mathbb{F}^{k\times k}$, then

$$\begin{aligned} &\{f_i(x), g_i(y)\}_{i\in\mathcal{I}}, \{f_i(x), g_i(y)\}_{i\in[n]} \leftarrow \mathtt{Share}(\boldsymbol{X}_1) \\ \equiv &\{f_i(x), g_i(y)\}_{i\in\mathcal{I}}, \{f_i(x), g_i(y)\}_{i\in[n]} \leftarrow \mathtt{Share}(\boldsymbol{X}_2). \end{aligned}$$

Proof. Let $\mathcal{I}' \subset [n]$ be subset of size $d+1-k$ such that $\mathcal{I} \subseteq \mathcal{I}'$. The last claim implies there is a one-to-one correspondence between a degree-(d,d) bivariate polynomial and a set of $(d+1)$ pairs of consistent degree-d polynomials $\{f_i(x), g_i(y)\}_{i\in[d+1]}$, which further implies a one-to-one correspondence between a degree-(d,d) bivariate polynomial and a set of $(d+1-k)$ pairs of consistent degree-d polynomials plus a $k \times k$ matrix defining the secret $\{f_i(x), g_i(y)\}_{i\in\mathcal{I}'} \cup \boldsymbol{X}$.

Based on this correspondence, we consider the following process of sharing a secret $\boldsymbol{X} = (X_{ij})_{ij}$:

$\mathtt{Share}'(\boldsymbol{X})$: sample random $d+1-k$ pairs of degree-d polynomials as $\{f_i(x),\ g_i(y)\}_{i\in\mathcal{I}'}$ such that $f_i(\alpha_j) = g_j(\alpha_i)$ for every $i,j \in \mathcal{I}'$, compute a degree-(d,d) bivariate polynomial such that

- for $i \in \mathcal{I}'$, $S(x,\alpha_i) = f_i(x)$ and $S(\alpha_i,y) = g_i(y)$;
- for $i,j \in [k]$, $S(\beta_i,\beta_j) = X_{ij}$,

and output $\{f_i(x), g_i(y)\}_{i\in[n]}$, where $f_i(x) := S(x,\alpha_i)$ and $g_i(y) := S(\alpha_i,y)$.

Then given $\boldsymbol{X} = (X_{ij})_{ij} \in \mathbb{F}^{k\times k}$, there is a one-to-one correspondence between a degree-(d,d) bivariate polynomial S satisfying $S(\beta_i, \beta_j) = X_{ij}$ for $i, j \in [k]$ and a set of $(d+1-k)$ pairs of consistent degree-d polynomials $\{f_i(x), g_i(y)\}_{i\in\mathcal{I}'}$, which implies

$$\begin{aligned}&\{f_i(x), g_i(y)\}_{i\in\mathcal{I}'}, \{f_i(x), g_i(y)\}_{i\in[n]} \leftarrow \mathtt{Share}(\boldsymbol{X})\\ \equiv&\{f_i(x), g_i(y)\}_{i\in\mathcal{I}'}, \{f_i(x), g_i(y)\}_{i\in[n]} \leftarrow \mathtt{Share}'(\boldsymbol{X}).\end{aligned}$$

Notice that the view of parties in $\mathcal{I}'$ generated by $\mathtt{Share}'(\boldsymbol{X})$ is independent of the secret $\boldsymbol{X}$. More precisely, we have that

$$\begin{aligned}&\{f_i(x), g_i(y)\}_{i\in\mathcal{I}'}, \{f_i(x), g_i(y)\}_{i\in[n]} \leftarrow \mathtt{Share}(\boldsymbol{X}_1)\\ \equiv&\{f_i(x), g_i(y)\}_{i\in\mathcal{I}'}, \{f_i(x), g_i(y)\}_{i\in[n]} \leftarrow \mathtt{Share}'(\boldsymbol{X}_1)\\ \equiv&\{f_i(x), g_i(y)\}_{i\in\mathcal{I}'}, \{f_i(x), g_i(y)\}_{i\in[n]} \leftarrow \mathtt{Share}'(\boldsymbol{X}_2)\\ \equiv&\{f_i(x), g_i(y)\}_{i\in\mathcal{I}'}, \{f_i(x), g_i(y)\}_{i\in[n]} \leftarrow \mathtt{Share}(\boldsymbol{X}_2),\end{aligned}$$

which implies the statement. □

A.2 Building Blocks

We list some building blocks that are useful for our constructions. Most of the functionalities are presented in the party elimination framework.

$\mathcal{F}_{\text{rand-sharing}}(\Sigma, N)$ aims to verifiably prepare random Σ-sharings. We follow [BH08] to instantiate this functionality. In particular, to prepare every group of $n' - 2t'$ Σ-sharings, each party first distributes a random Σ-sharing to all parties. Then all parties compute n' random Σ-sharings by applying a hyper-invertible matrix on original n' sharings and send their shares of $2t'$ of them to the first $2t'$ parties. These $2t'$ parties check the consistency of the sharings, where at least t' Σ-sharings are checked by honest parties. By property of the hyper-invertible matrix, the rest of $n' - 2t'$ Σ-sharings are linear combinations of $n' - t'$ Σ-sharings distributed by honest parties and t' Σ-sharings checked by honest parties. Then the $n' - 2t'$ Σ-sharings are guaranteed to be consistent.

The communication cost to prepare N Σ-sharings is $\mathsf{P2P}(O(N \cdot n' + (n')^2) \cdot |\Sigma|) + O(1) \times \mathsf{BA}(1) + O(1) \times \mathsf{BC}(O(\log N + \log n' + \log |\Sigma| + \log |\mathbb{F}|))$, amounting to $O(n' \cdot |\Sigma|)$ elements per Σ-sharing, where $|\Sigma|$ denotes the share size of a Σ-sharing.

Functionality 7: $\mathcal{F}_{\text{rand-sharing}}(\Sigma, N)$

$\mathcal{F}_{\text{rand-sharing}}$ prepares a number of N following kinds of random linear secret sharings for all active parties.

- Σ_1-sharing: a degree-$(t + 2k - 2)$ packed Shamir sharing $[\boldsymbol{o}]_{t+2k-2}$ with secret $\boldsymbol{o} = \boldsymbol{0} \in \mathbb{F}^k$.
- Σ_2-sharing: a degree-$(t + 2k - 2, t + 2k - 2)$ packed Shamir sharing $[\![\boldsymbol{O}]\!]_{t+2k-2}$ with secret $\boldsymbol{O} = \boldsymbol{0} \in \mathbb{F}^{k\times k}$.
- Σ_3-sharing: a degree-$(t+k-1, t+k-1)$ packed Shamir sharing $[\![\boldsymbol{R}]\!]_{t+k-1}$ together with degree-t Shamir sharing for each entry $[r_{ij}|j]_t$, where $\boldsymbol{R} = (r_{ij})_{ij}$ and $[r|j]_t$ denotes a random degree-t Shamir sharing with the secret r stored at the j-th position β_j.
- Σ_4-sharing: a random degree-t Shamir sharing $[r]_t$.
- Σ_5-sharing: a random degree-$(t + k - 1, t + k - 1)$ packed Shamir sharing $[\![\boldsymbol{R}]\!]_{t+k-1}$.
- Σ_6-sharing: a tuple of three degree-$(t + k - 1, t + k - 1)$ packed Shamir sharing in the form of $[\![\boldsymbol{O}]\!]_{t+k-1}$ with secret $\boldsymbol{O} = \boldsymbol{0} \in \mathbb{F}^{k\times k}$.

1. $\mathcal{F}_{\text{rand-sharing}}$ receives the set $\mathcal{C}$ of corrupted parties' identities, the set $\mathcal{P}_{\mathtt{active}}$ of all active parties' identities, and a public parameter N.

2. For each $\ell \in [N]$, $\mathcal{F}_{\text{rand-sharing}}$ receives active corrupted parties' shares $\{u_j\}_{j \in \mathcal{C} \cap \mathcal{P}_{\texttt{active}}}$ from the adversary, samples a random Σ-sharing such that for all $j \in \mathcal{C} \cap \mathcal{P}_{\texttt{active}}$, the j-th share of the sharing is u_j.
3. $\mathcal{F}_{\text{rand-sharing}}$ waits for an instruction from the adversary.
   - If $\mathcal{F}_{\text{rand-sharing}}$ receives `accept` from the adversary, $\mathcal{F}_{\text{rand-sharing}}$ distributes the shares to all active parties.
   - If $\mathcal{F}_{\text{rand-sharing}}$ receives `reject` from the adversary, $\mathcal{F}_{\text{rand-sharing}}$ receives a new disputed pair $E \subset \mathcal{P}_{\texttt{active}}$ containing at least one corrupted party from the adversary, and outputs `abort` together with E to all active parties.

$\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma, N)$ targets to check the consistency of a number of N Σ-sharings, where the whole Σ-sharing is fully determined by shares of honest parties. This functionality is not invoked in the framework of party elimination. We also follow [BH08] to instantiate this functionality. Concretely, for every group of $n-t$ Σ-sharings, all parties hold their shares of $n-t$ Σ-sharings, apply a Vandermatrix to compute n Σ-sharings which are linear combinations of original $n-t$ Σ-sharings, and reconstruct each of n sharings to a different party, who checks the consistency of the sharing. If all checks pass, this implies $n-t$ Σ-sharings checked by $n-t$ honest parties are consistent. Since there is a one-to-one correspondence between the original $n-t$ Σ-sharings and $n-t$ Σ-sharings checked by honest parties by property of Vandermonde matrix, the original $n-t$ Σ-sharings are guaranteed to be consistent.

The communication cost to check N Σ-sharings is $\mathsf{P2P}(O(N \cdot n + n^2) \cdot |\Sigma|) + O(1) \times \mathsf{BA}(1)$, amounting to $O(n \cdot |\Sigma|)$ elements per Σ-sharing, where $|\Sigma|$ denotes the share size of a Σ-sharing.

Functionality 8: $\mathcal{F}_{\text{vrfy-linear-abort}}(\Sigma, N)$

$\mathcal{F}_{\text{vrfy-linear-abort}}$ verifies whether the shares of all (honest) parties jointly form the following kinds of consistent linear secret sharings.

- Σ_1-sharing: a degree-$(t+2k-2)$ packed Shamir sharing $[\boldsymbol{o}]_{t+2k-2}$ with secret $\boldsymbol{o} = \mathbf{0} \in \mathbb{F}^k$.
- Σ_7-sharing: a pair of packed Shamir shairngs in the form of $([\boldsymbol{u}]_{t+k-1}, [\boldsymbol{u}]_{k-1})$, where the first degree-$(t+k-1)$ packed Shamir sharing and the second degree-$(k-1)$ packed Shamir sharing have the same secrets.

1. $\mathcal{F}_{\text{vrfy-linear-abort}}$ receives the set $\mathcal{C}$ of corrupted parties' identities.
2. $\mathcal{F}_{\text{vrfy-linear-abort}}$ receives honest parties' shares of $U_1, \ldots, U_N$ from the honest parties and sends them to the adversary.
3. $\mathcal{F}_{\text{vrfy-linear-abort}}$ checks if each group of shares form a valid Σ-sharing.
   - If all checks pass, $\mathcal{F}_{\text{vrfy-linear-abort}}$ receives an instruction from the adversary:
     - Upon receiving `reject`, output 0 to all parties.
     - Upon receiving `accept`, output 1 to all parties.
   - If one of the checks fails, $\mathcal{F}_{\text{vrfy-linear-abort}}$ outputs 0 to all parties.

$\mathcal{F}_{\text{batched-vrfy}}(P_r, \Sigma, N)$ targets to check the consistency of a number of N Σ-sharings which are distributed by an active party P_r. Importantly, in case of inconsistency, $\mathcal{F}_{\text{batched-vrfy}}$ will receive a disputed pair from the adversary and output it to all parties, which makes it different from $\mathcal{F}_{\text{vrfy-linear-abort}}$. The realization of $\mathcal{F}_{\text{batched-vrfy}}$ closely follows that of $\mathcal{F}_{\text{vrfy-linear-abort}}$, except that in the case of an inconsistency, P_r is responsible for identifying a disputed pair.

The communication to check N Σ-sharings is $\mathsf{P2P}(O(N \cdot n' + (n')^2) \cdot |\Sigma|) + O(1) \times \mathsf{BA}(1) + O(1) \times \mathsf{BC}(O(\log N + \log n' + \log |\Sigma| + \log |\mathbb{F}|))$, amounting $O(n' \cdot |\Sigma|)$ elements per Σ-sharing, where $|\Sigma|$ denotes the share size of a Σ-sharing.

Functionality 9: $\mathcal{F}_{\text{batched-vrfy}}(P_r, \Sigma, N)$

$\mathcal{F}_{\text{batched-vrfy}}$ checks whether all active (honest) parties hold N valid Σ-sharings distributed by an active party P_r.

- Σ_8-sharing: a pair of packed Shamir sharings in the form of $(\llbracket \boldsymbol{U} \rrbracket_{k-1}, \llbracket \boldsymbol{U} \rrbracket_{t+2k-2})$, where the first degree-$(k-1, k-1)$ packed Shamir sharing and the second degree-$(t+2k-2, t+2k-2)$ packed Shamir sharing have the same secrets $\boldsymbol{U} \in \mathbb{F}^{k \times k}$.
- Σ_9-sharing: a degree-$(k-1)$ packed Shamir sharing of the form $[\boldsymbol{u}]_{k-1}$.
- Σ_{10}-sharing: a pair of packed Shamir sharings of the form $([\boldsymbol{u}]_{k-1}, [\boldsymbol{u}]_{t+2k-2})$.
- Σ_{11}-sharing: a degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{o}]_{k-1}$ with secret $\boldsymbol{o} = \boldsymbol{0} \in \mathbb{F}^k$.

1. $\mathcal{F}_{\text{batched-vrfy}}$ receives the set $\mathcal{C}$ of corrupted parties' identities, the set $\mathcal{P}_{\texttt{active}}$ of active parties, and a public parameter N.
2. $\mathcal{F}_{\text{batched-vrfy}}$ receives honest parties' shares of $U_1, \ldots, U_N$ from the honest parties and sends them to the adversary.
3. $\mathcal{F}_{\text{batched-vrfy}}$ checks if each group of shares forms a valid Σ-sharing.
   - If all checks pass, $\mathcal{F}_{\text{batched-vrfy}}$ receives an instruction from the adversary:
     - Upon receiving `reject`, receive a disputed pair $E \subset \mathcal{P}_{\texttt{active}}$ containing at least one corrupted party, and output `abort` together with E to all parties.
     - Upon receiving `accept`, output `accept` to all parties.
   - If one of the checks fails, $\mathcal{F}_{\text{batched-vrfy}}$ receives a disputed pair $E \subset \mathcal{P}_{\texttt{active}}$ containing at least one corrupted party, and outputs `abort` together with E to all active parties.

$\mathcal{F}_{\text{triple}}(N)$ targets to verifiably prepare a number of N Beaver triples of degree-t with secrets for all active parties. We follow [BH08] to realize this functionality. In particular, all parties first invoke $\mathcal{F}_{\text{rand-sharing}}$ to verifiably prepare random degree-t Shamir sharings with secrets as $[a]_t$ and $[b]_t$ and a pair of double sharings $([r]_t, [r]_{2t})$. They locally compute $[e]_{2t} \leftarrow [a]_t \cdot [b]_t + [r]_{2t}$ and publicly and detectably reconstruct the secret e. All parties locally compute $[c]_t \leftarrow [e]_0 - [r]_t$, where $c = a \cdot b$.

The communication cost to prepare N Beaver triples of degree-t is $\mathsf{P2P}(O(N \cdot n' + (n')^2)) + O(1) \times \mathsf{BA}(1) + O(1) \times \mathsf{BC}(O(\log N + \log n' + \log |\mathbb{F}|))$, amounting to $O(n')$ elements per triple.

Functionality 10: $\mathcal{F}_{\text{triple}}(N)$

$\mathcal{F}_{\text{triple}}(N)$ prepares N Breaver triples for all active parties.

1. $\mathcal{F}_{\text{triple}}$ receives the set $\mathcal{C}$ of corrupted parties' identities, the set $\mathcal{P}_{\texttt{active}}$ of active parties' identities, and a public parameter N.
2. For each $\ell \in [N]$, $\mathcal{F}_{\text{triple}}$ receives corrupted parties' shares $\{u_j^{(1)}, u_j^{(2)}, u_j^{(3)}\}_{j \in \mathcal{C} \cap \mathcal{P}_{\texttt{active}}}$. $\mathcal{F}_{\text{triple}}$ randomly samples two elements a, b, computes $c \leftarrow a \cdot b$, and samples random degree-t Shamir sharings $([a]_t, [b]_t, [c]_t)$ such that for all $j \in \mathcal{C} \cap \mathcal{P}_{\texttt{active}}$, the j-th share of $([a]_t, [b]_t, [c]_t)$ is $(u_j^{(1)}, u_j^{(2)}, u_j^{(3)})$.
3. $\mathcal{F}_{\text{triple}}$ waits for an instruction from the adversary:
   - Upon receiving `accept`, for each $\ell \in [N]$, $j \in \mathcal{P}_{\texttt{active}}$, send to P_j its shares of $([a]_t, [b]_t, [c]_t)$.
   - Upon receiving `reject`, receive a new disputed pair $E \subset \mathcal{P}_{\texttt{active}}$ containing at least one corrupted party from the adversary, and output `abort` together with E to all active parties.

B Proofs of Propositions in Lemma 3

Proposition 6 (Restatement of Proposition 1). *The transcripts between P_2 and P_1 in executions $\texttt{exec}_3$ and $\texttt{exec}_2$ are identical. The transcripts between P_3 and P_1 in executions $\texttt{exec}_3$ and $\texttt{exec}_1$ are identical.*

The transcripts between P_2 and P_3 and transcripts between P_0 and each party in $\{P_1, P_2, P_3\}$ are identical in four executions $\mathtt{exec}_0, \mathtt{exec}_1, \mathtt{exec}_2$ and $\mathtt{exec}_3$.

Proof. We prove this statement by induction. First, by construction of $\mathtt{exec}_1$ and $\mathtt{exec}_2$, the transcripts between P_2 and P_3, as well as the transcripts between P_0 and each party in $\{P_1, P_2, P_3\}$, are identical across $\mathtt{exec}_0$, $\mathtt{exec}_1$ and $\mathtt{exec}_2$.

Consider the first message in $\mathtt{exec}_3$.

- If it is from P_0 to any party in $\{P_1, P_2, P_3\}$, then in $\mathtt{exec}_3$, since P_0 uses the same input I_0 and internal randomness r_0 as in $\mathtt{exec}_0$, the message in $\mathtt{exec}_3$ is identical to that in $\mathtt{exec}_0$, and therefore also identical to those $\mathtt{exec}_1$ and $\mathtt{exec}_2$.
- If it is from P_1 to P_2, by construction of $\mathtt{exec}_3$, since P_1 uses the messages that it would use in $\mathtt{exec}_2$ when interacting with P_2, this message is identical to that in $\mathtt{exec}_2$. Similarly, if it is from P_1 to P_3, by construction of $\mathtt{exec}_3$, since P_1 uses the messages that it would use in $\mathtt{exec}_1$ when interacting with P_3, this message is identical to that in $\mathtt{exec}_1$.
- If it is from P_2 (to any party in $\{P_0, P_1, P_3\}$), in $\mathtt{exec}_3$, since P_2 uses the same input I_2'' and internal randomness r_2'' as in $\mathtt{exec}_2$, the message in $\mathtt{exec}_3$ is identical to that in $\mathtt{exec}_2$. If it is from P_2 to P_3, it is the same across $\mathtt{exec}_0$, $\mathtt{exec}_1$, $\mathtt{exec}_2$ and $\mathtt{exec}_3$.
- If it is from P_3 (to any party in $\{P_0, P_1, P_2\}$), in $\mathtt{exec}_3$, since P_3 uses the same input I_3' and internal randomness r_3' as in $\mathtt{exec}_1$, the message in $\mathtt{exec}_3$ is identical to that in $\mathtt{exec}_1$. If it is from P_3 to P_2, it is the same across $\mathtt{exec}_0$, $\mathtt{exec}_1$, $\mathtt{exec}_2$ and $\mathtt{exec}_3$.

Now assume the first $T-1$ messages already satisfy the statement. We prove this holds for the T-th message.

- If it is from P_0 (to any party in $\{P_1, P_2, P_3\}$), by induction hypothesis, the messages previously sent or received by P_0 are identical in $\mathtt{exec}_3$ and $\mathtt{exec}_0$. Hence, the current views for P_0 (including its input I_0, internal randomness r_0, and messages sent or received by itself) are identical in $\mathtt{exec}_3$ and $\mathtt{exec}_0$. This implies this message sent from P_0 is identical in $\mathtt{exec}_3$ and $\mathtt{exec}_0$, and therefore also identical to those $\mathtt{exec}_1$ and $\mathtt{exec}_2$.
- If it is from P_1 (to any party in $\{P_0, P_2, P_3\}$), by construction of $\mathtt{exec}_3$, it satisfies the statement.
- If it is from P_2 (to any party in $\{P_0, P_1, P_3\}$), by induction hypothesis, the messages previously sent or received by P_2 are identical in $\mathtt{exec}_3$ and $\mathtt{exec}_2$. Hence, the current views for P_2 (including its input I_2'', internal randomness r_2'', and messages sent or received by itself) are identical in $\mathtt{exec}_3$ and $\mathtt{exec}_2$. This implies this message sent from P_2 is identical in $\mathtt{exec}_3$ and $\mathtt{exec}_2$. If this message is from P_2 to P_3, it is the same across $\mathtt{exec}_0$, $\mathtt{exec}_1$, $\mathtt{exec}_2$ and $\mathtt{exec}_3$.
- If it is from P_3 (to any party in $\{P_0, P_1, P_2\}$), by induction hypothesis, the messages previously sent or received by P_2 are identical in $\mathtt{exec}_3$ and $\mathtt{exec}_1$. Hence, the current views for P_3 (including its input I_3', internal randomness r_3', and messages sent or received by itself) are identical in $\mathtt{exec}_3$ and $\mathtt{exec}_1$. This implies this message sent from P_3 is identical in $\mathtt{exec}_3$ and $\mathtt{exec}_1$. If it is from P_3 to P_2, it is the same across $\mathtt{exec}_0$, $\mathtt{exec}_1$, $\mathtt{exec}_2$ and $\mathtt{exec}_3$.

By induction, the statement holds for all messages in $\mathtt{exec}_3$. □

Proposition 7 (Restatement of Proposition 2). *Let $(\boldsymbol{a}^{(3)}, \boldsymbol{b}^{(3)}, \boldsymbol{c}^{(3)})$ be the secrets of N packed triples determined by $2t+1$ parties' input (I_0, I_2'', I_3') in $\mathtt{exec}_3(d', d'')$. Then it holds that*

$$(\boldsymbol{a}^{(3)}, \boldsymbol{b}^{(3)}, \boldsymbol{c}^{(3)}) = (\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}) + (\boldsymbol{a}^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{c}^{(2)}) - (\boldsymbol{a}^{(0)}, \boldsymbol{b}^{(0)}, \boldsymbol{c}^{(0)}).$$

*Then $(\boldsymbol{a}^{(3)}, \boldsymbol{b}^{(3)}, \boldsymbol{c}^{(3)}) = (\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(2)}, \boldsymbol{a}^{(0)} * \boldsymbol{b}^{(2)} + \boldsymbol{a}^{(1)} * \boldsymbol{b}^{(0)} - \boldsymbol{a}^{(0)} * \boldsymbol{b}^{(0)})$, which implies that the input of $\mathtt{exec}_3(d', d'')$ **doesn't** satisfy the multiplication relation if there exists $\ell \in [N \cdot k]$ such that $a_\ell^{(1)} \neq a_\ell^{(0)}$ and $b_\ell^{(2)} \neq b_\ell^{(0)}$.*

Proof. We focus on the secrets of packed triples corresponding to input (I_2'', I_3') in $\texttt{exec}_3(d', d'')$. For simplicity of notations, we use $s(\bar{I}_2, \bar{I}_3)$ to denote the secrets defined by P_2's input $\bar{I}_2$ and P_3's input $\bar{I}_3$. By linearity of packed Shamir sharings, we have

$$s(I_2'', I_3') - s(I_2, I_3') = s(I_2'', I_3) - s(I_2, I_3),$$

which implies $s(I_2'', I_3') = s(I_2, I_3') + s(I_2'', I_3) - s(I_2, I_3)$ and thus

$$\begin{aligned}(\boldsymbol{a}^{(3)}, \boldsymbol{b}^{(3)}, \boldsymbol{c}^{(3)}) =& (\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}) + (\boldsymbol{a}^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{c}^{(2)}) - (\boldsymbol{a}^{(0)}, \boldsymbol{b}^{(0)}, \boldsymbol{c}^{(0)}) \\ =& (\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(0)}, \boldsymbol{a}^{(1)} * \boldsymbol{b}^{(0)}) + (\boldsymbol{a}^{(0)}, \boldsymbol{b}^{(2)}, \boldsymbol{a}^{(0)} * \boldsymbol{b}^{(2)}) \\ & - (\boldsymbol{a}^{(0)}, \boldsymbol{b}^{(0)}, \boldsymbol{a}^{(0)} * \boldsymbol{b}^{(0)}) \\ =& (\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(2)}, \boldsymbol{a}^{(1)} * \boldsymbol{b}^{(0)} + \boldsymbol{a}^{(0)} * \boldsymbol{b}^{(2)} - \boldsymbol{a}^{(0)} * \boldsymbol{b}^{(0)}).\end{aligned}$$

Notice that

$$\begin{aligned}\boldsymbol{a}^{(3)} * \boldsymbol{b}^{(3)} - \boldsymbol{c}^{(3)} &= \boldsymbol{a}^{(1)} * \boldsymbol{b}^{(2)} - (\boldsymbol{a}^{(1)} * \boldsymbol{b}^{(0)} + \boldsymbol{a}^{(0)} * \boldsymbol{b}^{(2)} - \boldsymbol{a}^{(0)} * \boldsymbol{b}^{(0)}) \\ &= (\boldsymbol{a}^{(1)} - \boldsymbol{a}^{(0)}) * (\boldsymbol{b}^{(2)} - \boldsymbol{b}^{(0)}),\end{aligned}$$

which means if there exists $\ell \in [N \cdot k]$ such that $a_\ell^{(1)} \neq a_\ell^{(0)}$ and $b_\ell^{(2)} \neq b_\ell^{(0)}$, then $\boldsymbol{a}^{(3)} * \boldsymbol{b}^{(3)} - \boldsymbol{c}^{(3)} \neq \boldsymbol{0}$. Consequently, in this case, $(\boldsymbol{a}^{(3)}, \boldsymbol{b}^{(3)}, \boldsymbol{c}^{(3)})$ doesn't satisfy the multiplication relation. □

Proposition 8 (Restatement of Proposition 3). *Let $\texttt{view}^{P_2}$ be a plausible view of P_2 with 1 as output, i.e., there exists a tuple $d = (I_0, I_1, I_2, I_3;\ r_0, r_1, r_2, r_3) \in \mathcal{D}$ such that $\texttt{view}^{P_2}(\pi, d) = \texttt{view}^{P_2}$ and $\texttt{output}(\pi, d) = 1$. Then for every $\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}} \in \mathbb{F}^{N \cdot k}$, there exists a tuple $\bar{d} = (\bar{I}_0, \bar{I}_1, I_2, \bar{I}_3; \bar{r_0}, \bar{r_1}, r_2, \bar{r_3}) \in \mathcal{D}$ of all parties' inputs and internal randomness such that $\texttt{view}^{P_2}(\pi, \bar{d}) = \texttt{view}^{P_2}$ and $(\bar{I}_0, \bar{I}_1, I_2, \bar{I}_3)$ forms degree-$(t + k - 1)$ packed Shamir sharings of $(\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}, \bar{\boldsymbol{a}} * \bar{\boldsymbol{b}})$.*

Let $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a} * \boldsymbol{b})$ be the tuple of the secrets defined by d. Let $\mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}} \subset \mathcal{D}$ be the set of all honest executions satisfying 1) the view of P_2 is $\texttt{view}^{P_2}$ and 2) the secret of the second sharing is $\boldsymbol{b}$. Then the number of possible distinct secrets of the first sharing among the executions in the set $\mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}}$ is $F^{N \cdot k}$, i.e.,

$$|\{\bar{\boldsymbol{a}} \in \mathbb{F}^{N \cdot k} \mid \exists\ \bar{d} \in \mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}} \text{ s.t. } \bar{d} \vdash \bar{\boldsymbol{a}}\}| = F^{N \cdot k}.$$

Proof. By perfect privacy against a (semi-honest) corrupted P_2, there exists a simulator $\mathcal{S}_2$ which takes as input the input and output of P_2, and outputs the view of P_2 such that for any inputs $(\bar{I}_0, \bar{I}_1, \bar{I}_3)$ of P_0, P_1, P_3 such that $(\bar{I}_0, I_2, \bar{I}_1, \bar{I}_3)$ forms valid degree-$(t + k - 1)$ packed Shamir sharings, it holds that

$$\mathcal{S}_2(I_2, F_{\text{vrfy-triple}}(\bar{I}_0, \bar{I}_1, I_2, \bar{I}_3)) \equiv \texttt{view}^{P_2}(\pi, \bar{I}_0, \bar{I}_1, I_2, \bar{I}_3),$$

where the function $F_{\text{vrfy-triple}}(\bar{I}_0, \bar{I}_1, I_2, \bar{I}_3)$ is defined to output 1 if the input tuple $(\bar{I}_0, \bar{I}_1, I_2, \bar{I}_3)$ corresponds to packed triples satisfying the multiplication relations, and 0 otherwise. Hence, for every $\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}} \in \mathbb{F}^k$, letting $(\bar{I}_0, \bar{I}_1, \bar{I}_3)$ be the input tuple from P_0, P_1 and P_3 such that $(\bar{I}_0, \bar{I}_1, I_2, \bar{I}_3)$ forms degree-$(t + k - 1)$ packed Shamir sharings of $(\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}, \bar{\boldsymbol{a}} * \bar{\boldsymbol{b}})$, by property of $F_{\text{vrfy-triple}}$, we have

$$\begin{aligned}\texttt{view}^{P_2}(\pi, I_0, I_1, I_2, I_3) &\equiv \mathcal{S}_2(I_2, F_{\text{vrfy-triple}}(I_0, I_1, I_2, I_3)) \\ &\equiv \mathcal{S}_2(I_2, F_{\text{vrfy-triple}}(\bar{I}_0, \bar{I}_1, I_2, \bar{I}_3)) \\ &\equiv \texttt{view}^{P_2}(\pi, \bar{I}_0, \bar{I}_1, I_2, \bar{I}_3).\end{aligned}$$

Then there exists a choice of a tuple of internal randomness from P_0, P_1 and P_3, $(\bar{r_0}, \bar{r_1}, \bar{r_3})$, such that $\texttt{view}^{P_2}(\pi, d) = \texttt{view}^{P_2}(\bar{I}_0, \bar{I}_1, I_2, \bar{I}_3; \bar{r_0}, \bar{r_1}, r_2, \bar{r_3})$.

By the reasoning above, fixing $\bar{\boldsymbol{b}} = \boldsymbol{b}$, for any $\bar{\boldsymbol{a}} \in \mathbb{F}^{N \cdot k}$, $\bar{\boldsymbol{a}}$ can lead to a satisfied execution in $\mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}}$. Notice that there are $F^{N \cdot k}$ possibilities of $\bar{\boldsymbol{a}} \in \mathbb{F}^{N \cdot k}$. Then we conclude the number of possible distinct secrets of the first sharing among the executions in $\mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}}$ is $F^{N \cdot k}$. □

Proposition 9 (Restatement of Proposition 4). *Given a view of P_2 $\mathtt{view}^{P_2}$ and a secret of the second sharing $\boldsymbol{b}$, we define the set $\mathcal{D}_{\mathtt{view}^{P_2},\boldsymbol{b}}$ as in the previous claim. For any two vectors of secrets $\bar{\boldsymbol{a}}_1, \bar{\boldsymbol{a}}_2 \in \mathbb{F}^{N\cdot k}$, the number of executions in $\mathcal{D}_{\mathtt{view}^{P_2},\boldsymbol{b}}$ with the first sharing's secret being $\bar{\boldsymbol{a}}_1$ is equal to the number of executions with the first sharing's secret being $\bar{\boldsymbol{a}}_2$, i.e.,*

$$|\{d \in \mathcal{D}_{\mathtt{view}^{P_2},\boldsymbol{b}} \mid d \vdash \bar{\boldsymbol{a}}_1\}| = |\{d \in \mathcal{D}_{\mathtt{view}^{P_2},\boldsymbol{b}} \mid d \vdash \bar{\boldsymbol{a}}_2\}|.$$

Proof. Intuitively, by privacy against a corrupted P_2, corrupted P_2 cannot distinguish different inputs according to its view. The formal proof goes as follows.

Given that the inputs from all parties form degree-$(t+k-1)$ packed triples satisfy the multiplication relations, a tuple $(\bar{\boldsymbol{a}}, \boldsymbol{b}, I_2)$ fully determines a tuple of inputs from all parties. For simplicity of notation, we use $(\bar{\boldsymbol{a}}, \boldsymbol{b}, I_2)$ to denote the input of all parties. Then by perfect privacy against a (semi-honest) corrupted P_2,

$$\mathtt{view}^{P_2}(\pi, \bar{\boldsymbol{a}}_1, \boldsymbol{b}, I_2) \equiv \mathcal{S}_2(I_2, 1) \equiv \mathtt{view}^{P_2}(\pi, \bar{\boldsymbol{a}}_2, \boldsymbol{b}, I_2).$$

This implies $\Pr_{d\leftarrow\mathcal{D}}[d \vdash \mathtt{view}^{P_2} \mid d \vdash \bar{\boldsymbol{a}}_1, \boldsymbol{b}, I_2] = \Pr_{d\leftarrow\mathcal{D}}[d \vdash \mathtt{view}^{P_2} \mid d \vdash \bar{\boldsymbol{a}}_2, \boldsymbol{b}, I_2]$. Then we get

$$\begin{aligned}
&\frac{|\{d \in \mathcal{D}_{\mathtt{view}^{P_2},\boldsymbol{b}} \mid d \vdash \bar{\boldsymbol{a}}_1\}|}{|\mathcal{D}_{\mathtt{view}^{P_2},\boldsymbol{b}}|} && (6)\\
=&\Pr_d[d \vdash \bar{\boldsymbol{a}}_1 \mid d \vdash \mathtt{view}^{P_2}, \boldsymbol{b}, I_2]\\
=&\frac{\Pr_d[d \vdash \mathtt{view}^{P_2}, \bar{\boldsymbol{a}}_1, \boldsymbol{b}, I_2]}{\Pr_d[d \vdash \mathtt{view}^{P_2}, \boldsymbol{b}, I_2]}\\
=&\frac{\Pr_d[d \vdash \mathtt{view}^{P_2}, \bar{\boldsymbol{a}}_1, \boldsymbol{b}, I_2]}{\Pr_d[d \vdash \bar{\boldsymbol{a}}_1, \boldsymbol{b}, I_2]} \cdot \frac{\Pr_d[d \vdash \bar{\boldsymbol{a}}_1, \boldsymbol{b}, I_2]}{\Pr_d[d \vdash \mathtt{view}^{P_2}, \boldsymbol{b}, I_2]}\\
=&\Pr_d[d \vdash \mathtt{view}^{P_2} \mid d \vdash \bar{\boldsymbol{a}}_1, \boldsymbol{b}, I_2] \cdot \frac{\Pr_d[d \vdash \bar{\boldsymbol{a}}_2, \boldsymbol{b}, I_2]}{\Pr_d[d \vdash \mathtt{view}^{P_2}, \boldsymbol{b}, I_2]} && (7)\\
=&\Pr_d[d \vdash \mathtt{view}^{P_2} \mid d \vdash \bar{\boldsymbol{a}}_2, \boldsymbol{b}, I_2] \cdot \frac{\Pr_d[d \vdash \bar{\boldsymbol{a}}_2, \boldsymbol{b}, I_2]}{\Pr_d[d \vdash \mathtt{view}^{P_2}, \boldsymbol{b}, I_2]} && (8)\\
=&\frac{\Pr_d[d \vdash \mathtt{view}^{P_2}, \bar{\boldsymbol{a}}_2, \boldsymbol{b}, I_2]}{\Pr_d[d \vdash \bar{\boldsymbol{a}}_2, \boldsymbol{b}, I_2]} \cdot \frac{\Pr_d[d \vdash \bar{\boldsymbol{a}}_2, \boldsymbol{b}, I_2]}{\Pr_d[d \vdash \mathtt{view}^{P_2}, \boldsymbol{b}, I_2]}\\
=&\frac{\Pr_d[d \vdash \mathtt{view}^{P_2}, \bar{\boldsymbol{a}}_2, \boldsymbol{b}, I_2]}{\Pr_d[d \vdash \mathtt{view}^{P_2}, \boldsymbol{b}, I_2]}\\
=&\Pr_d[d \vdash \bar{\boldsymbol{a}}_2 \mid d \vdash \mathtt{view}^{P_2}, \boldsymbol{b}, I_2]\\
=&\frac{|\{d \in \mathcal{D}_{\mathtt{view}^{P_2},\boldsymbol{b}} \mid d \vdash \bar{\boldsymbol{a}}_2\}|}{|\mathcal{D}_{\mathtt{view}^{P_2},\boldsymbol{b}}|},
\end{aligned}$$

where Equality 7 follows from the fact that d is sampled uniformly at random and Equality 8 holds because of the equality derived previously. This completes the proof of the claim. □

Upper bound $\Pr_{d\leftarrow\mathcal{D}}[\overline{\mathtt{event\ 1}}]$. Now we upper bound $\Pr_{d\leftarrow\mathcal{D}}[\overline{\mathtt{event\ 1}}]$ as follows. Let E_1 be the event $(C_0(d) < 3\cdot\mathbb{E}[C_0])$. For simplicity of notation, we abuse the notation E_1 to also denote the set of all executions

in which $C_0(d) < 3 \cdot \mathbb{E}[C_0]$, i.e., $E_1 = \{d \in \mathcal{D} \mid C_0(d) < 3 \cdot \mathbb{E}[C_0]\}$.

$$
\begin{aligned}
&\Pr_{d \leftarrow \mathcal{D}}[\overline{\texttt{event 1}}] \\
=& \sum_{\texttt{view}^{P_2}, \boldsymbol{b}} \Pr_{d \leftarrow \mathcal{D}}[\overline{\texttt{event 1}} \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] \\
=& \sum_{\texttt{view}^{P_2}, \boldsymbol{b}} \Pr_{d \leftarrow \mathcal{D}}[\overline{\texttt{event 1}} \wedge E_1 \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] + \Pr_{d \leftarrow \mathcal{D}}[\overline{\texttt{event 1}} \wedge \overline{E_1} \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] \\
=& \sum_{\texttt{view}^{P_2}, \boldsymbol{b}} \Pr_{d \leftarrow \mathcal{D}}[\overline{\texttt{event 1}} \mid E_1 \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] \cdot \Pr_{d \leftarrow \mathcal{D}}[E_1 \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] \\
&+ \Pr_{d \leftarrow \mathcal{D}}[\overline{\texttt{event 1}} \mid \overline{E_1} \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] \cdot \Pr_{d \leftarrow \mathcal{D}}[\overline{E_1} \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] \\
=& \sum_{\texttt{view}^{P_2}, \boldsymbol{b}} \Pr_{d \leftarrow \mathcal{D}}[\overline{\texttt{event 1}} \mid E_1 \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] \cdot \Pr_{d \leftarrow \mathcal{D}}[E_1 \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] \\
&+ \Pr_{d \leftarrow \mathcal{D}}[\overline{\texttt{event 1}} \mid \overline{E_1} \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] \cdot (\Pr_{d \leftarrow \mathcal{D}}[d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] - \Pr_{d \leftarrow \mathcal{D}}[E_1 \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}])
\end{aligned}
$$

$$
\begin{aligned}
<& \sum_{\texttt{view}^{P_2}, \boldsymbol{b}} \frac{1}{6} \cdot \frac{1}{\Pr_{d \leftarrow \mathcal{D}}[E_1 \mid d \vdash \texttt{view}^{P_2}, \boldsymbol{b}]} \cdot \Pr_{d \leftarrow \mathcal{D}}[E_1 \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] \\
&+ 1 \cdot (\Pr_{d \leftarrow \mathcal{D}}[d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] - \Pr_{d \leftarrow \mathcal{D}}[E_1 \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}]) \\
=& \sum_{\texttt{view}^{P_2}, \boldsymbol{b}} \frac{1}{6} \cdot \Pr_{d \leftarrow \mathcal{D}}[d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] + \Pr_{d \leftarrow \mathcal{D}}[d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] - \Pr_{d \leftarrow \mathcal{D}}[E_1 \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] \\
=& 1 + \frac{1}{6} - \Pr_{d \leftarrow \mathcal{D}}[E_1] \\
\leq& 1 + \frac{1}{6} - (1 - \frac{1}{3}) \\
=& \frac{1}{2}.
\end{aligned}
$$

The deduction above takes advantage of the following two inequalities:

1. $\Pr_{d \leftarrow \mathcal{D}}[\overline{\texttt{event 1}} \mid E_1 \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] < \frac{1}{6} \cdot \frac{1}{\Pr_{d \leftarrow \mathcal{D}}[E_1 | d \vdash \texttt{view}^{P_2}, \boldsymbol{b}]}$. We prove this as below.

   To compute the probability, we first introduce some notations. For simplicity, given an honest execution defined by d, we denote a party P's input concatenated with its transcript by $\sigma^P(\pi, d)$, i.e.,

   $$\sigma^P(\pi, d) = \texttt{input}^P(\pi, d) || \texttt{transcript}^P(\pi, d).$$

   Define $\mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}} \subset \mathcal{D}$ to the set of all executions in which the view of P_2 is $\texttt{view}^{P_2}$ and the secret of the second sharing is $\boldsymbol{b}$, i.e., $\mathcal{D}_{\texttt{view}^{P_2}, \boldsymbol{b}} = \{d \in \mathcal{D} \mid d \vdash \texttt{view}^{P_2}, \boldsymbol{b}\}$. Define $\texttt{Bad}_{\texttt{view}^{P_2}, \boldsymbol{b}, \sigma^{P_0}} \subset \mathcal{D}$ to be the set of all executions in which

   - the view of P_2 is $\texttt{view}^{P_2}$,
   - the secret of the second sharing is $\boldsymbol{b}$,
   - the input and transcript of P_0 is σ^{P_0},
   - the possibilities of distinct values of the secret of the first sharing is no more than $F^{\frac{N \cdot k}{2}}$.

Notice that

$$
\begin{aligned}
&\Pr_{d\leftarrow\mathcal{D}}[\overline{\texttt{event 1}} \mid E_1 \wedge d \vdash \texttt{view}^{P_2}, \boldsymbol{b}] \\
=&\frac{\sum_{\sigma^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\boldsymbol{b},\sigma^{P_0}}|}{|E_1 \cap \mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}|} && (9)\\
\leq&\frac{|\{\sigma^{P_0}(\pi, d) \mid d \in \mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}\}| \cdot \max_{\sigma^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\boldsymbol{b},\sigma^{P_0}}|}{|E_1 \cap \mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}|} && (10)\\
\leq&\frac{F^{3N+3\cdot\mathbb{E}[C_0]} \cdot \max_{\sigma^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\boldsymbol{b},\sigma^{P_0}}|}{|E_1 \cap \mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}|} && (11)\\
=&\frac{F^{3N+3\cdot\mathbb{E}[C_0]} \cdot \max_{\sigma^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\boldsymbol{b},\sigma^{P_0}}|}{|\mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}|} \cdot \frac{|\mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}|}{|E_1 \cap \mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}|} \\
\leq&\frac{F^{3N+3\cdot\mathbb{E}[C_0]} \cdot F^{\frac{N\cdot k}{2}}}{F^{N\cdot k}} \cdot \frac{|\mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}|}{|E_1 \cap \mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}|} && (12)\\
<&\frac{1}{6} \cdot \frac{|\mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}|}{|E_1 \cap \mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}|} && (13)\\
=&\frac{1}{6} \cdot \frac{1}{\Pr_{d\leftarrow\mathcal{D}}[E_1 \mid d \vdash \texttt{view}^{P_2}, \boldsymbol{b}]},
\end{aligned}
$$

where the step-by-step reasoning is given as follows. To show Equality 9, we group all executions $E_1 \cap \mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}$ according to P_0's input and transcript. Then `event` 1 doesn't happen if and only if the sampled execution d lies in the set $\texttt{Bad}_{\texttt{view}^{P_2},\boldsymbol{b},\sigma^{P_0}}$ for some σ^{P_0}.
Inequality 10 holds because

$$
\begin{aligned}
&\sum_{\sigma^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\boldsymbol{b},\sigma^{P_0}}| \\
\leq&\sum_{\sigma^{P_0}} \max_{\sigma^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\boldsymbol{b},\sigma^{P_0}}| \\
=&|\{\sigma^{P_0}(\pi, d) \mid d \in \mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}\}| \cdot \max_{\sigma^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\boldsymbol{b},\sigma^{P_0}}|.
\end{aligned}
$$

As for Inequality 11, because for any execution in E_1, the length of P_0's input together with its transcript is at most $3N + 3 \cdot \mathbb{E}[C_0]$, resulting in at most $F^{3N+3\cdot\mathbb{E}[C_0]}$ possibilities.
Regarding Inequality 12, by two claims above and the definition of $\texttt{Bad}_{\texttt{view}^{P_2},\boldsymbol{b}}$, we have

$$
\frac{\max_{\sigma^{P_0}} |\texttt{Bad}_{\texttt{view}^{P_2},\boldsymbol{b},\sigma^{P_0}}|}{|\mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}|} \leq \frac{F^{\frac{N\cdot k}{2}} \cdot |\{d \in \mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}}, \mid d \vdash \bar{\boldsymbol{a}}\}|}{F^{N\cdot k} \cdot |\{d \in \mathcal{D}_{\texttt{view}^{P_2},\boldsymbol{b}} \mid d \vdash \bar{\boldsymbol{a}}\}|} = \frac{F^{\frac{N\cdot k}{2}}}{F^{N\cdot k}},
$$

for any $\bar{\boldsymbol{a}} \in \mathbb{F}^{N\cdot k}$ representing the secret of the first sharing.
Finally, for Inequality 13, by assuming that $\mathbb{E}[C_0] < \frac{N\cdot k}{8}$ and setting $c_1 = \lceil \frac{24}{\log F} \rceil, c_2 = 24$, for all $N > c_1, k > c_2$, we have that

$$
\frac{F^{3N+3\cdot\mathbb{E}[C_0]} \cdot F^{\frac{N\cdot k}{2}}}{F^{N\cdot k}} < \frac{F^{3N}}{F^{\frac{N\cdot k}{8}}} \leq \frac{F^{3N}}{F^{\frac{25N}{8}}} = \frac{1}{F^{N/8}} < \frac{1}{6}.
$$

2. $\Pr_{d\leftarrow\mathcal{D}}[E_1] \geq 1 - \frac{1}{3}$. We prove this as below.
   By Markov's inequality, we have

$$
\Pr_{d\leftarrow\mathcal{D}}[C_0(d) \geq 3 \cdot \mathbb{E}[C_0]] \leq \frac{\mathbb{E}[C_0]}{3 \cdot \mathbb{E}[C_0]} = \frac{1}{3}.
$$

   Then we have $\Pr_{d\leftarrow\mathcal{D}}[E_1] = 1 - \Pr_{d\leftarrow\mathcal{D}}[C_0(d) \geq 3 \cdot \mathbb{E}[C_0]] \geq 1 - \frac{1}{3}$.

C The Figures

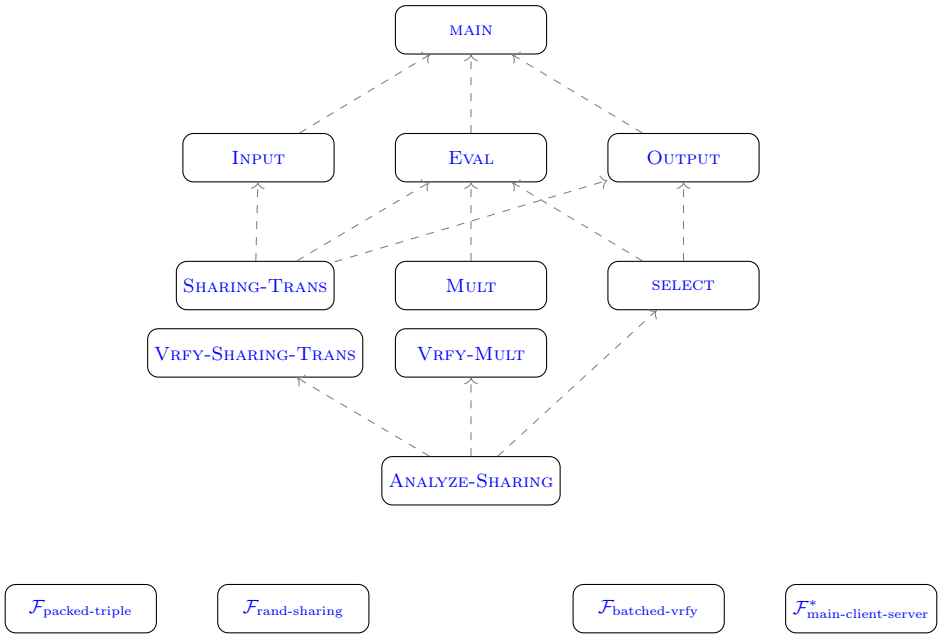


Fig. 1. Framework of perfectly secure MPC for general circuits in the preprocessing model.

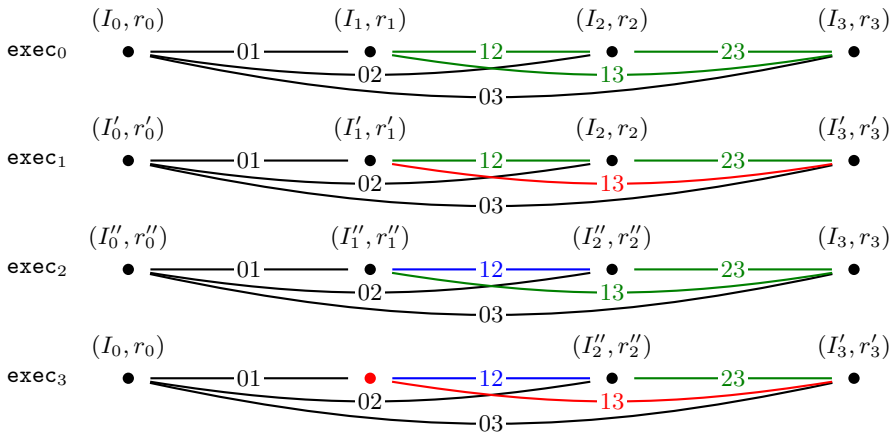


Fig. 2. Construction of four executions. In the figure, each execution is represented by four nodes and six edges. Each node corresponds to a party that takes (I, r) as input and internal randomness. Each edge labeled ij denotes the transcript between parties P_i and P_j, for $0 \leq i < j \leq 3$. In the first three executions, all parties honestly follow the protocol. In the fourth execution $\texttt{exec}_3$, party P_1 (highlighted in red) is malicious and behaves inconsistently across different parties. As proved in the first claim of the proof, the transcript between P_2 and P_1 in $\texttt{exec}_3$ is identical to that in $\texttt{exec}_2$, shown in blue; the transcript between P_3 and P_1 in $\texttt{exec}_3$ is identical to that in $\texttt{exec}_1$, shown in red. The transcript between P_2 and P_3 is identical across all four executions and is shown in green. Finally, the transcripts between P_0 and each party in $\{P_1, P_2, P_3\}$ are identical across all executions and are shown in black. Back to Lemma 3.