New Techniques for Information-Theoretic Asynchronous MPC with Abort

Xiaoyu Ji[1], Yifan Song[2]

[1] jixy23@mails.tsinghua.edu.cn, Tsinghua University
[2] yfsong@mail.tsinghua.edu.cn, Tsinghua University and Shanghai Qi Zhi Institute

Abstract. We study the communication complexity of information-theoretic asynchronous multiparty computation (AMPC) with optimal resilience $n = 3t + 1$ and malicious security. In this setting, the only known result with linear communication per gate is due to Goyal, Liu-Zhang, and Song [CRYPTO '24]. However, their construction incurs a large communication overhead of $\Omega(n^{14})$ elements that is independent of the circuit size, rendering their result only of theoretical interest. By additionally assuming a random oracle, Bandarupalli et al. [CCS '25] reduce the communication overhead to $\mathcal{O}(n^3)$ while maintaining the linear communication per gate, at the cost of only achieving malicious security with fairness.

In this work, we remove the random oracle assumption and design an information-theoretic AMPC protocol that achieves malicious security with abort. The communication complexity of our construction is $\mathcal{O}(|C|n + Dn^2 + n^3)$ field elements for an arithmetic circuit of size $|C|$ and depth D, assuming a functionality for Agreement on Common Set (ACS).

Our main technical contribution is a novel verification mechanism with the following guarantee: whenever verification succeeds, there exists a subset of at least $t+1$ honest parties whose local computations are mutually consistent, and the final output is correct with respect to their computation. In contrast to prior approaches that require all honest parties to hold consistent states and execute identical computations before verification, our mechanism tolerates inconsistencies among honest parties while still ensuring that the verified computation is correct for at least one such honest subset, if the verification succeeds.

1 Introduction

Secure multi-party computation (MPC) [Yao82, GMW87, BGW88, CCD88, RB89] allows a set of n parties to jointly evaluate a function over their private inputs while revealing nothing beyond the prescribed output, assuming that at most t parties may be corrupted by an adversary.

In the literature, most MPC protocols are designed assuming a synchronous network, where a global synchronous clock is available to all participants and every message is guaranteed to be delivered within a bounded amount of time. This feature simplifies the protocol construction and parties can rely on the strong timing guarantees to coordinate protocol steps. In the real world, however, communication is asynchronous: messages may be arbitrarily delayed due to unstable network connections. The nature of asynchrony introduces a fundamental challenge in the protocol construction: an honest party cannot distinguish between slow but honest participants and faulty participants. As a result, most of MPC protocols in the synchronous setting are no longer secure when running in the asynchronous network. Even for tasks that do not involve secrets, such as Byzantine agreements, asynchronous protocols are much more difficult to construct than the synchronous counterparts. In addition, the optimal resilience also differs between synchronous and asynchronous settings. Specifically, in the information-theoretic setting, statistical security can be achieved in the synchronous model when $t < n/2$ assuming broadcast [RB89]. In contrast, in the asynchronous model, statistical security requires $t < n/3$ [BKR94, ADS20].

Over the past decades, a long line of work [BKR94, PCR10, CP17, CP23, AAPP24, GLZS24] has focused on reducing the communication complexity of information-theoretic AMPC with guaranteed output delivery (GOD). In particular, [GLZS24] gives the first construction with amortized linear communication per gate. Concretely, for an arithmetic circuit of size $|C|$ and depth D, the communication complexity of their

construction is $\mathcal{O}(|C|n + Dn^2 + n^{14})$ field elements. However, the large communication overhead of $\mathcal{O}(n^{14})$ elements renders their result a pure theoretic interest, making their construction impractical even for a small number of parties.. While a recent work [JS26] improves [GLZS24] by reducing the communication overhead to $\mathcal{O}(n^5)$, it is still the main bottleneck of using AMPC in practice, especially for applications involving many participants.

To achieve a better concrete efficiency, one direction is to consider a weaker notion of *malicious security with abort*. In this setting, the protocol may abort prematurely but the secrecy of honest parties' inputs and outputs remains. Malicious security with abort has been extensively studied in the synchronous setting. In particular, the best-known information-theoretic synchronous protocols [GS20, GLO$^+$21] with optimal resilience can even achieve $\mathcal{O}(Cn + n^2)$ elements of communication for security with abort, compared to the best-known result with $\mathcal{O}(Cn + n^6)$ [GSZ20] for GOD. On the other hand, there has been very little study of asynchronous MPC for malicious security with abort in the literature. One exception is the recent work [BJK$^+$25], which constructs a malicious AMPC with abort[3] and achieves $\mathcal{O}(|C|n + Dn^2 + n^3)$ elements of communication assuming a random oracle, demonstrating the potential of this direction to address the issue of concrete efficiency of AMPC. We note that this $\mathcal{O}(n^3)$ communication overhead is nearly optimal: intuitively, in case each party needs to perform a broadcast, the communication overhead is at least $\mathcal{O}(n^3)$ since the lower bound of broadcast in the asynchronous setting is $\mathcal{O}(n^2)$ [DR85].

This leads to our following question:

Can we construct an information-theoretic AMPC protocol that achieves malicious security with abort in the optimal resilience setting, with linear communication and $\mathcal{O}(n^3)$ overhead?

1.1 Our Contributions

In this work, we answer this question affirmatively. Concretely, we obtain the following result.

Theorem 1. *Let $n = 3t + 1$, κ denote the security parameter, $\mathbb{F}$ be a finite field of size $2^{\Omega(\kappa)}$, and C be an arithmetic circuit over $\mathbb{F}$ of size $|C|$ and depth D. Let $\mathcal{F}_{\mathsf{acs}}$ be a functionality for* Agreement on Common Set*. There is a fully malicious asynchronous MPC protocol in the $\mathcal{F}_{\mathsf{acs}}$-hybrid model that securely computes C against at most t corrupted parties with security-with-abort. The achieved communication complexity is $\mathcal{O}(|C|n + Dn^2 + n^3)$ field elements plus $\mathcal{O}(1)$ instances of $\mathcal{F}_{\mathsf{acs}}$.*

Main Technical Contributions. In prior constructions of maliciously secure asynchronous MPC [GLZS24, JS26], the verification procedures crucially relies on the primitive of asynchronous complete secret sharing (ACSS), which allows a dealer to distribute degree-t Shamir sharings to all parties and ensures that all honest parties' shares are consistent (in the sense that their shares lie on a valid degree-t polynomial). However, the communication overhead of ACSS also becomes the main bottleneck in the previous constructions. For example, the $\mathcal{O}(n^{14})$ communication overhead in [GLZS24] is due to $\mathcal{O}(n^2)$ instances of ACSS [JLS24], each incurs $\mathcal{O}(n^{12})$ communication overhead. The recent work [JS26] improves [JLS24] and obtain a variant of ACSS that directly generates random degree-t Shamir sharings to all parties with $\mathcal{O}(n^5)$ communication overhead, and uses it to construct an AMPC protocol with the same communication overhead.

Our construction deviates from [GLZS24, JS26] and introduces a new verification mechanism *without* relying on ACSS. Very informally, we guarantee that whenever the verification succeeds, there exists a subset of at least $t+1$ honest parties whose local computations are mutually consistent, and the final output is correct with respect to their computation. Our mechanism thus tolerates inconsistency among honest parties' shares and intermediate states, while still providing a well-defined correctness guarantee. This new perspective enables a communication-efficient verification protocol in the security-with-abort setting.

[3] Their construction actually achieves fairness, which ensures that either all parties receive the function output or no party obtain the output.

2 Technical Overviews

In this section, we provide a high-level overview of our technique. We use n to denote the number of parties and t to denote the number of corrupted parties. We assume that $n = 3t + 1$. We write $[s]_t$ for a degree-t Shamir sharing of a secret s. The security parameter is denoted by κ, and we require the field size to be $2^{\Omega(\kappa)}$. Our goal is to construct an information-theoretic maliciously secure asynchronous MPC protocol in the $\mathcal{F}_{\mathsf{acs}}$-hybrid model with linear communication per gate and $\mathcal{O}(n^3)$ communication overhead in terms of field elements.

We begin in Section 2.1 with a variant of the AMPC in [JS26] that achieves only semi-honest security with send omission, where corrupted parties always follow the protocol but may drop their messages sent to honest parties, and the adversary can arbitrarily delay messages between honest parties. In Section 2.2, we identify the main obstacles that arise when extending the above protocol to the malicious setting. Sections 2.3 and 2.4 present our main technical contributions. Finally, in Section 2.5, we discuss some issues in the input/output phase and give our solutions.

2.1 Starting Point: Semi-Honest Variant of AMPC in [JS26]

The construction in [JS26] follows from the standard paradigm of constructing AMPC: (1) In the offline phase, all parties prepare a sufficient number of random Beaver triples, each shared using degree-t Shamir sharings. (2) Then in the online phase, all parties follow the standard Beaver triple technique to evaluate the circuit gate by gate. Notably, the online computation only requires public reconstruction of degree-t Shamir sharings, which can be performed with amortized linear communication per gate. Thus, we only focus on the preparation of random Beaver triples below.

We first introduce the semi-honest version of ACSS in [JS26], which follows from the standard technique of using bivariate polynomials. We refer to this protocol as $\Pi_{\mathtt{Sh}}$.

Distributing Degree-t Shamir sharings. The following protocol $\Pi_{\mathtt{Sh}}$ allows a dealer D to distribute $\mathcal{O}(n)$ degree-t Shamir sharings with $\mathcal{O}(n^2)$ communication. Thus, the communication per sharing is linear. Suppose D wishes to distribute $t + 1$ degree-t Shamir sharings $[s_0]_t, \ldots, [s_t]_t$. All parties do the following steps.

1. D samples a random degree-$(t, 2t)$ bivariate polynomial $F(x, y)$ such that $F(x, -\ell)$ is the degree-t polynomial associated with $[s_\ell]_t$ for all $\ell \in [0, t]$. Define $F(x, i), F(i, y)$ as each P_i's row and column polynomials.
2. **Distributing Columns.** D sends $F(i, y)$ to each P_i. Parties then execute a reliable agreement protocol to guarantee that at least $t+1$ honest parties have received their columns, and then terminate the sharing phase.
3. **Reconstructing Rows.** Each P_i sends the value $F(i, j)$ to party P_j. P_j collects values from at least $t + 1$ parties, which suffice to interpolate the degree-t row polynomial $F(x, j)$.
4. **Reconstructing Columns.** After reconstructing $F(x, i)$, each P_i sends $F(j, i)$ to P_j. P_j collects values from at least $2t + 1$ parties, which suffice to interpolate the degree-$2t$ column polynomial $F(j, y)$. This allows P_j to compute $F(j, -\ell)$ as his share of $[s_\ell]_t$ for all $\ell \in [0, t]$.

Preparation of Random Beaver Triples. Recall that a random Beaver triple is in the form of $([a]_t, [b]_t, [c]_t)$, where a, b are random field elements, and $c = a \cdot b$. In [JS26], the authors propose an efficient approach for preparing random Beaver triples with linear communication and $\mathcal{O}(n^3)$ overhead by combining the party virtualization technique [Bra87] and the triple extraction technique [CP17]. We give a brief review as follows.

Triple Extraction from [CP17]: Suppose the parties jointly hold N random Beaver triples $\{([a_i]_t, [b_i]_t, [c_i]_t)\}_{i=1}^N$, among which at most $T \leq (N - 1)/2$ are known to the adversary. The technique of [CP17] shows how to extract fresh random Beaver triples from this pool.

Let $L = \frac{N-1}{2}$. The protocol uses the N input triples to define three shared polynomials f, g, h such that $h = f \cdot g$, where f and g have degree L and h has degree $2L$. Then, for any evaluation point α, $(f(\alpha), g(\alpha), h(\alpha))$ forms a valid Beaver triple.

To be more concrete, the polynomial f and g are defined by $f(i) = a_i, g(i) = b_i$ for all $i \in \{1, \ldots, L+1\}$. All parties can locally compute shared polynomials $[f(\cdot)]_t, [g(\cdot)]_t$ from $\{([a_i]_t, [b_i]_t)\}_{i=1}^{L+1}$. As for the polynomial h, all parties set $[h(i)]_t = [c_i]_t$ for all $i \in \{1, \ldots, L+1\}$, and then compute $[h(i)]_t = [f(i) \cdot g(i)]_t$ for all $i \in \{L+2, \ldots, N\}$ by using the i-th random Beaver triple $([a_i]_t, [b_i]_t, [c_i]_t)$. In this way, all parties can locally compute the shared polynomial $[h(\cdot)]_t$ as well.

Since f and g are determined by $L+1$ points and the adversary learns at most T random Beaver triples, the parties can extract at most $L+1-T$ fresh random Beaver triples by evaluating the polynomials at another $L+1-T$ points (different from $1, \ldots, N$). The achieved amortized communication per fresh random Beaver triple is $\mathcal{O}\left(\frac{Nn}{N-2T+1}\right)$ field elements. In [CP17], this yields a quadratic communication per triple since their construction only guarantees $N = 2t+1$ and $T = t$.

Techniques in [JS26]: Building on [CP17], the authors of [JS26] show how to increase the ratio of unknown triples to the adversary so that $T/N < 1/4$, which suffices to achieve linear communication. Their main technique is party virtualization [Bra87].

Specifically, each ordered pair of parties (P_i, P_j) simulates a virtual party $V_{i,j}$ whose view is additively shared between P_i and P_j. Note that as long as one of P_i or P_j is honest, the view of $V_{i,j}$ remains hidden from the adversary. This results in $N' = n^2$ virtual parties, among which at most $T' = t^2$ are corrupted. Using these virtual parties, the protocol generates a pool of random Beaver triples as follows.

- **Stage 1**: Each virtual party $V_{i,j}$ prepares a random Beaver triple and invokes $\Pi_{\mathtt{Sh}}$ to distribute it to all parties. In [JS26], this step can be emulated by (P_i, P_j) with the help of all other parties with $O(n)$ communication.
- **Stage 2**: The parties agree on a subset of N virtual parties that successfully completed the distribution, and apply the extraction procedure of [CP17] to obtain $\Theta(N)$ fresh random Beaver triples.

Regarding the achievable value of N, note that a virtual party $V_{i,j}$ may terminate only when both P_i and P_j are honest. Since at least $2t+1$ parties are honest, N can be set to be $\frac{4}{9}N'$. Consequently, the protocol yields $N = \frac{4}{9}N'$ triples, among which at most $T = T' \leq \frac{1}{9}N' = \frac{1}{4}N$ are known to the adversary.

For simplicity, we defer the details and some optimizations for the whole protocol to Section 4. In summary, assuming a functionality $\mathcal{F}_{\mathtt{acs}}$ that helps the parties to agree on a common set of N virtual parties in **Stage 2**, the construction in [JS26] realizes a triple generation protocol with linear communication per triple and $\mathcal{O}(n^3)$ overhead. Together with the online phase, this yields an AMPC with linear communication and $\mathcal{O}(n^3)$ overhead that achieves semi-honest security with send omission.

2.2 Towards Malicious Security

In [JS26], the authors show that the above template can achieve malicious security with guaranteed output delivery when replace $\Pi_{\mathtt{Sh}}$ by a maliciously secure ACSS. To this end, the authors improve the ACSS construction in [JLS24] and obtain a variant of ACSS that directly generates random degree-t Shamir sharings to all parties with $\mathcal{O}(n^5)$ communication overhead, which leads to their final AMPC protocol with linear communication and $\mathcal{O}(n^5)$ overhead.

Recall that our focus is the malicious security with abort. To maintain the $\mathcal{O}(n^3)$ overhead as the semi-honest variant, a natural idea is to lift the security of $\Pi_{\mathtt{Sh}}$ to malicious security with abort while maintaining the linear communication and $\mathcal{O}(n^2)$ overhead. Unfortunately, we do not know how this can be achieved. When running $\Pi_{\mathtt{Sh}}$ in the fully malicious setting, a corrupted party P_i may send a wrong common point $F(i, j)$ (or $F(j, i)$) to P_j. As a result, honest parties may end up with incorrect shares *even if* the dealer D is honest.

One may attempt to adapt the synchronous approach to verify the shares of honest parties: first invoke the semi-honest secure $\Pi_{\mathtt{Sh}}$ to distribute shares, and then let all parties publicly reconstruct a random linear combination of their shares so that each party can locally check the correctness of his shares. Unfortunately, this approach fails in the asynchronous setting due to the adversary's ability to delay honest parties' messages.

In the asynchronous model, if the random challenge r is released *before* all honest parties' shares are received, then an adversary may decide the errors added on honest parties' shares based on the random

challenge r so that honest parties still receive incorrect shares but the verification passes. However, to ensure liveness, parties can only wait for responses from $2t+1$ parties. Thus, in the worst case, the adversary can selectively delay the delivery of up to t honest parties' shares while still allowing the protocol to proceed and generate the challenge. As a result, only $t+1$ honest parties will obtain their shares before the random challenge r is generated, while the rest of honest parties may only receive their shares after r is released. This issue essentially nullifies all verifications based on random challenges.

Our attempt is to directly use $\Pi_{\mathtt{Sh}}$ without ensuring consistency of shares among honest parties.

Problems of Using $\Pi_{\mathtt{Sh}}$ Directly. However, this attempt brings new difficulties in achieving malicious security.

Attack on the Public Reconstruction Procedure: Since honest parties may hold inconsistent shares, they can no longer rely on the error-correction property of degree-t Shamir sharings to guarantee a unique public reconstruction. This issue becomes particularly harmful in the online phase, where parties must perform public reconstruction when using Beaver triples to evaluate multiplication gates. If honest parties proceed with different reconstruction results, the authors in [GLS19] show a *double dipping* attack, which allows the adversary to extract information about honest parties' private inputs. We discuss how to address this issue in Section 2.4 later.

Attack on the Triple Verification: The triple generation protocol in [JS26] does not naturally achieve malicious security. To verify the correctness of all random Beaver triples, the idea in [JS26] is to reuse the shared polynomials f, g, h and verify the multiplication relation on a random evaluation point. Similar to the attempt of verifying honest parties' shares, this verification no longer works in our attempt either. We illustrate a concrete attack below in which verification succeeds at a random point r even though $h \not\equiv f \cdot g$. Without loss of generality, suppose that the first $t+1$ honest parties receive their shares from $\Pi_{\mathtt{Sh}}$, while the adversary delays the remaining honest parties' shares.

1. All parties generate a random challenge r and reconstruct $[f(r)]_t, [g(r)]_t, [h(r)]_t$.
2. The adversary collects the shares of $[f(r)]_t, [g(r)]_t, [h(r)]_t$ from these $t+1$ honest parties and interpolates the corresponding secrets.
3. The adversary computes $e = f(r) \cdot g(r) - h(r)$ and defines a degree-t sharing $[e]_t$ whose secret is e and whose shares for the first t honest parties are zero.
4. The adversary adds the remaining share of $[e]_t$ to the last honest party's share of $[h(r)]_t$, resulting in a sharing $[h'(r)]_t = [h(r)]_t + [e]_t = [f(r) \cdot g(r)]_t$. Then the adversary inserts errors on the last honest party's shares so that he would obtain his share of $[h'(r)]_t$.
5. During verification, the adversary delays messages from honest parties except the first t honest parties and the last honest party. Then the adversary uses corrupted parties' shares of $[f(r)]_t, [g(r)]_t, [h'(r)]_t$. As a result, the first t honest parties and the last honest party will reconstruct $f(r), g(r)$, and $h'(r)$, and therefore incorrectly accept the triples.

Remark 1. A careful reader may notice that in the above attack, with overwhelming probability, the $(t+1)$-th honest party will observe inconsistency with his share of $[h(r)]_t$ and the one $[h'(r)]_t$ reconstructed by all parties. Then this honest party would abort and notify other parties. However, the notification may be delayed by the adversary. As a result, even if an honest party observes incorrect computation, other honest parties may continue the computation.

Our Idea. Instead of attempting to enforce global consistency among all honest parties, which is fundamentally difficult in the asynchronous setting, we design verification protocols that certify the correctness of computation with respect to a subset of $t+1$ honest parties.

Concretely, successful verification guarantees that there exists a subset of $t+1$ honest parties such that all triples produced in the offline phase are correct with respect to their shares. In the online phase, there also exists a subset of $t+1$ honest parties such that the online evaluation proceeds correctly using these triples, with all public reconstruction results consistent with the shares held by this subset.

Crucially, the protocol neither identifies such a subset nor allows parties to distinguish which subset satisfies this property. Rather, verification only establishes the existence of such a subset. The subset of honest parties during the offline and online verification can also be different. Moreover, if multiple subsets of $t+1$ honest parties satisfy the verification conditions, then the secrets interpolated from all such subsets must be identical.

Additive Errors. Before moving on to the details of our construction, we clarify that although $\Pi_{\mathtt{Sh}}$ allows the adversary to insert errors on honest parties' shares *after* the sharing phase, when the secret is unknown to the adversary (e.g., a random sharing distributed by an honest dealer), any modification can only introduce *additive errors*. Concretely, if an honest party reconstructs a row or column polynomial, a corrupted party may alter some of the shares it receives from this party. Since the reconstruction is a linear combination of these shares, the effect of the adversary's modification manifests as an additive error on the resulting polynomial, and consequently, on the party's output shares.

2.3 Offline Verification

Our goal is to guarantee that once all parties pass the verification, there exists $t+1$ honest parties such that the triples interpolated by their shares are valid. Then we define the whole sharing of the verified triples based on these $t+1$ honest parties' shares, and for the rest of the honest parties who may hold inconsistent shares, the adversary can only insert additive errors at any time later. Our verification will ensure that if multiple subsets of $t+1$ honest parties satisfy the verification conditions, the triples interpolated from all such subsets are identical.

The Key Observation. Revisiting the triple verification procedure, during public reconstruction parties obtain $2t+1$ shares of $\{[f(r)]_t, [g(r)]_t, [h(r)]_t\}$, among which at least $t+1$ shares come from honest parties. If these $t+1$ honest parties receive their triple shares *before* the random challenge r is generated, then this verification is sound since the polynomials f, g, h determined by these $t+1$ honest parties' shares are already fixed before r is generated.

For the attack on verification in Section 2.2, it succeeds because the adversary can insert additive errors into the shares of an honest party who receives his triple shares *after* the random challenge r is generated. Once an honest party has received his shares, the adversary can no longer modify them by introducing new additive errors.

This observation suggests that if the adversary aims to prevent the existence of any subset of $t+1$ honest parties whose shares interpolate to correct triples, it must inject additive errors into the shares of a new honest party who has not yet received his triple shares at the time the random challenge is generated. Thus, our idea is to *repeat* the above verification $t+1$ times sequentially, each with a fresh random challenge. Then in each repetition, the adversary has to deliver the shares to a new honest party (or otherwise $f(r), g(r), h(r)$ would not satisfy the multiplication relation with overwhelming probability and all honest parties would abort). Since there are $2t+1$ honest parties and at least $t+1$ honest parties have received their shares before the first repetition, the adversary can carry out this attack for at most t repetitions. After all honest parties have received their shares, the attack no longer applies.

Ignoring the cost of generating random coins, each verification requires public reconstruction of the sharings $\{[f(r)]_t, [g(r)]_t, [h(r)]_t\}$, which costs $\mathcal{O}(n^2)$ field elements. Repeating this procedure for $t+1 = \mathcal{O}(n)$ rounds results in a total communication complexity of $\mathcal{O}(n^3)$ field elements.

A Subtle Issue. For a computationally unbounded adversary, after learning the challenge r in a verification, the adversary can check whether there exists a subset of $t+1$ honest parties whose shares have been delivered, and the secrets $\{f(r), g(r), h(r)\}$ interpolated from their shares satisfy the multiplication relation. If such a subset exists, the adversary no longer needs to insert errors into a new honest party's shares, but can ensure that $\{f(r), g(r), h(r)\}$ are reconstructed using shares from these $t+1$ honest parties and still pass the verification. When $|\mathbb{F}| = 2^{\Omega(\kappa)}$, the probability of this event in a single verification is at most $\binom{2t+1}{t+1}/|\mathbb{F}| \approx 2^{n-\kappa}$.

To make this probability negligible in κ, each verification now has to test multiple independent random challenges instead of a single one. Concretely, parties use $\lceil \frac{n+\kappa}{\kappa} \rceil$ random challenges in each verification. For any fixed subset of $t+1$ honest parties, the probability that accidental consistency occurs is then at most $2^{-(n+\kappa)}$. Taking a union bound over all such subsets yields an overall soundness error of at most $2^{-\kappa}$.

For communication complexity, the above modification introduces additional $\mathcal{O}(n^4)$-bit communication. In the following, we present an optimization that eliminates this overhead.

Initial Attempt. We first consider the following verification procedure, which is sufficient to achieve our goal for communication complexity. We also no longer need to generate public random coins.

- **Proposing Random Challenges.** Upon obtaining their shares of triples, each party P_i reliably broadcasts $\lceil \frac{n+\kappa}{\kappa} \rceil$ distinct random challenges.
- **Verification.** Upon receiving a challenge r from P_i, each party sends its shares of $\{[f(r)]_t, [g(r)]_t, [h(r)]_t\}$ to P_i. Once P_i receives $2t+1$ shares (including its own), it checks whether these shares lie on unique degree-t polynomials and the secrets satisfy the correct multiplication relation. If not, P_i sends a failure symbol $(\perp, P_i)$ to all parties and aborts. Otherwise, P_i sends (OK, P_i) to all parties.
- **Termination.** Upon receiving (OK, P_i) from $2t+1$ distinct parties P_i, a party terminates with `Success`. If a failure symbol $(\perp, P_i)$ is received first, the party aborts.

However, this construction is insecure due to the fact that all verifications are performed in parallel rather than sequentially. As a result, the adversary can first learn the random challenges proposed by multiple honest parties and then adaptively inject additive errors into honest parties' shares so as to satisfy all verifications simultaneously. Then different honest verifiers may rely on different subsets of shares, and the protocol may terminate successfully even though no single set of $t+1$ honest parties holds consistent triple shares.

To rule out this attack, we keep the same local verification procedure but add a structured acceptance process on top of it. Each successful verification is assigned an explicit acceptance level, and advancement to higher levels is permitted only after collecting sufficiently many acceptances at the previous level. As we show next, this iterative structure restores the existential guarantee that was missing in the above attempt.

Iterative Acceptance. Each party executes the same verification procedure as before (excluding the termination condition). In the following, whenever a party fails in the verification or receives a failure symbol $\perp$, it forwards $\perp$ to all parties and aborts. Each party P_i initializes a local counter $\mathsf{cnt}_i \leftarrow 0$.

1. **Initialization.** Whenever P_i successfully completes a verification for all its own $\lceil \frac{n+\kappa}{\kappa} \rceil$ distinct challenge r, it reconstructs and locally records the whole degree-t sharings $\{[f(r)]_t, [g(r)]_t, [h(r)]_t\}$. It then sends $(\mathsf{OK}, P_i, \mathsf{cnt}_i)$ to all parties.
2. **Iterative Acceptance.** Upon receiving $(\mathsf{OK}, P_*, \mathsf{cnt}_*)$ together with shares of $\{[f(r)]_t, [g(r)]_t, [h(r)]_t\}$ from $2t+1$ distinct parties P_* that all report the same counter value cnt_* as cnt_i, party P_i checks whether these shares are consistent with the degree-t sharings fixed during its own initialization. If so, P_i increments its counter $\mathsf{cnt}_i \leftarrow \mathsf{cnt}_i + 1$ and sends $(\mathsf{OK}, P_i, \mathsf{cnt}_i)$ to all parties. This step may repeat multiple times and does not rely on synchrony.
3. **Termination.** When P_i reaches $\mathsf{cnt}_i = t+2$, it terminates with `Success`.

Correctness and Termination. We show that if an honest party P_i terminates with `Success`, then there exists an honest party P_k whose verification relies on at least $t+1$ honest parties who received their shares before P_k proposes the random challenge.

Suppose an honest party P_i reaches $\mathsf{cnt}_i = t+2$, consider the set $\mathcal{S}$ of $2t+1$ parties P_* whose messages $(\mathsf{OK}, P_*, t+1)$ caused P_i to increment its counter from $t+1$ to $t+2$. Let P_{k_0} be the honest party in $\mathcal{S}$ that is the *last* (according to the adversarially controlled delivery order) to receive his shares. Since P_{k_0} sent $(\mathsf{OK}, P_{k_0}, t+1)$, then there exists a set $\mathcal{S}'$ of $2t+1$ parties whose messages (OK, P_*, t) caused this increment at P_{k_0}.

If at least $t+1$ honest parties in $\mathcal{S}'$ received their shares before P_{k_0}, then P_{k_0} is the target party P_k. Otherwise, there exists an honest party P_{k_1} in $\mathcal{S}'$ that receives its shares after P_{k_0}. In this case, we can apply the same argument recursively to P_{k_1}. In the ℓth iteration, either the honest party P_{k_ℓ} is the target

party P_k or there exists another honest party $P_{k_{\ell+1}}$ that receives shares after P_{k_ℓ}. At each step, the counter value strictly decreases by one, and there are at most t honest parties that will receive their shares after P_{k_0}. Therefore, after at most $t+1$ iterations, the process must reach an honest party whose acceptance relies on at least $t+1$ honest parties that received their shares earlier.

Termination follows since all honest parties eventually respond to each counter value. For the communication complexity, it costs $\mathcal{O}(n^3)$ field elements plus $\mathcal{O}(n)$ instances of reliably broadcast, and the input size for each broadcast protocol is $n\left\lceil\frac{n+\kappa}{\kappa}\right\rceil$ field elements. Refer to section 4.4 for detailed construction.

Summary. The above procedure establishes the following guarantee for offline triple verification: whenever the verification succeeds, there exists a subset of at least $t+1$ honest parties whose shares define polynomials f, g, h satisfying the multiplication relation $f \cdot g = h$. Consequently, the triples are well-defined with respect to this subset, while the remaining honest parties' shares can be interpreted as evaluations of these polynomials with additive errors that may be introduced by the adversary.

Importantly, we do not require global consistency among all honest parties. Correctness is certified solely through the existence of such a consistent subset. In the subsequent online phase, all parties rely on these well-defined triples for circuit evaluation. Since global consistency is not enforced, parties perform another verification in the online phase to ensure that the triples used in computation are indeed correct, which will be described later.

Remark 2. We stress that the above verification technique can not be used in the semi-honest secure $\Pi_{\mathtt{Sh}}$ to achieve a maliciously secure ACSS. In $\Pi_{\mathtt{Sh}}$, the dealer's secret is not uniquely fixed at the end of the sharing phase: due to possible additive errors inserted by the adversary, different subsets of $t+1$ honest parties may interpolate to different secrets. Our verification mechanism only guarantees the *existence* of a subset of $t+1$ honest parties whose views are mutually consistent.

In contrast, during offline triple verification, we do not aim to fix the specific values of the polynomials f, g, h. Instead, we only verify that there exists a subset of $t+1$ honest parties whose shares can interpolate valid polynomials f, g, h such that $f \cdot g = h$. The correctness guarantee is therefore relational rather than value-fixing. This distinction is crucial: while existential consistency suffices for verifying multiplication relations, it is insufficient for achieving maliciously secure secret sharing, where a unique dealer secret must be determined.

Omitted Difficulty in the Security Proof. Let f, g, h be the polynomials that are used to extract the random triples during the offline phase, and we use L to denote the degree of f, g. During the generation of f, g, h in the semi-honest secure construction, for $\ell \in [L+2, 2L+1]$, the adversary can choose values $\delta_\ell^{(f)}, \delta_\ell^{(g)}, \delta_\ell^{(h)}$ and insert an error $\delta_\ell^{(f)} \cdot f(\ell) + \delta_\ell^{(g)} \cdot g(\ell) + \delta_\ell^{(h)}$ to $h(\ell)$. Then, during the simulation of the offline verification, before learning the final verification result, it does not know whether $f \cdot g = h$ holds. But it needs to respond to the adversary's chosen challenge r, which contains the information a linear combination of $\{\delta_\ell^{(f)} \cdot f(\ell) + \delta_\ell^{(g)} \cdot g(\ell) + \delta_\ell^{(h)}\}_{\ell \in [L+2, 2L+1]}$. In the security proof, we carefully analyze that this linear combination would not reveal any information about the output triples or give the adversary a noticeable advantage to let honest parties' verification succeed, but $f \cdot g \neq h$.

2.4 Online Evaluation and Verification Procedure

Recall that when parties use a Beaver triple $\{[a]_t, [b]_t, [c]_t\}$ to evaluate a multiplication gate with input wires $([x]_t, [y]_t)$, they only publicly reconstruct the values $x + a$ and $y + b$. Our goal in the online phase is to guarantee that, once all parties pass the verification, there exist $t+1$ honest parties such that their shares of triples interpolate valid triples *fixed by the offline verification*, and their shares of $[x+a]_t$ and $[y+b]_t$ interpolate the public reconstruction values $x + a$ and $y + b$ used throughout the computation.

To improve the efficiency, we first defer the verification until the final output reconstruction and do verification for all circuit layers in parallel. As discussed in Section 2.2, directly using the semi-honest secure public reconstruction is not sufficient in the malicious setting, since there is no guarantee that all honest parties use the same public reconstruction values $x+a$ and $y+b$ during the computation. This may lead to the *double dipping* attack [GLS19], which compromises security. This issue also appears in prior work [BJK+25],

where it is shown that as long as all honest parties use the same reconstruction values $x + a$ and $y + b$ for computation, privacy is preserved even if the adversary injects errors into these values to break the correctness.

Therefore, we first focus on modifying the public reconstruction procedure so that all parties either terminate with the same reconstruction results or abort. Once this is ensured, the remaining verification can be safely deferred to the final stage.

Guaranteeing a Unique Public Reconstruction. To illustrate the idea, consider first the reconstruction of a single value x. Each party sends its reconstructed value to all parties and accepts x if it receives the same value from $2t+1$ parties. Then, for any two honest parties that terminate, there must exist at least one honest party whose message is received by both, implying that they accept the same reconstruction result.

When multiple reconstruction values need to be verified, we reduce the communication complexity by checking a random linear combination of these values. To ensure soundness, each party uses fresh pairwise randomness: party P_i samples a random value for each P_j, sends the randomness together with the corresponding linear combination of its reconstruction results, and P_j independently verifies consistency using its own reconstruction results. Since the pairwise randomness between two honest parties is unknown to the adversary at the time of reconstruction, the adversary cannot adaptively forge inconsistent reconstruction values that pass all checks, which guarantees soundness. Refer to Section 4.2 for detailed construction.

Online Verification. Now all parties can first securely evaluate the circuit using the above technique, during which all public reconstruction values $x + a, y + b$ are fixed. To achieve our verification goal, it is sufficient to check a random linear combination of all sharings $\{[x + a]_t, [y + b]_t\}$ generated in the online phase and verify that the used Beaver triples are valid.

For the verification of the random linear combination of all sharings $\{[x + a]_t, [y + b]_t\}$, although the adversary can still inject additive errors into some honest parties' shares, the procedure is analogous to triple verification in the offline phase: If a subset of $t + 1$ honest parties received their shares before the random challenge is generated, then either the public reconstruction results are correct with respect to their shares, or the verification fails for this subset of parties with overwhelming probability. Thus, if the adversary aims to prevent the existence of any subset of $t + 1$ honest parties whose shares are correct with respect to the publication results, it must inject additive errors into the shares of a new honest party and deliver the shares to this party. We can therefore apply the same technique to perform online verification. In addition, to ensure that the online computation and the Beaver triples are correct with respect to the *same* subset of $t + 1$ parties. All parties will verify the random Beaver triples again in parallel with the verification of the public reconstruction.

With more details, each party proposes a random challenge r and verifies a linear combination of all $\{[x + a]_t, [y + b]_t\}$ in the online phase based on r and the relation $f(r), g(r), h(r)$, where f, g, h are the polynomials generated in the offline phase. They follow the same iterative acceptance to decide the verification results. Once verification succeeds, we know that there exist $t+1$ honest parties whose computation is correct, meaning their output shares are all correct. For the remaining honest parties, the adversary can only insert additive errors into their output shares. In this way, the secret of the output is well-defined by the $t+1$ honest parties with correct shares, and all parties can securely reconstruct the output later. Refer to Section 5.1 for the detailed construction.

Summary. Once the online verification succeeds, all parties learn that there exist $t+1$ honest parties whose shares can interpolate both correct triples $\{a, b, c\}$ and the intermediate values $\{x + a, y + b\}$. With correct triples and these consistent public reconstruction values, the secret $\{x, y\}$ of each circuit wire is also well defined. Thus, these $t + 1$ honest parties' computations are all correct throughout the whole computation, and the output is well-defined as the secret interpolated from these $t + 1$ honest parties. Note that if there exist multiple such subsets of honest parties that can pass the verification, their shares must interpolate the same secrets values.

A Subtle Issue. One subtle point is that, when using a triple $\{a, b, c\}$ to evaluate the circuit, the randomness of $\{a, b, c\}$ is consumed when the public reconstruction values $\{x + a, y + b\}$ are released. Consider the worst case where all inputs x, y are known to the adversary (e.g. circuit's inputs are all from corrupted parties), so

the adversary can learn the full triples $\{a, b, c\}$. During triple verification, although we allow the adversary to insert additive errors, these errors can now depend on the secrets.

As a result, the adversary can inject additive errors $\delta^{(a)}, \delta^{(b)}$ to a and b, and $\delta^{(b)} \cdot a + \delta^{(a)} \cdot b + \delta^{(a)} \cdot \delta^{(b)}$ to c, effectively shifting secrets of the triple from $\{a, b, c\}$ to $\{a', b', c'\}$. By construction, $\{a', b', c'\}$ is still a valid triple. However, if online verification only guarantees the existence of $t + 1$ honest parties interpolating a valid triple, this is insufficient: the valid triple $\{a', b', c'\}$ may differ from the triple $\{a, b, c\}$ used to compute $x + a$ and $y + b$. In that case, the adversary could arbitrarily adjust the secrets of all circuit wires, and the output would not be well-defined.

Therefore, parties must verify that the $t + 1$ honest parties not only interpolate a valid triple, but also interpolate the triples fixed in the offline phase. This ensures that all circuit wires x and y are fixed since all $x + a, y + b, a, b$ are fixed. To achieve this, we rely on the fact that there are sufficiently many random points on f, g, h unknown to the adversary. Concretely, let L denote the degree of f, g and $h = f \cdot g$. As shown in Section 2.1, given $T < N/4$, there are approximately $L + 1 - T \approx N/4$ random points on f, g, h unknown to the adversary. If we select only half of these points as output triples, then even after the adversary learns all $N/8$ evaluation points used, there remain $N/8$ unknown points, preventing the adversary from interpolating f, g, h completely. Since each party verifies a random point on f, g, h to check $f \cdot g = h$, the verification of triples remains sound, and learning the used triples does not compromise it.

2.5 Input and Output Phase

Input Phase. A subtle issue here is that using $\Pi_{\mathtt{Sh}}$ is insufficient for distributing inputs because the adversary can insert additive errors to bias an honest dealer's input. Moreover, $\Pi_{\mathtt{Sh}}$ does not fix a corrupted dealer's input secret after the sharing phase, so the simulator cannot extract a corrupted dealer's input during simulation.

To address this, we assign an unused Beaver triple $\{[a]_t, [b]_t, [c]_t\}$ from the offline phase to each party P_i's input x. All parties send their shares of this triple to P_i. Using any $t + 1$ received shares, P_i interpolates the secrets $\{a, b, c\}$ and checks that $a \cdot b = c$. If the check fails, P_i detects a deviation from the protocol by some corrupted party and aborts. Otherwise, P_i reliably broadcasts the value $x + a$, and each party can locally compute its share of x as $[x]_t := (x + a) - [a]_t$.

During the online verification phase, the assigned triple is also verified. If some honest parties successfully completes the online verification, then, as argued in Section 2.4, the secret a is fixed during the offline phase, so the input x is uniquely determined by the public value $x + a$. In particular, when P_i is honest, since the Input phase happens before the Evaluation phase, a correct triple $\{[a]_t, [b]_t, [c]_t\}$ must be the one fixed from the offline verification.

Output Phase. We assume all parties receive the same output. As shown in Section 2.4, once parties succeed in the online verification, the output is well-defined (with respective to some subset of $t + 1$ honest parties). Since $\Pi_{\mathtt{Sh}}$ is secure up to additive errors, the secret interpolated from any subset of $t + 1$ honest parties can only deviate from the correct secret by an error learnt by the adversary. Thus, it is safe to perform public reconstruction to obtain the circuit output without any secrecy issue. However, to ensure the correctness of the reconstruction, all parties will execute the online verification procedure again.

3 Preliminary

3.1 UC Secure Model

We consider protocols executed by a set $\mathcal{P}$ of n parties $P_1, \ldots, P_n$. Our security analysis is carried out in the Universal Composability (UC) framework introduced by Canetti [Can01], which follows the *real/ideal world paradigm* [Can00]. The parties communicate over a network of asynchronous, secure point-to-point channels (see [CR98] for details of the asynchronous network model). These channels guarantee *eventual delivery*: every message that is sent is eventually delivered. Message scheduling is controlled by the adversary, who may arbitrarily (but finitely) delay messages and deliver them out of order.

We assume a fully malicious adversary that can completely control the behavior of corrupted parties. The authors in [KMTZ13, Coh16, CFG+25] introduce the entity activation order in the ideal world and the *request-based delayed output* mechanism used in our functionality.

Functionality of AMPC with Abort. We modify the functionality of $\mathcal{F}_{\mathsf{AMPC}}$ defined in [CFG+25] to the security-with-abort setting. Refer to Appendix C for details.

3.2 Shamir Secret Sharing Scheme

In this work, we will use the standard Shamir Secret Sharing Scheme [Sha79]. Let n be the number of parties and $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2n$. Let $\alpha_0, \ldots, \alpha_n$ be $n+1$ distinct non-zero elements in $\mathbb{F}$.

A *degree-d* Shamir sharing of $x \in \mathbb{F}$ is a vector $(x_1, \ldots, x_n)$ which satisfies that there exists a polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most d such that $f(\alpha_0) = x$ and $f(\alpha_i) = x_i$ for $i \in \{1, \ldots, n\}$. Each party P_i holds a share x_i and the whole sharing is denoted by $[x]_d$. We recall the properties of the Shamir secret sharing scheme:

- Linear Homomorphism:
$$\forall\ [x]_d, [y]_d,\ [x+y]_d = [x]_d + [y]_d.$$
- Multiplying two degree-d sharings yields a degree-$2d$ sharing. The secret value of the new sharing is the product of the original two secrets.
$$\forall\ [x]_d, [y]_d,\ [x \cdot y]_{2d} = [x]_d \cdot [y]_d.$$

3.3 Building Blocks

Our construction utilizes the following building blocks. Refer to appendix A for the detailed definition.

- **Reliable Broadcast $\mathcal{F}_{\mathsf{rbc}}$.** It allows parties to agree on the value of a sender, which can be realized with $\mathcal{O}(L \cdot n + n^2 \log n)$ communication bits for broadcasting a message of size L bits [ADD+22].
- **Agreement on a Common Set.** Functionality $\mathcal{F}_{\mathsf{acs}}$ allows parties to agree on a set of at least $n - t$ parties that satisfy a certain property. Combining the techniques in [Che24, ADD+22, DWZ23] can get an information-theoretically secure instantiation of $\mathcal{F}_{\mathsf{acs}}$[4].

4 Triple Generation

In this section, we give the construction of our triple generation protocol $\Pi_{\mathtt{Triple}}$, which achieves amortized linear communication per triple plus $\mathcal{O}(n^3)$ additive overhead in terms of field elements. We first introduce the detailed construction of the semi-honest secure secret sharing protocol $\Pi_{\mathtt{Sh}}$ in Section 2.1, then give the consistent public reconstruction protocol $\Pi_{\mathtt{PubRec}}$ in Section 4.2, finally define $\mathcal{F}_{\mathsf{Triple}}$, and the detailed construction of $\Pi_{\mathtt{Triple}}$.

4.1 Secret Sharing Protocol

Let N be the number of degree-t Shamir secret sharings to be distributed, the communication complexity of $\Pi_{\mathtt{Sh}}$ is $\mathcal{O}(Nn + n^2)$ field elements. As we discussed in Section 2.3, during the execution of $\Pi_{\mathtt{Sh}}$, the malicious adversary can insert additive errors to honest parties' shares even *after* the sharing phase.

[4] In this work, we do not count the cost of $\mathcal{F}_{\mathsf{acs}}$ since our construction is built in $\mathcal{F}_{\mathsf{acs}}$-hybrid model.

Protocol $\Pi_{\mathtt{Sh}}$

Parameter: Let N be the number of degree-t Shamir sharings to be distributed, $N' = N/(t+1)$, and $\beta_0, \ldots, \beta_t, \alpha_1, \ldots, \alpha_n$ be distinct field elements.

Input: Dealer D takes N degree-t Shamir secret sharing $[s_1]_t, \ldots, [s_N]_t$ as inputs.

Distributing Phase

1: D divides $[s_1]_t, \ldots, [s_N]_t$ into N' groups, each of size $t+1$. Let $[s_0^{(\ell)}]_t, \ldots, [s_t^{(\ell)}]_t$ be the sharing in the ℓth group, D randomly samples a degree-$(t, 2t)$ bivariate polynomials $F_\ell(x, y)$ such that $F_\ell(x, \beta_k) = [s_k^{(\ell)}]_t$ for all $k \in [0, t]$.

2: **Distributing Column Polynomials.** For each $\ell \in [N']$ and $P_i \in \mathcal{P}$, D computes degree-$2t$ column polynomial $g_{\ell,i}(y) = F_\ell(\alpha_i, y)$ and sends it to P_i.

3: **Termination.** All parties jointly invoke an instance of $\mathcal{F}_{\mathsf{ra}}$. Upon receiving the degree-$2t$ column polynomials from the dealer D, all parties set their input as 1 to $\mathcal{F}_{\mathsf{ra}}$. Upon terminating $\mathcal{F}_{\mathsf{ra}}$ with 1, all parties terminate the distributing phase and proceed to the recovery phase.

Recovery Phase

4: **Reconstruction of Row Polynomials.** Upon receiving $\{g_{\ell,i}(y)\}_{\ell \in [N']}$ from D, each party P_i sends $\{g_{\ell,i}(\alpha_j)\}_{\ell \in [N']}$ to each party P_j. Upon receiving $\{g_{\ell,j}(\alpha_i)\}_{\ell \in [N']}$ from $t+1$ distinct P_j, P_i interpolates a degree-t polynomial $f_{\ell,i}(x)$ based on these $t+1$ distinct $g_{\ell,j}(\alpha_i)$ for all $\ell \in [N']$. Then P_i sends $f_{\ell,i}(\alpha_j)$ to each party P_j for all $\ell \in [N']$.

5: **Reconstruction of Column Polynomials.** Upon receiving $\{f_{\ell,j}(\alpha_i)\}_{\ell \in [N']}$ from $2t+1$ distinct P_j, P_i interpolates a degree-$2t$ polynomial $g_{\ell,i}(y)$ based on these $2t+1$ distinct $f_{\ell,j}(\alpha_i)$ for all $\ell \in [N']$. Then P_i computes $\{g_{\ell,i}(\beta_k)\}_{\ell \in [N'], k \in [0,t]}$ as his share of $[s_1]_t, \ldots, [s_N]_t$ and terminates.

4.2 Consistent Public Reconstruction.

As we discussed in Section 2.4, parties first invoke an efficient public reconstruction protocol [DN07] to reconstruct the secrets, then execute a pair-wise verification to determine the output. The communication complexity is $\mathcal{O}(Nn + n^2)$ field elements for reconstructing N values.

Protocol $\Pi_{\mathtt{PubRec}}$

Parameter: Let N denote the number of input sharings.

Input: All parties take their shares of $[x_1]_t, \ldots, [x_N]_t$ as inputs.

If one party's input share is a failure symbol $\perp$, then he will send $\perp$ to all parties. Parties that receive $\perp$ will forward it to all parties and terminate with `Fail`.

1: Divide $[x_1]_t, \ldots, [x_N]_t$ into $N/(2t+1)$ groups, each of size $2t+1$. For each group, P_i does the following.
   (1). Let $[s^{(0)}]_t, \ldots, [s^{(2t)}]_t$ denote the degree-t Shamir secret sharings in this group. We define the degree-$2t$ polynomial $f(X) = s^{(0)} + s^{(1)} \cdot X + \cdots + s^{(2t)} \cdot X^{2t}$. P_i sends his share of $[f(\alpha_j)]_t = [s^{(0)}]_t + [s^{(1)}]_t \cdot \alpha_j + \cdots + [s^{(2t)}]_t \cdot \alpha_j^{2t}$ to each P_j.
   (2). Upon receiving $2t+1$ shares of $[f(\alpha_i)]_t$, P_i interpolates the secret $f(\alpha_i)$ and sends it to all parties.
   (3). Upon receiving $f(\alpha_j)$ from $2t+1$ distinct P_j, P_i interpolates the degree-$2t$ polynomial $f(X)$ and gets the coefficients $s^{(0)}, \ldots, s^{(2t)}$ from $f(x)$.

2: Upon getting the reconstitution result $x_1, \ldots, x_N$, each party P_i locally samples a random value $r_{i,j}$ for each P_j and computes $X_{i,j} = \sum_{\ell=1}^{N} r_{i,j}^{\ell} \cdot x_\ell$. Then P_i sends $(r_{i,j}, X_{i,j})$ to each party P_j.

3: **Pair-Wise Verification.** Upon receiving $(r_{j,i}, X_{j,i})$ from $2t+1$ distinct P_j, P_i checks whether $X_{j,i} = \sum_{\ell=1}^{N} r_{j,i}^{\ell} \cdot x_\ell$ is hold for all these $2t+1$ distinct P_j. If true, P_i terminates with $x_1, \ldots, x_N$. Otherwise, P_i terminates with a failure symbol `Fail`.

4.3 Triple Extraction

We modify the triple generation technique in [JS26] and give the following construction $\Pi_{\mathtt{GenTriple}}$, which allows all parties to generate random Beaver triples. We first recap a simplified version in [JS26], which allows each virtual party $V_{i,j} = (P_i, P_j)$ to distribute a random triple to all parties.

- **Step 1.** P_i and P_j invoke an instance of $\Pi_{\mathtt{Sh}}$ to share random degree-t sharings $\{[a^{(i)}]_t, [b^{(i)}]_t\}$ and $\{[a^{(j)}]_t, [b^{(j)}]_t\}$, respectively. Upon receiving shares, all parties locally compute their shares of $[a]_t := [a^{(i)}]_t + [a^{(j)}]_t, [b]_t := [b^{(i)}]_t + [b^{(j)}]_t$. These two sharings are the first two random sharings in the output triple.
- **Step 2.** For each $P_k \in \mathcal{P}$, let a_k, b_k denote P_k's shares of $[a]_t, [b]_t$. Then P_k computes $e_k = a_k \cdot b_k$ and randomly samples an additive secret sharing $\langle e_k \rangle = (e_k^{(i)}, e_k^{(j)})$ such that $e_k = e_k^{(i)} + e_k^{(j)}$. Then P_k distributes this additive sharing to P_i and P_j.
- **Step 3.** P_i and P_j agree on a common set $\mathcal{Q}$ of $2t+1$ parties that have distributed the additive shares to both of them. Then they interpolate a degree-$2t$ polynomial based on their additive shares, respectively. As a result, the secrets of their degree-$2t$ polynomials form an additive secret sharing $\langle a \cdot b \rangle$.
- **Step 4.** Let $\langle a \cdot b \rangle = (c^{(i)}, c^{(j)})$, then P_i and P_j invoke $\Pi_{\mathtt{Sh}}$ to distribute the sharings $[c^{(i)}]_t, [c^{(j)}]_t$, respectively. Upon receiving shares, all parties locally compute their shares of $[c]_t := [c^{(i)}]_t + [c^{(j)}]_t$. All parties hold their shares of triple $([a]_t, [b]_t, [c]_t)$.

Optimization. Directly execute the above procedure cause n^2 instances of $\Pi_{\mathtt{Sh}}$ and results in $\mathcal{O}(n^4)$ field elements additive overhead. To reduce the number of $\Pi_{\mathtt{Sh}}$ to $\mathcal{O}(n)$, we let each party $P_i \in \mathcal{P}$ first distribute n random sharings $\{[a^{(i,j)}]_t, [b^{(i,j)}]_t, [r^{(i,j)}]_t\}_{j\in[n]}$ via an instance of $\Pi_{\mathtt{Sh}}$, each one will be used to generate triple for the virtual party $V_{i,j}$.

For each $V_{i,j}$, parties set $[a]_t = [a^{(i,j)}]_t + [a^{(j,i)}]_t, [b]_t = [b^{(i,j)}]_t + [b^{(j,i)}]_t$. Upon getting the additive shares of $\langle a \cdot b \rangle$ in Step 4, P_j will use his secret $r^{(j,i)}$ to mask his additive share $c^{(j)}$ and send it to P_i. Then P_i can computes the secret $z^{(i,j)} = a \cdot b + r^{(i,j)} + r^{(j,i)}$. Upon getting $z^{(i,j)}$ for $2t+1$ distinct P_j, P_i reliably broadcasts them in parallel, and all parties can compute their shares of triples for these $2t+1$ distinct virtual parties $V_{i,j}$ in parallel. As a result, each party only invokes one instance of $\Pi_{\mathtt{Sh}}$ and reliably broadcasts, achieving $\mathcal{O}(n^3)$ additive overhead.

Finally, to avoid P_i and P_j performing $\mathcal{O}(n)$ round communication to agree on the set $\mathcal{Q}$, we let P_j choose his additive share of each P_k's $\langle e_k \rangle$. Upon receiving the additive share from P_j, each party P_k computes the additive share for P_i. Later, P_i only needs to send the set $\mathcal{Q}$ to P_j, and they can get their additive shares of the $\langle a \cdot b \rangle$.

Protocol $\Pi_{\mathtt{GenTriple}}$

Parameter: Let N be the number of Beaver triples to be prepared, $\tilde{N} = N + n\left\lceil \frac{n+\kappa}{\kappa} \right\rceil$, $N' = 9(\tilde{N} + 1/4)/n^2$. Let $\beta_1, \ldots, \beta_{4\tilde{N}+1}$ and $\gamma_1, \ldots, \gamma_N, \ldots, \gamma_{\tilde{N}}$ be distinct field elements.

1: **Distributing Random Sharings.** Each party P_i invokes the distributing phase of $\Pi_{\mathtt{Sh}}$ to distribute $3N'n$ random degree-t Shamir sharings $\{[a_\ell^{(i,j)}]_t, [b_\ell^{(i,j)}]_t, [r_\ell^{(i,j)}]_t\}_{\ell\in[N']}$ for all $j \in [n]$. All parties will invoke the recovery phase of $\Pi_{\mathtt{Sh}}$ when they terminate the distributing phase of $\Pi_{\mathtt{Sh}}$ invoked by dealer P_i.

2: **Generating Random Triples by Virtual Party.** For each virtual party $V_{i,j} = (P_i, P_j)$, all parties proceed the following steps after they terminate $\Pi_{\mathtt{Sh}}$ invoked by P_i, P_j and get their shares of $\{[a_\ell^{(i,j)}]_t, [b_\ell^{(i,j)}]_t, [r_\ell^{(i,j)}]_t\}_{\ell\in[N']}$, $\{[a_\ell^{(j,i)}]_t, [b_\ell^{(j,i)}]_t, [r_\ell^{(j,i)}]_t\}_{\ell\in[N']}$.

(1). For each party P_k and $\ell \in [N']$, P_j randomly samples a value $o_{\ell,k}$ and sends it to P_k.

(2). For all $\ell \in [N']$, all parties locally compute $[a_\ell]_t = [a_\ell^{(i,j)}]_t + [a_\ell^{(j,i)}]_t, [b_\ell]_t = [b_\ell^{(i,j)}]_t + [b_\ell^{(j,i)}]_t$. We denote P_k's share of $[a_\ell]_t, [b_\ell]_t$ as $a_{\ell,k}, b_{\ell,k}$.

(3). Upon receiving $o_{\ell,k}$ from P_j, each P_k computes and sends $d_{\ell,k} = a_{\ell,k} \cdot b_{\ell,k} - o_{\ell,k}$ to P_i for all $\ell \in [N']$.

(4). P_i initializes a set $\mathcal{Q} = \varnothing$. Upon receiving $\{d_{\ell,k}\}_{\ell\in[N']}$ from P_k, P_i adds P_k to $\mathcal{Q}$. Upon getting $|\mathcal{Q}| = 2t+1$, P_i sends $\mathcal{Q}$ to P_j.

(5). Upon receiving $\mathcal{Q}$ from P_i, P_j interpolates a degree-$2t$ sharing $[o_\ell]_{2t}$ based on $\{o_{\ell,k}\}_{k\in\mathcal{Q}}$ for each $\ell \in [N']$. Then P_j sends $\{o_\ell + r_\ell^{(j,i)}\}_{\ell\in[N']}$ to P_i.

(6). Upon receiving $\{o_\ell + r_\ell^{(j,i)}\}_{\ell\in[N']}$ from P_j, P_i first interpolates a degree-$2t$ sharing $[d_\ell]_{2t}$ based on $\{d_{\ell,k}\}_{k\in\mathcal{Q}}$, then computes $z_\ell = d_\ell + o_\ell + r_\ell^{(j,i)} + r_\ell^{(i,j)}$ for all $\ell \in [N']$.

Party P_i initializes a set $\mathtt{Ready}_i = \varnothing$. We denote (a_ℓ, b_ℓ, z_ℓ) by $(\tilde{a}_\ell^{(i,j)}, \tilde{b}_\ell^{(i,j)}, \tilde{z}_\ell^{(i,j)})$ for all $\ell \in [N']$. Upon getting $\{\tilde{z}_\ell^{(i,j)}\}_{\ell\in[N']}$, P_i adds P_j to $\mathtt{Ready}_i$.

3: **Distributing Random Triples.** For party P_i, upon getting $|\texttt{Ready}_i| = 2t+1$, he reliably broadcasts the set $\texttt{Ready}_i$ and $\{\tilde{z}_\ell^{(i,j)}\}_{\ell\in[N']}$ for all $P_j \in \texttt{Ready}_i$. Each party who receives message from P_i's broadcast will locally compute their shares of $[\tilde{c}_\ell^{(i,j)}]_t = \tilde{z}_\ell^{(i,j)} - [r_\ell^{(i,j)}]_t - [r_\ell^{(j,i)}]_t$ for all $\ell \in [N'], P_j \in \texttt{Ready}_i$. Then we view $\{[\tilde{a}_\ell^{(i,j)}]_t, [\tilde{b}_\ell^{(i,j)}]_t, [\tilde{c}_\ell^{(i,j)}]_t\}_{\ell\in[N']}$ as the triples generated by the virtual party $V_{i,j} = (P_i, P_j)$.

With random triples prepared by virtual parties, all parties follow the triple extraction technique [CP17] to generate fresh random Beaver triples. Upon termination $\Pi_{\texttt{GenTriple}}$, parties get shared polynomials f, g, h. Let L be the degree of f, g, then h is of degree-$2L$, and at most $L/2$ evaluation points on f, g, h are known to the adversary. There remain $L/2+1$ random evaluation points, and each of them forms a valid random Beaver triple.

Protocol $\Pi_{\texttt{GenTriple}}$

4: **Determine the Set of Successful Dealers.** All parties set the property Q as they receive the set $\texttt{Ready}_i$ from party P_i's broadcast. Then they invoke $\mathcal{F}_{\texttt{acs}}$ with property Q to agree on a set $\mathcal{D}$ of parties that successfully broadcast their set $\texttt{Ready}$.

5: **Extract Random Beaver Triples.** All parties do the following to extract random triples based on $\{[\tilde{a}_\ell^{(i,j)}]_t, [\tilde{b}_\ell^{(i,j)}]_t, [\tilde{c}_\ell^{(i,j)}]_t\}_{\ell\in[N']}$ for all $P_i \in \mathcal{D}, P_j \in \texttt{Ready}_i$. We denote $\mathcal{V}$ as the set of virtual parties such that each $\mathcal{V} = \{V_{i,j} | P_i \in \mathcal{D}, P_j \in \texttt{Ready}_i\}$, then the triples prepared by virtual parties in $\mathcal{V}$ is denoted by $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=1}^{|\mathcal{V}|}$, where $|\mathcal{V}| = \frac{4}{9}n^2 \cdot N' = 4\tilde{N}+1$.

(1). All parties set two polynomials f, g of degree $(|\mathcal{V}|-1)/2 = 2\tilde{N}$ such that $[f(\beta_i)]_t = [a_i]_t$ and $[g(\beta_i)]_t = [b_i]_t$ for all $i \in [2\tilde{N}+1]$.

(2). For all $i \in [2\tilde{N}+2, 4\tilde{N}+1]$, all parties locally compute their shares of $[f(\beta_i)+a_i]_t, [g(\beta_i)+b_i]_t$ and invoke $\Pi_{\texttt{PubRec}}$ to reconstruct $f(\beta_i)+a_i, g(\beta_i)+b_i$. Then locally compute:

$$\begin{aligned}[f(\beta_i)\cdot g(\beta_i)]_t = &(f(\beta_i)+a_i)\cdot(g(\beta_i)+b_i)\\ &-(f(\beta_i)+a_i)\cdot[b_i]_t\\ &-(g(\beta_i)+b_i)\cdot[a_i]_t + [c_i]_t.\end{aligned}$$

(3). All parties set a polynomial h of degree $4\tilde{N}+1$ such that $[h(\beta_i)]_t = [c_i]_t$ for all $i \in [2\tilde{N}+1]$ and $[h(\beta_i)]_t = [f(\beta_i)\cdot g(\beta_i)]_t$ for all $i \in [2\tilde{N}+2, 4\tilde{N}+1]$.

(4). All parties records their shares of $\{([f(\beta_i)]_t, [g(\beta_i)]_t), [h(\beta_i)]\}_{i\in[4\tilde{N}+1]}$.

It can be verified that the total communication complexity is $\mathcal{O}(Nn+n^3)$ field elements, with the help of the current best instantiation of the reliably broadcast protocol in the IT-secure setting [ADD+22].

4.4 Offline Verification

The following protocol describes the verification procedure we introduced in Section 2.3.

Protocol $\Pi_{\texttt{GenTriple}}$

Verifying the Multiplication relation of Triples:

6: Each party P_i initializes a counter $\texttt{cnt} = 0$ and reliably broadcasts $\lceil \frac{n+\kappa}{\kappa} \rceil$ random challenges not in $\{\gamma_1, \ldots, \gamma_N\}$. P_i will do the following steps for all these challenges in parallel, and we denote each challenge by r_i for simplicity.

7: Upon receiving r_j from each P_j, P_i computes $\{[f(r_j)]_t, [g(r_j)]_t, [h(r_j)]_t\}_{k\in[N]}$ based on his shares of $\{([f(\beta_i)]_t, [g(\beta_i)]_t), [h(\beta_i)]\}_{i\in[4\tilde{N}+1]}$ and sends them to P_j.

8: Upon receiving shares of $\{[f(r_i)]_t, [g(r_i)]_t, [h(r_i)]_t\}$ from $2t+1$ distinct P_j (the same set of parties for all random challenges), P_i checks whether these shares lie on degree-t polynomials $\{f_i(x), g_i(x), h_i(x)\}$ and $f_i(r_i)\cdot g_i(r_i) = h_i(r_i)$. If true, P_i sets $\texttt{cnt} = 1$ and sends $(\texttt{cnt}, P_i)$ to all parties. Otherwise, P_i sends $\perp$ to all parties and terminates with the failure symbol $\texttt{Fail}$.

9: From $\texttt{cnt} = 1$ to $t+1$, P_i proceeds the following steps. During this procedure, if P_i receives $\perp$ from other parties, P_i sends $\perp$ to all parties and terminates with the failure symbol $\texttt{Fail}$.

(1). Upon receiving $(\texttt{cnt}, P_j)$ and shares of $\{[f(r_i)]_t, [g(r_i)]_t, [h(r_i)]_t\}$ from $2t+1$ distinct P_j, if all these shares lie on $\{f_i(x), g_i(x), h_i(x)\}$, P_i sets $\texttt{cnt} \leftarrow \texttt{cnt} + 1$.
(2). If $\texttt{cnt} = t+1$, P_i reliably broadcasts (OK, P_i). Otherwise, P_i sends $(\texttt{cnt}, P_i)$ to all parties.
10: Upon getting $\texttt{cnt} = t+2$, P_i locally computes his shares of $\{[f(\gamma_i)]_t, [g(\gamma_i)]_t, [h(\gamma_i)]_t\}_{i\in[N]}$ as his output shares of triples. Then P_i terminates with his shares of triples and $\{([f(\beta_i)]_t, [g(\beta_i)]_t), [h(\beta_i)]\}_{i\in[4\tilde{N}+1]}$.

4.5 Offline Phase

Finally, we define the functionality $\mathcal{F}_{\mathsf{Triple}}$ and the construction of $\Pi_{\texttt{Triple}}$. Let N be the number of triples to be prepared, L be each party's maximum number of requests for verifying the random triples. The first **Generation of the Random Triples** is the triple generation protocol invoked in the offline phase, and the second **Request for Verifying the Random Triples** is invoked later in the online phase to verify the used triples.

Remark 3. Recall that later in the online phase, all parties still need to verify the correctness of the used triples for circuit evaluation. In that verification, each party also needs to propose $\lceil \frac{n+\kappa}{\kappa} \rceil$ distinct random challenges. The parameter L denotes the maximum number of times each party can propose these challenges for verification. In our construction, all parties invoke this verification twice in the offline phase. Then each party will propose its challenges for $L = 2$ times.

The total communication complexity of $\Pi_{\texttt{Triple}}$ is $\mathcal{O}(Nn + Ln^3)$ field elements for generating N random Beaver triples. During the request for verification invoked by each party, it costs $\mathcal{O}(Ln^2)$ field elements. Refer to Appendix B for the security proof of Lemma 1.

Functionality $\mathcal{F}_{\mathsf{Triple}}$

Public Input: Integer N, L and distinct field elements $\alpha_1, \ldots, \alpha_L$. The integer N denotes the number of random Beaver triples to be prepared, and the integer L denotes the maximum number of each party's requests for verifying the triples.
$\mathcal{F}_{\mathsf{Triple}}$ runs with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an ideal adversary $\mathcal{S}$.

Generation of the Random Triples

1: Upon receiving request $(\texttt{Init}, \texttt{Triple})$ from $2t+1$ parties, for all $\ell \in [N]$, the functionality randomly samples $a_\ell, b_\ell, c_\ell \in \mathbb{F}$ such that $c_\ell = a_\ell \cdot b_\ell$. Then the functionality randomly samples three random degree-t Shamir sharings $([a_\ell]_t, [b_\ell]_t, [c_\ell]_t)$ based on the secrets a_ℓ, b_ℓ, c_ℓ.
2: The functionality distributes the shares of $\{[a_\ell]_t, [b_\ell]_t, [c_\ell]_t\}_{\ell\in[N]}$ to all parties as request-based delayed outputs.
- Upon receiving a set of shares $\{u_{\ell,j}, v_{\ell,j}, w_{\ell,j}\}_{j\in\mathcal{C}orr}$ of corrupted parties from $\mathcal{S}$, if the output has not been delivered to any honest parties, the functionality randomly samples the sharings $([a_\ell]_t, [b_\ell]_t, [c_\ell]_t)$ again based on the shares of corrupted parties and the secrets a_ℓ, b_ℓ, c_ℓ.
- Upon receiving request $(\texttt{InsertShareError}, P_i, \{e_\ell^{(a)}, e_\ell^{(b)}, e_\ell^{(c)}\}_{\ell\in[N]})$ from $\mathcal{S}$, if the output has not been delivered to P_i, the functionality adds $\{e_\ell^{(a)}, e_\ell^{(b)}, e_\ell^{(c)}\}_{\ell\in[N]}$ to P_i's shares of $\{[a_\ell]_t, [b_\ell]_t, [c_\ell]_t\}_{\ell\in[N]}$, respectively, and uses them as P_i's output.
- Upon receiving request $(\texttt{abort}, P_i)$ from $\mathcal{S}$, if the output has not been delivered to P_i, the functionality changes P_i's output to a failure symbol $\texttt{Fail}$.

Request for Verifying the Random Triples

3: For each party P_i, the functionality initializes a counter $\texttt{cnt}_i = 1$, two sets $\mathcal{H}_i = \mathcal{M}_i = \varnothing$. If $\texttt{cnt}_i \le L$, the functionality does the following steps.
- Upon receiving a request $(\texttt{VerifyStart}, P_i)$ from party $2t+1$ distinct parties (include party P_i), it sets $\texttt{cnt}_i \leftarrow \texttt{cnt}_i + 1$ and sends $(\texttt{VerifyStart}, P_i)$ to all parties as a request-based delayed output. It adds the identity of the honest parties who have received their output shares in Step 2 to the set $\mathcal{H}_i$.
- Upon receiving a request $(\texttt{VerifyProceed}, P_i, P_j)$ from a party P_j, if the output shares in Step 2 have been delivered to P_j, the functionality sends $(\texttt{VerifyProceed}, P_i, P_j)$ to P_i as a request-based delayed output and adds P_j to $\mathcal{M}_i$.

4: Upon receiving a request $(\texttt{VerifyEnd}, P_i, \mathcal{M})$ from party P_i, if $\texttt{cnt}_i \leq L, \mathcal{M} \subseteq \mathcal{M}_i$ and $|\mathcal{M}| \geq 2t+1$, the functionality does the following steps to decide an output bit $b \in \{0,1\}$, and sends it to P_i as a request-based delayed output.
   (1). It checks whether the output shares of honest parties in $\mathcal{M}$ lie on degree-t polynomials. If not, it sets $b \leftarrow 0$ as the final output for P_i. Otherwise, it proceeds.
   (2). It interpolates the secrets $\{a'_\ell, b'_\ell, c'_\ell\}_{\ell \in [N]}$ based on the output shares of honest parties in $\mathcal{M}$. Then it checks whether $\{a'_\ell, b'_\ell, c'_\ell\} = \{a_\ell, b_\ell, c_\ell\}$ for all $\ell \in [N]$. If true, it sets $b \leftarrow 1$. Otherwise, it sets $b \leftarrow 0$. Then it sends b to $\mathcal{S}$.
   (3). Upon receiving request $(\texttt{Decide}, b')$ from $\mathcal{S}$, if the output bit b has not been delivered to P_i and $|\mathcal{H}_i \cap \mathcal{M}| \leq t$, it changes P_i's output to b'.

Protocol $\Pi_{\texttt{Triple}}$

Parameter. Let N be the number of triples to be prepared, L be each party's maximum number of requests for verifying the random triples, κ be the security parameter, and $\tilde{N} = N + (L+1)n\left\lceil \frac{n+\kappa}{\kappa} \right\rceil$. Let $\gamma_1, \ldots, \gamma_N, \ldots, \gamma_{\tilde{N}}, \beta_1, \ldots, \beta_{4\tilde{N}+1}$ denote distinct field elements.

Generation of the Random Triples

1: All parties invoke $\Pi_{\texttt{GenTriple}}$ with input parameter $\tilde{N} - n\left\lceil \frac{n+\kappa}{\kappa} \right\rceil$ and the field elements $\gamma_1, \ldots, \gamma_{\tilde{N}}, \beta_1, \ldots, \beta_{4\tilde{N}+1}$. Upon terminating $\Pi_{\texttt{GenTriple}}$, for each party P_i:
   – If the output is $\texttt{Fail}$, this party terminates with the failure symbol $\texttt{Fail}$.
   – Otherwise, this party gets his shares of triples and $\{([f(\beta_i)]_t, [g(\beta_i)]_t, [h(\beta_i)]_t)\}_{i \in [4\tilde{N}+1]}$, where the degree of f, g is $2\tilde{N}$ and $h = f \cdot g$. This party will output the first N shares of triples, denoted by $\{[f(\gamma_i)]_t, [g(\gamma_i)]_t, [h(\gamma_i)]_t\}_{i \in [N]}$.

Request for Verifying the Random Triples

All parties locally initialize a counter $\texttt{cnt}_i = 1$ for each party P_i. When $\texttt{cnt}_i \leq L$, they proceed to the following steps. Each party P_i locally initializes a set $\mathcal{M}_i = \varnothing$.

1: Upon receiving a request $(\texttt{VerifyStart}, P_i)$ from the environment, P_i reliably broadcasts $\left\lceil \frac{n+\kappa}{\kappa} \right\rceil$ random challenges $r_\ell \notin \{\gamma_1, \ldots, \gamma_N\}$ for all $\ell \in [\left\lceil \frac{n+\kappa}{\kappa} \right\rceil]$. Each party P_j who receives the request $(\texttt{VerifyStart}, P_i)$ from the environment will participate in P_i's broadcast protocol. Upon terminating P_i's broadcast, all parties set $\texttt{cnt} \leftarrow \texttt{cnt} + 1$.
2: For each party P_j, upon receiving a request $(\texttt{VerifyProceed}, P_i, P_j)$ from the environment, if P_j has got his shares of $\{[f(\beta_i)]_t, [g(\beta_i)]_t, [h(\beta_i)]_t)\}_{i \in [4\tilde{N}+1]}$, for each random challenge r_ℓ he received from P_i's broadcast, he locally computes his shares of $\{[f(r_\ell)]_t, [g(r_\ell)]_t, [h(r_\ell)]_t)\}$ and sends it to P_i. When party P_i receives shares of $\{[f(r_\ell)]_t, [g(r_\ell)]_t, [h(r_\ell)]_t)\}$ from P_j, he outputs $(\texttt{VerifyProceed}, P_i, P_j)$ and adds P_j to $\mathcal{M}_i$.
3: Upon receiving a request $(\texttt{VerifyEnd}, P_i, \mathcal{M})$ from the environment, if $\mathcal{M} \subseteq \mathcal{M}_i$ and $|\mathcal{M}| \geq 2t+1$, for all $\ell \in [\left\lceil \frac{n+\kappa}{\kappa} \right\rceil]$, P_i does the following steps to decide an output bit $b \in \{0,1\}$.
   – P_i checks whether all shares of $\{[f(r_\ell)]_t, [g(r_\ell)]_t, [h(r_\ell)]_t)\}$ received from parties in $\mathcal{M}$ lie on degree-t polynomials. If not, it sets $b \leftarrow 0$. Otherwise, it proceeds.
   – P_i interpolates the secrets $\{f(r_\ell), g(r_\ell), h(r_\ell)\}$ based on the shares received from parties in $\mathcal{M}$, then it checks whether $f(r_\ell) \cdot g(r_\ell) = h(r_\ell)$. If true, it sets $b \leftarrow 1$. Otherwise, it sets $b \leftarrow 0$.

Lemma 1. *Protocol $\Pi_{\texttt{Triple}}$ securely computes $\mathcal{F}_{\texttt{Triple}}$ in the $\mathcal{F}_{\texttt{acs}}$-hybrid model against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t < n/3$ parties.*

Communication Complexity. We analyze the communication complexity as follows.

– During the generation of triples, each party invokes one instance of $\Pi_{\texttt{Sh}}$ and reliably broadcast with input size $\mathcal{O}(N/n + n)$, and the total communication complexity is $\mathcal{O}(Nn + n^3)$ field elements for all parties. Parties also invoke one instance of $\mathcal{F}_{\texttt{acs}}$ to do agreement.
– For the rest of the verification procedure, each party reliably broadcasts $\mathcal{O}(n^2)$ field elements each time, which costs $\mathcal{O}(n^3)$ field elements in total for all parties. For the rest of steps, parties exchange their counter messages, which costs $\mathcal{O}(n^3)$ field elements in total.

5 Main Protocol

5.1 Online Verification

Our goal is to verify whether there exists $t+1$ honest parties that use valid triples (verified in the offline phase) to evaluate all multiplication gates, and the secrets interpolated by their shares of $\{[u_\ell]_t, [v_\ell]_t\}_{\ell\in[C]}$ are the same as the public reconstruction results. The core idea is explained in Section 2.4.

Protocol $\Pi_{\texttt{VerifyEval}}$

Parameter. Let C denote the number of gates to be verified.

Input. All parties take the public reconstruction results $\{u_\ell, v_\ell\}_{\ell\in[C]}$ and their shares of $\{[u_\ell]_t, [v_\ell]_t\}_{\ell\in[C]}$ as input, which are generated when all parties use triples to evaluate the circuit.

If one party's input share is a failure symbol $\perp$, then he will send $\perp$ to all parties. Parties that receive $\perp$ will forward it to all parties and terminate with `Fail`.

For each P_i:

1: P_i initializes a counter $\texttt{cnt} = 0$, randomly samples $\left\lceil \frac{n+\kappa}{\kappa} \right\rceil$ distinct challenges, and reliably broadcasts them. We denote each challenge by r_i, and all parties will do the following steps for all challenges in parallel. Each party that enters this step will send a request $(\texttt{VerifyStart}, P_i)$ to $\mathcal{F}_{\mathsf{Triple}}$.

2: Upon receiving r_j from P_j's broadcast, P_i sends request $(\texttt{VerifyProceed}, P_j, P_i)$ to $\mathcal{F}_{\mathsf{Triple}}$, computes and sends his share of $[S_j]_t$ (defined below) to P_j.

$$[S_j]_t := \sum_{\ell\in[C]} r_j^{\ell-1}([u_\ell]_t + r_j^C \cdot [v_\ell]_t)$$

3: Upon receiving share of $[S_i]_t$ and $(\texttt{VerifyProceed}, P_i, P_j)$ from $2t+1$ distinct P_j (the same set of parties for all challenges), let $\mathcal{M}$ denote the set of these $2t+1$ distinct P_j, P_i does the following steps.

- P_i sends a request $(\texttt{VerifyEnd}, P_i, \mathcal{M})$ to $\mathcal{F}_{\mathsf{Triple}}$, gets an output bit $b \in \{0,1\}$, and checks whether $b = 1$.
- P_i checks whether the shares of $[S_i]_t$ from parties in $\mathcal{M}$ lie on a degree-t polynomial $S_i(x)$. If true, then P_i further checks whether $S_i(\alpha_0) \stackrel{?}{=} \sum_{\ell\in[C]} r_i^{\ell-1}(u_\ell + r_i^C \cdot v_\ell)$.

If both the above conditions hold, P_i sets $\texttt{cnt} = 1$ and sends $(\texttt{cnt}, P_i)$ to all parties. Otherwise, P_i sends $\perp$ to all parties and terminates with the failure symbol `Fail`.

4: From $\texttt{cnt} = 1$ to $t+1$, P_i proceeds the following steps. During this procedure, if P_i receives $\perp$ from other parties, P_i sends $\perp$ to all parties and terminates with the failure symbol `Fail`.

(1). Upon receiving $(\texttt{cnt}, P_j)$ and shares of $[S_i]_t$ from $2t+1$ distinct P_j, if $P_j \notin \mathcal{M}$, P_i adds P_j. Then P_i sends a request $(\texttt{VerifyEnd}, P_i, \mathcal{M})$ to $\mathcal{F}_{\mathsf{Triple}}$. If the output is 1, P_i sets $\texttt{cnt} \leftarrow \texttt{cnt} + 1$. Otherwise, P_i sends $\perp$ to all parties.

(2). If $\texttt{cnt} = t+1$, P_i reliably broadcasts (OK, P_i). Otherwise, P_i sends $(\texttt{cnt}, P_i)$ to all parties.

5: Upon getting $\texttt{cnt} = t+2$, P_i terminates with `Success`.

5.2 Circuit Evaluation

Based on the ideas we introduced in Section 2.4 and 2.5, we get the construction of $\Pi_{\texttt{AMPC-Abort}}$. Let C and D denote the circuit size and depth, the communication complexity is $\mathcal{O}(Cn + Dn^2 + n^3)$ field elements plus $\mathcal{O}(1)$ instance of $\mathcal{F}_{\mathsf{acs}}$. Refer to Appendix C for the security proof of Lemma 2.

Protocol $\Pi_{\texttt{AMPC-Abort}}$

Parameter. Let $|C|$ denote the circuit size, C_I denote the input size.

Offline Phase

Preparation of Beaver Triples. All parties invoke $\mathcal{F}_{\mathsf{Triple}}$ to prepare $|C| + C_I$ random Beaver triples.

Input Phase

1: **Distributing Inputs.** All parties do the following steps.

(1). They assign an unused triple $\{[a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t\}$ to each party P_i's input $x^{(i)}$, then they send their shares of $\{[a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t\}$ to P_i.

(2). Each party P_i uses the first $t+1$ shares of $\{[a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t\}$ he received to interpolate the secrets $\{a^{(i)}, b^{(i)}, c^{(i)}\}$. Then P_i verifies that whether $a^{(i)} \cdot b^{(i)} = c^{(i)}$. If true, he reliably broadcasts $x^{(i)} + a^{(i)}$. Otherwise, he reliably broadcasts a failure symbol $\perp$.

2: **Agreement the Set of the Inputs.** Each party P_i sets the property Q as P_i terminates reliably broadcast led by dealer P_j. Then all parties invoke $\mathcal{F}_{\mathsf{acs}}$ with property Q to agree on a set $\mathcal{D}$ of parties that successfully share their inputs. For every $P_i \notin \mathcal{D}$, all parties set their shares of P_i's input as a default value 0.

3: **Computing Shares of Input Wires.** For each dealer $P_i \in \mathcal{D}$, all parties check whether the broadcast message is a failure symbol $\perp$. If not, all parties locally compute their shares of P_i's input sharings $[x^{(i)}]_t = x^{(i)} + a^{(i)} - [a^{(i)}]_t$. Otherwise, all parties terminate with `Fail`. We define $\{x_\ell + a_\ell\}_{\ell \in [C+1, C+C_I]} := \{x^{(i)} + a^{(i)}\}_{i \in \mathcal{D}}$.

Protocol $\Pi_{\texttt{AMPC-Abort}}$

Evaluation Phase

1: **Evaluating the Circuit.** All parties do the following to evaluate the circuit layer by layer:

- **Addition Gate:** With input sharings $[x]_t, [y]_t$, all parties locally compute the output sharing $[z]_t = [x]_t + [y]_t$.
- **Multiplication Gate:** All parties evaluate every $t+1$ multiplication gate in each layer in parallel. Let $([a_i]_t, [b_i]_t, [c_i]_t)_{i=1}^{t+1}$ denote the random Beaver triples assigned to these $t+1$ gates, and $\{([x_i]_t, [y_i]_t)\}_{i=1}^{t+1}$ be the corresponding input sharings. All parties invoke an instance of $\Pi_{\texttt{PubRec}}$ with their shares of $([x_i + a_i]_t, [y_i + b_i]_t)_{i=1}^{t+1}$ as input to reconstruct $\{(x_i + a_i, y_i + b_i)\}_{i=1}^{t+1}$. Upon terminating $\Pi_{\texttt{PubRec}}$, if one party does not terminate with `Fail`, he computes his output share $[z_i]_t$ as follows:

$$[z_i]_t = (x_i + a_i)(y_i + b_i) - (x_i + a_i)[b_i]_t - (y_i + b_i)[a_i]_t + [c_i]_t.$$

  Otherwise, he sets the failure symbol $\perp$ as his share of $[z_i]_t$.

2: **Verification of the Evaluation.** All parties invoke $\Pi_{\texttt{VerifyEval}}$ to do verification. They take the public reconstruction results $\{x_\ell + a_\ell, y_\ell + b_\ell\}_{\ell \in [C+C_I]}$ and their shares of $\{[x_\ell + a_\ell]_t, [y_\ell + b_\ell]_t\}_{\ell \in [C+C_I]}$ as the input for $\Pi_{\texttt{VerifyEval}}$. Upon terminating $\Pi_{\texttt{VerifyEval}}$, for each party:

- If the output is `Success`, then this party proceeds to the output phase.
- Otherwise, this party sets the failure symbol $\perp$ as his share of all output wires.

Output Phase

1: **Reconstruct the Output.** All parties invoke $\Pi_{\texttt{PubRec}}$ with their output shares $[y]_t$ to reconstruct the circuit output y.

2: **Verification of the Output.** All parties invoke $\Pi_{\texttt{VerifyEval}}$ in the same way as in the Evaluation Phase to do verification, except that they additionally take the public reconstruction output values and their shares of output wires as inputs.

- If a party terminates with `Success` and has got the circuit output y, then he terminates with y.
- Otherwise, this party terminates with `Fail`.

Lemma 2. *Protocol $\Pi_{\texttt{AMPC-Abort}}$ securely computes $\mathcal{F}_{\mathsf{AMPC}}$ in the $\{\mathcal{F}_{\mathsf{acs}}, \mathcal{F}_{\mathsf{Triple}}\}$-hybrid model against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t < n/3$ parties.*

Communication Complexity. We analyze the communication complexity as follows.

- In the offline phase, all parties first prepare $C + C_I$ random triples, which costs $\mathcal{O}((C + C_I)n + n^3)$ field elements.
- In the input phase, all parties help each party to reconstruct a triple, then let this party reliably broadcast $x+a$. The total communication complexity is $\mathcal{O}(C_I n + n^3)$ field elements. Parties also invoke one instance of $\mathcal{F}_{\mathsf{acs}}$ to do agreement.

– In the evaluation phase, all parties invoke Beaver triple technique to evaluate the circuit, which costs $\mathcal{O}(Cn + Dn^2 + n^3)$ field elements. Parties also invoke $\Pi_{\texttt{VerifyEval}}$ to do verification, which costs $\mathcal{O}(n^3)$ field elements.
– In the output phase, parties publicly reconstruct the output and invoke $\Pi_{\texttt{VerifyEval}}$ to do verification, which costs $\mathcal{O}(n^3)$ field elements in total.

References

[AAPP24] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Perfect asynchronous mpc with linear communication overhead. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 280–309. Springer, 2024.

[ADD+22] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 399–417, 2022.

[ADS20] Ittai Abraham, Danny Dolev, and Gilad Stern. Revisiting asynchronous fault tolerant computation with optimal resilience. In Yuval Emek and Christian Cachin, editors, *39th ACM PODC*, pages 139–148. ACM, August 2020.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BJK+25] Akhil Bandarupalli, Xiaoyu Ji, Aniket Kate, Chen-Da Liu-Zhang, Daniel Pöllmann, and Yifan Song. Velox: Scalable fair asynchronous mpc from lightweight cryptography. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 1799–1813, 2025.

[BKR94] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In Jim Anderson and Sam Toueg, editors, *13th ACM PODC*, pages 183–192. ACM, August 1994.

[Bra87] Gabriel Bracha. An $\mathcal{O}(\log n)$ expected rounds randomized byzantine generals protocol. *Journal of the ACM (JACM)*, 34(4):910–920, 1987.

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

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[CFG+23] Ran Cohen, Pouyan Forghani, Juan Garay, Rutvik Patel, and Vassilis Zikas. Concurrent asynchronous byzantine agreement in expected-constant rounds, revisited. In Guy Rothblum and Hoeteck Wee, editors, *Theory of Cryptography*, pages 422–451, Cham, 2023. Springer Nature Switzerland.

[CFG+25] Ran Cohen, Pouyan Forghani, Juan Garay, Rutvik Patel, and Vassilis Zikas. Is it even possible? on the parallel composition of asynchronous mpc protocols. In *Theory of Cryptography Conference*, pages 33–63. Springer, 2025.

[Che24] Jinyuan Chen. Ociormvba: Near-optimal error-free asynchronous mvba. *arXiv preprint arXiv:2501.00214*, 2024.

[Coh16] Ran Cohen. Asynchronous secure multiparty computation in constant time. In *Public-Key Cryptography–PKC 2016*, pages 183–207. Springer, 2016.

[CP17] Ashish Choudhury and Arpita Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Transactions on Information Theory*, 63(1):428–468, 2017.

[CP23] Ashish Choudhury and Arpita Patra. On the communication efficiency of statistically secure asynchronous mpc with optimal resilience. *Journal of Cryptology*, 36(2):13, 2023.

[CR98] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience, 1998.

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer, 2007.

[DR85] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.

[DWZ23] Sisi Duan, Xin Wang, and Haibin Zhang. Fin: Practical signature-free asynchronous common subset in constant time. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 815–829, 2023.

[GLO+21] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. ATLAS: Efficient and scalable MPC in the honest majority setting. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 244–274, Virtual Event, August 2021. Springer, Cham.

[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.

[GLZS24] 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, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 170–206. Springer, Cham, August 2024.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GS20] Vipul Goyal and Yifan Song. Malicious security comes free in honest-majority mpc. *Cryptology ePrint Archive*, 2020.

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority MPC. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 618–646. Springer, Cham, August 2020.

[JLS24] Xiaoyu Ji, Junru Li, and Yifan Song. Linear-communication asynchronous complete secret sharing with optimal resilience. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 418–453. Springer, Cham, August 2024.

[JS26] Xiaoyu Ji and Yifan Song. Statistically secure asynchronous mpc with linear communication and $\mathcal{O}(n^5)$ additive overhead. *Cryptology ePrint Archive*, 2026.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *Theory of Cryptography Conference*, pages 477–498. Springer, 2013.

[PCR10] Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. Efficient statistical asynchronous verifiable secret sharing with optimal resilience. In Kaoru Kurosawa, editor, *ICITS 09*, volume 5973 of *LNCS*, pages 74–92. Springer, Berlin, Heidelberg, December 2010.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

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

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd FOCS*, pages 80–91. IEEE Computer Society Press, November 1982.

A Additional Preliminaries

We describe functionalities for the agreement primitives, following the descriptions from [CFG+23, CFG+25].

Reliable Broadcast. We describe the functionality $\mathcal{F}_{\mathsf{rbc}}$ for reliable broadcast. When a party P_s inputs a value v to the functionality as the sender, we will say that "P_s (reliably) broadcasts value v". Moreover, when some party P_j receives an output v in a reliable broadcast functionality with sender P_i, we will say that "P_j receives output v from P_i's reliable broadcast", and we will omit specifying the sender if the context is clear. The functionality of $\mathcal{F}_{\mathsf{rbc}}$ from [CFG+23].

Functionality $\mathcal{F}_{\mathsf{rbc}}$

The functionality $\mathcal{F}_{\mathsf{rbc}}$ runs with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, a resiliency threshold $t \leq n/3$, and an ideal adversary $\mathcal{S}$. Start a session once a message of the form $(\mathtt{send}, \mathtt{sid}, \cdot)$ with $\mathtt{sid} = (\mathcal{P}, P_s, \mathtt{sid}')$ is received from (the sender) $P_s \in \mathcal{P}$. For each $P_i \in \mathcal{P}$, initializes $\mathtt{part}_i := 0$ and delay values $D_i^{\mathtt{in}} = D_i^{\mathtt{out}} := 1$. Also initialize $y := \perp$.

- Upon receiving $(\mathtt{delay}, \mathtt{sid}, P_i, \mathtt{type}, D)$ from $\mathcal{S}$ for $P_i \in \mathcal{P}$, with $\mathtt{type} \in \{\mathtt{in}, \mathtt{out}\}$, and $D \in \mathbb{Z}$ represented in unary notation, update $D_i^{\mathtt{type}} := \max(1, D_i^{\mathtt{type}} + D)$.
- Upon receiving $(\mathtt{send}, \mathtt{sid}, v)$ from $P_i = P_s$ (or $\mathcal{S}$ on behalf of corrupted P_s), runs the *Input Submission Procedure*, record $y := v$, and send $(\mathtt{leakage}, \mathtt{sid}, v)$ to $\mathcal{S}$.
- Upon receiving $(\mathtt{replace}, \mathtt{sid}, v)$ from $\mathcal{S}$ (v can be $\perp$), if P_s is corrupted and no honest party has received their output yet, then update $y := v$.
- Upon receiving $(\mathtt{fetch}, \mathtt{sid})$ from $P_i \in \mathcal{P}$ (or $\mathcal{S}$ on behalf of corrupted P_i), run the *Input Submission* and *Output Release Procedures*, and send $(\mathtt{fetched}, \mathtt{sid}, v)$ to $\mathcal{S}$ and any messages set by the *Output Release Procedure* to P_i.

Input Submission Procedure: If $\mathtt{part}_i = 0$ then do:

1. Update $D_i^{\mathtt{in}} = D_i^{\mathtt{in}} - 1$.
2. If $D_i^{\mathtt{in}} = 0$ then set $\mathtt{part}_i = 1$.

Output Release Procedure: If $\sum_{j=1}^{n} \mathtt{part}_j \geq n - t$ and $y \neq \perp$, then do:

1. Update $D_i^{\mathtt{out}} = D_i^{\mathtt{out}} - 1$.
2. If $D_i^{\mathtt{out}} = 0$ then set $(\mathtt{output}, \mathtt{sid}, y)$ to be sent to P_i.

Agreement on a Common Subset. The agreement on a common subset (ACS) primitive allows the parties to agree on a set of at least $n - t$ parties that satisfy a certain property (a so-called ACS property).

Definition 1. *Let $\mathcal{P}$ be a set of n parties and let Q be a property that can be influenced by multiple protocols running in parallel. Every party $P_i \in \mathcal{P}$ can decide for every party $P_j \in \mathcal{P}$ based on the protocols running in parallel whether P_j satisfies the property towards P_i or not. If it does, we say P_i* likes *P_j* for *Q or simply P_i* likes *P_j if the property Q is clear from the context. We require that once a party likes another party, it cannot unlike it. Such a property Q is called an* ACS property *if for every pair of uncorrupted parties $(P_i, P_j) \in \mathcal{P}^2$ we have that P_i will eventually like P_j.*

We state the traditional property-based formalization of ACS.

Definition 2. *Let Π be an n-party protocol where all parties take as input a global ACS property Q and each party P_i outputs a set S_i of parties. We say that Π is a t-resilient ACS protocol for Q if the following holds whenever up to t parties are corrupted:*

- *Consistency: Each honest party outputs the same set $S_i = S$.*
- *Set quality: Each output set has size at least $n - t$, and for each $P_i \in S$ there exists at least one honest party P_j that likes P_i for Q.*

– *Termination: All honest parties eventually terminate.*

We also describe a functionality for ACS. In the functionality, each party can input $k \in [n]$. And it is guaranteed that every party receives at least $n - t$ such indices. Moreover, any index k input by a party P_i will also be eventually input by P_j.

For an ACS property Q, we will say that the parties invoke $\mathcal{F}_{\mathsf{acs}}$, meaning that each party P_i inputs k to the functionality as soon as P_i likes P_k.

Functionality $\mathcal{F}_{\mathsf{acs}}$

The functionality $\mathcal{F}_{\mathsf{rbc}}$ runs with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, a resiliency threshold $t \leq n/3$, and an ideal adversary $\mathcal{S}$. On first activation, verify $\texttt{sid} = (\mathcal{P}, \texttt{sid}')$ where $|\mathcal{P}| = n$. For each $P_i \in \mathcal{P}$, initialize $S_i := \varnothing$ (submitted elements), $P_i := \varnothing$ (pending inputs), a priority order $\rho_i := \perp$, and delays $D_i^{\mathsf{in}} = D_i^{\mathsf{out}} := 1$. Also, initialize $A := \perp$ and $Y := \perp$.

- Upon receiving $(\texttt{delay}, \texttt{sid}, P_i, \texttt{type}, D)$ from $\mathcal{S}$, where $\texttt{type} \in \{\texttt{in}, \texttt{out}\}$ and $D \in \mathbb{Z}$ (represented in unary), set $D_i^{\texttt{type}} := \max(1, D_i^{\texttt{type}} + D)$.
- Upon receiving $(\texttt{input}, \texttt{sid}, k)$ from $P_i \in \mathcal{P}$ (or $\mathcal{S}$ for corrupted P_i), with $k \in [n]$, set $P_i := P_i \cup \{k\}$, run the *Input Submission Procedure*, and send $(\texttt{leakage}, \texttt{sid}, P_i, k)$ to $\mathcal{S}$.
- Upon receiving $(\texttt{order}, \texttt{sid}, P_i, \rho)$ from $\mathcal{S}$, where ρ encodes a total order on $[n]$ (e.g., a permutation/ranking), record $\rho_i := \rho$.
- Upon receiving $(\texttt{adv-input}, \texttt{sid}, X)$ from $\mathcal{S}$, if $X \subseteq [n]$ and $|X| \geq n - t$, set $A := X$.
- Upon receiving $(\texttt{fetch}, \texttt{sid})$ from $P_i \in \mathcal{P}$ (or $\mathcal{S}$ for corrupted P_i), run the *Input Submission Procedure* and the *Output Release Procedure*, then send $(\texttt{fetched}, \texttt{sid}, P_i)$ to $\mathcal{S}$ along with any messages sent to P_i by the *Output Release Procedure*.

Input Submission Procedure. If P_i has provided at least one input, then:

1. $D_i^{\mathsf{in}} = D_i^{\mathsf{in}} - 1$.
2. If $D_i^{\mathsf{in}} = 0$ and $P_i \neq \varnothing$, choose the next element to submit as follows:
   - If $\rho_i \neq \perp$, let $k^\star$ be the element of P_i with the smallest rank under ρ_i;
   - Otherwise, arbitrarily choose $k^\star$ from P_i.

   Update $P_i := P_i \setminus \{k^\star\}$ and $S_i := S_i \cup \{k^\star\}$, then reset $D_i^{\mathsf{in}} := 1$.

Output Release Procedure. For each value $k \in [n]$, count how many parties have already submitted k (i.e., in how many S_j it appears). Call a value *eligible* if it appears in at least $t + 1$ parties' submitted sets. If there are at least $n - t$ eligible values, then:

1. $D_i^{\mathsf{out}} = D_i^{\mathsf{out}} - 1$.
2. If $D_i^{\mathsf{out}} = 0$ and $Y = \perp$, then set Y as follows:
   - If $A \neq \perp$ and every element of A is eligible and $|A| \geq n - t$, set $Y := A$;
   - Otherwise, set Y to be the set of all eligible values (which is of size at least $n - t$).

   In any case, send $(\texttt{output}, \texttt{sid}, Y)$ to P_i.

B Security Proof for Triple Generation

We first construct the simulator of the sub-protocols as follows.

The simulation of $\Pi_{\texttt{Sh}}$:

Simulator $\mathcal{S}_{\texttt{Sh}}$

For honest dealer D

Distributing Phase

1: For each corrupted party P_i and $\ell \in [N']$, $\mathcal{S}$ randomly samples degree-t row polynomials $\tilde{f}_{\ell,i}(x)$ and degree-$2t$ column polynomial $\tilde{g}_{\ell,i}(y)$ such that $\tilde{f}_{\ell,i}(\alpha_j) = \tilde{g}_{\ell,j}(\alpha_i)$ for all corrupted P_i, P_j and sends them to P_i on behalf of D.

2: When each honest party P_i receives his column polynomials from D, $\mathcal{S}$ sets P_i's input of $\mathcal{F}_{\mathsf{ra}}$ as 1. $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{ra}}$. For each honest party, when he terminates $\mathcal{F}_{\mathsf{ra}}$ with 1, $\mathcal{S}$ considers that this honest party terminates the distributing phase.

Recover Phase

3: During the reconstruction of Row Polynomials:

- For each honest P_i, when he receives $\{g_{\ell,j}(\alpha_i)\}_{\ell\in[N']}$ from $t+1$ distinct P_j, we denote the set of these $t+1$ parties as $\mathcal{M}_{i,0}$. $\mathcal{S}$ interpolates degree-t polynomial $\delta^{(f)}_{\ell,i}(x)$ for each $\ell \in [N']$ such that $\delta^{(f)}_{\ell,i}(\alpha_j) = 0$ for each honest $P_j \in \mathcal{M}_{i,0}$ and $\delta^{(f)}_{\ell,i}(\alpha_j) = g_{\ell,i}(\alpha_j) - \tilde{g}_{\ell,i}(\alpha_j)$ for each corrupted $P_j \in \mathcal{M}_{i,0}$.
- For each corrupted P_j, $\mathcal{S}$ sends $\tilde{f}_{\ell,i}(\alpha_j) + \delta^{(f)}_{\ell,i}(\alpha_j)$ for all $\ell \in [N']$ to P_j on behalf of each honest P_i (after P_i has interpolated his row polynomials).

4: During the reconstruction of Column Polynomials:

- For each honest P_i, when he receives $\{f_{\ell,j}(\alpha_i)\}_{\ell\in[N']}$ from $2t+1$ distinct P_j, we denote the set of these $2t+1$ parties as $\mathcal{M}_{i,1}$. $\mathcal{S}$ interpolates degree-$2t$ polynomial $\delta^{(g)}_{\ell,i}(y)$ for each $\ell \in [N']$ such that $\delta^{(g)}_{\ell,i}(\alpha_j) = \delta^{(f)}_{\ell,j}(\alpha_i)$ for each honest $P_j \in \mathcal{M}_{i,0}$ and $\delta^{(g)}_{\ell,i}(\alpha_j) = f_{\ell,j}(\alpha_i) - \tilde{f}_{\ell,j}(\alpha_i)$ for each corrupted $P_j \in \mathcal{M}_{i,0}$.
- $\mathcal{S}$ computes $\{\delta^{(g)}_{\ell,i}(\beta_k)\}_{\ell\in[N'],k\in[0,t]}$ for each honest P_i. Each $\delta^{(g)}_{\ell,i}(\beta_k)$ is an additive error on this honest party's share.

For corrupted dealer D

Distributing Phase

1: $\mathcal{S}$ honestly follows the protocol to execute each honest party and $\mathcal{F}_{\mathsf{ra}}$.

2: Upon all parties terminate $\mathcal{F}_{\mathsf{ra}}$ with 1, let $\mathcal{H}$ be the set of the first $t+1$ honest parties who receives their column polynomials $g_{\ell,i}(y)$ from the dealer D, $\mathcal{S}$ interpolates degree-$(t,2t)$ bivariate polynomial $\tilde{F}_\ell(x,y)$ based on $\{g_{\ell,i}(y)\}_{i\in\mathcal{H}}$ for all $\ell \in [N']$.

3: For each corrupted P_i, $\mathcal{S}$ computes $g_{\ell,i}(y) = \tilde{F}_\ell(\alpha_i, y)$ for all $\ell \in [N']$.

Recover Phase

4: $\mathcal{S}$ honestly follows the protocol to execute each honest party. When an honest party has reconstructed his column polynomials $g_{\ell,i}(y)$, $\mathcal{S}$ computes $\delta^{(g)}_{\ell,i}(y) = \tilde{F}_\ell(\alpha_i, y) - g_{\ell,i}(y)$.

5: $\mathcal{S}$ computes $\{\delta^{(g)}_{\ell,i}(\beta_k)\}_{\ell\in[N'],k\in[0,t]}$ for each honest P_i. Each $\delta^{(g)}_{\ell,i}(\beta_k)$ is an additive error on this honest party's share. $\mathcal{S}$ also computes corrupted parties' shares.

The simulation of $\Pi_{\mathtt{PubRec}}$:

Simulator $\mathcal{S}_{\mathtt{PubRec}}$

$\mathcal{S}$ takes the whole sharings $[x_1]_t, \ldots, [x_N]_t$ and the additive errors on honest parties' shares of $[x_1]_t, \ldots, [x_N]_t$ as input.

1: In Step 1, $\mathcal{S}$ does the following simulation for each group of sharings $[s^{(0)}]_t, \ldots, [s^{(t)}]_t$.

(1). In Step (1), we denote honest P_i's additive errors on his shares by $\delta_i^{(0)}, \ldots, \delta_i^{(t)}$. For each honest party P_i, we denote his shares of $\{[s^{(k)}]_t\}_{k\in[0,t]}$ as $\{s_i^{(k)}\}_{k\in[0,t]}$, $\mathcal{S}$ sends $\sum_{k=0}^{t}(s_i^{(k)}+\delta_i^{(k)})\cdot\alpha_j^k$ to each corrupted P_j.

(2). In Step (2), for each honest party P_i, when P_i receives $2t+1$ messages from all parties, if one of them is the failure symbol $\perp$, $\mathcal{S}$ forwards it to all parties and sets `Fail` as P_i's final output. Otherwise, $\mathcal{S}$ checks whether these $2t+1$ shares lie on degree-t polynomials. If not, $\mathcal{S}$ also forwards the failure symbol to all parties and sets `Fail` as P_i's final output. Otherwise, $\mathcal{S}$ does the following steps.

- $\mathcal{S}$ follows the protocol to compute each corrupted party P_j's share of $[f(\alpha_i)]_t$, denoted by $f_{i,j}$. Let $\tilde{f}_{i,j}$ denote the message received from corrupted P_j, $\mathcal{S}$ computes $\delta^{(f)}_{i,j} = f_{i,j} - \tilde{f}_{i,j}$.
- For each honest P_j, $\mathcal{S}$ computes $\delta^{(f)}_{i,j} = \sum_{k=0}^{t} \delta_i^{(k)} \cdot \alpha_j^k$.

Then $\mathcal{S}$ interpolates a degree-t polynomial $\delta_i^{(f)}(x)$ such that $\delta_i^{(f)}(\alpha_j) = \delta^{(f)}_{i,j}$ for these $2t+1$ distinct P_j. $\mathcal{S}$ also computes $f(\alpha_i)$ and sends $f(\alpha_i) + \delta_i^{(f)}(\alpha_0)$ to all corrupted parties on behalf of honest P_i.

(3). In Step (3), $\mathcal{S}$ honestly executes each honest party to compute their reconstruction results, and we denote each honest P_i's reconstruction results as $s_i^{(0)}, \ldots, s_i^{(t)}$. If the honest party received a failure symbol or these $2t+1$ values do not lie on a degree-t polynomial, $\mathcal{S}$ forwards the failure symbol to all parties and sets `Fail` as the honest party's final output.

2: In Step 2, for each honest party P_i and corrupted P_j, $\mathcal{S}$ honestly follows the protocol to sample $r_{i,j}$, computes $X_{i,j}$ and sends $X_{i,j}$ to P_j on behalf of P_i. For each honest P_j, $\mathcal{S}$ simulates the behavior of sending messages from P_i to P_j.

3: In Step 3, $\mathcal{S}$ honestly executes each honest party to do the simulation. If an honest P_i accepts his reconstruction results, let $\mathcal{M}_i$ be the set of $2t+1$ distinct P_j that P_i received their $X_{j,i}$, then $\mathcal{S}$ checks whether $s_i^{(k)} = s_j^{(k)}$ are true for all $k \in [0, 2t]$, all groups of sharings, and all honest $P_j \in \mathcal{M}_i$. If not, $\mathcal{S}$ aborts the simulation.

4: Let P_j be the first honest party who accepts his reconstruction results, $\mathcal{S}$ computes the additive error on the public reconstruction output $\delta^{(k)} = s^{(k)} - s_j^{(k)}$ for $k \in [0, 2t]$ and all groups of sharings.

Simulator $\mathcal{S}_{\texttt{GenTriple}}$

In the following, we use $\mathcal{C}orr$ to denote the set of corrupted parties. We also assume that there are exactly t corrupted parties. If $|\mathcal{C}orr| = 2\tilde{N} < t$, we will view the first $t - 2\tilde{N}$ honest parties as corrupted parties.

1: **Distributing Random Sharings.** $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{Sh}}$ for each party P_i. Upon terminating $\mathcal{S}_{\texttt{Sh}}$ invoked by P_i,
- $\mathcal{S}$ learns corrupted parties' shares of $\{[a_\ell^{(i,j)}]_t, [b_\ell^{(i,j)}]_t, [r_\ell^{(i,j)}]_t\}_{\ell \in [N'], j \in [n]}$.
- For each honest party P_j who has received his shares, $\mathcal{S}$ learns the additive errors on his output shares, denoted by $\{\delta_{\ell,j}^{(i,j,a)}, \delta_{\ell,j}^{(i,j,b)}, \delta_{\ell,j}^{(i,j,r)}\}_{\ell \in [N']}$. When the dealer P_i is corrupted, $\mathcal{S}$ knows P_j's output shares and the whole sharing chosen by P_i.

2: **Preparing Random Triples by Party P_i.** $\mathcal{S}$ does the following steps. For simplicity, we simplify the notation of sharings distributed by P_i, P_j by $\{[a_\ell^{(i)}]_t, [b_\ell^{(i)}]_t, [r_\ell^{(i)}]_t\}_{\ell \in [N']}$ and $\{[a_\ell^{(j)}]_t, [b_\ell^{(j)}]_t, [r_\ell^{(j)}]_t\}_{\ell \in [N']}$, and the honest P_k's additive errors are denoted by $\{\delta_{\ell,k}^{(i,a)}, \delta_{\ell,k}^{(i,b)}, \delta_{\ell,k}^{(i,r)}\}_{\ell \in [N']}$ and $\{\delta_{\ell,k}^{(j,a)}, \delta_{\ell,k}^{(j,b)}, \delta_{\ell,k}^{(j,r)}\}_{\ell \in [N']}$.
- When both P_i and P_j are honest:
  (1). In Step (1), for each corrupted party P_k, $\mathcal{S}$ randomly samples values as $\{\tilde{o}_{\ell,k}\}_{\ell \in [N']}$ and sends them to P_k on behalf of P_j. For each honest P_k, $\mathcal{S}$ simulates the behavior of sending messages from P_j to P_k.
  (2). In Step (2), $\mathcal{S}$ computes the additive errors on each honest P_k's shares of $[a_\ell]_t, [b_\ell]_t$ as follows:

$$\tilde{\delta}_{\ell,k}^{(a)} = \delta_{\ell,k}^{(i,a)} + \delta_{\ell,k}^{(j,a)}, \tilde{\delta}_{\ell,k}^{(b)} = \delta_{\ell,k}^{(i,b)} + \delta_{\ell,k}^{(j,b)}.$$

  (3). In Step (3), $\mathcal{S}$ computes $\tilde{d}_{\ell,k} = a_{\ell,k} \cdot b_{\ell,k} - \tilde{o}_{\ell,k}$ for each corrupted party P_k.
  (4). In Step (4), when P_i gets $\mathcal{Q}$ of size $2t+1$, for each corrupted $P_k \in \mathcal{Q}$, $\mathcal{S}$ computes $\delta_{\ell,k}^{(d)} = d_{\ell,k} - \tilde{d}_{\ell,k}$. For each honest $P_k \in \mathcal{Q}$, $\mathcal{S}$ sets $\delta_{\ell,k}^{(d)}$ to be the following form.

$$\delta_{\ell,k}^{(d)} = a_{\ell,k} \cdot \tilde{\delta}_{\ell,k}^{(b)} + b_{\ell,k} \cdot \tilde{\delta}_{\ell,k}^{(a)} + \tilde{\delta}_{\ell,k}^{(a)} \cdot \tilde{\delta}_{\ell,k}^{(b)}.$$

  Where $a_{\ell,k}, b_{\ell,k}$ are P_k's shares of $[a_\ell]_t, [b_\ell]_t$, $\mathcal{S}$ also simulates the behavior of sending messages from P_i to P_j.
  (5). In Step (5), $\mathcal{S}$ simulates the behavior of sending messages from P_j to P_i.

(6). In Step (6), let $\delta_\ell^{(z)}$ denote the error term on the reconstruction result z_ℓ and $\mathcal{H}_i$ be the set of honest parties in $\mathcal{Q}$, $\mathcal{S}$ sets the form of $\delta_\ell^{(z)}$ as follows:

$$\begin{aligned}
\delta_\ell^{(z)} &= \sum_{k\in\mathcal{Q}} \delta_{\ell,k}^{(d)} \cdot I_k(\alpha_0) \\
&= \sum_{k\in\mathcal{H}_i} \left(a_{\ell,k}\cdot\tilde{\delta}_{\ell,k}^{(b)} + b_{\ell,k}\cdot\tilde{\delta}_{\ell,k}^{(a)} + \tilde{\delta}_{\ell,k}^{(a)}\cdot\tilde{\delta}_{\ell,k}^{(b)}\right)\cdot I_k(\alpha_0) + \sum_{k\in\mathcal{Q}\setminus\mathcal{H}_i} \delta_{\ell,k}^{(d)}\cdot I_k(\alpha_0) \\
&= \sum_{k\in\mathcal{H}_i}\left((a_\ell\cdot I_0'(\alpha_k) + \sum_{u\in\mathcal{C}orr} a_{\ell,u}\cdot I_u'(\alpha_k))\cdot\tilde{\delta}_{\ell,k}^{(b)}\right) \\
&+ \sum_{k\in\mathcal{H}_i}\left((b_\ell\cdot I_0'(\alpha_k) + \sum_{u\in\mathcal{C}orr} b_{\ell,u}\cdot I_u'(\alpha_k))\cdot\tilde{\delta}_{\ell,k}^{(a)}\right) + \sum_{k\in\mathcal{Q}\setminus\mathcal{H}_i}\delta_{\ell,k}^{(d)}\cdot I_k(\alpha_0) \\
&= \sum_{k\in\mathcal{H}_i}(\tilde{\delta}_{\ell,k}^{(b)}\cdot a_\ell + \tilde{\delta}_{\ell,k}^{(a)}\cdot b_\ell)I_0'(\alpha_k) + \sum_{k\in\mathcal{Q}\setminus\mathcal{H}_i}\delta_{\ell,k}^{(d)}\cdot I_k(\alpha_0) \\
&+ \sum_{k\in\mathcal{H}_i}\sum_{u\in\mathcal{C}orr}(\tilde{\delta}_{\ell,k}^{(b)}a_{\ell,u} + \tilde{\delta}_{\ell,k}^{(a)}b_{\ell,u})I_u'(\alpha_k)
\end{aligned}$$

where $I_k(x), I_u'(x)$ are Lagrange interpolation polynomial. We can simplify $\delta_\ell^{(z)}$ into the following form:

$$\delta_\ell^{(z)} = \alpha_\ell\cdot a_\ell + \beta_\ell\cdot b_\ell + \gamma_\ell.$$

where α_ℓ, β_ℓ, and γ_ℓ can be computed by $\mathcal{S}$. Then $\mathcal{S}$ randomly samples a value as $z_\ell + \delta_\ell^{(z)}$.

– When P_i is honest and P_j is corrupted:

(1). In Step (1), $\mathcal{S}$ receives $o_{\ell,k}$ from P_j on behalf of honest P_k.

(2). In Step (2), $\mathcal{S}$ computes each honest P_k's additive errors $\tilde{\delta}_{\ell,k}^{(a)}, \tilde{\delta}_{\ell,k}^{(b)}$.

(3). In Step (3), $\mathcal{S}$ simulates the behavior of sending messages from P_k to P_i.

(4). In Step (4), upon getting $\mathcal{Q}$ of size $2t+1$, for each corrupted $P_k\in\mathcal{Q}$, $\mathcal{S}$ computes $o_{\ell,k} = a_{\ell,k}\cdot b_{\ell,k} - d_{\ell,k}$. Then $\mathcal{S}$ interpolates the sharing $[\tilde{o}_\ell]_{2t}$ based on $\{o_{\ell,k}\}_{k\in\mathcal{Q}}$. Then $\mathcal{S}$ sends $\mathcal{Q}$ to P_j on behalf of P_i.

(5). In Step (5), upon receiving the message from P_j, $\mathcal{S}$ computes the difference between the received messages and $\tilde{o}_\ell + r_\ell^{(j)}$, denoted by $\delta_\ell^{(o)}$.

(6). $\mathcal{S}$ sets $\delta_\ell^{(z)}$ to be the following form:

$$\delta_\ell^{(z)} = \alpha_\ell\cdot a_\ell + \beta_\ell\cdot b_\ell + \gamma_\ell.$$

where $\alpha_\ell = \sum_{k\in\mathcal{H}_i}\tilde{\delta}_{\ell,k}^{(b)}\cdot I_0'(\alpha_k), \beta_\ell = \sum_{k\in\mathcal{H}_i}\tilde{\delta}_{\ell,k}^{(a)}\cdot I_0'(\alpha_k)$, and γ_ℓ is as follows.

$$\gamma_\ell = \sum_{k\in\mathcal{H}_i}\sum_{u\in\mathcal{C}orr}(\tilde{\delta}_{\ell,k}^{(b)}a_{\ell,u} + \tilde{\delta}_{\ell,k}^{(a)}b_{\ell,u})I_u'(\alpha_k) + \sum_{k\in\mathcal{Q}\setminus\mathcal{H}_i}\delta_{\ell,k}^{(d)}\cdot I_k(\alpha_0) + \delta_\ell^{(o)}.$$

– When P_i is corrupted and P_j is honest:

(1). In Step (1), For each corrupted party P_k, $\mathcal{S}$ randomly samples values as $\{o_{\ell,k}\}_{\ell\in[N']}$ and sends them to P_k on behalf of P_j. For each honest P_k, $\mathcal{S}$ simulates the behavior of sending messages between P_j and P_k.

(2). In Step (2), for each honest party P_k, $\mathcal{S}$ randomly samples value as $d_{\ell,k}$ and sends it to P_i on behalf of P_k.

(3). In Step (5), upon receiving the set $\mathcal{Q}$ from P_i, if $\mathcal{S}$ has sent messages to P_i on behalf of all honest $P_k\in\mathcal{Q}$, $\mathcal{S}$ randomly samples a value $\tilde{o}_\ell$ as $o_\ell + r_\ell^{(j)}$ and sends it to P_i on behalf of P_j.

(4). $\mathcal{S}$ locally computes $d_{\ell,k}$ for all corrupted $P_k\in\mathcal{Q}$, when $\mathcal{S}$ receives z_ℓ from P_i's broadcast later, $\mathcal{S}$ computes the difference between the received message z_ℓ and $d_{\ell,k} + \tilde{o}_\ell + r_\ell^{(i)}$, denoted by $\delta_\ell^{(o)}$. Then $\mathcal{S}$ does the same thing as the case when P_i is honest and P_j is corrupted to set the form of $\delta_\ell^{(z)} = \alpha_\ell\cdot a_\ell + \beta_\ell\cdot b_\ell + \gamma_\ell$.

– When both P_i and P_j are corrupted, $\mathcal{S}$ honestly execute each honest party to do simulation. For each triple $[\tilde{a}_\ell^{(i,j)}]_t, [\tilde{b}_\ell^{(i,j)}]_t, [\tilde{c}_\ell^{(i,j)}]_t$, $\mathcal{S}$ computes the corresponding additive errors on honest P_k's shares as follows:

$$\tilde{\delta}_{\ell,k}^{(a)} = \delta_{\ell,k}^{(i,a)} + \delta_{\ell,k}^{(j,a)}, \tilde{\delta}_{\ell,k}^{(b)} = \delta_{\ell,k}^{(i,b)} + \delta_{\ell,k}^{(j,b)}.$$

where $\tilde{\delta}_{\ell,k}^{(a)}, \tilde{\delta}_{\ell,k}^{(b)}$ denote the additive errors on P_k's shares of $[\tilde{a}_\ell^{(i,j)}]_t, [\tilde{b}_\ell^{(i,j)}]_t$. For the additive error $\tilde{\delta}_{\ell,k}^{(c)}$ on P_k's share of $[\tilde{c}_\ell^{(i,j)}]_t$, later when all honest parties receive the messages $\tilde{z}_\ell$ from P_i's broadcast, $\mathcal{S}$ computes:

$$\tilde{\delta}_{\ell,k}^{(c)} = \tilde{z}_\ell + \delta_{\ell,k}^{(i,r)} + \delta_{\ell,k}^{(j,a)} - a_\ell \cdot b_\ell - r_\ell^{(i)} - r_\ell^{(j)}.$$

3: **Distributing Random Triples by Party P_i.** $\mathcal{S}$ honestly follows the protocol to execute each honest party P_i to broadcast the set $\texttt{Ready}_i$ and the corresponding $\{\tilde{z}_\ell^{(i,j)}\}_{\ell\in[N']}$. Upon receiving $\{\tilde{z}_\ell^{(i,j)}\}_{\ell\in[N']}$, $\mathcal{S}$ computes corrupted parties' shares of $[\tilde{c}_\ell^{(i,j)}]_t = \tilde{z}_\ell^{(i,j)} - [r_\ell^{(i,j)}]_t - [r_\ell^{(j,i)}]_t$ for all $\ell \in [N']$. For triples generated by virtual party $V_{i,j}$ consist of corrupted P_i, P_j, $\mathcal{S}$ knows the whole sharings $([\tilde{a}_\ell^{(i,j)}]_t, [\tilde{b}_\ell^{(i,j)}]_t, [\tilde{c}_\ell^{(i,j)}]_t)$, honest parties' shares and their additive errors. Otherwise, if at least one party on $V_{i,j}$ is honest:

– For each honest party P_k, $\mathcal{S}$ knows the additive errors on their shares of $[\tilde{a}_\ell^{(i,j)}]_t, [\tilde{b}_\ell^{(i,j)}]_t$, denoted by $\delta_{\ell,k}^{(i,j,a)}, \delta_{\ell,k}^{(i,j,b)}$, respectively.

– The error $\delta_{\ell,k}^{(i,j,c)}$ on honest P_k's share of $[\tilde{c}_\ell^{(i,j)}]_t$ is in the form of $\delta_\ell^{(i,j,z)} - \delta_{\ell,k}^{(i,j,r)} - \delta_{\ell,k}^{(j,i,r)}$, and $\delta_\ell^{(i,j,z)}$ is as follows:

$$\delta_\ell^{(i,j,z)} = \alpha_\ell^{(i,j)} \cdot \tilde{a}_\ell^{(i,j)} + \beta_\ell^{(i,j)} \cdot \tilde{b}_\ell^{(i,j)} + \gamma_\ell.$$

Where $\alpha_\ell^{(i,j)}, \beta_\ell^{(i,j)}$, and $\gamma_\ell^{(i,j)}$ are known to $\mathcal{S}$. For simplicity, we set $\gamma_{\ell,k} = \gamma_\ell - \delta_{\ell,k}^{(i,j,r)} - \delta_{\ell,k}^{(j,i,r)}$, then the error $\delta_{\ell,k}^{(i,j,c)}$ on honest P_k's share of $[\tilde{c}_\ell^{(i,j)}]_t$ is in the form of:

$$\delta_{\ell,k}^{(i,j,c)} = \alpha_\ell^{(i,j)} \cdot \tilde{a}_\ell^{(i,j)} + \beta_\ell^{(i,j)} \cdot \tilde{b}_\ell^{(i,j)} + \gamma_{\ell,k}.$$

4: **Determine the Set of Successful Dealers.** $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{acs}}$ and learns the set of dealers $\mathcal{D}$.

5: **Extract Random Beaver Triples.** In the following, we will say a virtual party $V_{i,j}$ is corrupted if both P_i and P_j are corrupted. Otherwise, we will say $V_{i,j}$ is honest. We denote the triples by $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=1}^{|\mathcal{V}|}$. For each honest party P_k, we denote his errors on his triples as $\{\delta_{i,k}^{(a)}, \delta_{i,k}^{(b)}, \delta_{i,k}^{(c)}\}_{i=1}^{|\mathcal{V}|}$. $\mathcal{S}$ proceeds as follows.

(1). In Step (1), $\mathcal{S}$ follows the protocol to set corrupted parties' shares of $[f(\beta_i)]_t$ and $[g(\beta_i)]_t$.

(2). In Step (2), $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{PubRec}}$ and takes the following values as inputs.

- Corrupted parties' shares of $[f(\beta_i) + a_i]_t, [g(\beta_i) + b_i]_t$.
- For each honest party P_k, $\mathcal{S}$ interpolates a degree-$2\tilde{N}$ polynomial $\Delta_{*,k}^{(a)}(x)$ such that $\Delta_{*,k}^{(a)}(\beta_i) = \delta_{i,k}^{(a)}$ for all $i \in [2\tilde{N}+1]$. Then $\mathcal{S}$ computes $\Delta_{*,k}^{(a)}(\beta_i) + \delta_{i,k}^{(a)}$ for all $i \in [2\tilde{N}+2, 4\tilde{N}+1]$ as the additive errors on P_k's shares of $[f(\beta_i)+a_i]_t$. $\mathcal{S}$ does the same thing to get the additive errors on P_k's shares of $[g(\beta_i) + b_i]_t$. $\mathcal{S}$ takes these additive errors as inputs.

Upon terminating $\mathcal{S}_{\texttt{PubRec}}$, $\mathcal{S}$ learns the public reconstruction results $\overline{f(\beta_i) + a_i}, \overline{g(\beta_i) + b_i}$ and the additive errors on the public reconstruction results $\lambda_i^{(1)}, \lambda_i^{(2)}$ for all $i \in [2\tilde{N}+2, 4\tilde{N}+1]$. Then $\mathcal{S}$ computes corrupted parties' shares of $[f(\beta_i) \cdot g(\beta_i)]_t$ as follows.

$$\begin{aligned}[f(\beta_i) \cdot g(\beta_i)]_t = &(\overline{f(\beta_i) + a_i}) \cdot (\overline{g(\beta_i) + b_i}) \\ &- (\overline{f(\beta_i) + a_i}) \cdot [b_i]_t \\ &- (\overline{g(\beta_i) + b_i}) \cdot [a_i]_t + [c_i]_t.\end{aligned}$$

(3). In Step (3), $\mathcal{S}$ follows the protocol to compute corrupted parties' shares of $[h(\beta_i)]_t$ for all $i \in [4\tilde{N}+1]$. For each honest party P_k, for all $i \in [2\tilde{N}+1]$, $\mathcal{S}$ sets the error term $\delta_{i,k}^{(h)} = \delta_{i,k}^{(c)} = \alpha_i a_i + \beta_i b_i + \gamma_{i,k}$. For all $i \in [2\tilde{N}+2, 4\tilde{N}+1]$, $\mathcal{S}$ sets the error $\delta_{i,k}^{(h)}$ on his share as follows:

$$\begin{aligned}\delta_{i,k}^{(h)} = &(\overline{f(\beta_i) + a_i}) \cdot (\overline{g(\beta_i) + b_i}) - (\overline{f(\beta_i) + a_i} - \lambda_i^{(1)}) \cdot (\overline{g(\beta_i) + b_i} - \lambda_i^{(2)}) \\ &- (\overline{f(\beta_i) + a_i}) \cdot \delta_{i,k}^{(b)} - (\overline{g(\beta_i) + b_i}) \cdot \delta_{i,k}^{(a)} + \delta_{i,k}^{(c)} - \lambda_i^{(1)} \cdot b_{i,k} - \lambda_i^{(2)} \cdot a_{i,k}\end{aligned}$$

By replacing $a_{i,k}, b_{i,k}$ by $a_i \cdot I_0''(\alpha_k) + \sum_{p\in\mathcal{C}orr} a_{i,p} \cdot I_p''(\alpha_k), b_i \cdot I_0''(\alpha_k) + \sum_{p\in\mathcal{C}orr} b_{i,p} \cdot I_p''(\alpha_k)$, and $\delta_{i,k}^{(c)} = \alpha_i \cdot a_i + \beta_i \cdot b_i + \gamma_{i,k}$, we can rewrite $\delta_{i,k}^{(h)}$ as follows:

$$\delta_{i,k}^{(h)} = (\alpha_i - I_0''(\alpha_k)\lambda_i^{(2)})a_i + (\beta_i - I_0''(\alpha_k)\lambda_i^{(1)})b_i + \gamma_{i,k}'.$$

where $\gamma_{i,k}'$ is as follows:

$$\begin{aligned}\gamma_{i,k}' = &(\overline{f(\beta_i) + a_i}) \cdot (\overline{g(\beta_i) + b_i}) - (\overline{f(\beta_i) + a_i} - \lambda_i^{(1)}) \cdot (\overline{g(\beta_i) + b_i} - \lambda_i^{(2)}) \\ &- (\overline{f(\beta_i) + a_i}) \cdot \delta_{i,k}^{(b)} - (\overline{g(\beta_i) + b_i}) \cdot \delta_{i,k}^{(a)} + \gamma_{i,k} \\ &- \sum_{p\in\mathcal{C}orr} (\lambda_i^{(2)} \cdot a_{i,p} + \lambda_i^{(1)} \cdot b_{i,p}) \cdot I_p''(\alpha_k).\end{aligned}$$

(4). For all $i \in [4\tilde{N}+1]$, $\mathcal{S}$ sets $h'(\beta_i) = h(\beta_i) + A_i \cdot f(\beta_i) + B_i \cdot g(\beta_i)$, where A_i, B_i is computed as follows:
- For all $i \in [2\tilde{N}+1]$, we set $A_i = \alpha_i, B_i = \beta_i$.
- For all $i \in [2\tilde{N}+2, 4\tilde{N}+1]$, we set $A_i = \alpha_i - \lambda_i^{(2)}, B_i = \beta_i - \lambda_i^{(1)}$.

(5). In Step (4), $\mathcal{S}$ follows the protocol to compute corrupted parties' shares of $([f(\gamma_i)]_t, [g(\gamma_i)]_t, [h'(\gamma_i)]_t)$. For each honest party P_k, $\mathcal{S}$ records his additives errors on his shares of $([f(\beta_i)]_t, [g(\beta_i)]_t, [h'(\beta_i)]_t)$ for all $i \in [4\tilde{N}+1]$.

Simulator $\mathcal{S}_{\texttt{VerifyTriple}}$

The simulation of the verification procedure, we use L to denote the maximum number of request for each party to do verification of triples (defined in $\Pi_{\texttt{Triple}}$).
We use $\{[f(\beta_\ell)]_t, [g(\beta_\ell)]_t\}_{\ell\in[2\tilde{N}+1]}, \{[h(\beta_\ell)]_t\}_{\ell\in[4\tilde{N}+1]}$ to denote the input triples. Let $\mathcal{M} \in [4\tilde{N}+1]$ denote the set of corrupted virtual parties who generate the triples during the $\Pi_{\texttt{Triple}}$, and $\mathcal{W} := [4\tilde{N}+1] \setminus \mathcal{M}$. Note that here for a corrupted virtual party V_ℓ where $\ell \in [2\tilde{N}+2, 4\tilde{N}+1]$, the simulator can compute the sharings $[f(\beta_\ell)]_t, [g(\beta_\ell)]_t, [h(\beta_\ell)]_t$ based on V_ℓ's triples and the public reconstruction result, so also view this triples $[f(\beta_\ell)]_t, [g(\beta_\ell)]_t, [h(\beta_\ell)]_t$ are generated by the corrupted virtual parties. $\mathcal{S}$ takes the following values as input:

- Corrupted parties' shares of $\{[f(\beta_\ell)]_t, [g(\beta_\ell)]_t\}_{\ell\in[2\tilde{N}+1]}, \{[h(\beta_\ell)]_t\}_{\ell\in[4\tilde{N}+1]}$. For each $\ell \in \mathcal{M}$, the whole sharings $\{[f(\beta_\ell)]_t, [g(\beta_\ell)]_t, [h(\beta_\ell)]_t\}$.
- Each honest party P_i's additive errors on $[f(\beta_\ell)]_t, [g(\beta_\ell)]_t, [h(\beta_\ell)]_t$, which are in the form of $\delta_{\ell,i}^{(f)}, \delta_{\ell,i}^{(g)}, \delta_{\ell,i}^{(h)}$.
- The coefficients A_ℓ, B_ℓ in the following relation between $h(\beta_\ell)$ and $f(\beta_\ell), g(\beta_\ell)$ for all $\ell \in [4\tilde{N}+1]$.

$$h(\beta_\ell) = f(\beta_\ell) \cdot g(\beta_\ell) + A_\ell \cdot f(\beta_\ell) + B_\ell \cdot g(\beta_\ell).$$

$\mathcal{S}$ first invokes error corrections on $\{A_\ell\}_{\ell\in\mathcal{W}}, \{B_\ell\}_{\ell\in\mathcal{W}}$ to interpolate degree-$2\tilde{N}$ polynomials. Note that the size of $\mathcal{W}$ is at least $3\tilde{N}+1$.

- If there are more than $(L+1)n\left\lceil\frac{n+\kappa}{\kappa}\right\rceil \leq \tilde{N}/2$ errors on these values, $\mathcal{S}$ proceeds to **Process 1**.
- Otherwise, let $\mathcal{Q}$ be the set of the index of these errors, $\mathcal{S}$ randomly samples the whole sharings $([f(\beta_\ell)]_t, [g(\beta_\ell)]_t, [h(\beta_\ell)]_t)$ based on shares of corrupted parties for all $\ell \in \mathcal{Q}$. Then $\mathcal{S}$ proceeds to the **Process 2**. Let $A(x), B(x)$ be the degree-L polynomials interpolated from $\{A_\ell\}_{\ell\in\mathcal{W}}, \{B_\ell\}_{\ell\in\mathcal{W}}$ by the error correction.

Process 1

1: In Step 1, $\mathcal{S}$ honestly executes each honest P_i to sample random challenges and broadcasts them.
2: In Step 2, $\mathcal{S}$ first initializes a set $\mathcal{R}^{(f)} = \mathcal{R}^{(g)} = \varnothing$. Upon receiving r_j from P_j, $\mathcal{S}$ does the following steps.
(1). $\mathcal{S}$ locally computes corrupted parties' shares of $[f(r_j)]_t$. The secret $f(r_j)$ is in the following form:

$$f(r_j) = \sum_{\ell\in\mathcal{W}} f(\beta_\ell) \cdot I_\ell(r_j) + \sum_{\ell\in\mathcal{M}} f(\beta_\ell) \cdot I_\ell(r_j).$$

If $\mathcal{S}$ receives the challenge r_j for the first time, it randomly samples the whole sharing $[f(r_j)]_t$ based on shares of corrupted parties, sets $\{c_{\ell,j}\}_{\ell\in\mathcal{W}} = \{I_\ell(r_j)\}_{\ell\in\mathcal{W}}, C_j^{(f)} = f(r_j)$, and adds $(r_j, \{c_{\ell,j}\}_{\ell\in\mathcal{W}}, C_j^{(f)})$

to $\mathcal{R}^{(f)}$. Otherwise, $\mathcal{S}$ uses the previously sampled $[f(r_j)]_t$ and proceeds. Let $f_i^{(j)}$ denote honest P_i's shares of $[f(r_j)]_t$, $\mathcal{S}$ sends the following values to P_j as P_i's share of $[f(r_j)]_t$ on behalf of P_i:

$$f_i^{(j)} + \sum_{\ell \in [2\tilde{N}+1]} \delta_{\ell,i}^{(f)} \cdot I_\ell(r_j).$$

Where $I_\ell(x)$ denotes the Lagrange interpolation function. $\mathcal{S}$ does the same thing for P_i's share of $[g(r_j)]_t$.

(2). The secret $h(r_j)$ is in the following form:

$$\begin{aligned} h(r_j) &= \sum_{\ell \in [4\tilde{N}+1]} h(\beta_\ell) \cdot I_\ell(r_j) \\ &= \sum_{\ell \in [4\tilde{N}+1]} (f(\beta_\ell) \cdot g(\beta_\ell) + A_\ell \cdot f(\beta_\ell) + B_\ell \cdot g(\beta_\ell)) \cdot I_\ell(r_j) \\ &= f(r_j) \cdot g(r_j) + \sum_{\ell \in [4\tilde{N}+1]} (A_\ell \cdot f(\beta_\ell) + B_\ell \cdot g(\beta_\ell)) \cdot I_\ell(r_j) \\ &= f(r_j) \cdot g(r_j) + \delta_j + \sum_{\ell \in \mathcal{W}} A'_{\ell,j} \cdot f(\beta_\ell) + B'_{\ell,j} \cdot g(\beta_\ell) \end{aligned}$$

where $A'_{\ell,j} = A_\ell \cdot I_\ell(r_j), B'_{\ell,j} = B_\ell \cdot I_\ell(r_j), \delta_j = \sum_{\ell \in \mathcal{M}} A'_{\ell,j} \cdot f(\beta_\ell) + B'_{\ell,j} \cdot g(\beta_\ell)$. $\mathcal{S}$ first checks whether $\{A'_{\ell,j}\}_{\ell \in \mathcal{W}}$ are linear dependent to some values $\{c_{\ell,k}\}_{\ell \in \mathcal{W}} \in \mathcal{R}^{(f)}$.

- If true, $\mathcal{S}$ computes the secret $\sum_{\ell \in \mathcal{W}} A'_{\ell,j} \cdot f(\beta_\ell)$ accordingly.
- Otherwise, $\mathcal{S}$ randomly samples a value C_j as $\sum_{\ell \in \mathcal{W}} A'_{\ell,j} \cdot f(\beta_\ell)$, sets $\{c_{\ell,j}\}_{\ell \in \mathcal{W}} = \{A'_{\ell,j}\}_{\ell \in \mathcal{W}}$, and adds $(r_j, \{c_{\ell,j}\}_{\ell \in \mathcal{W}}, C_j)$ to $\mathcal{R}^{(f)}$.

$\mathcal{S}$ does the same thing for $\{B'_{\ell,j}\}_{\ell \in \mathcal{W}}$. Then $\mathcal{S}$ computes the secret $h(r_j)$, corrupted parties' shares of $[h(r_j)]_t$, and randomly samples the whole sharing $[h(r_j)]_t$ based on corrupted parties' shares and the secret. Let $h_i^{(j)}$ denote honest P_i's shares of $[h(r_j)]_t$, $\mathcal{S}$ sends the following value to P_j as P_i's share of $[h(r_j)]_t$ on behalf of P_i:

$$h_i^{(j)} + \sum_{\ell \in [4\tilde{N}+1]} \delta_{\ell,i}^{(h)} \cdot I_\ell(r_j).$$

(3). For the rest of the steps, $\mathcal{S}$ honestly executes each honest party to do the simulation. If honest parties eventually terminate with `Success`, $\mathcal{S}$ aborts the simulation.

Process 2

1: In Step 1, $\mathcal{S}$ honestly executes each honest P_i to sample random challenges and broadcasts them.

2: In Step 2, Upon receiving r_j from P_j, $\mathcal{S}$ does the following steps.

(1). $\mathcal{S}$ does the same thing as in **Process 1** to generate honest P_i's shares of $[f(r_j)]_t, [g(r_j)]_t$ and sends them to P_j.

(2). $\mathcal{S}$ computes the secret $h(r_j)$ as follows:

$$\begin{aligned} h(r_j) = f(r_j) \cdot g(r_j) + A(r_j) \cdot f(r_j) + B(r_j) \cdot g(r_j) \\ + \sum_{\ell \in \mathcal{Q}} ((A_\ell - A(\beta_\ell)) \cdot f(\beta_\ell) + (B_\ell - B(\beta_\ell)) \cdot g(\beta_\ell)) \cdot I_\ell(r_j). \end{aligned}$$

Then $\mathcal{S}$ locally computes corrupted parties' shares of $[h(r_j)]_t$, and randomly samples the whole sharing $[h(r_j)]_t$ based on the secret and shares of corrupted parties. Let $h_i^{(j)}$ denote honest P_i's shares of $[h(r_j)]_t$, $\mathcal{S}$ sends the following value to P_j as P_i's share of $[h(r_j)]_t$ on behalf of P_i:

$$h_i^{(j)} + \sum_{\ell \in [4\tilde{N}+1]} \delta_{\ell,i}^{(h)} \cdot I_\ell(r_j).$$

3: For the rest of the steps, $\mathcal{S}$ honestly executes each honest party to do the simulation. When any honest party terminates with `Success`, $\mathcal{S}$ learns the set $\mathcal{M}$ which contains $t+1$ honest parties, and the secrets of triples interpolated by their shares should be valid (satisfy the multiplication relation, but note that $\mathcal{S}$ cannot interpolate these secrets). In this case, $\mathcal{S}$ additionally does the following verification.

– $\mathcal{S}$ interpolates degree-$2\tilde{N}$ polynomials $A'(x), B'(x)$ such that each $A'(\beta_\ell), B'(\beta_\ell)$ is interpolated by $\{\delta^{(f)}_{\ell,i}\}_{i\in\mathcal{M}}, \{\delta^{(g)}_{\ell,i}\}_{i\in\mathcal{M}}$ for all $\ell \in [2\tilde{N}+1]$. Then $\mathcal{S}$ checks whether $A'(x) = A(x), B'(x) = B(x)$. If true, $\mathcal{S}$ proceeds. Otherwise, $\mathcal{S}$ aborts the simulation.
– $\mathcal{S}$ interpolates a degree-$4\tilde{N}$ polynomial $C(x)$ such that each $C(\beta_\ell)$ is interpolated $\{\delta^{(h)}_{\ell,i}\}_{i\in\mathcal{M}}$ for all $\ell \in [4\tilde{N}+1]$. Then $\mathcal{S}$ checks whether $C(x) = A(x)\cdot B(x)$. If true, $\mathcal{S}$ proceeds. Otherwise, $\mathcal{S}$ aborts the simulation.

If $\mathcal{S}$ succeeds the verification, then we know that the parties in $\mathcal{M}$ hold valid triples where the secrets lie on polynomials $f' = f + A, g' = g + B$, and $h' = h + C$, and it satisfies $h' = f' \cdot g'$. Then we can view parties in $\mathcal{M}$ hold correct shares of $\{[f'(\beta_\ell)]_t, [g'(\beta_\ell)]_t\}_{\ell\in[2\tilde{N}+1]}, \{[h'(\beta_\ell)]_t\}_{\ell\in[4\tilde{N}+1]}$. For the rest of the parties, $\mathcal{S}$ also views that they hold shares of $\{[f'(\beta_\ell)]_t, [g'(\beta_\ell)]_t\}_{\ell\in[2\tilde{N}+1]}, \{[h'(\beta_\ell)]_t\}_{\ell\in[4\tilde{N}+1]}$, and will update the additive errors accordingly.

Simulator $\mathcal{S}$

Generation of the Random Triples

1: $\mathcal{S}$ sends request $(\texttt{Init}, \texttt{Triple})$ to $\mathcal{F}_{\mathsf{Triple}}$ on behalf of honest parties. $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{GenTriple}}$. Upon terminating, $\mathcal{S}$ learns corrupted parties' shares of triples and the additive errors on honest parties' shares. Then $\mathcal{S}$ sends these values to $\mathcal{F}_{\mathsf{Triple}}$.
2: $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{VerifyTriple}}$. For each honest party P_i terminates with $\texttt{Fail}$, $\mathcal{S}$ sends $(\texttt{abort}, P_i)$ to $\mathcal{F}_{\mathsf{Triple}}$.

Request for Verifying the Random Triples

When $\mathcal{S}$ proceeds to the following simulation, then during the invocation of $\mathcal{S}_{\texttt{VerifyTriple}}$, $\mathcal{S}$ must proceed to Process 2. In this case, $\mathcal{S}$ views each honest P_i as a corrupted party, then does the same thing as in Process 2 of $\mathcal{S}_{\texttt{VerifyTriple}}$ to respond to each corrupted party's challenge.

The hybrid arguments are as follows.

$\mathbf{Hyb}_0$: $\mathcal{S}$ honestly executes the protocol, and outputs the views of $\mathcal{A}$ in the real world.

$\mathbf{Hyb}_1$: In the following, we focus on the simulation of $\Pi_{\texttt{GenTriple}}$.

$\mathbf{Hyb}_{1.1}$: During $\Pi_{\texttt{Sh}}$, for each honest dealer, $\mathcal{S}$ first generates corrupted parties' shares, then randomly samples the whole degree-t sharing based on corrupted parties' shares and the secret. According to the property of Shamir secret sharing, this adjustment makes no difference. The distribution of $\mathbf{Hyb}_{1.1}$ and $\mathbf{Hyb}_1$ is identical.

$\mathbf{Hyb}_{1.2}$: In Step 2 of $\Pi_{\texttt{GenTriple}}$, for virtual party $V_{i,j}$:

– If both P_i and P_j are honest, $\mathcal{S}$ delays the generation of messages between P_i, P_j and honest parties until Step 3 of $\Pi_{\texttt{GenTriple}}$.
– If P_i is honest and P_j is corrupted, $\mathcal{S}$ delays the shares of honest parties distributed by P_i until Step 3 of $\Pi_{\texttt{GenTriple}}$, and only simulates the behavior of sending messages between honest parties and P_i.

The distribution of $\mathbf{Hyb}_{1.2}$ and $\mathbf{Hyb}_{1.1}$ is identical.

$\mathbf{Hyb}_{1.3}$: In Step 2 of $\Pi_{\texttt{GenTriple}}$ (preparing random triples by virtual parties), for virtual party $V_{i,j}$: If P_i is corrupted and P_j is honest, $\mathcal{S}$ changes the generation of honest P_k's $o_{\ell,k}$ by randomly sampling $d_{\ell,k}$ and recomputes $o_{\ell,k} = a_{\ell,k}\cdot b_{\ell,k} - d_{\ell,k}$. Since both $o_{\ell,k}$ and $d_{\ell,k}$ are randomly sampled, the distribution of $\mathbf{Hyb}_{1.3}$ and $\mathbf{Hyb}_{1.2}$ is identical.

$\mathbf{Hyb}_{1.4}$: In Step 2 of $\Pi_{\texttt{GenTriple}}$ (preparing random triples by virtual parties), for virtual party $V_{i,j}$: If P_i is corrupted and P_j is honest, $\mathcal{S}$ changes the generation of P_j's $r^{(j,i)}_\ell$ by first randomly sampling $o_\ell + r^{(j,i)}_\ell$ and then recomputes $r^{(j,i)}_\ell = (o_\ell + r^{(j,i)}_\ell) - o_\ell$. Since both $o_\ell + r^{(j,i)}_\ell$ and $r^{(j,i)}_\ell$ are randomly sampled, the distribution of $\mathbf{Hyb}_{1.4}$ and $\mathbf{Hyb}_{1.3}$ is identical.

$\mathbf{Hyb}_{1.5}$: During $\Pi_{\texttt{Sh}}$ (in Step 1 of $\Pi_{\texttt{GenTriple}}$), for each honest dealer, $\mathcal{S}$ delays the generation of honest parties' shares until Step 3 of $\Pi_{\texttt{GenTriple}}$. Since in $\mathbf{Hyb}_{1.4}$, honest parties do not need to use their shares distributed by honest dealers, the distribution of $\mathbf{Hyb}_{1.5}$ and $\mathbf{Hyb}_{1.4}$ is identical.

$\mathbf{Hyb}_{1.6}$: In Step 3 of $\Pi_{\texttt{GenTriple}}$ (distributing triples by virtual parties), for virtual party $V_{i,j}$: If P_i is honest, $\mathcal{S}$ changes the generation of P_i's $r^{(i,j)}_\ell$ by first randomly sampling $\tilde{z}^{(i,j)}_\ell$ and recomputes $r^{(i,j)}_\ell =$

$\tilde{z}_\ell^{(i,j)} - (d_\ell + o_\ell + r_\ell^{(j,i)})$. Since both $\tilde{z}_\ell^{(i,j)}$ and $r_\ell^{(i,j)}$ are randomly sampled, the distribution of $\mathbf{Hyb}_{1.6}$ and $\mathbf{Hyb}_{1.5}$ is identical.

$\mathbf{Hyb}_2$: In the following, we focus on the simulation of Step 5 of $\Pi_{\texttt{GenTriple}}$ (extraction of random triples).

$\mathbf{Hyb}_{2.1}$: Let $\mathcal{D}$ denote the set of virtual parties all parties agreed in Step 4 of $\Pi_{\texttt{GenTriple}}$, and the triples distributed by each virtual party $V_i \in \mathcal{D}$ is denoted by $([a_i]_t, [b_i]_t, [c_i]_t)$. In this hybrid, for each honest $V_i \in \mathcal{D}$, we change the way of determining the first two $([a_i]_t, [b_i]_t)$ for all $i \in [2\tilde{N}+1]$. At a high level, we first change the way of generating the shared polynomial $[f(\cdot)]_t, [g(\cdot)]_t$ and then decide the degree-t Shamir sharings distributed by V_i based on $[f(\cdot)]_t, [g(\cdot)]_t$.

Let $\mathcal{D}' = \{V_1, \ldots, V_{2\tilde{N}+1}\}$, for each corrupted $V_i \in \mathcal{D}$:

- If $V_i \in \mathcal{D}'$, $\mathcal{S}$ has received the $[a_i]_t$ and can set $[f(\beta_i)]_t = [a_i]_t$.
- If $V_i \in \mathcal{D} \setminus \mathcal{D}'$, $\mathcal{S}$ randomly samples a degree-t Shamir sharing as $[f(\beta_i)]_t$ given the shares of corrupted parties.

Note that for each $[f(\gamma_i)]_t$, it is a linear combination of $\{[f(\beta_i)]_t\}_{i\in\mathcal{D}'}$. Then for all $i \in [\tilde{N}+1]$, $\mathcal{S}$ computes corrupted parties' shares of $[f(\gamma_i)]_t$ and randomly samples a degree-t Shamir secret sharing as $[f(\gamma_i)]_t$ based on the shares of corrupted parties.

Now $\mathcal{S}$ has fixed $\tilde{N}+1$ points $f(\gamma_i)$, with $\tilde{N}$ corrupted virtual parties points $f(\beta_i)$, $\mathcal{S}$ randomly samples a degree-$2\tilde{N}$ polynomial $f(\cdot)$ based on these points and the whole $[f(\cdot)]_t$ based on shares of corrupted parties. Then for all honest $V_i \in \mathcal{D}'$, we set $[a_i]_t = [f(\beta_i)]$. For the rest of honest virtual parties in $\mathcal{D}\setminus\mathcal{D}'$, $\mathcal{S}$ randomly samples a degree-t Shamir sharing $[a_i]_t$ based on the shares of corrupted parties. We do the same thing for $[b_i]_t$.

To show that $\mathbf{Hyb}_{2.1}$ and $\mathbf{Hyb}_2$ have the same distribution, it is sufficient to show that the degree-t Shamir sharings of honest virtual parties generated in the above approach are identically distributed to those in $\mathbf{Hyb}_2$. To this end, it is sufficient to show that the distribution of the shared polynomial $[f(\cdot)]_t$ in both hybrids is identical. In $\mathbf{Hyb}_2$, $[f(\cdot)]_t$ is a random shared polynomial given shares of $[f(\beta_i)]_t$ for all corrupted $V_i \in \mathcal{D}'$. In $\mathbf{Hyb}_{2.1}$, the only difference is that we additionally fix $[f(\beta_i)]_t$ for all corrupted $V_i \in \mathcal{D} \setminus \mathcal{D}'$ and $[f(\beta_i)]_t$ for all $i \in [\tilde{N}]$. However, those degree-t Shamir sharings are randomly sampled. Therefore, the obtained shared polynomial $[f(\cdot)]_t$ has the same distribution as that in $\mathbf{Hyb}_2$.

$\mathbf{Hyb}_{2.2}$: In this hybrid, In this hybrid, for each corrupted V_i where $i \in [2\tilde{N}+2, 4\tilde{N}+1]$, $\mathcal{S}$ first randomly samples values as $f(\beta_i)+a_i, g(\beta_i)+b_i$, then computes $f(\beta_i) = f(\beta_i)+a_i-a_i, g(\beta_i) = g(\beta_i)+b_i-b_i$. Finally, $\mathcal{S}$ randomly samples $[f(\beta_i)]_t, [g(\beta_i)]_t$ based on secrets $f(\beta_i), g(\beta_i)$ and shares of corrupted parties. $\mathbf{Hyb}_{2.2}$ and $\mathbf{Hyb}_{2.1}$ have the same distribution.

$\mathbf{Hyb}_{2.3}$: In this hybrid, for each honest V_i where $i \in [2\tilde{N}+2, 4\tilde{N}+1]$, instead of first randomly sampling degree-t Shamir sharings $[a_i]_t, [b_i]_t$, $\mathcal{S}$ first randomly samples the whole $[f(\beta_i)+a_i]_t, [g(\beta_i)+b_i]_t$, then compute $[a_i]_t = [f(\beta_i)+a_i]_t - [f(\beta_i)]_t, [b_i]_t = [g(\beta_i)+b_i]_t - [g(\beta_i)]_t$. $\mathbf{Hyb}_{2.3}$ and $\mathbf{Hyb}_{2.2}$ have the same distribution.

$\mathbf{Hyb}_{2.4}$: In this hybrid, for each honest $V_i \in \mathcal{D}$, we change the way of determining the third sharing $[c_i]_t$ in each random Beaver triple as follows. At a high level, we first determine the shared polynomial $[h(\cdot)]_t$ and then decide the degree-t Shamir sharings distributed by honest parties.

$\mathcal{S}$ first computes each corrupted V_i's shares of $[h(\cdot)]_t$ as follows:

- If $V_i \in \mathcal{D}'$, set $[h(\beta_i)]_t = [c_i]_t$.
- If $V_i \in \mathcal{D} \setminus \mathcal{D}'$, follows the protocol to compute $[h(\beta_i)]_t$.

Then $\mathcal{S}$ can compute the linear combination of $\{[h(\beta_i)]_t\}_{i\in\mathcal{D}}$ to get corrupted parties' shares of $[h(\gamma_i)]_t$ for all $i \in [\tilde{N}+1]$. Then $\mathcal{S}$ computes a degree-t Shamir sharing $[h(\beta_i)]_t$ based on shares of corrupted parties and the secret $h(\gamma_i) = f(\gamma_i) \cdot g(\gamma_i)$.

Now we have fixed $2\tilde{N}+1$ degree-t Shamir sharing in $[h(\cdot)]_t$, $\mathcal{S}$ computes $h(\beta_i) = f(\beta_i)g(\beta_i)$ and randomly samples $[h(\beta_i)]_t$ based on the $h(\beta_i)$ and shares of corrupted parties for the first $2\tilde{N}$ honest $V_i \in \mathcal{D}$. With these L degree-t Shamir sharings, $\mathcal{S}$ interpolates $[h(\beta_i)]_t$ for the rest of honest dealers.

Finally, for each honest $V_i \in \mathcal{D}'$, $\mathcal{S}$ sets $[c_i]_t = [h(\beta_i)]_t$. For each honest party $V_i \in \mathcal{D} \setminus \mathcal{D}'$, $\mathcal{S}$ sets $[c_i]_t = [h(\beta_i)]_t - (f(\beta_i)+a_i) \cdot (g(\beta_i)+b_i) + (g(\beta_i)+b_i) \cdot [a_i]_t + (f(\beta_i)+a_i) \cdot [b_i]_t$.

To show that $\mathbf{Hyb}_{2.4}$ and $\mathbf{Hyb}_{2.3}$ have the same distribution, it is sufficient to show that the degree-t Shamir sharings of honest virtual parties generated in the above approach are identically distributed to

those in $\mathbf{Hyb}_{2.3}$. To this end, it is sufficient to show that the distribution of the shared polynomial $[h(\cdot)]_t$ in both hybrids is identical. In both hybrids, $[h(\cdot)]_t$ is a random shared polynomial given $h(\cdot) = f(\cdot)g(\cdot)$, $[h(\beta_i)]_t$ for all corrupted $V_i \in \mathcal{D}$, and the shares of corrupted parties. In $\mathbf{Hyb}_{2.4}$, the only difference is that we additionally choose $[h(\gamma_i)]_t$ for all $i \in [\tilde{N}+1]$. However, those degree-t Shamir sharings are randomly sampled. Therefore, the obtained shared polynomial $[h(\cdot)]_t$ has the same distribution as that in $\mathbf{Hyb}_{2.3}$.

$\mathbf{Hyb}_{2.5}$: In this hybrid, for the public reconstruction, let P_i be the first party to get the reconstruction results. Then, when $\mathcal{S}$ learns a new honest party P_j terminates with a different public reconstruction result, $\mathcal{S}$ aborts the simulation. The difference between $\mathbf{Hyb}_{2.5}$ and $\mathbf{Hyb}_{2.4}$ is whether $\mathcal{S}$ aborts the protocol. In this case, for honest P_i, P_j, they both accept $2t+1$ parties' challenges. Then there exists at least one honest P_k who both sends challenges to P_i and P_j. Let r_i, r_j be the challenge from P_k to P_i and P_j, respectively. Let N be the number of degree-t sharing to be reconstructed during this procedure, and we use $(s_1^{(i)}, \ldots, s_N^{(i)})$, $s_1^{(j)}, \ldots, s_N^{(j)}$, and $s_1^{(k)}, \ldots, s_N^{(k)}$ to denote the output of P_i, P_j and P_k. Then we know that:

$$\sum_{\ell=1}^{N} r_i^{\ell} \cdot s_{\ell}^{(i)} = \sum_{\ell=1}^{N} r_i^{\ell} \cdot s_{\ell}^{(k)}$$
$$\sum_{\ell=1}^{N} r_j^{\ell} \cdot s_{\ell}^{(j)} = \sum_{\ell=1}^{N} r_j^{\ell} \cdot s_{\ell}^{(k)}$$

Also, the challenges r_i, r_j are uniformly distributed and unknown to the adversary. According to the Schwartz–Zippel lemma, we know that the probability that there exists $\ell \in [N]$ such that $s_{\ell}^{(i)} \neq s_{\ell}^{(k)}$ is at most $N/|\mathbb{F}|$. It is the same for $s_{\ell}^{(j)} \neq s_{\ell}^{(k)}$. Then in case $s_{\ell}^{(i)} \neq s_{\ell}^{(j)}$, it happens in: (1) $s_{\ell}^{(i)} = s_{\ell}^{(k)}$ but $s_{\ell}^{(j)} \neq s_{\ell}^{(k)}$, (2) $s_{\ell}^{(i)} \neq s_{\ell}^{(k)}$ and $s_{\ell}^{(j)} = s_{\ell}^{(k)}$, (3) $s_{\ell}^{(i)} \neq s_{\ell}^{(k)}$, $s_{\ell}^{(j)} \neq s_{\ell}^{(k)}$ and $s_{\ell}^{(i)} \neq s_{\ell}^{(j)}$. The probability is at most $(1 - N/|\mathbb{F}|) \cdot N/|\mathbb{F}| + N/|\mathbb{F}| \approx 2N/|\mathbb{F}|$. When taking the union bound for all parties, the total probability is at most $2Nn^2/|\mathbb{F}|$, which is negligible in κ when the field size $|F| = 2^{\Omega(\kappa)}$. Therefore, the distribution of $\mathbf{Hyb}_{2.5}$ and $\mathbf{Hyb}_{2.4}$ is statistically close.

$\mathbf{Hyb}_{2.6}$: In this hybrid, we delay the generation of the triples generated by honest virtual parties until the verification of triples. Note that these have never used in the simulation so far. For each output triple $([f(\gamma_i)]_t, [g(\gamma_i)]_t, [h(\gamma_i)]_t)$, it is a random multiplication triple given the shares of corrupted parties. $\mathbf{Hyb}_{2.6}$ and $\mathbf{Hyb}_{2.5}$ have the same distribution.

$\mathbf{Hyb}_{2.7}$: In this hybrid, we delay the generation of the whole polynomial f, g, h until the honest party needs to use his share of triples to respond to a corrupted party's challenge r. For an honest party P_i's challenge, $\mathcal{S}$ only simulates the behavior of sending messages to P_i on behalf of all honest parties. $\mathbf{Hyb}_{2.7}$ and $\mathbf{Hyb}_{2.6}$ have the same distribution.

In the following, we focus on the simulation of the verification procedure of triples. We let $\mathcal{S}$ try to invoke error corrections on $\{A_\ell\}_{\ell \in \mathcal{W}}, \{B_\ell\}_{\ell \in \mathcal{W}}$ (defined in the simulator for the triple verification) to interpolate degree-L polynomials.

$\mathbf{Hyb}_3$: In the following hybrid arguments, we focus on the case that there are more than $(L+1)n \left\lceil \frac{n+\kappa}{\kappa} \right\rceil \leq \tilde{N}/2$ errors on these values, then $\mathcal{S}$ cannot interpolate the degree-$2\tilde{N}$ polynomials $A(x), B(x)$. We also use the notation P_i' to denote the ith party whose triple shares are delivered by the adversary. $\mathcal{S}$ initializes a set $\mathcal{H}_i = \varnothing$ for each honest party P_i'. For each challenge proposed by honest parties, $\mathcal{S}$ also reveals it to the adversary, which allows the adversary to learn each honest party's local verification messages. Since these verification messages can be public, this adjustment does not make a difference.

$\mathbf{Hyb}_{3.0}$: In this hybrid, for each random challenge r, $\mathcal{S}$ does the same thing as in the construction of the simulator to randomly sample a value map to the corresponding linear combination results. Then $\mathcal{S}$ randomly samples the whole sharing based on shares of corrupted parties and the secrets. In the following, we will show that no honest party will eventually terminate with `Success` with overwhelming probability. In this case, there is no privacy information, and $\mathcal{S}$ can generate these values without affecting the output distribution. If any honest party terminates with `Success`, $\mathcal{S}$ aborts the simulation.

From $i = 1$ to n, we consider the following hybrid argument.

$\mathbf{Hyb}_{3.i}$: In this hybrid, if P_i' is corrupted, $\mathcal{S}$ honestly execute honest parties to response his challenge. If P_i' is honest, we delay the generation of triples after P_i' has proposed his random challenges, $\mathcal{S}$ sets $\mathcal{H}_i$ to be the set of honest parties who have received their shares of triples. Upon P_i' getting the shares of $\{f(r), g(r), h(r)\}$ from $2t+1$ parties, which form the set $\mathcal{M}$, $\mathcal{S}$ checks whether $|\mathcal{M} \cap \mathcal{H}_i| \geq t+1$. If true and P_i' can pass the verification $f'(r) \cdot g'(r) = h'(r)$, $\mathcal{S}$ aborts. We show that the probability that $\mathcal{S}$ will abort is negligible. According to the Schwartz-Zippel lemma, this also implies that with overwhelming probability, no subset of $t+1$ honest parties in $\mathcal{M} \cap \mathcal{H}_i$ can interpolate the f, g, h such that $f \cdot g = h$.

For a fixed set of $t+1$ honest parties, $\mathcal{S}$ uses the additive errors on their shares of triples to interpolate the polynomials $\Delta^{(f)}(x), \Delta^{(g)}(x)$ and $\Delta^{(h)}(x)$. In case these parties' shares can interpolate the correct f, g, h, then we know that for all $i \in \mathcal{W}$, the following equation holds.

$$h(\beta_i) + \Delta^{(h)}(\beta_i) = (f(\beta_i) + \Delta^{(f)}(\beta_i)) \cdot (g(\beta_i) + \Delta^{(g)}(\beta_i)).$$

Recall that $h(\beta_i) = f(\beta_i) \cdot g(\beta_i) + A_i \cdot f(\beta_i) + B_i \cdot g(\beta_i)$, then the above equation will imply the following relation:

$$(A_i - \Delta^{(g)}(\ell)) \cdot f(\beta_i) + (B_i - \Delta^{(f)}(\beta_i)) \cdot g(\beta_i) + \Delta^{(h)}(\beta_i) - \Delta^{(f)}(\beta_i) \cdot \Delta^{(g)}(\beta_i) = 0$$

We use L to denote the degree of f, g. Since there are at most $\tilde{N}$ triples known to the adversary. In the following, we want to prove that, if there exist more than $\tilde{N} + (L+1)n \left\lceil \frac{n+\kappa}{\kappa} \right\rceil$ distinct i such that $A_i \neq \Delta^{(g)}(\beta_i), B_i \neq \Delta^{(f)}(\beta_i)$, the above relation will hold with negligible probability.

- In this case, we know that there exists at least $(L+1)n \left\lceil \frac{n+\kappa}{\kappa} \right\rceil + 1$ distinct i such that $A_i \neq \Delta^{(g)}(\beta_i), B_i \neq \Delta^{(f)}(\beta_i)$. However, during the verification process, the adversary can learn at most $(L+1)n \left\lceil \frac{n+\kappa}{\kappa} \right\rceil$ distinct linear combination of $\sum_{i \in \mathcal{W}} (A_i \cdot f(\beta_i) + B_i \cdot g(\beta_i)) \cdot I_i(r)$, where $I_i(x)$ denote Lagrange interpolation function. Let `Space` denote the linear space spanned by these $(L+1)n \left\lceil \frac{n+\kappa}{\kappa} \right\rceil$ linear combinations, we know that there exists at least one $i \in \mathcal{W}$, the value $(A_i - \Delta^{(g)}(\ell)) \cdot f(\beta_i) + (B_i - \Delta^{(f)}(\beta_i)) \cdot g(\beta_i)$ is not in `Space`. Since both $f(\beta_i), g(\beta_i)$ are uniformly random, the adversary can only predict the value with negligible probability.

To summarize, for a fixed set of $t+1$ parties in $\mathcal{M} \cap \mathcal{H}_i$, one random challenge r, the probability that these parties' shares of triple can interpolate valid f, g, h is bounded by $\mathcal{O}(\tilde{N}/|\mathbb{F}|)$. By taking the union bound over all subsets of $t+1$ honest parties and all random challenges proposed by P_i', the probability is negligible in the security parameter. Therefore, the output distribution between $\mathbf{Hyb}_{3.i}$ and $\mathbf{Hyb}_{3.i-1}$ is statistically close.

$\mathbf{Hyb}_4$: In the following hybrid arguments, we focus on the case that there are no more than $(L+1)n \left\lceil \frac{n+\kappa}{\kappa} \right\rceil \leq \tilde{N}/2$ errors on these values, then $\mathcal{S}$ cannot interpolate the degree-$2\tilde{N}$ polynomials $A(x), B(x)$.

$\mathbf{Hyb}_{4.1}$: In this hybrid, for each challenge proposed by honest parties, $\mathcal{S}$ also reveals it to the adversary, which allows the adversary to learn each honest party's local verification messages. Since these verification messages can be public, this adjustment does not make a difference. The output distribution in $\mathbf{Hyb}_{4.1}$ and $\mathbf{Hyb}_4$ is identical.

$\mathbf{Hyb}_{4.2}$: In this hybrid, for all honest virtual parties in $\mathcal{Q}$ (defined in the simulator), $\mathcal{S}$ still generate their triples. For the rest of the honest virtual parties, $\mathcal{S}$ further delays the generation of their triples until the first challenge is released. The output distribution in $\mathbf{Hyb}_{4.2}$ and $\mathbf{Hyb}_{4.1}$ is identical.

$\mathbf{Hyb}_{4.3}$: In this hybrid, for each challenge r, we change the generation of the secrets $\{f(\beta), g(\beta), h(\beta)\}$ by first randomly sampling $\{f(r), g(r)\}$ and recomputes $h(r)$ as we described in the simulator. Then $\mathcal{S}$ recomputes $\{f(\beta), g(\beta), h(\beta)\}$. In $\mathbf{Hyb}_4$, $\{f(\beta), g(\beta), h(\beta)\}$ is generated by an honest virtual party and is uniformly random. In $\mathbf{Hyb}_{4.2}$, we randomly samples $\{f(r), g(r), h(r)\}$ and recomputes $\{f(\beta), g(\beta), h(\beta)\}$. Since there is a one-to-one map between $\{f(\beta), g(\beta), h(\beta)\}$ and $\{f(r), g(r), h(r)\}$, this adjustment makes no difference. The output distribution in $\mathbf{Hyb}_{4.3}$ and $\mathbf{Hyb}_{4.2}$ is identical.

$\mathbf{Hyb}_{4.4}$: In this hybrid, when an honest party terminates the verification with `Success`, $\mathcal{S}$ does the additional verification as shown in the construction of the simulator. If this condition happens, $\mathcal{S}$ aborts the simulation. In this case, it means the adversary finds polynomials A', B', and C' such that $(f(r)+A'(r))(g(r)+$

$B'(r)) = h(r) + C'(r)$, where f, g, h are unknown to the adversary. The probability that this can happen is negligible in the security parameter. The output distribution in $\mathbf{Hyb}_{4.4}$ and $\mathbf{Hyb}_{4.3}$ is statistically close.

$\mathbf{Hyb}_{4.5}$: In this hybrid, during the **Request for Verifying the Random Triples**, for each honest party P_i's verification, $\mathcal{S}$ delays the generation of triples and does the same thing as in $\mathbf{Hyb}_{4.4}$ to decide P_i's output bit b. The output distribution in $\mathbf{Hyb}_{4.5}$ and $\mathbf{Hyb}_{4.4}$ is statistically close.

$\mathbf{Hyb}_5$: In this hybrid, $\mathcal{S}$ no longer generates honest parties' shares of triples since they are never used. $\mathcal{S}$ will send shares of corrupted parties and the additive errors to $\mathcal{F}_{\mathsf{Triple}}$ to let it distribute triples to all parties. The output distribution in $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_{4.5}$ is identical.

Note that $\mathbf{Hyb}_5$ corresponds to the ideal world, then $\Pi_{\mathtt{Triple}}$ securely computes $\mathcal{F}_{\mathsf{Triple}}$.

C Security Proof for AMPC

The functionality of $\mathcal{F}_{\mathsf{AMPC}}$ is defined below.

Functionality $\mathcal{F}_{\mathsf{AMPC}}$

$\mathcal{F}_{\mathsf{AMPC}}$ proceeds as follows, running with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, an adversary $\mathcal{S}$ and a n-party function $f : (\{0,1\}^* \cup \{\bot\})^n \to \{0,1\}^* \cup \{\bot\}$. At the first activation, verify that $\mathtt{sid} = (\mathcal{P}, \mathtt{sid}')$ where $\mathcal{P}$ is a player set of size n. For each $P_i \in \mathcal{P}$, initialize x_i and y_i to a default value $\bot$, set the participation flags $\mathtt{part}_i^{\mathsf{in}} = \mathtt{part}_i^{\mathsf{out}} := 0$, and set the delay values $D_i^{\text{in-part}} = D_i^{\text{out-part}} = D_i^{\text{out-rel}} := 1$ (where "in-part" stands for input participation, "out-part" for output participation, and "out-rel" for output release). Also initialize the core set $\mathcal{CS}$ to $\bot$.

- Upon receiving $(\mathtt{delay}, \mathtt{sid}, P_i, \mathtt{type}, D)$ from $\mathcal{S}$ for $P_i \in \mathcal{P}$, with $\mathtt{type} \in \{\mathtt{in\text{-}part}, \mathtt{out\text{-}part}, \mathtt{out\text{-}rel}\}$, and $D \in \mathbb{Z}$ represented in unary, update $D_i^{\mathtt{type}} = \max(1, D_i^{\mathtt{type}} + D)$.
- Upon receiving $(\mathtt{input}, \mathtt{sid}, x)$ from $P_i \in \mathcal{P}$ (or $\mathcal{S}$ on behalf of corrupted P_i), record $x_i := x$, run the *Input Submission Procedure*, and send $(\mathtt{leakage}, \mathtt{sid}, P_i, l)$ to $\mathcal{S}$, where $l := (|x_1|, \ldots, |x_n|)$.
- Upon receiving $(\mathtt{core\text{-}set}, \mathtt{sid}, \mathcal{I})$ from $\mathcal{S}$, if $\mathcal{I} \subseteq \mathcal{P}$ with $|\mathcal{I}| \geq n - t$ and $\mathtt{part}_j^{\mathsf{in}} = 1$ for all $P_j \in \mathcal{I}$, and the functionality has not been evaluated yet (i.e., $y_1 = \cdots = y_n = \bot$), then record $\mathcal{CS} := \mathcal{I}$.
- Upon receiving $(\mathtt{fetch}, \mathtt{sid})$ from $P_i \in \mathcal{P}$ (or $\mathcal{S}$ on behalf of corrupted P_i), run the *Input Submission*, *Output Generation*, and *Output Release* Procedures (in that order), and send $(\mathtt{fetched}, \mathtt{sid}, P_i)$ to $\mathcal{S}$ and any messages sent by the *Output Release Procedure* to P_i.
- Upon receiving $(\mathtt{abort}, \mathtt{sid}, P_i)$ from $\mathcal{S}$, if the output y_i has not been delivered to P_i, then update $y_i := \mathtt{Fail}$.

Input Submission Procedure. If P_i has already provided input then do:

1. Update $D_i^{\text{in-part}} := D_i^{\text{in-part}} - 1$.
2. If $D_i^{\text{in-part}} = 0$, set $\mathtt{part}_i^{\mathsf{in}} := 1$.

Output Generation Procedure. If $\sum_{j=1}^{n} \mathtt{part}_j^{\mathsf{in}} \geq n - t$ then do:

1. Update $D_i^{\text{out-part}} := D_i^{\text{out-part}} - 1$.
2. If $D_i^{\text{out-part}} = 0$, set $\mathtt{part}_i^{\mathsf{out}} := 1$.
3. If $\sum_{j=1}^{n} \mathtt{part}_j^{\mathsf{out}} \geq t + 1$, and $\mathcal{CS} \neq \bot$, and $y_i = \bot$, then evaluate $(y_1, \ldots, y_n) := f(x'_1, \ldots, x'_n)$, where $x'_j := x_j$ for each $P_j \in \mathcal{CS}$ and $x'_j := \bot$ otherwise. Additionally, set $(\mathtt{adv\text{-}output}, \mathtt{sid}, (a_1, \ldots, a_n))$ to be sent to $\mathcal{S}$, where $a_j := y_j$ if P_j is corrupted and $a_j := \bot$ otherwise.

Output Release Procedure. If $\sum_{j=1}^{n} \mathtt{part}_j^{\mathsf{out}} \geq n - t$ then do:

1. Update $D_i^{\text{out-rel}} := D_i^{\text{out-rel}} - 1$.
2. If $D_i^{\text{out-rel}} = 0$, then do the following. First, if $\mathcal{CS} = \bot$, set $\mathcal{CS} := \{P_j \in \mathcal{P} \mid \mathtt{part}_j^{\mathsf{in}} = 1\}$. Now, if $y_i = \bot$, then choose $r \leftarrow \{0,1\}^*$ and evaluate $(y_1, \ldots, y_n) := f(x'_1, \ldots, x'_n; r)$, where $x'_j := x_j$ for each $P_j \in \mathcal{CS}$ and $x'_j := \bot$ otherwise. Finally, set $(\mathtt{output}, \mathtt{sid}, (\mathcal{CS}, y_i))$ to be sent to P_i (to be delivered as a response to a $\mathtt{fetch}$ request).

Construction of the simulator:

Simulator $\mathcal{S}_{\texttt{VerifyEval}}$

$\mathcal{S}$ takes the following values as input:

- Honest parties' additive errors on their shares of triples $\{[a_\ell]_t, [b_\ell]_t, [c_\ell]_t\}_{\ell\in[C]}$.
- The whole sharings $\{[u_\ell]_t, [v_\ell]_t\}_{\ell\in[C]}$, the public reconstruction results $\{u_\ell, v_\ell\}_{\ell\in[C]}$, which have been generated during the simulation of the online phase.

1: In Step 1, $\mathcal{S}$ honestly executes each honest P_i to sample random challenge and broadcasts them. $\mathcal{S}$ sends request $(\texttt{VerifyStart}, P_i)$ to $\mathcal{F}_{\mathsf{Triple}}$ on behalf of each honest party.
2: In Step 2, Upon receiving r_j from P_j, $\mathcal{S}$ does the following steps.
(1). $\mathcal{S}$ sends request $(\texttt{VerifyProceed}, P_j, P_i)$ to $\mathcal{F}_{\mathsf{Triple}}$ on behalf of P_i.
(2). $\mathcal{S}$ follows the protocol to compute honest parties' shares of $[S_j]_t$, and sends it to P_j.
3: In Step 3, for each honest party P_i, $\mathcal{S}$ does the following steps.
(1). $\mathcal{S}$ simulates the behavior of sending request $(\texttt{VerifyEnd}, P_i, \mathcal{M})$ to $\mathcal{F}_{\mathsf{Triple}}$ on behalf of P_i.
(2). $\mathcal{S}$ honestly execute P_i to checks whether the shares of $[S_i]_t$ from parties in $\mathcal{M}$ lie on a degree-t polynomial $S_i(x)$. If true, $\mathcal{S}$ further checks whether $S_i(\alpha_0) \stackrel{?}{=} \sum_{\ell\in[C]} r_i^{\ell-1}(u_\ell + r_i^C \cdot v_\ell)$.
4: $\mathcal{S}$ emulates the behavior of $\mathcal{F}_{\mathsf{Triple}}$ as follows for both honest and corrupted P_i. Upon receiving a request $(\texttt{VerifyStart}, P_i)$ or Upon receiving a request $(\texttt{VerifyProceed}, P_i, P_j)$, $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{Triple}}$. Upon receiving a request $(\texttt{VerifyEnd}, P_i, \mathcal{M})$ from party P_i, if $\texttt{cnt}_i \le L, \mathcal{M} \subseteq \mathcal{M}_i$ and $|\mathcal{M}| \ge 2t+1$, $\mathcal{S}$ does the following steps.
(1). $\mathcal{S}$ has got the additive errors on shares of $\{[a_\ell]_t, [b_\ell]_t, [c_\ell]_t\}_{\ell\in[C]}$ for all honest parties in $\mathcal{M}$. It checks whether these additive errors lie on degree-t polynomials. If not, it sets $b \leftarrow 0$ as the final output for P_i. Otherwise, it proceeds.
(2). For all $\ell \in [C]$, $\mathcal{S}$ interpolates degree-t polynomials $\delta_\ell^{(a)}(x), \delta_\ell^{(b)}(x), \delta_\ell^{(b)}(x)$ based on these honest parties' additive errors and checks whether $\delta_\ell^{(a)}(\alpha_0) = \delta_\ell^{(b)}(\alpha_0) = \delta_\ell^{(c)}(\alpha_0) = 0$. If true, it sets $b \leftarrow 1$. Otherwise, it sets $b \leftarrow 0$. Then it sends b to the adversary $\mathcal{A}$.
(3). Upon receiving request $(\texttt{Decide}, b')$ from $\mathcal{S}$, if the output bit b has not been delivered to P_i and $|\mathcal{H}_i \cap \mathcal{M}| \le t$ ($\mathcal{H}_i$ and $\mathcal{M}$ defined in $\mathcal{F}_{\mathsf{Triple}}$), it changes P_i's output to b'.
5: For the rest of the steps, $\mathcal{S}$ honestly executes each honest party to do the simulation. If any honest party terminates with `Success`, $\mathcal{S}$ learns a set $\mathcal{M}$ which contains $t+1$ honest parties, and the parties in $\mathcal{M}$ use valid triples to evaluate the circuit, and these parties' shares of output wires are correct. Then $\mathcal{S}$ interpolates the secrets $\{u'_\ell, v'_\ell\}_{\ell\in[C]}$ from shares $\{[u_\ell]_t, [v_\ell]_t\}_{\ell\in[C]}$ of for honest parties in $\mathcal{M}$, and checks whether $\{u'_\ell, v'_\ell\} = \{u_\ell, v_\ell\}$ for all $\ell \in [C]$. If not, $\mathcal{S}$ aborts the simulation.

Simulator $\mathcal{S}$

In the following simulation, when $\mathcal{S}$ learns that an honest party P_i terminates `Fail` or `abort` in any sub-protocols, it will send $(\texttt{Fail}, P_i)$ to $\mathcal{F}_{\mathsf{AMPC\text{-}Ab}}$ in the end.

Offline Phase

1: $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Triple}}$ as follows.
- During the generation of random triples. $\mathcal{S}$ only generates shares of corrupted parties and no longer generates the secrets of triples. The rest of the steps are the same.
- During the request for verifying the random triples.
  - In Step 3, it honestly emulates $\mathcal{F}_{\mathsf{Triple}}$.
  - In Step 4: in Step (1), instead of verifying whether the output shares of honest parties in $\mathcal{M}$ lie on degree-t polynomials, $\mathcal{S}$ checks whether the additive errors of these honest parties on their shares of triples lie on degree-t polynomials $\{\delta_\ell^{(a)}(x), \delta_\ell^{(b)}(x), \delta_\ell^{(c)}(x)\}$; in Step (2), instead of verifying whether each $\{a'_\ell, b'_\ell, c'_\ell\} = \{a_\ell, b_\ell, c_\ell\}$, $\mathcal{S}$ directly checks whether $\delta_\ell^{(a)}(\alpha_0) = \delta_\ell^{(b)}(\alpha_0) = \delta_\ell^{(c)}(\alpha_0) = 0$.

As a result, $\mathcal{S}$ learns corrupted parties' shares of $\{[a_\ell]_t, [b_\ell]_t, [c_\ell]_t\}_{\ell\in[C_I+C]}$. For each honest party that has received his shares from $\mathcal{F}_{\mathsf{Triple}}$, $\mathcal{S}$ learns his additives errors on his shares of $\{[a_\ell]_t, [b_\ell]_t, [c_\ell]_t\}_{\ell\in[C_I+C]}$.

Input Phase

1: **Distributing Inputs.** $\mathcal{S}$ does the following steps.

(1). In Step 1, for each corrupted P_i, $\mathcal{S}$ randomly samples values $a^{(i)}]_t, b^{(i)}]_t$ and computes $c^{(i)}]_t = a^{(i)}]_t \cdot b^{(i)}]_t$. Then $\mathcal{S}$ randomly samples the whole sharings $\{[a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t\}$ based on shares of corrupted parties and the secrets. With the whole sharings $\{[a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t\}$ and the additive errors on honest parties' shares, $\mathcal{S}$ honestly executes each honest party. For each honest P_i, $\mathcal{S}$ simulates the behavior of sending messages from honest parties to P_i. $\mathcal{S}$ also records the shares he received from corrupted parties, and computes the difference between the received values and the corrupted parties' shares sampled by $\mathcal{S}$ before.

(2). In Step 2, for each honest party P_i, let $\mathcal{H}_i$ denote the set of the identities of the first $t+1$ parties who sent shares to P_i. Then $\mathcal{S}$ interpolates a degree-t polynomial $\delta_i^{(a)}(x)$ such that: for each corrupted party $P_j \in \mathcal{H}_i$, $\delta_i^{(a)}(\alpha_j)$ is the difference computed above. For each honest party $P_j \in \mathcal{H}_i$, $\delta_i^{(a)}(\alpha_j)$ is P_j's additive error on his shares of $[a^{(i)}]_t$. $\mathcal{S}$ does the same thing to interpolate $\delta_i^{(b)}, \delta_i^{(c)}$. Then $\mathcal{S}$ checks whether $\delta_i^{(a)}(\alpha_0) = \delta_i^{(b)}(\alpha_0) = \delta_i^{(c)}(\alpha_0) = 0$. If not, $\mathcal{S}$ reliably broadcasts $\perp$ on behalf of P_i. Otherwise, $\mathcal{S}$ randomly samples a value as $x^{(i)} + a^{(i)}$ and reliably broadcasts it on behalf of P_i.

Evaluation Phase

1: **Evaluating the Circuit.** $\mathcal{S}$ does the following steps.
- For addition gate, $\mathcal{S}$ computes corrupted parties' share of $[z]_t$, and each honest party's additive error on his share of $[z]_t$.
- For every $t+1$ multiplication gates in one layer, $\mathcal{S}$ first randomly samples the whole sharings $([x_i + a_i]_t, [y_i + b_i]_t)_{i=1}^{t+1}$ based on shares of corrupted parties. Then it invokes $\mathcal{S}_{\texttt{PubRec}}$ to do simulation, with the sharings $([x_i + a_i]_t, [y_i + b_i]_t)_{i=1}^{t+1}$ and honest parties' additive errors on their shares of $([x_i + a_i]_t, [y_i + b_i]_t)_{i=1}^{t+1}$ as inputs. Upon getting the public reconstruction results, $\mathcal{S}$ computes corrupted parties' shares of $\{[z_i]_t\}_{i\in[t+1]}$, and each honest party's additive error on his share of $\{[z_i]_t\}_{i\in[t+1]}$.

2: **Verification of the Evaluation.** $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{VerifyEval}}$ and learns honest parties' output of $\Pi_{\texttt{VerifyEval}}$ at the end of $\mathcal{S}_{\texttt{VerifyEval}}$. If $\mathcal{S}$ learns that any honest party gets `Success` at the end of $\mathcal{S}_{\texttt{VerifyEval}}$, then $\mathcal{S}$ also learns that there exists a set $\mathcal{M}$ of size $t+1$ such that the parties in $\mathcal{M}$ are all honest, and they use valid triples to evaluate the circuit correctly.

3: **Extract Corrupted Parties' Input.** For the input sharing $[x_i]_t$ distributed by each corrupted $P_i \in \mathcal{D}$, $\mathcal{S}$ uses shares of all honest parties in $\mathcal{M}$ to interpolate the secret x_i.

Output Phase

1: $\mathcal{S}$ sends the set $\mathcal{D}$ and the inputs x_i for all corrupted $P_i \in \mathcal{D}$ to $\mathcal{F}_{\mathsf{AMPC\text{-}Ab}}$ and gets the output y. Then $\mathcal{S}$ randomly samples the whole sharing $[y]_t$ based on the secret y and shares of corrupted parties. Then $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{PubRec}}$ with sharing $[y]_t$ and honest parties' additive errors as inputs.

2: Upon terminating $\Pi_{\texttt{PubRec}}$, $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{VerifyEval}}$ again and learns honest parties' output of $\Pi_{\texttt{VerifyEval}}$ at the end of $\mathcal{S}_{\texttt{VerifyEval}}$.
- For each honest party terminates with `Success`, $\mathcal{S}$ deliveries the output from $\mathcal{F}_{\mathsf{AMPC\text{-}Ab}}$ to this honest party.
- Otherwise, $\mathcal{S}$ sends $(\texttt{Fail}, P_i)$ to $\mathcal{F}_{\mathsf{AMPC\text{-}Ab}}$.

The hybrid arguments are as follows.

Hyb$_0$: $\mathcal{S}$ honestly executes the protocol, and outputs the views of $\mathcal{A}$ in the real world.

Hyb$_1$: In this hybrid, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Triple}}$ in the offline phase as follows. During the Generation of the Random Triples:

- In Step 1, $\mathcal{S}$ delays the generation of the secrets of triples until the first honest party needs to use his shares in the online phase.
- In Step 2, it records corrupted parties' shares of triples, the additive errors on honest parties' shares of triples, and the $(\texttt{abort}, P_i)$ instruction.

During the Request for Verifying the Random Triples:

- In Step 3, it honestly emulates $\mathcal{F}_{\mathsf{Triple}}$.
- In Step 4: in Step (1), instead of verifying whether the output shares of honest parties in $\mathcal{M}$ lie on degree-t polynomials, $\mathcal{S}$ checks whether the additive errors of these honest parties on their shares of

triples lie on degree-t polynomials $\{\delta_\ell^{(a)}(x), \delta_\ell^{(b)}(x), \delta_\ell^{(c)}(x)\}$; in Step (2), instead of verifying whether each $\{a'_\ell, b'_\ell, c'_\ell\} = \{a_\ell, b_\ell, c_\ell\}$, $\mathcal{S}$ directly checks whether $\delta_\ell^{(a)}(\alpha_0) = \delta_\ell^{(b)}(\alpha_0) = \delta_\ell^{(c)}(\alpha_0) = 0$.

This adjustment delays the generation of the secrets of triples and honest parties' shares of triples, the distribution of $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ is identical.

$\mathbf{Hyb}_2$: In the following hybrid arguments, we focus on the simulation of the Input Phase.

$\mathbf{Hyb}_{2.1}$: In this hybrid, for each honest dealer P_i, $\mathcal{S}$ change the generation of the secret $a^{(i)}$ by first randomly samples $x^{(i)} + a^{(i)}$ and recomputes $x^{(i)} = x^{(i)} + a^{(i)} - a^{(i)}$. The distribution of $\mathbf{Hyb}_{2.1}$ and $\mathbf{Hyb}_2$ is identical.

$\mathbf{Hyb}_{2.2}$: In this hybrid, $\mathcal{S}$ delays the generation of the whole sharings $\{[a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t\}$ which is assigned to an honest dealer P_i until the end of the input phase. For honest parties who should send their shares of $\{[a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t\}$ to P_i, $\mathcal{S}$ only simulates this behavior of sending messages. Then for each honest party P_i, instead of verifying $a^{(i)} \cdot b^{(i)} = c^{(i)}$, $\mathcal{S}$ interpolates degree-t polynomials $\delta^{(a)}(x), \delta^{(b)}(x), \delta^{(c)}(x)$ as shown in the construction of simulator, and only checks whether $\delta^{(a)}(\alpha_0) = \delta^{(b)}(\alpha_0) = \delta^{(c)}(\alpha_0) = 0$. If not, $\mathcal{S}$ reliably broadcasts $\perp$ on behalf of P_i.

The difference between $\mathbf{Hyb}_{2.2}$ and $\mathbf{Hyb}_{2.1}$ is that P_i will not reliably broadcasts $\perp$ in case not all $\delta^{(a)}(\alpha_0), \delta^{(b)}(\alpha_0), \delta^{(c)}(\alpha_0)$ are zero but the following equation hold.

$$(a^{(i)} + \delta^{(a)}(\alpha_0)) \cdot (b^{(i)} + \delta^{(b)}(\alpha_0)) = c^{(i)} + \delta^{(c)}(\alpha_0).$$

This is equivalent to the following equation.

$$b^{(i)} = (-\delta^{(b)}(\alpha_0) \cdot a^{(i)} + \delta^{(c)}(\alpha_0) - \delta^{(a)}(\alpha_0) \cdot \delta^{(b)}(\alpha_0))/\delta^{(a)}(\alpha_0).$$

Since honest P_i will reliably broadcast $x^{(i)} + a^{(i)}$, then if $x^{(i)}$ is not random, then the adversary can learn the distribution of $a^{(i)}$. However, $b^{(i)}$ is still random and is uniformly distributed from the view of $\mathcal{A}$. Therefore, the probability that the above equation holds is at most $1/|\mathbb{F}|$, which is negligible in the security parameter κ if $|\mathbb{F}| = 2^{\Omega(\kappa)}$. By taking the union bound for all honest P_i, this probability is at most $n/|\mathbb{F}|$. Therefore, the distribution of $\mathbf{Hyb}_{2.2}$ and $\mathbf{Hyb}_{2.1}$ is statistically close.

$\mathbf{Hyb}_3$: In this hybrid, during the evaluation phase, for every $t+1$ multiplication gates, we change the generation of the secrets $\{a_i, b_i\}_{i=1}^{t+1}$. $\mathcal{S}$ first randomly samples values as $\{x_i + a_i, y_i + b_i\}_{i=1}^{t+1}$, then recomputes $a_i = x_i + a_i - x_i$, $b_i = y_i + b_i - y_i$. Since both of the secrets are randomly sampled, the distribution of $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_{2.2}$ is identical.

$\mathbf{Hyb}_4$: In this hybrid, during the evaluation phase, for every $t+1$ multiplication gates, we change the generation of the sharings $\{[x_i + a_i]_t, [y_i + b_i]_t\}_{i=1}^{t+1}$ by interpolating it from the secret $x_i + a_i, y_i + b_i$ and shares of corrupted parties. Then $\mathcal{S}$ further delays the generation of honest parties' shares of $[x_i]_t, [y_i]_t$ until Step 2 of the Evaluation phase. Since honest parties' input shares for each gate have not been used so far, the distribution of $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ is identical.

$\mathbf{Hyb}_5$: In the following hybrid arguments, we focus on the simulation of $\Pi_{\texttt{VerifyEval}}$. We set P'_i to be the identity of the party who broadcast his challenge r_i in the ith time order, $\mathcal{S}$ initializes a set $\mathcal{H}_i = \varnothing$ for each honest party P'_i.

From $i = 1$ to n, we consider the following hybrid argument. Note that each party will broadcast his challenge after he gets his shares of $\{[u_\ell]_t, [v_\ell]_t\}_{\ell \in [C]}$.

$\mathbf{Hyb}_{5.i}$: In this hybrid, if P'_i is corrupted, $\mathcal{S}$ honestly execute honest parties to response his challenge. If P'_i is honest, $\mathcal{S}$ sets $\mathcal{H}_i$ to be the set of honest parties who have received their shares of $\{[u_\ell]_t, [v_\ell]_t\}_{\ell \in [C]}$. Upon P'_i getting the set $\mathcal{M}$ (the set of parties who sent shares of $[S_i]_t$ to P'_i), $\mathcal{S}$ checks whether $|\mathcal{M} \cap \mathcal{H}_i| \geq t+1$. If not, then during the emulation of $\mathcal{F}_{\mathsf{Triple}}$, $\mathcal{S}$ allows the adversary $\mathcal{A}$ to change the output bit b for P'_i.

In case $|\mathcal{M} \cap \mathcal{H}_i| \geq t+1$, $\mathcal{S}$ interpolates the secrets $\{u'_\ell, v'_\ell\}_{\ell \in [C]}$ from shares $\{[u_\ell]_t, [v_\ell]_t\}_{\ell \in [C]}$ of for honest parties in $\mathcal{M}$, and checks whether $\{u'_\ell, v'_\ell\} = \{u_\ell, v_\ell\}$ for all $\ell \in [C]$. If not, but $\mathcal{S}$ finds that the output of $\mathcal{F}_{\mathsf{Triple}}$ for P'_i is 1 and the equation $S_i(\alpha_0) = \sum_{\ell \in [C]} r_i^{\ell-1}(u_\ell + r_i^C \cdot v_\ell)$ holds for P'_i, $\mathcal{S}$ aborts the simulation. The difference between $\mathbf{Hyb}_{5.i}$ and $\mathbf{Hyb}_{5.i-1}$ is that $\mathcal{S}$ will not abort the simulation in $\mathbf{Hyb}_{5.i-1}$ if the following condition happens:

$$\sum_{\ell \in [C]} r_i^{\ell-1}(u_\ell + r_i^C \cdot v_\ell) = \sum_{\ell \in [C]} r_i^{\ell-1}(u'_\ell + r_i^C \cdot v'_\ell), \exists \ell \in [C], \{u_\ell, v_\ell\} \neq \{u'_\ell, v'_\ell\}.$$

We rewrite the above equation as follows.

$$\sum_{\ell\in[C]} r_i^{\ell-1}(u_\ell - u'_\ell + r_i^C \cdot (v_\ell - v'_\ell)) = 0$$

There exists $\ell \in [C]$ such that the coefficient $u_\ell - u'_\ell$ or $v_\ell - v'_\ell$ is non-zero. Therefore, r_i can be viewed as a root for the equation $f(x) = \sum_{\ell\in[C]} x^{\ell-1}(u_\ell - u'_\ell + x^C \cdot (v_\ell - v'_\ell))$. Since the coefficients are fixed before P_i proposes his challenge r_i. Therefore, for a fixed set $\mathcal{M}$, the probability (for a single challenge r_i) that the adversary can guess the root r_i is at most $C/|\mathbb{F}|$, which is negligible in the security parameter κ if $|\mathbb{F}| = 2^{\Omega(\kappa)}$. By taking the union bound for all subsets $\mathcal{M}$ and $\lceil \frac{n+\kappa}{\kappa} \rceil$ distinct random challenges, the probability that the adversary can correctly guess is at most:

$$2^n \cdot (C/|\mathbb{F}|)^{\lceil \frac{n+\kappa}{\kappa} \rceil} = C/2^\kappa.$$

which is negligible in the security parameter κ. Therefore, the distribution of $\mathbf{Hyb}_{5.i}$ and $\mathbf{Hyb}_{5.i-1}$ is statistically close.

$\mathbf{Hyb}_6$: In this hybrid, when $\mathcal{S}$ learns that one honest party P_k terminates $\Pi_{\texttt{VerifyEval}}$ with `Success`, we argue that there exists at least one honest P_i such that he sends request $(\texttt{VerifyEnd}, P_i, \mathcal{M})$ to $\mathcal{F}_{\mathsf{Triple}}$, where $\mathcal{M} \subseteq \mathcal{M}_i$ (defined in $\mathcal{F}_{\mathsf{Triple}}$) and $|\mathcal{M} \cap \mathcal{H}_i| \geq t+1$. Thus, the adversary cannot change the output bit b for this honest party P'_i. The argument is as follows.

Proof. In this case, we know that this honest party P_k who outputs `Success` has received $(t+1, P_j)$ from $2t+1$ distinct P_j.

Let $\mathcal{H}_0^{(i)}, \mathcal{H}_1^{(i)}$ denote the set of parties who receive their shares of triples before/after an honest party P_i. For the last honest party P_i who sent $(t+1, P_i)$ among these $2t+1$ parties, we know that he has received (t, P_j) from $2t+1$ distinct P_j, each of them considers the verification result to be true. Then we know that:

- If $t+1$ honest parties of them come from $\mathcal{H}_0^{(i)}$, we have met the requirement, the party P_i is the desired party and these $t+1$ honest parties in $\mathcal{H}_0^{(i)}$ form the set $\mathcal{H}_i$.
- Otherwise, at least one honest party of them comes from $\mathcal{H}_1^{(i)}$, and we denote this honest party by P_k.

For this honest party P_k, similarly, we know that he has received $(t-1, P_j)$ from $2t+1$ distinct P_j. Then either $t+1$ honest parties of them come from $\mathcal{H}_0^{(k)}$, or at least one honest P'_k comes from $\mathcal{H}_1^{(k)}$. Since the size of $\mathcal{H}_1^{(k)}$ is bounded by t, in the worst case, after at most t times of recursion, we know that there exists an honest party who receives his shares after all the other honest parties, and this honest party meets our requirement. □

The distribution of $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ is identical.

$\mathbf{Hyb}_7$: In this hybrid, we change the computation of the output sharing $[y]_t$. Instead of computing it by $\mathcal{S}$ with honest parties' inputs, $\mathcal{S}$ first extracts corrupted parties' input based on shares of honest parties in $\mathcal{M}$, then $\mathcal{S}$ sends the set $\mathcal{D}$ and all corrupted parties' inputs to $\mathcal{F}_{\mathsf{AMPC\text{-}Ab}}$ and gets the circuit output y. Finally, $\mathcal{S}$ computes the whole sharing $[y]_t$ based on shares of corrupted parties and the secret y. The distribution of $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ is identical.

$\mathbf{Hyb}_8$: In this hybrid, during the $\Pi_{\texttt{VerifyEval}}$ in the Output phase, $\mathcal{S}$ does a similar thing as in $\mathbf{Hyb}_6$ to learn the identity party P_i, his challenge r_i and the request set $\mathcal{M}$. Following the same argument in $\mathbf{Hyb}_5$, the distribution of $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ is statistically close.

$\mathbf{Hyb}_9$: In this hybrid, if an honest party eventually terminates with `Success`, then $\mathcal{S}$ delivers the output from $\mathcal{F}_{\mathsf{AMPC\text{-}Ab}}$ to this honest party. Otherwise, $\mathcal{S}$ sends $(\texttt{Fail}, P_i)$ to $\mathcal{F}_{\mathsf{AMPC\text{-}Ab}}$. $\mathcal{S}$ also no longer generates shares of honest parties since they are never used. The distribution of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ is identical.

Note that $\mathbf{Hyb}_9$ corresponds to the ideal world, then $\Pi_{\texttt{AMPC-Abort}}$ securely computes $\mathcal{F}_{\mathsf{AMPC\text{-}Ab}}$.