Concretely Efficient Asynchronous MPC from Lightweight Cryptography

Akhil Bandarupalli[1], Xiaoyu Ji[2], Aniket Kate[3], Chen-Da Liu-Zhang[4], and Yifan Song[5]

[1] abandaru@purdue.edu, Purdue University
[2] jixy23@mails.tsinghua.edu.cn, Tsinghua University
[3] aniket@purdue.edu, Purdue University & Supra Research
[4] chen-da.liuzhang@hslu.ch, Lucerne University of Applied Sciences and Arts & Web3 Foundation
[5] yfsong@mail.tsinghua.edu.cn, Tsinghua University and Shanghai Qi Zhi Institute

Abstract. We consider the setting of asynchronous multi-party computation (AMPC) with optimal resilience $n = 3t + 1$ and linear communication complexity, and employ only "lightweight" cryptographic primitives, such as random oracle hash.

In this model, we introduce two concretely efficient AMPC protocols for a circuit with $|C|$ multiplication gates: a protocol achieving fairness with $\mathcal{O}(|C| \cdot n + n^3)$ field elements of communication, and a protocol achieving guaranteed output delivery with $\mathcal{O}(|C| \cdot n + n^5)$ field elements. These protocols significantly improve upon the best prior AMPC protocol in this regime communicating $\mathcal{O}(|C| \cdot n + n^{14})$ elements. To achieve this, we introduce novel variants of asynchronous complete secret sharing (ACSS) protocols with linear communication in the number of sharings, providing different abort properties.

By combining the AMPC protocols, one can achieve an AMPC with guaranteed output with an optimistic communication that is similar to the AMPC with fairness.

1 Introduction

Multi-Party Computation (MPC) [Yao82, GMW87, BGW88, CCD88, RB89] enables n mutually distrustful parties to compute any function on their private inputs. Moreover, it is guaranteed that the adversary does not learn any information about the inputs apart from what can be inferred from the output.

The cryptographic literature has studied MPC for more than forty years and the last decade has seen tremendous progress towards making it practical. However, most existing MPC systems still rely on strong networking assumptions such as (bounded) synchrony and broadcast channels that make their practicability questionable, especially for low-latency application scenarios. In the *synchronous* model, messages are assumed to be delivered within a fixed time frame. In high-throughput low-latency application scenarios in the real world, we cannot set synchronous time bounds generously and thus unpredictable delays must be tolerated. This makes most existing synchronous MPC systems inadequate. While protocols designed in the *asynchronous* model are resilient to such delays, the current designs may not scale as n grows: Indeed, current asynchronous MPC (AMPC) protocols struggle to scale to hundreds of parties either due to their 1) communication complexity from the information-theoretic (IT) approach [BKR94, PCR10, PCR08, CP23, GLZS24, LYK+19] which does not rely on any cryptographic assumptions or 2) computational complexity from the usage of threshold cryptography for common coins, additive homomorphic encryption/commitments and/or non-interactive zero-knowledge proofs [CP15, Coh16, BKLZL20, HNP05, HNP08, CHLZ21]. Concretely, the best-known IT-secure AMPC with $t < n/3$ resilience protocols require $\mathcal{O}(nC + \kappa n^{14})$ [GLZS24] for a circuit with C gates and n parties. Current computational AMPC protocols [BKR94, PCR10, PCR08, CP15, CP23, LYK+19] rely on significant "heavy-weight" public-key cryptography and/or non-interactive zero-knowledge proofs and have high computational costs as n grows.

Lightweight Cryptography. In this space between these two extremes of heavy-weight number-theoretic public-key cryptography and IT cryptography, hash-based cryptography is an interesting lightweight middle-ground option: a cryptographic hash computation is 100x to 1000x faster than an exponentiation in the discrete logarithm setting. Recently, hash-based constructions have been considered for asynchronous distributed random beacons [BBB+24], asynchronous (distributed) agreement on common

subset [DDL+24], and asynchronous verifiable secret sharing [SS24]. Among those, HashRand [BBB+24] demonstrated that the hash-based design is indeed practically more efficient than both computational designs as well as information-theoretic designs.

Moreover, as Quantum computer development picks up the pace, the world has begun adopting post-quantum secure cryptography in many frontiers; e.g., Apple recently converted their messaging service to be post-quantum secure. It is imperative to develop scalable AMPC protocols with post-quantum security. As contemporary hash functions like SHA3 also behave as Random Oracles against a polynomial-time quantum adversary, hash-based AMPC constructions can offer post-quantum security similar to IT-secure constructions.

However, building an AMPC protocol based on Hash functions with comparable communication efficiency as heavy-weight cryptography-based AMPC protocols is challenging as hashes are not additively homomorphic.

1.1 Our Contributions

In this work, we consider the setting of AMPC with linear communication and optimal resilience $t < n/3$ [BOKR94, ADS20] active corruptions. Given the above considerations, we ask whether it is possible to construct AMPC based on lightweight cryptography while still maintaining concrete communication complexity in a similar range as AMPC based on heavyweight cryptography:

Can we achieve concretely efficient AMPC with linear communication and optimal resilience from lightweight cryptography?

We answer in the affirmative by presenting scalable AMPC protocols that combine the computational efficiency of IT-secure protocols with practical communication overhead, without compromising post-quantum security. Our contributions are divided into two parts.

1. Security with Fairness: Our first result features an AMPC with security with fairness and communication complexity $\mathcal{O}(n)$ elements per multiplication, and $\mathcal{O}(n^3)$ elements of additive overhead. Note that such an overhead is minimal since any AMPC requires each party to reliably broadcast their input, and the cost for each reliable broadcast incurs inherently a quadratic term $\mathcal{O}(n^2)$ [DR85]. More precisely, we achieve the following:

Theorem 1. *Let $n = 3t + 1$ and κ denote the security parameter. For a finite field $\mathbb{F}$ of size $2^{\Omega(\kappa)}$ and any circuit C of size $|C|$ and depth D, there is an AMPC protocol computing the circuit that is secure against at most t corrupted parties with security with fairness. Let the input and output size be C_I and C_O respectively, the achieved communication complexity is $\mathcal{O}((|C| + C_I) \cdot n + D \cdot n^2 + C_O \cdot n^3)$ field elements.*

2. Guaranteed Output Delivery: We then move on to the stronger setting of guaranteed output delivery. In this case, we achieve communication complexity of $\mathcal{O}(n)$ elements per multiplication, and $\mathcal{O}(n^5)$ elements of additive overhead. More precisely:

Theorem 2. *Let $n = 3t + 1$ and κ denote the security parameter. For a finite field $\mathbb{F}$ of size $2^{\Omega(\kappa)}$ and any circuit C of size $|C|$ and depth D, there is a fully malicious asynchronous MPC protocol computing the circuit that is secure against at most t corrupted parties with guaranteed output delivery. Let the input and output size be C_I and C_O respectively, the achieved communication complexity is $\mathcal{O}((|C|+C_I+C_O) \cdot n + D \cdot n^2 + n^5)$ field elements.*

By executing the asynchronous MPC protocols with fairness and guaranteed output delivery in sequence, and using the output from the fair MPC if it did not fail and the guaranteed output MPC otherwise, we achieve the optimal communication when all parties are honest (additive $\mathcal{O}(n^3)$ elements overhead), while achieving guaranteed output delivery.

Corollary 1. *Let $n = 3t + 1$ and κ denote the security parameter. For a finite field $\mathbb{F}$ of size $2^{\Omega(\kappa)}$ and any circuit C of size $|C|$ and depth D, there is a fully malicious asynchronous MPC protocol computing the circuit that is secure against at most t corrupted parties with guaranteed output delivery. Let the input and output sizes be C_I and C_O respectively. The optimistic communication complexity is $\mathcal{O}((|C| + C_I) \cdot$*

$n + D \cdot n^2 + C_O \cdot n^3$) *field elements when all parties are honest. In the worst case, the communication is* $\mathcal{O}((|C| + C_I) \cdot n + D \cdot n^2 + C_O \cdot n^3 + n^5)$ *field elements.*

Computational Efficiency. On top of communication efficiency, our protocols are computationally efficient. Our AMPC protocol with fairness requires $O(n^3)$ Hash computations per party, independent of circuit size. Further, our AMPC with GOD protocol requires $O(n|C|+n^5)$ Hash computations per party. In comparison, protocols based on homomorphic cryptography like [CP15, AJM+23] require $\Omega(n|C|)$ computations per party, where each such operation is 100× to 1000× more expensive than a Hash computation.

Reduction to Small Field. All the above results assume a finite field $\mathbb{F}$ of size at least $2^{\Omega(\kappa)}$. In Section 6, we show how to reduce the field size requirement and obtain the following theorems.

Theorem 3. *Let* $n = 3t + 1$ *and* κ *denote the security parameter. For a finite field* $\mathbb{F}$ *of size at least* $n + 1$ *and any circuit* C *of size* $|C|$ *and depth* D*, there is an AMPC protocol computing the circuit that is secure against at most* t *corrupted parties with security with fairness. Let the input and output size be* C_I *and* C_O *respectively, the achieved communication complexity is* $\mathcal{O}((|C| + C_I) \cdot n + D \cdot n^2 + C_O \cdot n^3)$ *field elements plus* $\mathcal{O}(\kappa \cdot n^3)$ *bits.*

Theorem 4. *Let* $n = 3t + 1$ *and* κ *denote the security parameter. For a finite field* $\mathbb{F}$ *of size at least* $n + 1$ *and any circuit* C *of size* $|C|$ *and depth* D*, there is a fully malicious asynchronous MPC protocol computing the circuit that is secure against at most* t *corrupted parties with guaranteed output delivery. Let the input and output size be* C_I *and* C_O *respectively, the achieved communication complexity is* $\mathcal{O}((|C| + C_I + C_O) \cdot n + D \cdot n^2 + n^5)$ *field elements plus* $\mathcal{O}(\kappa \cdot n^5)$ *bits.*

1.2 Related Work

The communication complexity in AMPC has been the subject of a very significant line of work.

Information-Theoretic MPC. In the IT setting, the first protocol with optimal resilience $t < n/3$ was provided by Ben-Or, Kelmer, and Rabin [BKR94]. The works [PCR10, PCR08] achieved $\mathcal{O}(n^5)$ field elements per multiplication, which was further improved in [CP23] to $\mathcal{O}(n^4)$. The works [GLZS24, JLS24] recently improved the scope to $\mathcal{O}(n)$ elements, but the additive overhead is $\Omega(n^{14})$ elements, making it impractical. In the case of $t < n/4$ and perfect security, the recent work [AAPP24] achieves linear communication $\mathcal{O}(n)$ elements per multiplication with an additive overhead of $\mathcal{O}(n^5)$ elements.

Cryptographic MPC. There are several communication-efficient protocols with optimal resilience $t < n/3$ under different assumptions. However, these works make use of heavy cryptography, typically in the form of threshold (somewhat homomorphic) encryption and/or non-interactive zero-knowledge proofs, which increases considerably the computational overhead. Our protocols only make use of hash functions, which are orders of magnitude faster.

The works [HNP05, HNP08, CHLZ21] make use of an additive homomorphic encryption, with [HNP08, CHLZ21] communicating $\mathcal{O}(n^2)$ elements per multiplication. The work [CP15] achieves $\mathcal{O}(n)$ elements per multiplication at the cost of using somewhat-homomorphic encryption. The work [CHLZ21] also achieves linear cost using additive-homomorphic encryption for $t < (1-\epsilon)n/3$, but considers the atomic-send model. The works [Coh16, BKLZL20] achieve a communication independent of the circuit size using fully-homomorphic encryption. Finally, the works [LYK+19, DGKN09] introduce AMPC protocols where the preprocessing phase may not terminate, i.e. they are not live. This undesirable condition is more critical than the standard security with abort, as in the latter case, parties realize that the protocol failed (they obtain $\perp$ as output).

There also exist works that use homomorphic commitments to achieve ACSS with linear communication [AJM+23] per secret. However, this protocol requires $O(n)$ discrete-log operations per secret, which is a scalability bottleneck.

Lightweight Protocols. There are no MPC protocols that make use of lightweight cryptography, but some have appeared for concrete functionalities, including asynchronous distributed random beacons [BBB+24], asynchronous common subset [DDL+24], and asynchronous verifiable secret sharing [BKP11, SS24]. The work [BKP11] introduces an ACSS protocol with $O(n^3L)$ communication for sharing L secrets. [SS24] improve this complexity to $O(nL + \kappa n^2 \log(n))$ bits when the dealer is honest using Hash-based Zero-Knowledge proofs and Pseudorandom functions (PRFs). However, a malicious dealer can increase communication to $O(n^2L + \kappa n^3)$ bits, which is too expensive to build linear AMPC.

2 Technical Overview

In the following, we use $[s]_t$ to denote a degree-t Shamir secret sharing of s and $\alpha_{-t+1}, \ldots, \alpha_n$ to denote distinct field elements.

We give a high-level overview of the main techniques used in this work. Following [CP23, GLZS24], an asynchronous MPC (AMPC) can be obtained from the following three steps. The first step is to realize an *asynchronous complete secret sharing* (ACSS) [PCR09] protocol which ensures that all honest parties can obtain their shares of a degree-t Shamir sharing $[s]_t$ distributed by a dealer. Then, the second step is to prepare Beaver triples [Bea92] with the help of ACSS in the offline phase. After preparing a sufficient number of Beaver triples, all parties only need to do public reconstruction in the online phase, which can be achieved with linear communication complexity and high concrete efficiency.

For the preparation of Beaver triples, [GLZS24] first achieves linear communication per triple in the information-theoretic setting, while for concrete efficiency, the overhead is $\mathcal{O}(n^7)$ (regardless of the costs of ACSS). Their idea is to run two different processes for triple generation in parallel and argue that all parties will eventually get the triples from one process. However, their construction incurs overheads of $\mathcal{O}(n^5)$ in the first process and $\mathcal{O}(n^7)$ in the second process. Our first technical contribution is a new construction for the second process that is conceptually simpler and more concretely efficient. Relying on the random oracle hash, we manage to reduce the overheads of both processes to $\mathcal{O}(n^3)$. One drawback of our technique, however, is that our construction can only achieve malicious security with abort (and fairness) rather than guaranteed output delivery as achieved in [GLZS24]. Our second technical contribution is to use the party-elimination framework [BTH06] to compile our protocol to achieve guaranteed output delivery with an additive overhead $\mathcal{O}(n^5)$. To the best of our knowledge, we are the first to use the party-elimination framework in the asynchronous network setting.

Another problem is the concrete efficiency of ACSS. In the information-theoretic setting, [JLS24] gives the first ACSS protocol with linear cost per sharing but with an additive overhead of $\mathcal{O}(n^{12})$. We note that when targeting malicious security with abort, we only need a weaker security guarantee on the ACSS protocol: Instead of requiring all honest parties to eventually obtain their shares, we only require that all honest parties will eventually terminate the protocol either with the correct shares or a failure symbol. Note that this is different from an *asynchronous verifiable secret sharing* (AVSS) protocol which does not ensure all honest parties obtain their shares even if all parties follow the protocol. Based on random oracle hash, we give a concretely efficient ACSS protocol that only achieves malicious security with abort. Concretely, the cost per sharing is linear and the additive overhead is $\mathcal{O}(n^2)$. When constructing our GOD protocol, we upgrade the above ACSS protocol to achieve identifiable abort where an honest party may either receive his correct shares or a proof that can be used to accuse the corrupted dealer. Our ACSS protocol with identifiable abort also supports efficient public reconstruction: When the corrupted deal is accused, all parties may together recover the secrets shared by this dealer with linear cost. This will be an important building block for our GOD protocol as we will elaborate later.

In the following, we start by recapping the techniques in [GLZS24] and then introduce our new solution for preparing Beaver triples. Next we show how to achieve malicious security with fairness and GOD.

2.1 Overview of Previous Approach and Our New Construction

Triple Generation Framework in [GLZS24]. To prepare random Beaver triples, the authors in [GLZS24] design two processes where each process can achieve a linear cost per triple but does not guarantee termination when executed alone. On the other hand, they show that the first process will terminate when at least ϵt corrupted parties participate, while the second process will terminate when no more than ϵt corrupted parties participate. Then they connect these two processes by requiring that a party can only participate in the second process if he has participated in the first process. In this way, at least one process will eventually terminate and all parties will get their triples with linear cost.

In [GLZS24], the first process is adapted from the triple extraction process introduced in [CP17]. At a high level, the idea is to first let all parties distribute Beaver triples $([a]_t, [b]_t, [c_t])$ through ACSS where $c = a \cdot b$, and then jointly extract random Beaver triples which are unknown to each party. Recall that in the asynchronous setting, corrupted parties may never participate in the execution. Therefore, to ensure termination, all honest parties can only expect to agree on a set of size $L = 2t + 1$ of successful dealers

and in this case, the triple extraction process can generate $(L+1)/2-t=1$ random triple. This leads to a quadratic communication cost per triple in [CP17]. To save a factor of $\mathcal{O}(n)$, the idea of the first process is to let all parties wait for a set of size $L=(2+\epsilon)t+1$ of successful dealers. Then all parties can extract $(L+1)/2-t=\mathcal{O}(n)$ Beaver triples and thus achieve the linear cost per triple. However, this requires at least ϵt corrupted parties participating in the first process. Since corrupted parties may never participate in this process, this process may never terminate.

To counter the above malicious strategy, the second process targets for the case where at most ϵt corrupted parties participate. In addition, to link these two processes, a party P_i will only accept P_j's messages in the second process if P_i terminates P_j's ACSS protocol in the first process. In this way, a corrupted party must first participate in the first process to participate in the second process.

For the second process, the authors in [GLZS24] adapt the triple extraction process to work with packed Beaver triples [GPS22], where each packed Beaver triple can be transformed to $\mathcal{O}(n)$ standard Beaver triples. To share packed Beaver triples, the authors in [GLZS24] design an efficient sharing protocol for degree-$(1+\epsilon)t$ packed Shamir sharings, which only works when the corruption threshold is smaller than ϵt. Also the smaller corruption threshold allows them to utilize the error correction property for higher-degree polynomials.

Potential Solution to Achieve Linear Communication Based on [DN07]. We note that the second process in [GLZS24] is quite involved and complicated. Our starting point is the well-known DN technique [DN07] to prepare random Beaver triples in the synchronous setting:

1. All parties first prepare random sharings $([a]_t,[b]_t)$ and a pair of random *double sharings* $([r]_t,[r]_{2t})$.
2. All parties locally compute $[z]_{2t}=[a]_t\cdot[b]_t+[r]_{2t}$ and send it to P_{king}. Then P_{king} reconstructs and sends z to all parties.
3. After receiving z, all parties locally compute $[c]_t=z-[r]_t$.

When using the DN technique [DN07] in the asynchronous setting, we have to address the following two issues: (1) efficiently preparing double sharings, and (2) avoiding a malicious P_{king} not sending any result back. To better explain our idea, let us first assume that all messages sent by corrupted parties are honestly computed (but corrupted parties may choose to not send some messages). We will consider the malicious security case in the next subsection.

Following from the observation in [GLO+21], for *double sharings*, we may instead prepare $([r]_t,[o]_{2t})$ where $[o]_{2t}$ is a random degree-$2t$ Shamir secret sharing of 0. This allows us to decouple the relation of these two sharings. Then we can prepare $([a]_t,[b]_t,[r]_t)$ through standard techniques in [DN07] and ACSS, and the remaining problem is how to prepare $[o]_{2t}$ with linear communication. Again, relying on the techniques in [DN07], this question is further reduced to allowing a single dealer to distribute degree-$2t$ Shamir sharings of 0 with linear communication. Recall that in the second process, it is sufficient to deal with the case where there are at most ϵt corrupted parties. Let $\alpha_{-(1-\epsilon)t+1},\ldots,\alpha_n$ be distinct field elements, with a smaller corruption threshold, the following simple sharing protocol works:

1. Suppose the dealer D wants to share $[o_1]_{2t},\ldots,[o_{(1-\epsilon)t}]_{2t}$. D first encodes these ϵt degree-$2t$ Shamir sharings into a random degree-$(2t,(2-\epsilon)t)$ bivariate polynomial $F(x,y)$ such that $F(x,\alpha_{-i+1})=[o_i]_{2t}$ for all $i\in\{1,\ldots,(1-\epsilon)t\}$. Then D sends $F(x,\alpha_i)$ to P_i.
2. After receiving $F(x,\alpha_i)$ from D, P_i broadcasts $(\texttt{support},P_i,D)$ and sends $F(\alpha_j,\alpha_i)$ to each party P_j. If at least $2t+1$ parties support D, the sharing phase succeeds.
3. After receiving $(2-\epsilon)t+1$ evaluations of $F(\alpha_j,y)$, P_j reconstructs $F(\alpha_j,y)$ and recovers his shares of $[o_1]_{2t},\ldots,[o_{(1-\epsilon)t}]_{2t}$.

First notice that the communication cost per sharing is linear. To see why it works, note that when $2t+1$ parties support D, there are at least $2t+1-\epsilon t$ honest party P_i who will send $F(\alpha_j,\alpha_i)$ to P_j. Thus, every honest P_j can eventually obtain his shares.

For the second issue, we follow the idea in [GLZS24] to let each party perform as P_{king} to prepare $\mathcal{O}(1/n)$ fraction of random Beaver triples.

2.2 Our Solution for AMPC with Fairness

Towards Malicious Security with Abort. Unfortunately, our new protocol is not sufficient to achieve malicious security. To see this, note that P_{king} can only expect to receive $2t+1$ shares of $[z]_{2t}$, which are

just enough to reconstruct the secret. This means that even a single incorrect share would result in an incorrect triple and P_{king} cannot detect it with the $2t+1$ shares he received.

On the other hand, we manage to show that the random Beaver triples prepared using our approach achieve malicious security up to additive attacks, i.e., for each triple $([a]_t, [b]_t, [c]_t)$, the adversary may choose an arbitrary constant d such that $c = a \cdot b + d$. Thus, we follow the verification protocol in [GLZS24] to check the correctness of the prepared triples. If the verification fails, the protocol aborts. This allows us to achieve malicious security with abort.

One small issue is that the protocol may not terminate due to an honest party aborting the protocol earlier. Indeed, when an honest party aborts, the corruption threshold of the remaining parties can be $t \geq n/3$. To address this, when an honest party finds the computation fails on his part, he will continue to participate in the protocol execution while sending a failure symbol whenever he needs to send a message. A party that receives a failure symbol from some other party will also regard the computation as failing. This ensures the termination of our final protocol.

Security with Abort ACSS. So far, all previous discussion assumes an ACSS protocol. As we mentioned in the introduction, known solutions for ACSS with linear cost either incur a large communication overhead or a large computation overhead. We note that when targeting for malicious security with abort. It is sufficient to achieve a weaker guarantee where each honest party will eventually terminate the protocol with his correct shares or a failure symbol.

Our starting point is a distributed Zero-Knowledge (dZK) proof from [ABCP23], which allows a prover (the dealer) to prove that a distributed set of verifiers possess shares from a degree-t polynomial. It works as follows. Verifier P_ℓ possesses L shares $f_i(\alpha_\ell)$ for $i \in [1, L]$. The prover first samples a random degree-t polynomial $Y(x)$. Then, it generates commitments $\boldsymbol{C}[\ell]$ to points $Y(\alpha_\ell)$ for $\ell \in [1, n]$ using a Random Oracle H, and broadcasts them. The verifiers sample a random point p and ask the prover to broadcast $r(x) := Y(x) - \sum_{i=1}^{L} p^i f_i(x)$. Then, each verifier P_ℓ checks if $H(r(\alpha_\ell) + \sum_{i=1}^{L} p^i f_i(\alpha_\ell)) \stackrel{?}{=} \boldsymbol{C}[\ell]$. If the degrees of $f_i(x), Y(x) > t$ and $r(x)$ must have degree-t, then $\sum_{i=1}^{L} c_i \cdot p^i = 0$, where c_i is the coefficient of the higher degree term in $f_i(x)$. However, according to the Schwartz-Zippel lemma, a non-zero polynomial evaluates to zero on a randomly sampled point with probability $\leq \frac{L}{|S|}$. This proof technique can be made non-interactive by using the Fiat-Shamir heuristic, where the prover creates the challenge point p by applying H on generated commitments.

This dZK proof technique can be trivially used to build an Asynchronous Verifiable Secret Sharing (AVSS) protocol. In this protocol, the dealer sends shares of polynomials $f_i(x)$ to parties over private channels and broadcast commitments and the dZK proof using Reliable Broadcast (RBC). At least $t+1$ honest parties that participate in the dealer's RBC successfully verify and possess valid shares of the secrets. However, a corrupted dealer might never send shares to t honest parties, and force the protocol to terminate with $t+1$ honest parties.

We address this issue by enabling the $t+1$ parties who terminated with shares to help other honest parties interpolate their shares. In this protocol, the dealer encodes the share polynomials into larger degree-$(2t, t)$ bivariate polynomials $F_i(x, y)$, where each F_i packs $t+1$ degree-t share polynomials f. the dealer sends the row and column polynomials $F_i(x, \alpha_\ell), F_i(\alpha_\ell, y)$ to party P_ℓ. Each party verifies its shares using the dZK proof. Then, parties participate in a share interpolation phase where they send common points on each other's share polynomials. Each party receives a sufficient number of points on its row polynomials and eventually reconstructs its shares. However, a corrupted party can send wrong shares to an honest party, which might cause it to output wrong shares. Therefore, each party verifies its shares using the dZK proof. If the verification succeeds, the party outputs its shares. Otherwise, it outputs `abort`. Further, like in AVSS, parties can reconstruct the share polynomials by using dZK proofs to prove the validity of shares, followed by Lagrange interpolation.

In summary, this protocol utilizes bivariate polynomials over an underlying AVSS protocol to achieve ACSS with abort. The communication complexity of this construction is linear per sharing plus an additive overhead $\mathcal{O}(n^2)$.

From Malicious Security with Abort to Fairness. We note that the above ACSS protocol is also an AVSS protocol. Our next step is to compile our AMPC protocol to achieve fairness. We focus on the case where all parties should receive the same function output. At a high level,

1. During the input phase, each party P_i also shares a random degree-t Shamir sharing $[r_i]_t$ using our ACSS protocol.

2. Without loss of generality, suppose the first $t+1$ parties successfully share their random sharings. We use $[r]_t := [r_1]_t + \ldots + [r_{t+1}]_t$ as a random mask for the final output y.
3. After running our AMPC protocol that achieves malicious security with abort, each party either outputs $y + r$ or a failure symbol. Now all parties run a multi-value BA protocol to agree on the final output. If the final output is $y + r$, all parties verifiably reconstructs $r_1, \ldots, r_{t+1}$. Otherwise, all parties abort.

In this way, if all parties fail to agree on the output $y+r$, the function output y is perfectly protected by r_i generated by an honest party P_i. On the other hand, after all parties agree on the output $y + r$, by the property of AVSS, corrupted parties cannot prevent honest parties from reconstructing the mask r and learning y.

2.3 From Security with Fairness to GOD

Our next goal is to achieve malicious security with guaranteed output delivery. We note that our current AMPC protocol may fail due to the following two points:

- The random Beaver triples prepared in the preprocessing phase are incorrect.
- Honest parties do not obtain their shares of degree-t Shamir sharings, leading to failure in the online phase.

Note that if the random Beaver triples are all correct and all honest parties obtain their shares, then the protocol is guaranteed to succeed. We will first address the second issue.

Public Reconstruction with Party Elimination Framework. With the help of Beaver triples, the online phase only involves public reconstruction of degree-t Shamir sharings. In particular, for each $[x]_t$ to be reconstructed in the online phase, it can be written as $[x]_t = \sum_{i=1}^{n} [x_i]_t$ where $[x_i]_t$ is a linear combination of the sharings dealt by P_i. To ensure the success of reconstruction, our idea is to first augment the ACSS protocol to achieve *identifiable abort*, where an honest party either receives his correct shares or a proof that can be used to accuse the corrupted dealer, and support *efficient public reconstruction*. Then in the online phase, all parties may perform the public reconstruction of $[x]_t$ as follows.

Step 1: Check the Existence of degree-t Sharing. For $i \in [n]$, each party checks whether he has shares of $[x_i]_t$. If true, he computes his shares of $[x]_t$ and broadcasts it. Otherwise, he broadcasts the proof to accuse the corrupted P_i.
Step 2: Do Public Reconstruction with ACSS Proof. Each party waits to receive messages from others:
- When he first receives enough shares of $[x]_t$ and succeeds in reconstructing secret x by online error correction, he sets the reconstruction result as x.
- When he first receives an ACSS proof, he sets the reconstruction result as $\perp$.

Step 3: Agreement on Public Reconstruction Result. All parties run an agreement protocol to agree on the same reconstruction result in Step 2. If the agreement result is not $\perp$, all parties output the result and terminate. Otherwise, all parties continue to agree on a corrupted dealer P_i.
Step 4: Public Reconstruction of Corrupted Dealer's Secrets. In case a corrupted dealer P_i is identified, all parties reconstruct the secrets shared by P_i, compute x_i by the linear combination of P_i's secrets, and replace their shares of $[x_i]_t$ by x_i.

Note that whenever a corrupted dealer P_i is identified, all parties will replace $[x_i]_t$ by the constant value x_i. This ensures that the public reconstruction procedure will not fail due to P_i again. Thus, all parties can eventually reconstruct the secret x by repeating the above four steps and removing corrupted dealers. To achieve linear communication per reconstruction, we combine our technique with the efficient public reconstruction protocol in [DN07].

Security with Identifiable Abort ACSS. According to the analysis of the requirements of ACSS for public reconstruction, we present our next ACSS protocol, which achieves security with identifiable abort. In this protocol, each honest party outputs its shares when the dealer is honest and only outputs `abort` when the dealer behaves maliciously. Further, each party that outputs `abort` also outputs a verifiable proof which can implicate a malicious dealer. This proof allows all honest parties to definitively identify

that the dealer is malicious, and can be forwarded to and verified by other parties as well. Therefore, this protocol is strictly stronger than the previous protocol, where any Byzantine faulty party (not just the dealer) can make an honest party output `abort`.

We augment the existing protocol by creating dZK proofs for the entire bivariate polynomial. In this approach, the dealer creates commitments and dZK proofs for bivariate polynomials, which results in commitments being $n \times n$ matrices and the dZK proof polynomial being a degree-$(2t, t)$ bivariate polynomial. This technique is analogous to using AVSS to share each party's column polynomials. In detail, in the column interpolation phase, the dZK proof allows an honest party to distinguish between valid and invalid shares in its column. As at least $t+1$ honest parties possess valid shares on every other party's column polynomial, each party will successfully interpolate its column.

If the dealer is honest, each honest party participates in the row interpolation phase by sending points on its column polynomial. Again, parties use the dZK proof to identify correct points on their rows. Eventually, after receiving $2t+1$ valid points, each party interpolates its rows and shares of $f_i(x)$. However, a corrupted dealer can broadcast invalid commitments for a party P_i's column. In this case, P_i will not be able to participate in the row interpolation phase because it cannot generate a valid proof that these points are on its column. Therefore, parties cannot recognize the difference between correct and incorrect points on their rows. However, P_i can implicate the dealer by broadcasting the $t+1$ valid points it received on its column polynomial. Other parties can verify this proof by reconstructing P_i's columns and checking if the dealer's commitments are malformed. Therefore, parties either output shares or eventually agree that the dealer is corrupted. We ensure that this protocol has linear cost in both cases by using additional techniques like batching.

Another important requirement for our ACSS is that the reconstruction of the dealer's secret should be communication efficient. We introduce an efficient method to publicly reconstruct a batch of L secrets $f_1(\alpha_0), \ldots, f_L(\alpha_0)$ with linear cost. We adopt the public reconstruction technique in [DN07] into an AVSS protocol. In detail, the dealer forms groups of share polynomials $f_i(x)$ of size $t+1$ and forms degree-t polynomials $g_k(x) := \sum_{m=1}^{t+1} f_m(x) \cdot \alpha_k^m$ for $k \in [1, n]$. Then, it runs an AVSS protocol for each $g_k(x)$. In the reconstruction phase, parties reconstruct $g_k(x)$ to party P_k. Then, P_k reconstructs $g_k(\alpha_0) = \sum_{m=1}^{t+1} f_m(\alpha_0) \cdot \alpha_k^{m-1}$, which is equivalent to evaluating $\phi(x) := \sum_{m=1}^{t+1} f_m(\alpha_0) \cdot x^{m-1}$ at point α_k. Finally, each honest party uses Online Error Correction to reconstruct $\phi(x)$ and the $t+1$ secrets within it. The amortized costs are linear with the secret number L.

Triple Generation with Party Elimination Framework. Finally, we tackle the first issue: the prepared random Beaver triples can be incorrect in the preprocessing phase. Recall that the triple generation step is done by running two processes in parallel and the first process is identical to that in [GLZS24]. Thus, from the analysis in [GLZS24], the triples generated from the first process are guaranteed to be correct.

For the second process, our idea is to verify the Beaver triples led by different kings separately. In case the check fails, all parties help P_{king} to identify a corrupted party. Then this corrupted party is removed and all parties restart the preparation step. More concretely, when the verification fails, all parties will send their shares of $([a]_t, [b]_t, [r]_t, [o]_{2t})$ to P_{king}. The second process may fail due to the following reasons.

- Not all honest parties have their shares of $[a]_t, [b]_t, [r]_t$.
- A corrupted dealer distributes an incorrect degree-$2t$ Shamir sharing of 0.
- A corrupted party sends an incorrect share to P_{king}.

For the first case, P_{king} would receive a proof and he will just use this proof to implicate a corrupted dealer. For the second case, we augment the sharing protocol for degree-$2t$ Shamir sharings of 0 with identifiable abort as well. When the verification fails, all parties help P_{king} to recover the degree-$2t$ Shamir sharings of 0 dealt by each dealer. Then P_{king} can generate a proof to implicate a corrupted dealer in case a corrupted dealer distributes an incorrect degree-$2t$ Shamir sharing of 0. In the third case, after P_{king} reconstructs $[a]_t, [b]_t, [r]_t$ and $[o]_{2t}$, he may compute the correct sharing of $[z]_{2t}$ and check which party sends an incorrect share. To be able to implicate this corrupted party, when a party P_i sends his share of $[z]_{2t}$ to P_{king}, we ask P_i to broadcast a commitment of his share and then provide the opening to P_{king}. Later, P_{king} may use the opening of this commitment to prove that this message is indeed from P_i.

3 Preliminaries

We denote the security parameter by κ and require the field size to be $2^{\Omega(\kappa)}$.

3.1 Model

We consider protocols among a set $\mathcal{P}$ of n parties $P_1, \ldots, P_n$. Our protocols are proven secure in the model by Canetti [Can00]. Parties have access to a network of point-to-point asynchronous and secure channels (for details of the asynchronous network model, we refer the reader to [CR98]). Asynchronous channels guarantee *eventual* delivery, meaning that messages sent are eventually delivered, and the adversary does the scheduling of the messages. In particular, the adversary can arbitrarily (but finitely) delay all messages sent and deliver them out of order. We also consider the fully malicious adversary, that can completely control the behavior of corrupted parties.

Functionality of Asynchronous MPC. We define the functionality for AMPC with fairness in Appendix C.8 and use the AMPC with GOD in [CP23] (see Appendix D.9).

3.2 Agreement Primitives

Our construction makes use of the following agreement primitives and we give the definitions of them in Appendix A.1.

- **Reliable Broadcast**. It allows the parties to agree on the value of a sender without requiring termination if the sender is corrupted. [DXR21] shows that when $t < n/3$, there is a t-resilient protocol with communication complexity $\mathcal{O}(L \cdot n + \kappa \cdot n^2)$ for broadcasting L bits messages in the random oracle model.
- **Byzantine Agreement**. It allows parties to agree on a common message. For $t < n/3$, t-resilient binary asynchronous Byzantine agreement with communication complexity $\mathcal{O}(n^2)$ can be achieved given a common coin (see e.g. [MMR15]). Multi-valued Byzantine agreement with communication complexity $\mathcal{O}(L \cdot n + \kappa \cdot n^2 \log(n))$ can be achieved in the random oracle model and given a common coin, where L is the size of the message (see [NRS+20]).
- **Reliable Agreement.** Reliable agreement is the agreement version of the reliable broadcast where all parties have input. Compared to the standard byzantine agreement, a reliable agreement only guarantees the termination when honest parties provide matching input. For $t < n/3$, t-resilient reliable agreement protocol can be achieved with communication complexity $\mathcal{O}(L \cdot n^2)$ for agreeing on L bits message (see [DDL+24]).
- **Agreement on a Common Set.** If allows parties to agree on a set of at least $n - t$ parties that satisfy a certain property. For $t < n/3$, t-resilient agreement on a common set protocol can be achieved with communication complexity $\mathcal{O}(\kappa \cdot n^3)$ in the random oracle model (see [DDL+24]).

3.3 Merkle Trees Commitments

In this work, we use Merkle trees to instantiate vector commitments [CF13] and denote it by $\mathsf{MT} = (\mathsf{MT.Setup}, \mathsf{MT.Com}, \mathsf{MT.Open}, \mathsf{MT.Vfy})$. Let n be a power-of-two, given a vector $\boldsymbol{x}$ of size n and collision-resistant hash function, the Merkle trees commitment is constructed by a binary tree, where the leaves are something related to elements in $\boldsymbol{x}$, each node is the hash of its children and the root is commitment. We denote the i-th element in $\boldsymbol{x}$ as $\boldsymbol{x}[i]$, to promise the hiding property, the leaves are the commitment of each element $\boldsymbol{x}[i]$. The commitment of $\boldsymbol{x}[i]$ here can be realized by any standard commitment scheme, in particular, we can use $H(\boldsymbol{x}[i], r)$ as the commitment of $\boldsymbol{x}[i]$ where H denotes the random oracle and r is a random element.

The commitment is the root, and the opening of commitment at position i consists of the corresponding leaf x_i, random value r, and all the sibling values of all the nodes in the path from this leaf x_i till the root, which is logarithmic in the size of $\boldsymbol{x}$. In the following, we denote the i-th opening of $\boldsymbol{x}[i]$ as $\mathsf{op}[\boldsymbol{x}|i]$.

4 Achieving Malicious Security with Fairness

4.1 Security with Abort ACSS

The functionality $\mathcal{F}_{\text{ACSS-Abort}}$ defined below allows each honest party to output either a share of the secret or `abort`. The adversary can force an honest party to output `abort` by sending $(\texttt{abort}, P_i)$ to $\mathcal{F}_{\text{ACSS-Abort}}$. Additionally, $\mathcal{F}_{\text{ACSS-Abort}}$ enables parties to reconstruct the shared secrets when requested by $t+1$ parties.

Functionality $\mathcal{F}_{\text{ACSS-Abort}}$

Public Input: $(\alpha_0, \ldots, \alpha_n), L$

$\mathcal{F}_{\text{ACSS-Abort}}$ runs with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, a dealer $D \in \mathcal{P}$, and an adversary $\mathcal{S}$.

1: Upon receiving L degree-t polynomials $q_1(\cdot), \ldots, q_L(\cdot)$ from D, for each $P_i \in \mathcal{P}$, send an requested-based delayed output $q_1(\alpha_i), \ldots, q_L(\alpha_i)$ to P_i.
   - Upon receiving a request $(\texttt{abort}, P_i)$ from $\mathcal{S}$, if the output of P_i has not been delivered, change the output of P_i by `abort`. Otherwise, ignore this request.

2: Upon receiving `Public-Recon` from $t+1$ parties, send an requested-based delayed output $q_1(\cdot), \ldots, q_L(\cdot)$ to each $P_j \in \mathcal{P}$.

For the construction of ACSS with abort. We adopt the distributed Zero-Knowledge Proofs in Atapoor *et al.* [ABCP23] to the asynchronous setting. The main idea of this protocol is to combine the dZK proof with bivariate polynomials, where each honest party has partial information about another honest party's shares. Digging deep, the dealer, who wishes to share L degree-t polynomials among the parties denoted by $f_i(x)$, splits the L polynomials into groups of $t+1$ polynomials and encodes each group into a degree-$(2t, t)$ bivariate polynomial, denoted by $F_i(x, y)$. Then, the dealer creates an $n \times n$ matrix of evaluation points and distributes the i-th row and column to party P_i.

Protocol $\Pi_{\text{ACSS-ab}}$

Let $\alpha_{-t}, \ldots, \alpha_0, \ldots, \alpha_n$ be distinct field elements.

Dealer D Protocol

1: D possesses a list of L degree-t polynomials $f_1(x), \ldots, f_L(x)$. D divides them into groups containing $t+1$ polynomials each, denoted by $f_{i,j}(x)$ for $i \in [1, \frac{L}{t+1}], j \in [1, t+1]$, where $f_{i,j}(x) = f_{(i-1)*(t+1)+j}$.

2: **Encode Sharings**: For each $\{f_{i,j}\}_{j=1}^{t+1}$, D samples a degree-$(2t, t)$ bivariate polynomial $F_i(x, y)$ where $F_i(\alpha_{-j+1}, y) = f_{i,j}(x)$, for $j \in [1, t+1]$, $i \in [1, \frac{L}{t+1}]$.

3: **Commitments**: Dealer D samples a random degree-(t, t) bivariate polynomial $Y(x, y)$. Then, D computes commitment vector $\boldsymbol{C}$ for the share polynomials as $\boldsymbol{C}[i] = H(f_1(\alpha_i), \ldots, f_L(\alpha_i), Y(\alpha_0, \alpha_i))$, for $i \in [1, n]$.

4: **Distributed Zero-Knowledge (dZK) proofs**: D runs the following steps.
   (a) D samples a random degree-t polynomial $f_0(x)$ and a random degree-(t, t) bivariate nonce polynomial $Y_0(x, y)$.
   (b) It computes vector $\boldsymbol{\mathcal{C}}$ as $\boldsymbol{\mathcal{C}}[i] = H(f_0(\alpha_i), Y_0(\alpha_0, \alpha_i))$, and $d = H(\boldsymbol{C}, \boldsymbol{\mathcal{C}})$.
   (c) Finally, it computes polynomial $r(x)$ as follows.

$$r(x) := f_0(x) - \sum_{i \in [1, L]} d^i f_i(x)$$

5: **Send Shares and Broadcast Commitments**: D reliably broadcasts $\boldsymbol{C}, \boldsymbol{\mathcal{C}}, r(x)$. Further, D sends $\langle \mathit{Shares}, \{F_i(x, \alpha_j)\}, Y(x, \alpha_j), Y_0(\alpha_j, y)), \{F_i(\alpha_j, y)\}, Y(\alpha_j, y), Y_0(\alpha_j, y))\rangle$ to party $P_j \in \mathcal{P}$.

The dealer then computes *unconditionally hiding* commitments of shares. It randomly samples a degree-(t, t) nonce polynomial $Y(x, y)$, and computes commitments $\boldsymbol{C}[i] := H(f_1(\alpha_i), \ldots, f_L(\alpha_i), Y(\alpha_0, \alpha_i))$ for $i \in [1, n]$. These commitments leak no information about the shares because of the pigeon-hole principle and H being a random oracle.

Protocol $\Pi_{\mathsf{ACSS\text{-}ab}}$

Participant Party Protocol

1: **Verifying shares**: A participant party P_j that receives a $\langle Shares \rangle$message executes the following steps to verify their correctness.
   (a) **Verify Commitments**: For each $k \in [t+1], i \in [L/(t+1)]$, P_j computes $f'_{(i-1)*(t+1)+k}(\alpha_j) = F'_i(\alpha_{-k+1}, \alpha_j)$. It then computes $\boldsymbol{C}'[j] = H(f'_1(\alpha_j), \ldots, f'_L(\alpha_j), Y'(\alpha_0, \alpha_j))$ and verifies if $\boldsymbol{C}'[j] \stackrel{?}{=} \boldsymbol{C}[j]$.
   (b) **Verify dZK Proof**: Finally, P_j computes $d = H(\boldsymbol{C}, \boldsymbol{\mathcal{C}})$. Then, it verifies if the following equation is true.
$$H(r(\alpha_j) + \sum_{i \in [1,L]} d^i f'_i(\alpha_j), {Y_0}'(\alpha_0, \alpha_j)) \stackrel{?}{=} \boldsymbol{\mathcal{C}}[j]$$
2: **Run Reliable Agreement**: Party P_ℓ inputs 1 to $\mathcal{F}_{\mathsf{ra}}$ on successfully verifying its shares. After terminating $\mathcal{F}_{\mathsf{ra}}$ with output 1, all parties terminate the sharing phase and move to the next phase.

Share Interpolation Phase

In the following, each party who accepts his shares from D before terminating the sharing phase still participates in the following procedures, but he will not verify their shares again (so he will output his shares and not abort).

3: **Send common shares on rows**: P_j computes and sends the message $\langle Row, (F_1(\alpha_k, \alpha_j), \ldots, F_{\frac{L}{t+1}}(\alpha_k, \alpha_j), Y(\alpha_k, \alpha_j), Y_0(\alpha_k, \alpha_j)) \rangle$ to party P_k.
4: **Reconstruct columns**: On receiving $t+1$ shares on its column polynomial, P_k reconstructs $F'_i(\alpha_k, y), Y'(\alpha_k, y), {Y_0}'(\alpha_k, y)$.
5: **Send common shares on columns**: P_k computes and sends message $\langle Column, (F'_1(\alpha_k, \alpha_\ell), \ldots, F'_{\frac{L}{t+1}}(\alpha_k, \alpha_\ell), Y'(\alpha_k, \alpha_\ell), {Y_0}'(\alpha_k, \alpha_\ell)) \rangle$ to party P_ℓ.
6: **Reconstruct rows**: Upon receiving $2t+1$ valid shares on its row, P_ℓ reconstructs the row polynomials $F'_i(x, \alpha_\ell), Y'(x, \alpha_\ell), {Y_0}'(x, \alpha_\ell)$.
7: **Verify commitments and dZK proofs**: P_ℓ reconstructs its row polynomials and verifies commitments and dZK proofs. If the verification succeeds, then P_ℓ outputs its shares and terminates the protocol. Otherwise, it outputs $\langle Abort \rangle$ and terminates.

Public Reconstruction Phase

8: **Reconstruction**: Each party P_ℓ who gets its shares will send the message $\langle PubRec, (f'_1(\alpha_\ell), \ldots, f'_L(\alpha_\ell), Y'(\alpha_0, \alpha_\ell), {Y_0}'(\alpha_0, \alpha_\ell)) \rangle$ to each P_j.
   On receiving a $\langle PubRec \rangle$ from party P_ℓ, P_j computes commitment $\boldsymbol{C}'[\ell] := H(f'_1(\alpha_\ell), \ldots, f'_L(\alpha_\ell), Y'(\alpha_0, \alpha_\ell))$ and checks $\boldsymbol{C}'[\ell] \stackrel{?}{=} \boldsymbol{C}[\ell]$. Further, it checks the dZK proof by verifying if $H(r(\alpha_\ell) + \sum_{j \in [1,L]} d^j f'_j(\alpha_\ell), {Y_0}'(\alpha_0, \alpha_\ell)) \stackrel{?}{=} \boldsymbol{\mathcal{C}}[\ell]$. If the verification succeeds, P_j accepts the point. It then waits for $t+1$ valid points and interpolates $f_1(x), \ldots, f_L(x)$.

The dealer then generates a dZK proof by randomly sampling a degree-t polynomial $f_0(x)$ and a degree-(t,t) bivariate nonce polynomial $Y_0(x,y)$. The dealer generates unconditionally hiding commitments of $f_0(x)$ by computing $\boldsymbol{\mathcal{C}}[i] := H(f_0(\alpha_i), Y_0(\alpha_0, \alpha_i))$. Then, the dealer follows the Fiat-Shamir heuristic and computes a succinct commitment $d = H(\boldsymbol{C}, \boldsymbol{\mathcal{C}})$ over the commitment transcript to act as the challenge point for the dZK proof. Finally, the dealer computes $r(x) = f_0(x) - \sum_{i \in [1,L]} d^i f_i(x)$. It uses RBC to broadcast $\boldsymbol{C}, \boldsymbol{\mathcal{C}}, r$, and sends row and column polynomials of F_i, Y, Y_0 over private channels.

A party P_ℓ receiving shares from dealer verifies them using the dZK proof. If the verification succeeds, P_ℓ inputs 1 to a Reliable Agreement instance $\Pi_{\mathtt{ra}}$. This primitive is the agreement version of reliable broadcast, where every party has an input and sends an ECHO message for its own input, rather than the broadcaster's value. If a party terminates $\Pi_{\mathtt{ra}}$ with output 1, then at least $t+1$ honest parties verified their shares and input 1 to $\Pi_{\mathtt{ra}}$.

The $t+1$ honest parties that received their shares enable all other parties to interpolate their own shares. Each party P_j first sends $F_i(\alpha_j, \alpha_k)$ to parties $P_k \in \mathcal{P}$. Upon receiving $t+1$ points on its column, a party (that did not receive shares from the dealer) interpolates its column polynomial. Then, P_k sends points on its row polynomials to other parties. Upon receiving $2t+1$ points on its row polynomial, it interpolates its row polynomials. Finally, it verifies the L interpolated shares of polynomials $f_i(x)$ using the dZK proof broadcast by D. If the verification succeeds, then it outputs the shares, otherwise it aborts the protocol.

In this protocol, all parties output their shares only when D is honest and no other party behaves maliciously. Note that a single faulty party can make an honest party abort the protocol by sending it the wrong shares in the column and row interpolation phase, even when the dealer is honest.

Public Reconstruction. An honest party P_j that does not output `abort` sends its shares and the corresponding nonces $Y(\alpha_0, \alpha_j), Y_0(\alpha_0, \alpha_j)$. A party receiving these shares verifies them using the commitments and dZK proofs broadcast by the dealer. Then, it waits for $t+1$ valid shares and uses Lagrange interpolation to verify the shares.

Lemma 1. *Protocol $\Pi_{\mathsf{ACSS\text{-}ab}}$ securely computes $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t < n/3$ parties.*

We prove Lemma 1 and analyze the costs in Appendix B.1.

4.2 Preparing Random Degree-t Shamir Sharings and Weak Public Reconstruction

In our construction for AMPC with fairness, we need to let all parties prepare random degree-t Shamir sharings and do public reconstruction. As we introduced in section 2.2, we only require a weaker version of them and allow all parties' output can be a failure symbol. We define the functionality $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}, \mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ and give the corresponding construction $\Pi_{\mathtt{randSh\text{-}Weak}}, \Pi_{\mathtt{pubRec\text{-}Weak}}$ for them, refer to Appendix C.1 and C.2 for more details.

4.3 Preparing Random Beaver Triples with Additive Error

We design protocol $\Pi_{\mathtt{triple\text{-}Add\text{-}Weak}}$ for preparing random Beaver triples with additive errors which realizes $\mathcal{F}_{\mathsf{triple\text{-}add\text{-}Weak}}$ below. The additive errors mean that for each output Beaver triple $([a]_t, [b]_t, [c]_t)$, the shares of honest parties form valid degree-t Shamir sharings, and $d = c - a \cdot b$ is known to the adversary.

Functionality $\mathcal{F}_{\mathsf{triple\text{-}add\text{-}Weak}}$

Public Input: N

$\mathcal{F}_{\mathsf{triple\text{-}add\text{-}Weak}}$ runs with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$.

1: Wait to receive the number N of random Beaver triples to be prepared and the identities of corrupted parties.
2: For all $i \in [N]$, randomly samples a_i, b_i, c_i such that $c_i = a_i \cdot b_i$.
3: For all $i \in [N]$, wait to receive a set of shares $\{u_{i,j}, v_{i,j}, w_{i,j}\}_{j \in \mathcal{C}orr}$ of corrupted parties from $\mathcal{S}$ as well as an additive error d_i. Then sample three random degree-t Shamir sharings $([a_i]_t, [b_i]_t, [c_i + d_i]_t)$ based on the shares of corrupted parties and the secrets $a_i, b_i, c_i + d_i$.
4: For each party P_j, send a request-based delayed output of shares of $\{([a_i]_t, [b_i]_t, [c_i]_t)\}_{i=1}^N$ to P_j.
   – Upon receiving a request $(\mathtt{Fail}, P_j)$ from $\mathcal{S}$, if the output of P_j has not been delivered, change the output of P_j by `Fail`.

Recall that our high-level idea is to run two different processes in parallel. We will ensure that no matter how corrupted parties behave, at least one of the two processes will succeed, and the successful process will produce random Beaver triples (with additive errors) and the communication complexity is linear per triple. The description of these two processes is as follows.

Process 1. As we introduced in section 2.2, here we let all parties wait for $L = 2t + (t-1)/2$ successful dealers. Since $L > 2t + 1$, we cannot use the ACS protocol in a black box way to reach an agreement on the successful dealers. Instead, we modify the ACS protocol in [DDL+24] and present it in Appendix A.1. Then all parties extract random Beaver triples from those shared by successful dealers. Note that process 1 will only terminate if at least $(t-3)/2$ corrupted parties terminate their $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$. The detailed construction of process 1 $\Pi_{\mathtt{tripleExt\text{-}Weak}}$ is shown in Appendix C.4.

Process 2. In the second process, we will use the DN technique [DN07] to prepare random Beaver triples. We let all parties invoke $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$ to prepare random degree-t Shamir sharings and design $\Pi_{\mathtt{randShareZero\text{-}Weak}}$ for the preparation of degree-$2t$ Shamir sharings of zero. Note that only when at most $(t+1)/2$ corrupted parties participate in $\Pi_{\mathtt{randShareZero\text{-}Weak}}$, all honest parties can eventually get their shares of zero. The construction of $\Pi_{\mathtt{randShareZero\text{-}Weak}}$ (see Appendix C.3) is similar to $\Pi_{\mathtt{randSh\text{-}Weak}}$ and we replace $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ by $\Pi_{\mathtt{Sh2tZero\text{-}Weak}}$.

Protocol $\Pi_{\texttt{Sh2tZero-Weak}}$

Let $\beta_1, \ldots, \beta_{(t+1)/2}, \alpha_0, \alpha_1, \ldots, \alpha_n$ be distinct field elements and N be the number of degree-$2t$ Shamir sharings of zero.

Dealer D

1: **Distributing Shares:** Divide these N sharings into L disjoint sets, each of size $(t+1)/2$. For ℓ-th set of sharings, denoted by $[o_1]_{2t}, \ldots, [o_{(t+1)/2}]_{2t}$, prepare random degree-$(2t, t+(t-1)/2)$ bivariate polynomials $F_\ell(x, y)$ where $F_\ell(x, \beta_i) = [o_i]_{2t}$ for all $i \in [(t+1)/2]$. Let $f_\ell^{(i)}(x) = F_\ell(x, \alpha_i), g_\ell^{(i)}(y) = F_\ell(\alpha_i, y)$. Then send $\{f_\ell^{(i)}(x)\}_{\ell \in [L]}$ to party P_i.

Party P_i

1: **Receiving Row Polynomial:** Upon receiving $\{f_\ell^{(i)}(x)\}_{\ell \in [L]}$ from the dealer, send $\{f_\ell^{(i)}(\alpha_j)\}_{\ell \in [L]}$ to party P_j. Then send $(\mathsf{support}, P_i, D)$ to all parties.

2: **Reconstructing Column Polynomial:** Upon receiving $\{f_\ell^{(j)}(\alpha_i)\}_{\ell \in [L]}$ from $t + (t+1)/2$ distinct parties P_j, interpolate column polynomials $\{g_\ell^{(i)}(y)\}_{\ell \in [L]}$.

3: **Termination procedure:** Only after accepting $\{g_\ell^{(i)}(y)\}_{\ell \in [L]}$: Upon receiving $(\mathsf{support}, P_j, D)$ from $2t+1$ distinct parties P_j such that for each P_j, P_i has received outputs from the $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ invoked by dealer P_j in $\Pi_{\texttt{tripleExt-Weak}}$, locally computes his shares of $[o_1]_{2t}, \ldots, [o_{(t+1)/2}]_{2t}$ (by evaluating each $g_\ell^{(i)}(y)$ at positions $\beta_1, \ldots, \beta_{(t+1)/2}$ for all $\ell \in [L]$) and terminates.

Following the DN technique, we ask P_{king} to help all parties reconstruct the secret of $[z]_{2t} = [a]_t \cdot [b]_t + [r]_t + [o]_{2t}$ and design $\Pi_{\texttt{tripleKingDN-Weak}}$ as follows.

Process $\Pi_{\texttt{tripleKingDN-Weak}}$

Let N be the number of Beaver triples prepared by P_{king} and each party's inputs be $\{[a_\ell]_t, [b_\ell]_t, [r_\ell]_t, [o_\ell]_{2t}\}_{\ell=1}^N$. Each party whose input sharings are $\perp$ will output $\perp$.

1: Each party P_i locally compute $[z_\ell]_{2t} := [a_\ell]_t \cdot [b_\ell]_t + [r_\ell]_t + [o_\ell]_{2t}$.

2: Each party P_i sends his shares of $\{[z_\ell]_{2t}\}_{\ell \in [N]}$ to P_{king}.

3: If P_{king} first receives $2t+1$ parties' $\{[z_\ell]_{2t}\}_{\ell \in [N]}$, he reconstructs $\{z_\ell\}_{\ell \in [N]}$ and reliably broadcast them. Otherwise, if P_{king} first receives $\perp$ from one party, he reliably broadcasts the failure symbol $\perp$.

4: All parties wait to receive the message from P_{king}, if they receive $\{z_\ell\}_{\ell \in [N]}$ from the dealer, they locally compute $[c_\ell]_t = z_\ell - [r_\ell]_t$ for all $\ell \in [N]$. Otherwise, they output $\perp$ when they receive $\perp$.

To prevent corrupted P_{king} from not broadcasting secrets or a failure symbol, we design $\Pi_{\texttt{tripleDN-Weak}}$. At a high level, we let each party act as P_{king} and lead an instance of $\Pi_{\texttt{tripleKingDN-Weak}}$ to reconstruct $\mathcal{O}(1/n)$ fraction of the secrets. Since the instances of $\Pi_{\texttt{tripleKingDN-Weak}}$ led by corrupted parties may never terminate, all parties will invoke an ACS protocol to agree on $2t+1$ kings who successfully reconstruct the secrets to all parties.

Process $\Pi_{\texttt{tripleDN-Weak}}$

Let N be the number of Beaver triples to be prepared and $N' = N/(2t+1)$.

Preparation

1: All parties invoke $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$ to prepare $3N' \cdot n$ random degree-t Shamir sharings, each party who receives `Fail` from $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$ will send $\perp$ to all parties. All parties invoke $\Pi_{\texttt{randShareZero-Weak}}$ to prepare $N' \cdot n$ random degree-$2t$ Shamir sharings of 0.

2: Each party locally divide these Shamir secret sharings into n groups such that each group contains $3N'$ random degree-t sharings and N' degree-$2t$ sharings of 0. The i-th group is denoted by $\{[a_\ell^{(i)}]_t, [b_\ell^{(i)}]_t, [r_\ell^{(i)}]_t, [o_\ell^{(i)}]_{2t}\}_{\ell=1}^{N'}$ for $i \in [n]$.

Generation

1: **Generation of Random Triples:**
Each party P_j acts as P_{king} and leads an instance of $\Pi_{\texttt{tripleKingDN}}$, all parties participates in the $\Pi_{\texttt{tripleKingDN}}$ led by P_j with their shares of $\{[a_\ell^{(j)}]_t, [b_\ell^{(j)}]_t, [r_\ell^{(j)}]_t, [o_\ell^{(j)}]_{2t}\}_{\ell=1}^{N'}$.

2: **Determine the Set of Successful Kings and Output:**

Each party sets the property Q as he terminates the $\Pi_{\texttt{tripleKingDN}}$ led by one P_{king}, and all parties invoke $\mathcal{F}_{\mathsf{acs}}$ with property Q to agree on a set $\mathcal{D}$ of successful kings with size $|\mathcal{D}| = 2t+1$. Each party checks whether he gets all shares of Beaver triples from each $P_j \in \mathcal{D}$. If true, he outputs his shares of $\{[a_\ell^{(j)}]_t, [b_\ell^{(j)}]_t, [c_\ell^{(j)}]_t\}_{\ell=1}^{N'}$ for all $P_j \in \mathcal{D}$. Otherwise, he outputs `Fail`.

Putting it all Together. Based on the above two processes, we give the construction of $\Pi_{\texttt{triple-Add-Weak}}$ which realizes $\mathcal{F}_{\mathsf{triple\text{-}add\text{-}Weak}}$. The communication complexity is $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits. We prove lemma 2 and analyze the communication complexity in Appendix C.5.

Protocol $\Pi_{\texttt{triple-Add-Weak}}$

1: **Run Process 1 and Process 2:**
All parties execute $\Pi_{\texttt{tripleExt-Weak}}$ and $\Pi_{\texttt{tripleDN-Weak}}$ in parallel to prepare N random Beaver triples.
2: **Agree on Successful Process:**
For each party P_i, if the first process first succeeds, he sets $b_i = 0$; otherwise, he sets $b_i = 1$. Then P_i sends b_i to $\mathcal{F}_{\mathsf{ba}}$. Upon receiving b from $\mathcal{F}_{\mathsf{ba}}$, if $b = 0$, he takes the output of the first process as the final output; otherwise, he takes the output of the second process as the final output.

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

4.4 Generating Beaver Triples without Additive Error

We design $\Pi_{\texttt{triple-Weak}}$ to prepare random Beaver triples without additive errors, which realizes $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$ defined below. At a high level, all parties first invoke $\mathcal{F}_{\mathsf{triple\text{-}add\text{-}Weak}}$ to prepare triples, then check for additive errors. If errors are found, they output a failure symbol. The verification process follows the approach in [NV18, BSFO12] and we give the detailed construction in C.6. The communication complexity of $\Pi_{\texttt{triple-Weak}}$ is $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits. We prove lemma 3 and analyze the costs in Appendix C.7.

Functionality $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$

Public Input: N
$\mathcal{F}_{\mathsf{triple\text{-}Weak}}$ runs with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, a dealer $D \in \mathcal{P}$, and an adversary $\mathcal{S}$.
1: Wait to receive the number N of random Beaver triples to be prepared and the identities of corrupted parties.
2: For all $i \in [N]$, randomly samples a_i, b_i, c_i such that $c_i = a_i \cdot b_i$.
3: For all $i \in [N]$, wait to receive a set of shares $\{u_{i,j}, v_{i,j}, w_{i,j}\}_{j \in \mathcal{C}orr}$ of corrupted parties from $\mathcal{S}$. Then sample three random degree-t Shamir sharings $([a_i]_t, [b_i]_t, [c_i]_t)$ based on the shares of corrupted parties and the secrets a_i, b_i, c_i.
4: For each party P_j, send a request-based delayed output of shares of $\{([a_i]_t, [b_i]_t, [c_i]_t)\}_{i=1}^{N}$ to P_j.
– Upon receiving a request $(\texttt{Fail}, P_j)$, if the output of P_j has not been delivered, change the output of P_j by `Fail`.

Protocol $\Pi_{\texttt{triple-Weak}}$

Let N be the number of Beaver triples to be prepared.
1: **Preparing Random Beaver Triples with Additive Errors:**
All parties invoke $\mathcal{F}_{\mathsf{triple\text{-}add\text{-}Weak}}$ to prepare $2(N+1)$ random Beaver triples, denoted by $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=0}^{2N+1}$. If a party receives `Fail` in $\mathcal{F}_{\mathsf{triple\text{-}add\text{-}Weak}}$, this party uses $\perp$ as his input for $\Pi_{\texttt{tripleVerify-Weak}}$.
2: **Verifying the Prepared Beaver Triples:**
All parties invoke $\Pi_{\texttt{tripleVerify-Weak}}$ and use their shares of $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=0}^{2N+1}$ or $\perp$ as input. Upon terminating $\Pi_{\texttt{tripleVerify-Weak}}$, each party will output his shares of $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=1}^{N}$ or `Fail`.

Lemma 3. *Let κ denote the security parameter, for a finite field $\mathbb{F}$ of size $2^{\Omega(\kappa)}$, protocol $\Pi_{\texttt{triple-Weak}}$ securely computes $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$ in the $\mathcal{F}_{\mathsf{triple\text{-}add\text{-}Weak}}$-hybrid model against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t < n/3$ parties.*

4.5 Main Protocol for Malicious Security with Fairness

First, we let all parties invoke $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$ to prepare Beaver triples during the offline phase. Then, following the brief outline for the online phase introduced in section 2.2, we achieve malicious security with fairness AMPC with linear communication costs and obtain the following theorem. We refer the reader to Appendix C.8 for more detailed construction, security proof, and cost analysis.

Theorem 1. *Let $n = 3t + 1$ and κ denote the security parameter. For a finite field $\mathbb{F}$ of size $2^{\Omega(\kappa)}$ and any circuit C of size $|C|$ and depth D, there is an AMPC protocol computing the circuit that is secure against at most t corrupted parties with security with fairness. Let the input and output size be C_I and C_O respectively, the achieved communication complexity is $\mathcal{O}((|C| + C_I) \cdot n + D \cdot n^2 + C_O \cdot n^3)$ field elements.*

5 From Security with Fairness to Guaranteed Output Delivery

5.1 Security with Identifiable Abort ACSS

In $\Pi_{\mathsf{ACSS\text{-}ab}}$, a single malicious party can make an honest party abort the protocol, even if D was honest. We strengthen this guarantee to ensure that nodes will output `abort` only when dealer D is malicious. Further, we also ensure that every honest party will terminate either with its shares or with verifiable proof that implicates a malicious dealer. We present the functionality $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ and defer the detailed protocol description and proof of Lemma 4 to Appendix B.3.

Functionality $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$

Public Input: $(\alpha_0, \ldots, \alpha_n), N$

$\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ runs with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, a dealer $D \in \mathcal{P}$, and an adversary $\mathcal{S}$.

1: Upon receiving the set of corrupted parties $\mathcal{P}_{\mathcal{C}orr}$, if $D \in \mathcal{P}_{\mathcal{C}orr}$, initialize $\mathcal{P}_{\text{proof}} = \mathcal{P}_{\mathcal{C}orr}$. Otherwise, set $\mathcal{P}_{\text{proof}} = \varnothing$.

2: Upon receiving N degree-t polynomials $q_1(\cdot), \ldots, q_N(\cdot)$ from D, for each party $P_i \in \mathcal{P}$, send an requested-based delayed output $q_1(\alpha_i), \ldots, q_N(\alpha_i)$ to P_i.
   - Upon receiving a request $(\texttt{proof}, P_i)$ from $\mathcal{S}$, if $D \in \mathcal{P}_{\mathcal{C}orr}$ and the output of P_i has not been delivered, change the output of P_i by $(\texttt{Corrupt}, D)$ and add P_i in $\mathcal{P}_{\text{proof}}$. Otherwise, ignore this request.

3: Upon receiving `Broadcast-Proof` from P_i, if $P_i \in \mathcal{P}_{\text{proof}}$, send an requested-based delayed output $(\texttt{Corrupt}, D)$ to each $P_j \in \mathcal{P}$.

4: Upon receiving `Public-Recon` from $t + 1$ parties, send an requested-based delayed output $q_1(\alpha_0), \ldots, q_N(\alpha_0)$ to each $P_j \in \mathcal{P}$.

In this protocol, D generates a dZK proof for the whole share polynomials $F(x, y)$. This dZK proof enables any honest party to verify if a point is from a given bivariate polynomial. D follows the same framework as $\Pi_{\mathsf{ACSS\text{-}ab}}$ - broadcasts commitments and the dZK proof, while delivering shares over private channels. In the column interpolation phase, an honest party uses the expanded dZK proof to verify if a point is on its column polynomial. As at least $t+1$ parties verified their shares before proceeding to this phase, each party P_i will eventually reconstruct its column. Then, P_i must send points on its column to enable other parties to interpolate their rows. However, a corrupted D can craft commitments such that the commitments of specific shares on P_i's column do not match the broadcast commitments. When this happens, other parties will not be able to verify P_i's shares, which prevents row interpolation.

However, P_i can implicate D and prove its malice with the $t + 1$ valid points it received on its column polynomial. P_i compiles an $\langle \mathit{Abort} \rangle$ message by attaching its accepted column shares. A party that receives this message can verify the commitments and dZK proofs of these points, interpolate the ith column of all F_is, and verify the malformed commitments/dZK proofs. Note that this scenario can only occur when D is malicious, i.e., if D is honest, malicious parties cannot compile such a proof.

Reducing proof size. The size of the proof required to implicate a corrupted D is $O(n)$ field elements on each of the $\frac{L}{t+1}$ polynomials, which is $O(L)$ field elements. If $O(n)$ honest parties broadcast their proofs, then the overall communication becomes $O(n^2L)$ field elements. Therefore, to reduce this complexity, D splits the $\frac{L}{t+1}$ bivariate polynomials into n batches, and creates separate commitments and dZK proofs for each batch. With this change, a party only has to compile a proof for one batch to implicate D, which only has size $O(\frac{L}{t})$. Therefore, $O(n)$ honest parties broadcasting their proofs costs $O(nL)$ communication.

Lemma 4. *Protocol $\Pi_{\mathsf{ACSS\text{-}id}}$ securely computes $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t < n/3$ parties.*

5.2 Public Reconstruction and Sub-Circuit Evaluation

We integrate the public reconstruction process into the so-called *sub-circuit evaluation* protocol which is mainly used in the online phase. We first show some details for public reconstruction based on the party elimination framework.

Batch Reconstruction and Agreement with Linear Costs. To ensure the whole public reconstruction achieves linear communication cost, steps 2 and 3 of public reconstruction introduced in section 2.3 should also achieve linear cost. We design $\Pi_{\texttt{BatchPubRec}}$ and $\Pi_{\texttt{Agreement}}$ to achieve them, respectively (see Appendix D.1), which will be used as sub-protocols later. At a high level,

- For $\Pi_{\texttt{BatchPubRec}}$, we batch L degree-t Shamir sharings and follow the idea of weak public reconstruction, except that each party will set his reconstruction result as $\perp$ when he receives an ACSS proof.
- For $\Pi_{\texttt{Agreement}}$, we invoke two instances of BA protocol. All parties take their reconstruction result as the input for the first BA. Upon terminating it with an output, they check whether it equals their reconstruction result. If true, they set input 1 for the second BA and 0 otherwise. This can ensure that if all parties terminate the second BA with 1, then the output of the first BA must be some honest party's input. That can prevent all parties from agreeing on an incorrect result chosen by the adversary.

Sub-Circuit Evaluation. We note that with the help of Beaver triples, the online protocol only needs to do public reconstruction. We integrate the public reconstruction protocol into the $\Pi_{\texttt{SubCktEval}}$ (see Appendix D.2), which allows all parties to evaluate a circuit with the help of Beaver triples: In the beginning, all parties take a target circuit C' as input. We assume that all parties hold degree-t Shamir sharings of the inputs of C' as well as a sufficient number of random Beaver triples. If the circuit does not have an output layer, all parties will skip Step 3. This is the case when evaluating a segment of the whole circuit in the online phase. If all parties only want to perform public reconstruction, we may think that the circuit C' outputs its inputs. In this case, they can skip the check for Beaver triples in Step 1 and the circuit evaluation in Step 2.

To summarize, later we will use $\Pi_{\texttt{SubCktEval}}$ in a black box way to do public reconstruction in the offline phase to prepare triples. For the online phase, we will divide the circuit into several sub-circuits and let all parties invoke $\Pi_{\texttt{SubCktEval}}$ to evaluate them in order. The benefit is that when all parties fail at the current sub-circuit, they do not need to re-execute the previous sub-circuits, only the current sub-circuit.

5.3 Preparation of Beaver Triples

In this subsection, we modify the two processes in Section 4 to guarantee that all parties can eventually generate Beaver triples without additive errors.

Upgrade Process 1 to GOD. Based on $\Pi_{\texttt{tripleExt-Weak}}$, we do the following adjustments. Refer to Appendix D.4 for detailed construction and cost analysis.

- We replace $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ by $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to ensure that each party who does not receive his shares from the corrupted dealer can accuse this dealer later.

– We first check whether the triples prepared by each dealer are correct. Following by [CP23], we can let each dealer additionally prepare $\{[h_i(\alpha_\ell)]_t\}_{\ell=N'+1}^{2N'}$ and then all parties have sufficient shares to interpolate their shares of $[h_i(r)]_t$ where r is the challenge point. For each corrupted dealer who provides incorrect shares, all parties will replace their shares with zero. As a result, there are no additive errors in the extracted triples.
– We invoke $\mathcal{F}_{\mathsf{coin}}$ (see Appendix A.2) to prepare the challenge point r. For the reconstruction of each dealer P_i's $\{[f_i(r)]_t, [g_i(r)]_t, [h_i(r)]_t\}$, since there are only n values and we can directly use $\Pi_{\texttt{SubCktEval}}$ to do reconstruction. Under the party elimination framework, it may fail for at most t times while the communication costs are at most $\mathcal{O}(n^3)$ field elements.
– For extraction, we do public reconstruction through $\Pi_{\texttt{SubCktEval}}$ for each $i \in [L'+2, L]$ in order. That is because we need to divide the reconstruction of whole $\mathcal{O}(N)$ values into $\mathcal{O}(n)$ segments to keep linear communication costs.

Overview of Process 2 to GOD. Based on $\Pi_{\texttt{tripleDN-Weak}}$, we first replace $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$ and $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ by $\Pi_{\texttt{randSh}}$ (see Appendix D.3) and $\Pi_{\texttt{SubCktEval}}$, then we focus on how to prepare Beaver triples without additive errors. Recall that we use the party elimination framework here, we divide the generation of N triples into n segments and all parties will either generate $\mathcal{O}(N/n)$ triples or agree on a corrupted party in each segment. To be more concrete:

First, each party acts as P_{king} and leads an instance of $\Pi_{\texttt{tripleKingDN}}$ to generate $\mathcal{O}(1/n^2)$ fraction of triples in each segment. After agreeing on a set of successful kings, all parties invoke $\Pi_{\texttt{tripleVerify}}$ (see Appendix D.5) to check whether the prepared Beaver triples by each P_{king} in this set are correct. If there are $t+1$ distinct P_{king} provides valid triples, all parties output these Beaver triples. Otherwise, all parties will help each P_{king} identify a corrupted party. As a result, they can expect to receive a proof to accuse a corrupted party from an honest P_{king} among these $t+1$ distinct kings. Finally, all parties will agree to eliminate a corrupt party and execute the current segment again.

Recall the three issues introduced in Section 2.3 that cause the additive errors in triples, the first one is solved by the ACSS proofs. For the latter two issues, to ensure that honest P_{king} can later convince all parties which party causes the errors, we let the dealer and parties commit their degree-$2t$ shares of $[o]_{2t}$ and $[z]_{2t}$. Later, P_{king} can open the commitments to prove that the accused corrupted parties indeed provided incorrect messages.

From $\Pi_{\texttt{randShareZero-Weak}}$ to $\Pi_{\texttt{randShareZero}}$. To prevent the corrupted dealer from distributing incorrect shares and allow honest parties to accuse the corrupted dealer later, we modify $\Pi_{\texttt{Sh2tZero-Weak}}$ by adding commitments on the shares and get $\Pi_{\texttt{Sh2tZero}}$ below and $\Pi_{\texttt{randShareZero}}$ (see Appendix D.3).

Protocol $\Pi_{\texttt{Sh2tZero}}$

Let $\beta_1, \ldots, \beta_{(t+1)/2}, \alpha_0, \alpha_1, \ldots, \alpha_n$ be distinct field elements and N be the number of degree-$2t$ Shamir sharings of zero.

Dealer D

1: **Distributing Shares:** The same as $\Pi_{\texttt{Sh2tZero-Weak}}$, except that additionally prepare random degree-$(2t, t+(t-1)/2)$ bivariate polynomial $\bar{F}_\ell(x,y)$ and let $\bar{f}_\ell^{(i)}(x) = \bar{F}_\ell(x, \alpha_i)$, $\bar{g}_\ell^{(i)}(y) = \bar{F}_\ell(\alpha_i, y)$ for all $\ell \in [L]$. Then send $\{f_\ell^{(i)}(x), \bar{f}_\ell^{(i)}(x)\}_{\ell \in [L]}$ to party P_i.
2: **Computing Commitments:** Define:

$$\begin{aligned}
\boldsymbol{f}_*^{(i)}(\alpha_j) &:= (H(f_1^{(i)}(\alpha_j), \bar{f}_1^{(i)}(\alpha_j)), \ldots, H(f_L^{(i)}(\alpha_j), \bar{f}_L^{(i)}(\alpha_j))) \\
g_\ell^{(i)}(\beta_*) &:= g_\ell^{(i)}(\beta_1) || \cdots || g_\ell^{(i)}(\beta_{(t+1)/2}) \\
\bar{g}_\ell^{(i)}(\beta_*) &:= \bar{g}_\ell^{(i)}(\beta_1) || \cdots || \bar{g}_\ell^{(i)}(\beta_{(t+1)/2})
\end{aligned}$$

Locally compute:

$$\begin{aligned}
\mathsf{cm\text{-}row}_j^{(i)} &= \mathsf{MT.Com}(\boldsymbol{f}_*^{(i)}(\alpha_j)) \quad \forall i, j \in [n] \\
\mathsf{cm\text{-}col}_\ell^{(i)} &= H(g_\ell^{(i)}(\beta_*), \bar{g}_\ell^{(i)}(\beta_*)) \quad \forall i \in [n], \ell \in [L]
\end{aligned}$$

Reliably broadcast $\{\mathsf{cm\text{-}row}_j^{(i)}\}_{i,j \in [n]}, \{\mathsf{cm\text{-}col}_\ell^{(i)}\}_{i \in [n], \ell \in [L]}$.

Party P_i

1: **Verifying Row Polynomial:** Upon receiving $\{f_\ell^{(i)}(x), \bar{f}_\ell^{(i)}(x)\}_{\ell\in[L]}$ and $\{\mathsf{cm\text{-}row}_j^{(i)}\}_{j\in[n]}$ from the dealer, locally verify that $\mathsf{cm\text{-}row}_j^{(i)} \stackrel{?}{=} \mathsf{MT.Com}(\boldsymbol{f}_*^{(i)}(\alpha_j)) \quad \forall j \in [n]$. If true, send $\{f_\ell^{(i)}(\alpha_j), \bar{f}_\ell^{(i)}(\alpha_j)\}_{\ell\in[L]}$ to each party P_j and $(\texttt{support}, P_i, D)$ to all parties.

2: **Reconstructing Column Polynomial:** Upon receiving $\mathsf{cm\text{-}row}_i^{(j)}$ from the dealer and $\{f_\ell^{(j)}(\alpha_i), \bar{f}_\ell^{(j)}(\alpha_i)\}_{\ell\in[L]}$ from party P_j, verify that $\mathsf{cm\text{-}row}_i^{(j)} \stackrel{?}{=} \mathsf{MT.Com}(\boldsymbol{f}_*^{(j)}(\alpha_i))$. If true, accept $\{f_\ell^{(j)}(\alpha_i), \bar{f}_\ell^{(j)}(\alpha_i)\}_{\ell\in[L]}$ and locally compute the Merkle tree opening $\mathsf{op}[\boldsymbol{f}_*^{(j)}(\alpha_i)|\ell] = \mathsf{MT.Open}(\boldsymbol{f}_*^{(j)}(\alpha_i), \ell) \quad \forall \ell \in [L]$. Upon accepting it from $t + (t+1)/2$ distinct parties P_j, interpolate column polynomials $\{g_\ell^{(i)}(y), \bar{g}_\ell^{(i)}(y)\}_{\ell\in[L]}$.

3: **Verifying Column Polynomial:** Upon interpolating $\{g_\ell^{(i)}(y), \bar{g}_\ell^{(i)}(y)\}_{\ell\in[L]}$ and receiving $\{\mathsf{cm\text{-}col}_\ell^{(i)}\}_{\ell\in[L]}$ from the dealer, locally verify that $\mathsf{cm\text{-}col}_\ell^{(i)} \stackrel{?}{=} H(g_\ell^{(i)}(\beta_*), \bar{g}_\ell^{(i)}(\beta_*)) \quad \forall \ell \in [L]$. If true, accept $\{g_\ell^{(i)}(y)\}_{\ell\in[L]}$. Otherwise, locally generate $\texttt{ShareProof}$ which contains $t + (t+1)/2$ distinct P_j's $f_\ell^{(j)}(\alpha_i), \bar{f}_\ell^{(j)}(\alpha_i)$ and the Merkle tree opening $\mathsf{op}[\boldsymbol{f}_*^{(j)}(\alpha_i)|\ell]$. Each party who receives this proof can interpolate P_i's column polynomial and check whether it matches the dealer's commitment.

4: **Termination procedure:** The same as $\Pi_{\texttt{Sh2tZero-Weak}}$, except that P_i checks whether he has terminated the $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by dealer P_j in $\Pi_{\texttt{tripleExt}}$ and P_i will additionally output his column polynomial $\{g_\ell^{(i)}(y)\}_{\ell\in[L]}$.

Modification on Preparing Triples with P_{king}. Compared to the previous construction, we additionally let each party commit his shares of $[z]_{2t}$ when he sends it to P_{king}. Then if later P_{king} detects a corrupted party sends incorrect shares to him, he can use this commitment to accuse this corrupted party.

Process $\Pi_{\texttt{tripleKingDN}}$

The construction is same as $\Pi_{\texttt{tripleKingDN-Weak}}$, except that:

1: Denote P_i's share of $[z_\ell]_{2t}$ as $z_\ell^{(i)}$, P_i randomly samples a value $\nu^{(i)}$ and computes commitment $\tau^{(i)} = H(z_1^{(i)}||\cdots||z_N^{(i)}, \nu^{(i)})$. Then P_i reliably broadcasts $\tau^{(i)}$ and sends $\nu^{(i)}$ to P_{king}. P_{king} will accepts if P_i's shares $\{z_\ell^{(i)}\}_{\ell=1}^N$ match his commitment.

2: When P_{king} first receives $(\texttt{Proof}, D)$ from P_i and $(\texttt{Corrupt}, D)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ rather than succeeds in reconstructing secrets, P_{king} reliably broadcasts the identity of P_i and lets all parties wait to receive $(\texttt{Corrupt}, D)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$.

3: If all parties receive the identity of a party P_i, they wait to receive $(\texttt{Proof}, D)$ from P_i and $(\texttt{Corrupt}, D)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. Once these messages are received, they terminate and output the identity of the corrupted party D.

Detecting Corruptions. Based on the high-level idea introduced in Section 2.3 about how to detect corrupted parties, we design $\Pi_{\texttt{faultLoc}}$ (see Appendix D.6) to let all parties help P_{king} to identify an active corrupted party.

Upgrade Process 2 to GOD. Finally, we give the construction of $\Pi_{\texttt{tripleDN}}$. During the preparation phase, we need to let all parties prepare $n(n+t)$ groups of sharings because we use the party elimination framework. The factor n is used for all kings and the factor $n+t$ is because we may fail for additional t times due to corrupted parties. Refer to Appendix D.7 for detailed cost analysis.

Process $\Pi_{\texttt{tripleDN}}$

Let N be the number of Beaver triples to be prepared and $N' = N/(t+1)$.

Preparation.

1: All parties invoke $\Pi_{\texttt{randSh}}$ and $\Pi_{\texttt{randShareZero}}$ to prepare $(6N'+3n)(n+t)$ random degree-t Shamir sharings and $(2N'+n)(n+t)$ random degree-$2t$ Shamir sharings of 0 respectively.

2: For degree-t Shamir sharings, all parties locally divide them into $n(n+t)$ groups, each of size $6N'/n+3$. For degree-$2t$ Shamir sharing of 0, all parties have divided them into $n(n+t)$ groups in $\Pi_{\texttt{randShareZero}}$. Then we combine every group of degree-t and $2t$ Shamir sharing and denote the (i,j)-th group as $\{[a_\ell^{(i,j)}]_t, [b_\ell^{(i,j)}]_t, [r_\ell^{(i,j)}]_t, [o_\ell^{(i,j)}]_{2t}\}_{\ell=1}^{2N'/n+1}$ for $i \in [n+t], j \in [n]$.

Generation.

Divide the generation of N random Beaver triples into n segments. For each segment, let i be the group index, all parties use the $(i,*)$-th group $\{[a_\ell^{(i,j)}]_t, [b_\ell^{(i,j)}]_t, [r_\ell^{(i,j)}]_t, [o_\ell^{(i,j)}]_{2t}\}_{\ell\in[2N'/n+1],j\in[n]}$ to generate N/n random Beaver triples as follows.

1: **Accusation against Corrupted Dealer:**
Each party who receives $(\texttt{Corrupt}, D)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ (during $\Pi_{\texttt{randSh}}$) reliably broadcasts $(\texttt{Proof}, D)$ and sends request `Broadcast-Proof` to the $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by D. Each party only does this for one active corrupted D once in each segment.

2: **Generation of Random Triples:**
Each party P_j acts as P_{king} and leads an instance of $\Pi_{\texttt{tripleKingDN}}$, all parties participates in the $\Pi_{\texttt{tripleKingDN}}$ led by P_j with their shares of $\{[a_\ell^{(i,j)}]_t, [b_\ell^{(i,j)}]_t, [r_\ell^{(i,j)}]_t, [o_\ell^{(i,j)}]_{2t}\}_{\ell\in[2N'/n+1]}$.

3: **Determine the Set of Successful Kings and Outputs:**

(1). Each party sets the property Q as he terminates the $\Pi_{\texttt{tripleKingDN}}$ led by one P_{king}, and all parties invoke $\mathcal{F}_{\mathsf{acs}}$ with property Q to agree on a set $\mathcal{D}$ of successful kings with size $|\mathcal{D}| = 2t+1$.

(2). For the outputs of each $P_{\mathsf{king}} \in \mathcal{D}$, all parties proceed if all kings provide shares of Beaver triples. Otherwise, assume that P_k is the corrupted party with the smallest index in the outputs, all parties move to Step 4-(3) with the identity of P_k.

4: **Verification on Triples:**
All parties execute $\Pi_{\texttt{tripleVerify}}$ with $(2N'/n+1)(2t+1)$ shares of Beaver triples prepared in the instances of $\Pi_{\texttt{tripleKingDN}}$ led by all kings in $\mathcal{D}$:

- If there are at least $t+1$ kings who pass the verification, all parties store the total $N'/n\cdot(t+1) = N/n$ (output of the triples prepared by the first $t+1$ successful kings in $\Pi_{\texttt{tripleVerify}}$), set the group index $i = i+1$ and move to the next segment.
- Otherwise, all parties execute $\Pi_{\texttt{faultLoc}}$ for the first $t+1$ king who fails the verification and then the following steps to agree on a corrupted party.

  (1). All parties execute n instances of $\mathcal{F}_{\mathsf{ba}}$. Upon learning a corrupted P_k at the end of each instance of $\Pi_{\texttt{faultLoc}}$, each party sets his input of the k-th $\mathcal{F}_{\mathsf{ba}}$ as 1.

  (2). If a party terminates any $\mathcal{F}_{\mathsf{ba}}$ with output 1, he sets his input to 0 for the rest of the $\mathcal{F}_{\mathsf{ba}}$ (unless already set to 1) and waits for all instances of $\mathcal{F}_{\mathsf{ba}}$ to terminate. When all parties terminate all instances of $\mathcal{F}_{\mathsf{ba}}$, assuming that the k-th $\mathcal{F}_{\mathsf{ba}}$ with the smallest index and output 1, they agree on the corrupted party P_k.

  (3). All parties do the following things for corrupted party P_k:
  - If P_k has distributed degree-t Shamir sharings, all parties send `Public-Recon` to the $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by P_k and wait to receive his secrets from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. Upon receiving the secrets, all parties locally replace their degree-t Shamir sharings distributed by P_k with the secrets.
  - If P_k has distributed degree-$2t$ Shamir sharings, all parties change their $2t$ Shamir sharings distributed by P_k with zero.

  Then all parties set the group index $i = i+1$ and execute the current segment again.

Putting it all Together. We give the construction of $\Pi_{\texttt{triple}}$ in Appendix D.8 and the communication complexity is $\mathcal{O}(N\cdot n + n^5)$ elements plus $\mathcal{O}(N\cdot n\kappa + \kappa\cdot n^5)$ bits for preparing N Beaver triples.

5.4 Main Protocol for Malicious Security with GOD

We first let all parties execute $\Pi_{\texttt{triple}}$ to prepare triples in the offline phase. Then in the online phase, we divide the circuit into t disjoint sub-circuits $C_1, \ldots, C_t$ (sorted by the topology), each containing $|C|/t$ multiplication gates, then all parties invoke $\Pi_{\texttt{SubCktEval}}$ to evaluate each sub-circuit in order. Recall that when all parties' output of $\Pi_{\texttt{SubCktEval}}$ is the identity of a corrupted party, they will eliminate this party, locally update their Shamir shares, and evaluate the current sub-circuit again. Therefore, during the circuit evaluation process, all parties will invoke $\Pi_{\texttt{SubCktEval}}$ for at most $2t$ times and the communication complexity for the online phase still remains linear. We can do the same thing for the output reconstruction. As a result, we obtain the following theorem. We refer the reader to Appendix D.9 for detailed construction, security proof, and cost analysis.

Theorem 2. *Let $n = 3t+1$ and κ denote the security parameter. For a finite field $\mathbb{F}$ of size $2^{\Omega(\kappa)}$ and any circuit C of size $|C|$ and depth D, there is a fully malicious asynchronous MPC protocol computing the circuit that is secure against at most t corrupted parties with guaranteed output delivery. Let the input and output size be C_I and C_O respectively, the achieved communication complexity is $\mathcal{O}((|C|+C_I+C_O)\cdot n + D\cdot n^2 + n^5)$ field elements.*

6 Reducing Field Size

In this section, following the techniques in [GLZS24], we show how to relax the requirement of a large finite field in our constructions. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq n+1$ and $\mathbb{G}$ be an extension field of $\mathbb{F}$ such that $|\mathbb{G}| \geq 2^\kappa$, where κ is the security parameter. Let $m := [\mathbb{G} : \mathbb{F}]$ denote the extension degree.

Our goal is to evaluate an arithmetic circuit C over the finite field $\mathbb{F}$.

Step 1: Reduction from $\mathbb{G}$ to $\mathbb{F}$ in $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}, \mathcal{F}_{\mathsf{ACSS\text{-}id}}$. For any element $x \in \mathbb{G}$, it can be viewed as m elements $(x_1, \ldots, x_m)$ in $\mathbb{F}$. Therefore, for a degree-t Shamir sharing $[x]_t$ over $\mathbb{G}$ with the underlying degree-t polynomial $f(X) = a_0 + \cdots + a_t X^t$ over $\mathbb{G}$, if we consider each coefficient $a_k \in \mathbb{G}$ as m elements $\{a_{k,j}\}_{j=1}^m$ over $\mathbb{F}$, $f(X)$ can be viewed as m degree-t polynomials $\{f_j(X) = a_{0,j} + \cdots + a_{t,j}X^t\}_{j=1}^m$ over $\mathbb{F}$.

If we assign an evaluation point $\alpha_i \in \mathbb{F} \subset \mathbb{G}$ to each party P_i, each party can locally splits his share $f(\alpha_i) \in \mathbb{G}$ to $(f_1(\alpha_i), \ldots, f_m(\alpha_i)) \in \mathbb{F}^m$. Then all parties together transform $[x]_t$ over $\mathbb{G}$ to $\{[x_j]_t\}_{j=1}^m$ over $\mathbb{F}$.

Therefore, to share degree-t Shamir sharings over $\mathbb{F}$, the dealer can concatenate m degree-t Shamir sharings over $\mathbb{F}$ to a degree-t Shamir sharing over $\mathbb{G}$.

Step 2: Generating Triples Over $\mathbb{F}$ From Triples Over $\mathbb{G}$. We follow [GLZS24] to use the technique of Reverse Multiplication-Friendly Embeddings (RMFE) [CCXY18]. We first review the notion of RMFE below.

We say a pair of $\mathbb{F}$-linear maps (ϕ, ψ), where $\phi : \mathbb{F}^k \to \mathbb{G}$ and $\psi : \mathbb{G} \to F^k$, is a (k, m)-RMFE if for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$,

$$\boldsymbol{x} * \boldsymbol{y} = \psi(\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y}))$$

where $*$ denotes the coordinate-wise multiplication. From [CCXY18], there exists a family of RMFEs such that $k = \Theta(m)$. Relying on a (k, m)-RMFE, all parties can locally apply the following two operations:

- Let $[x_1]_t, \ldots, [x_k]_t$ be degree-t Shamir sharings over $\mathbb{F}$. If each party applies ϕ on his k shares, then all parties together hold a degree-t Shamir sharing of $\phi(\boldsymbol{x})$.
- Let $[y]_t$ be a degree-t Shamir sharing over $\mathbb{G}$. If each party applies ψ on his share and obtains a vector of k elements in $\mathbb{F}^k$, then all parties together hold k degree-t Shamir sharings of the elements in $\psi(y)$.

Based on these two operations, the high-level idea of generating triples over $\mathbb{F}$ is as follows. We first let all parties prepare a sufficient number of random degree-t Shamir sharings over $\mathbb{F}$ with the help of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}, \mathcal{F}_{\mathsf{ACSS\text{-}id}}$ over $\mathbb{F}$. Then we will transform every k of them into one sharing over $\mathbb{G}$. With two transformed sharings $[a]_t, [b]_t$, we consume one random Beaver triple over $\mathbb{G}$ to compute $[c]_t = [a \cdot b]_t$. After getting $([a]_t, [b]_t, [c]_t)$ over $\mathbb{G}$, we can again convert them back to k triples over $\mathbb{F}$.

To be more concrete, suppose all parties want to prepare N random Beaver triples over $\mathbb{F}$. Let m be the extension degree of $\mathbb{G}$ and k be an integer such that there exists a (k, m)-RMFE. By [CCXY18], $k = \Theta(m)$. Let $N' = \lceil N/k \rceil$. The construction is as follows.

For the malicious security with fairness construction, we first realize $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$ over $\mathbb{F}$ based on $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ over $\mathbb{F}$, then:

1. All parties invoke $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$ to prepare $2N' \cdot k$ random degree-t Shamir sharings $\{[a_j^{(i)}]_t, [b_j^{(i)}]_t\}_{i \in [N'], j \in [k]}$.
2. All parties invoke $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$ to prepare N' random Beaver triples $\{[\bar{u}^{(i)}]_t, [\bar{v}^{(i)}]_t, [\bar{w}^{(i)}]_t\}$ over $\mathbb{G}$.
3. For all $i \in [N']$, let $u^{(i)} = \phi(a_1^{(i)}, \ldots, a_k^{(i)})$ and $v^{(i)} = \phi(b_1^{(i)}, \ldots, b_k^{(i)})$. All parties locally convert $\{([a_j^{(i)}]_t, [b_j^{(i)}]_t)\}_{j=1}^k$ to $([u^{(i)}]_t, [v^{(i)}]_t)$ over $\mathbb{G}$.
4. For all $i \in [N']$, all parties consume a random Beaver triple over $\mathbb{G}$ to compute $[w^{(i)}]_t = [u^{(i)} \cdot v^{(i)}]_t$ as follows.
   (1). Compute their shares of $[u^{(i)} + \bar{u}^{(i)}]_t, [v^{(i)} + \bar{v}^{(i)}]_t$ for all $i \in [N']$.
   (2). Invoke $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ to reconstruct $u^{(i)} + \bar{u}^{(i)}, v^{(i)} + \bar{v}^{(i)}$ for all $i \in [N']$.
   (3). Locally compute:
   $$[w^{(i)}]_t = (u^{(i)} + \bar{u}^{(i)}) \cdot (v^{(i)} + \bar{v}^{(i)}) - (u^{(i)} + \bar{u}^{(i)})[\bar{v}^{(i)}]_t - (v^{(i)} + \bar{v}^{(i)})[\bar{u}^{(i)}]_t + [\bar{w}^{(i)}]_t$$
5. By the property of RMFEs, we have $\psi(w^{(i)}) = (c_1^{(i)}, \ldots, c_k^{(i)})$, where $c_j^{(i)} = a_j^{(i)} \cdot b_j^{(i)}$ for all $j \in [k]$. For all $i \in [N']$, all parties locally convert $[w^{(i)}]_t$ to $[c_1^{(i)}]_t, \ldots, [c_k^{(i)}]_t$.

6. Each party who gets his shares of $([a_j^{(i)}]_t, [b_j^{(i)}]_t, [c_j^{(i)}]_t)$ will output them. Otherwise, each party who receives `Fail` from $\mathcal{F}_{\text{randSh-Weak}}, \mathcal{F}_{\text{triple-Weak}}$ or $\mathcal{F}_{\text{pubRec-Weak}}$ will output `Fail`.

For the GOD construction, the differences are as follows:

- In step 1, replace $\mathcal{F}_{\text{randSh-Weak}}$ over $\mathbb{F}$ by $\Pi_{\texttt{randSh}}$ over $\mathbb{F}$, which can be realized by $\mathcal{F}_{\text{ACSS-id}}$ over $\mathbb{F}$.
- In step 2, replace $\mathcal{F}_{\text{triple-Weak}}$ by $\Pi_{\texttt{triple}}$.
- In step 4, we divide the reconstruction of all $\{u^{(i)} + \bar{u}^{(i)}, v^{(i)} + \bar{v}^{(i)}\}_{i\in[N']}$ into n segments and all parties invoke $\Pi_{\texttt{SubCktEval}}$ to reconstruct the values in each segment in order. If their output of $\Pi_{\texttt{SubCktEval}}$ is the identity of a corrupted party, all parties first reconstruct the secrets of degree-t Shamir sharings distributed by this party. Then they update their shares with these secrets locally and invoke $\Pi_{\texttt{SubCktEval}}$ for the current segment again.

Step 3: The Online Phase. After getting a sufficient number of random Beaver triples over $\mathbb{F}$, all parties follow the online phase of $\Pi_{\texttt{main-Fair}}$ or $\Pi_{\texttt{main-GOD}}$ to evaluate the circuit.

In summary, we obtain the following theorems based on the above techniques.

Theorem 3. *Let $n = 3t + 1$ and κ denote the security parameter. For a finite field $\mathbb{F}$ of size at least $n + 1$ and any circuit C of size $|C|$ and depth D, there is an AMPC protocol computing the circuit that is secure against at most t corrupted parties with security with fairness. Let the input and output size be C_I and C_O respectively, the achieved communication complexity is $\mathcal{O}((|C| + C_I) \cdot n + D \cdot n^2 + C_O \cdot n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits.*

Theorem 4. *Let $n = 3t + 1$ and κ denote the security parameter. For a finite field $\mathbb{F}$ of size at least $n + 1$ and any circuit C of size $|C|$ and depth D, there is a fully malicious asynchronous MPC protocol computing the circuit that is secure against at most t corrupted parties with guaranteed output delivery. Let the input and output size be C_I and C_O respectively, the achieved communication complexity is $\mathcal{O}((|C| + C_I + C_O) \cdot n + D \cdot n^2 + n^5)$ field elements plus $\mathcal{O}(\kappa \cdot n^5)$ bits.*

7 Conclusion

In this work, we presented a suite of concretely efficient Asynchronous MPC protocols using only lightweight cryptography. Our first protocol offers fairness against a malicious adversary and is hyper-efficient with a concrete communication overhead of only $O(n^3)$ field elements. The second protocol offers Guaranteed Output Delivery with overhead of only $O(n^5)$ field elements. We introduced several new primitives and techniques in distributed cryptography by employing Hash functions for verifiably detecting malicious behavior, and collectively handling corrupted parties in an asynchronous network.

Acknowledgements. X. Ji and Y. Song were supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61361136003. C. Liu-Zhang and Y. Song were supported in part by the ETH Zurich Leading House Research Partnership Grant RPG-072023-19. A. Bandarupalli and A. Kate were supported in part by Sui Academic Research Award and NIFA/USDA award number 2021-67021-34252.

References

[AAPP24] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Perfect asynchronous MPC with linear communication overhead. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 280–309. Springer, Cham, May 2024.

[ABCP23] Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. VSS from distributed ZK proofs and applications. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part I*, volume 14438 of *LNCS*, pages 405–440. Springer, Singapore, December 2023.

[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.

[AJM+23] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part I*, page 39–70, Berlin, Heidelberg, 2023. Springer-Verlag.

[BBB+24] Akhil Bandarupalli, Adithya Bhat, Saurabh Bagchi, Aniket Kate, and Michael K. Reiter. Hashrand: Efficient asynchronous random beacon without threshold cryptographic setup. ACM CCS 2024 (to appear), 2024. `https://eprint.iacr.org/2023/451`.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[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.

[BKLZL20] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 353–380. Springer, Cham, November 2020.

[BKP11] Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 590–609, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[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.

[BOKR94] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '94, page 183–192, New York, NY, USA, 1994. Association for Computing Machinery.

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

[BTH06] Zuzana Beerliova-Trubiniova and Martin Hirt. Efficient multi-party computation with dispute control. In *Theory of Cryptography Conference*, pages 305–328. Springer, 2006.

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

[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.

[CCXY18] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 395–426. Springer, Cham, August 2018.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16*, pages 55–72. Springer, 2013.

[CGHZ16] Sandro Coretti, Juan A. Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multi-party computation based on one-way functions. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 998–1021. Springer, Berlin, Heidelberg, December 2016.

[CHLZ21] Annick Chopard, Martin Hirt, and Chen-Da Liu-Zhang. On communication-efficient asynchronous MPC with adaptive security. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 35–65. Springer, Cham, November 2021.

[Coh16] Ran Cohen. Asynchronous secure multiparty computation in constant time. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 183–207. Springer, Berlin, Heidelberg, March 2016.

[CP15] Ashish Choudhury and Arpita Patra. Optimally resilient asynchronous MPC with linear communication complexity. In *Proc. Intl. Conference on Distributed Computing and Networking (ICDCN)*, pages 1–10, 2015.

[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.

[DDL+24] Sourav Das, Sisi Duan, Shengqi Liu, Atsuki Momose, Ling Ren, and Victor Shoup. Asynchronous consensus without trusted setup or public-key cryptography. *Cryptology ePrint Archive*, 2024.

[DGKN09] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 160–179. Springer, Berlin, Heidelberg, March 2009.

[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.

[DXR21] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2705–2721. ACM Press, November 2021.

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

[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.

[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.

[GPS22] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 3–32. Springer, Cham, August 2022.

[HNP05] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multiparty computation with optimal resilience (extended abstract). In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 322–340. Springer, Berlin, Heidelberg, May 2005.

[HNP08] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Asynchronous multi-party computation with quadratic communication. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 473–485. Springer, Berlin, Heidelberg, July 2008.

[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.

[LYK+19] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew K. Miller. Honeybadgermpc and asynchromix: Practical asynchronous MPC and its application to anonymous communication. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 887–903. ACM, 2019.

[MMR15] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with $t < n/3$, $o(n^2)$ messages, and $o(1)$ expected time. *J. ACM*, 62(4), 8 2015.

[MR17] Achour Mostéfaoui and Michel Raynal. Signature-free asynchronous byzantine systems: from multivalued to binary consensus with t< n/3 t< n/3, o (nˆ 2) o (n 2) messages, and constant time. *Acta Informatica*, 54:501–520, 2017.

[NRS+20] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H Vaidya, and Zhuolun Xiang. Improved extension protocols for byzantine broadcast and agreement. *arXiv preprint arXiv:2002.11321*, 2020.

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

[PCR08] Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Efficient asynchronous multiparty computation with optimal resilience. Cryptology ePrint Archive, Report 2008/425, 2008.

[PCR09] Arpita Patra, Ashish Choudhary, and C Pandu Rangan. Efficient statistical asynchronous verifiable secret sharing with optimal resilience. In *International Conference on Information Theoretic Security*, pages 74–92. Springer, 2009.

[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.

[SS24] Victor Shoup and Nigel P. Smart. Lightweight asynchronous verifiable secret sharing with optimal resilience. *J. Cryptol.*, 37(3):27, 2024.

[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

A.1 Definitions of Agreement Primitives

We describe functionalities for the agreement primitives, following the descriptions from [CGHZ16, Coh16].

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.

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

$\mathcal{F}_{\mathsf{rbc}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$, where one of the parties is the sender P_s, and the adversary $\mathcal{S}$. Initialize $y = \perp$.

1: Upon receiving an input v from party P_s (the sender, or the adversary on behalf of the corrupted sender), set the output to $y = v$ and send v to the adversary.
2: Upon receiving v from the adversary, if P_s is corrupted and no party has received their output, then set $y = v$.
3: When the output is y is set to be some value v, the functionality outputs y as a request-based delayed output to all parties.

Byzantine Agreement. We describe the functionality $\mathcal{F}_{\mathsf{ba}}$ for Byzantine agreement.

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

$\mathcal{F}_{\mathsf{ba}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and the ideal adversary $\mathcal{S}$. Let $\mathcal{I} = \mathcal{H}$, where $\mathcal{H}$ is the set of honest parties. For each party P_i, initialize x_i and y_i to $\perp$. Let the message length be L.

1: Upon receiving $\mathcal{P}'$ from $\mathcal{S}$, with $|\mathcal{P}'| \leq t$, if no party has received output, then set $\mathcal{I} = \mathcal{H} \setminus \mathcal{P}'$.
2: Upon receiving a message $m \in \{0,1\}^L$ from party P_i, do as follows.
   – If any party or $\mathcal{S}$ has received an output y, then ignore this message; otherwise, set $x_i = m$.
   – If $x_i \neq \perp$ for every $P_i \in \mathcal{I}$, then set $y_j = y$ for every $j \in [n]$, where $y = x$ if all inputs $x_j = x$ for $P_j \in \mathcal{I}$, for some $x \neq \perp$. Otherwise, set $y = x_j$ for $P_j \notin \mathcal{H}$ with the smallest index.
   – Send m to $\mathcal{S}$.
3: When the output y_i is set to be some value y, the functionality outputs y as a request-based delayed output to P_i.

Reliable Agreement. We describe the functionality $\mathcal{F}_{\mathsf{ra}}$ for Reliable Agreement.

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

$\mathcal{F}_{\mathsf{ra}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and the ideal adversary $\mathcal{S}$. Let $\mathcal{I} = \mathcal{H}$, where $\mathcal{H}$ is the set of honest parties. For each party P_i, initialize x_i and y_i to $\perp$. Set $\mathsf{AdvDeliver} = 0$.

1: Upon receiving $\mathcal{P}'$ from $\mathcal{S}$, with $|\mathcal{P}'| \leq t$, if no party has received output, then set $\mathcal{I} = \mathcal{H} \setminus \mathcal{P}'$.
2: Upon receiving a bit message m from party P_i, do as follows.
   – If any party or $\mathcal{S}$ has received an output y, then ignore this message; otherwise, set $x_i = m$.
   – If $x_i \neq \perp$ for every $P_i \in \mathcal{I}$, then set $y_j = y$ for every $j \in [n]$, where $y = x$ if all inputs $x_j = x$ for $P_j \in \mathcal{I}$, for some $x \neq \perp$. Otherwise, set $\mathsf{AdvDeliver} = 1$.
   – Send m to $\mathcal{S}$.
3: Upon receiving v from the adversary, if $\mathsf{AdvDeliver} = 1$, then set $y_i = v$ for every $i \in [n]$.
4: When the output y_i is set to be some value y, the functionality outputs y as a request-based delayed output 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 $\mathcal{F}_{\mathsf{acs}}$

$\mathcal{F}_{\mathsf{acs}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and the adversary $\mathcal{S}$. Initialize $S_i = \varnothing$ for every $i \in [n]$, and $S = \perp$.

1: Upon receiving an index k from P_i, add index k to S_i. Then forward k to $\mathcal{S}$. If $|S_i| \geq n - t$, then we say that P_i is ready. If $n - t$ honest parties are ready, set S to be the set of indices k such that there is some honest party that inputs k.
2: Upon receiving S' from $\mathcal{S}$, check that $|S'| \geq n - t$, and that for every $k \in S'$, there is some honest party that has input k. If so, then set $S = S'$.
3: Upon setting S, output it to all parties as a request-based delayed output.

[DDL+24] gives a construction of ACS protocol which only requires random oracle assumption. For their construction, at a high level, they first design the VABA protocol defined below, which can be considered as a special ACS protocol that allows all parties to agree on the identity of one party. Then they let each party propose a potential set and jointly invoke one instance of VABA to agree on one party's proposal. As a result, after agreeing on one party's identity, all parties can wait to receive the set broadcast by this party. Inspired by their construction, we give Π^{Q}_{acs} which only modifies the size of the agreement set from $2t+1$ to L (in Step 3) to make it applicable in our construction. The communication complexity is $\mathcal{O}(\kappa \cdot n^3)$.

Definition 3. *Let Π be an n-party protocol where all parties take as input a global ACS property Q and each party P_i outputs the identity of a party P_{k_i}. We say that Π is a t-resilient VABA protocol for Q if the following holds whenever up to t parties are corrupted:*

- *Consistency: Each honest party outputs the same party's identity $P_{k_i} = P_k$.*
- *Validity: There exists at least one honest party that likes P_k for Q.*
- *Termination: All honest parties eventually terminate.*

Protocol Π^{Q}_{acs}

Let Q be the property needed to be satisfied, L be the size of agreement set.

1: Each party P_i initializes sets $\texttt{Valid}_i = \varnothing$.
2: For each party P_j, when P_i considers P_j satisfies the property Q, he adds P_j to $\texttt{Valid}_i$.
3: When $|\texttt{Valid}_i| = L$, P_i reliably broadcasts set $I_i = \texttt{Valid}_i$. P_i can still update his $\texttt{Valid}_i$.

4: Each party P_i set the property Q' as he receives $I_j \subseteq \texttt{Valid}_i, |I_j| = L$ from P_j. Then all parties invoke VABA with property Q' to agree on a party P_k.
5: Upon terminating the VABA protocol and getting the identity of P_k, all parties wait to receive the agreement set of size L from P_k's reliable broadcast.

A.2 Preparing Random Coin

The description of $\mathcal{F}_{\mathsf{coin}}$ appears below. Such functionality can be realized by first preparing a random degree-t Shamir sharing $[r]_t$ and then using $\mathcal{F}_{\mathsf{pubRec}}$ to reconstruct the secrets to all parties. Following [CP23], $\mathcal{F}_{\mathsf{coin}}$ can be realized with communication complexity $\mathcal{O}(n^3)$ elements.

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

$\mathcal{F}_{\mathsf{coin}}$ proceeds as follows, running with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$.
1: Upon receiving $2t + 1$ parties' requests, the trusted party samples a random value r.
2: The trusted party sends r to all parties.
3: All honest parties output the results received from the trusted party. Corrupted parties may output anything they want.

A.3 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 $\alpha_1, \ldots, \alpha_n$ be n 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 $f(0) = x$ 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.
$$\forall\ [x]_d, [y]_d,\ [x \cdot y]_{2d} = [x]_d \cdot [y]_d.$$

Packed Shamir Sharings. The packed Shamir secret sharing, introduced by Franklin and Yung [FY92], is a generalization of the standard Shamir secret sharing scheme. Let k be the number of secrets to pack in one sharing. Let $\beta_1, \ldots, \beta_k$ be k distinct elements that are different from $\alpha_1, \ldots, \alpha_n$ in $\mathbb{F}$. A *degree-d* ($d \geq k - 1$) packed Shamir sharing of $\boldsymbol{x} = (x_1, \ldots, x_k) \in \mathbb{F}^k$ is a vector $(x_1, \ldots, x_n)$ for which there exists a polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most d such that $f(\beta_i) = x_i$ for all $i \in \{1, 2, \ldots, k\}$, and $f(\alpha_i) = x_i$ for all $i \in \{1, 2, \ldots, n\}$.

Reconstructing a degree-d packed Shamir sharing requires $d+1$ shares and can be done by Lagrange interpolation. For a random degree-d packed Shamir sharing of $\boldsymbol{x}$, any $d-k+1$ shares are independent of the secret $\boldsymbol{x}$. If $d - (k - 1) \geq t$, then knowing t of the shares does not leak anything about the k secrets. In particular, a sharing of degree $t + (k - 1)$ keeps hidden the underlying k secret.

B Proofs of Our Malicious Secure ACSS Construction in the Random Oracle

In the following, we prove the security of our protocol by constructing an ideal adversary $\mathcal{S}$. $\mathcal{S}$ needs to interact with the environment $\mathcal{Z}$ and with the ideal functionalities. $\mathcal{S}$ constructs virtual real-world honest parties and runs the real-world adversary $\mathcal{A}$. For simplicity, we just let $\mathcal{S}$ communicate with $\mathcal{A}$ on

behalf of honest parties and the ideal functionality of sub-protocols in our proof. In order to simulate the communication with $\mathcal{Z}$, every message that $\mathcal{S}$ receives from $\mathcal{Z}$ is sent to $\mathcal{A}$, and likewise, every message sent from $\mathcal{A}$ sends to $\mathcal{Z}$ is forwarded by $\mathcal{S}$. Each time an honest party needs to send a message to another honest party, $\mathcal{S}$ will tell $\mathcal{A}$ that a message has been delivered such that $\mathcal{A}$ can tell $\mathcal{S}$ the arrival time of this message to help $\mathcal{S}$ instruct the functionalities to delay the outputs in the ideal world. For each request-based delayed output that needs to be sent to an honest party, we let $\mathcal{S}$ delay the output in default until we say $\mathcal{S}$ allows the functionality to send the output.

B.1 Proof for ACSS with Abort

Lemma 5. *Protocol $\Pi_{\mathsf{ACSS\text{-}ab}}$ securely computes $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ in the $\{\mathcal{F}_{\mathsf{rbc}}, \mathcal{F}_{\mathsf{ra}}\}$-hybrid model and the Random Oracle model against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t = \frac{n-1}{3}$ parties.*

Proof. **Termination**: We first show that either all honest parties terminate $\Pi_{\mathsf{ACSS\text{-}ab}}$ or no honest party terminates it. Then, we show that each terminating honest party either outputs a share or $\langle Abort \rangle$. The $\mathcal{F}_{\mathsf{rbc}}$ and $\mathcal{F}_{\mathsf{ra}}$ functionalities ensure totality i.e. if an honest party terminates, then eventually every honest party will terminate. The $\mathcal{F}_{\mathsf{ra}}$ functionality also requires at least $t+1$ honest parties to input 1, which ensures at least $t+1$ honest parties receive their row and column polynomials from the dealer D.

In case honest parties receive output from $\mathcal{F}_{\mathsf{ra}}$, they proceed to the share interpolation phase. In this phase, each honest party receives at least $t+1$ points on its column polynomials in the form of $\langle Row \rangle$ messages. This is because the honest parties that input 1 to $\mathcal{F}_{\mathsf{ra}}$ will send $\langle Row \rangle$ messages to every other honest party. Every honest party will interpolate a column polynomial. Then, each honest party will also send common points on row polynomials in the form of $\langle Column \rangle$ messages. As there are $2t+1$ honest parties, each party will receive sufficient points on its row polynomials. Therefore, every honest party will interpolate a row polynomial. It then verifies if its row polynomial is correct based on commitments delivered by $\mathcal{F}_{\mathsf{rbc}}$. Finally, based on the result of this verification, each honest party either outputs its shares or outputs $\langle Abort \rangle$.

In case honest parties do not receive output from $\mathcal{F}_{\mathsf{ra}}$, no party terminates the protocol.

Next, we prove the security of $\Pi_{\mathsf{ACSS\text{-}ab}}$ using a simulator $\mathcal{S}_{\mathsf{ACSS\text{-}Abort}}$ and hybrid arguments.

Simulator $\mathcal{S}$

When D is honest

Sharing Phase

1: Let $\mathcal{P}$ denote the set of all parties, $\mathcal{H}$ denote the set of honest parties, and $\mathcal{P}_{\mathcal{C}orr}$ denote the set of corrupted parties. The simulator $\mathcal{S}$ receives L shares of corrupted parties $\mathcal{P}_{\mathcal{C}orr}$ from the functionality $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$. We denote these shares as $s_{i,j}$ for $i \in [1, L]$ and $j \in \mathcal{P}_{\mathcal{C}orr}$.

2: **Encoding Shares**: For each share polynomial $i \in [1, L]$, set $f_i(\alpha_j) = s_{i,j}$ for $j \in \mathcal{P}_{\mathcal{C}orr}$, $i \in [1, L]$.
- **Row Polynomials**: $\mathcal{S}$ randomly samples $\frac{L}{t+1}$ degree-$2t$ row polynomials $F_i(x, \alpha_j)$ such that $F_i(-\alpha_{k+1}, \alpha_j) = f_{(i-1)*(t+1)+k}(\alpha_j)$ for $j \in \mathcal{P}_{\mathcal{C}orr}$.
- **Column Polynomials**: It also randomly samples degree-t column polynomials $F_i(\alpha_j, y)$: $F_i(\alpha_j, \alpha_k) = F_i(\alpha_k, \alpha_j)$ for $P_j, P_k \in \mathcal{P}_{\mathcal{C}orr}$.
- **Nonce Polynomials**: It also randomly samples degree-t nonce polynomials $Y_i(\alpha_j, y) : Y_i(\alpha_j, \alpha_k) = Y_i(\alpha_k, \alpha_j)$ and blinding nonce polynomials $Y_0(\alpha_j, y) : Y_0(\alpha_j, \alpha_k) = Y_0(\alpha_k, \alpha_j)$ for $P_j, P_k \in \mathcal{P}_{\mathcal{C}orr}$.

3: **Commitments**: $\mathcal{S}$ simulates the random oracle H by randomly sampling n values $\boldsymbol{C}[i]$ for $i \in [1, n]$ as the output of H. For $P_j \in \mathcal{P}_{\mathcal{C}orr}$, $\mathcal{S}$ maps the input $(f_1(\alpha_j), \ldots, f_L(\alpha_j), Y(\alpha_0, \alpha_j))$ to the value $\boldsymbol{C}[j]$. If these values have been mapped to some inputs queried by $\mathcal{A}$, $\mathcal{S}$ aborts.

4: **dZK proofs**:
- $\mathcal{S}$ next samples a vector of n random values as $\boldsymbol{\mathcal{C}}[i]$ for $i \in [1, n]$. If these values have been mapped to some inputs queried by $\mathcal{A}$, $\mathcal{S}$ aborts.
- $\mathcal{S}$ samples a random value as d and maps input $(\boldsymbol{C}, \boldsymbol{\mathcal{C}})$ to value d. If d has been mapped to some inputs queried by $\mathcal{A}$, $\mathcal{S}$ aborts.
- $\mathcal{S}$ samples a random degree-t polynomial $r(x)$. It also computes $f_0(\alpha_j) := r(\alpha_j) + \sum_{i \in [1,L]} d^i f_i(\alpha_j)$ for $P_j \in \mathcal{P}_{\mathcal{C}orr}$.
- It maps $(f_0(\alpha_j), Y_0(\alpha_0, \alpha_j))$ to value $\boldsymbol{\mathcal{C}}[j]$ for $j \in \mathcal{P}_{\mathcal{C}orr}$.

$\mathcal{S}$ also intercepts $\mathcal{A}'s$ calls to the RO H and replies with the mapped output if the input is already mapped. Otherwise, $\mathcal{S}$ picks a random value and maps the input to this value.

5: **Send shares and broadcast commitments**: $\mathcal{S}$ sends row, column, and nonce polynomials to parties $P_j \in \mathcal{P}_{\mathcal{C}orr}$ on behalf of D. Further, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{rbc}}$ to deliver $\boldsymbol{C}, \boldsymbol{\mathcal{C}}, r(x)$ to parties in $\mathcal{P}_{\mathcal{C}orr}$.

6: In the following, $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{ra}}$. For each honest party, $\mathcal{S}$ sets his input for $\mathcal{F}_{\mathsf{ra}}$ as 1 when this honest party receives $\langle Share \rangle$ from the dealer. When $\mathcal{S}$ learns the output of $\mathcal{F}_{\mathsf{ra}}$ is 1, $\mathcal{S}$ continues to do the following simulation.

7: **Column Interpolation**: $\mathcal{S}$ sends common shares on row polynomials to parties in $\mathcal{P}_{\mathcal{C}orr}$. Further, $\mathcal{S}$ also notifies $\mathcal{A}$ of scheduled messages between the simulated honest parties. Note that the adversary $\mathcal{A}$ controls and provides the order of messages among honest parties to $\mathcal{S}$. Consider the set T of the first $t+1$ $\langle Row \rangle$ messages received by a simulated honest party P_k. $\mathcal{S}$ defines degree-t polynomials $\Delta F_1(\alpha_k, y), \ldots, \Delta F_{\frac{L}{t+1}}(\alpha_k, y)$ such that for all $i \in \{1, \ldots, \frac{L}{t+1}\}$, $\Delta F_i(\alpha_k, \alpha_\ell) = 0$ when $P_\ell \in T$ is an honest party, and $\Delta F_i(\alpha_k, \alpha_\ell)$ is the difference between the received share from P_ℓ and the correct share when $P_\ell \in T$ is a corrupted party. Similarly, $\mathcal{S}$ also defines $\Delta Y(\alpha_k, y), \Delta Y_0(\alpha_k, y)$ in a similar way.

8: **Row Interpolation Phase**: Then, $\mathcal{S}$ also sends common shares on column polynomials $\langle Column, (F_1'(\alpha_k, \alpha_\ell), \ldots, F'_{\frac{L}{t+1}}(\alpha_k, \alpha_\ell), Y'(\alpha_k, \alpha_\ell), {Y_0}'(\alpha_k, \alpha_\ell)) \rangle$ to parties in $P_\ell \in \mathcal{P}_{\mathcal{C}orr}$ on behalf of honest party P_k:

- For all $i \in \{1, \ldots, \frac{L}{t+1}\}$, $\mathcal{S}$ sets $F_i'(\alpha_k, \alpha_\ell) = F_i(\alpha_k, \alpha_\ell) + \Delta F_i(\alpha_k, \alpha_\ell)$.
- $\mathcal{S}$ sets $Y'(\alpha_k, \alpha_\ell) = Y(\alpha_k, \alpha_\ell) + \Delta Y(\alpha_k, \alpha_\ell)$ and ${Y_0}'(\alpha_k, \alpha_\ell) = Y_0(\alpha_k, \alpha_\ell) + \Delta Y_0(\alpha_k, \alpha_\ell)$.

Note that the adversary $\mathcal{A}$ controls and provides the order of messages among honest parties to $\mathcal{S}$. Consider the set T' of the first $2t+1$ $\langle Column \rangle$ messages received by a simulated honest party P_ℓ. $\mathcal{S}$ defines degree-$2t$ polynomials $\Delta F_1(x, \alpha_\ell), \ldots, \Delta F_{\frac{L}{t+1}}(x, \alpha_\ell)$ such that for all $i \in \{1, \ldots, \frac{L}{t+1}\}$:

- For honest party $P_k \in T'$, if P_k receives his row polynomials before $\mathcal{S}$ learns output 1 from $\mathcal{F}_{\mathsf{ra}}$, $\mathcal{S}$ has computed $\Delta F_i(\alpha_k, \alpha_\ell)$. Otherwise, $\mathcal{S}$ sets $\Delta F_i(\alpha_k, \alpha_\ell) = 0$.
- For corrupted party $P_k \in T'$, $\Delta F_i(\alpha_k, \alpha_\ell)$ is the difference between the received share from P_k and the correct share.

Similarly, $\mathcal{S}$ also defines $\Delta Y(x, \alpha_\ell), \Delta Y_0(x, \alpha_\ell)$ in a similar way.

Then $\mathcal{S}$ verifies that:

- For all $i \in \{1, \ldots, \frac{L}{t+1}\}$, $\Delta F_i(-\alpha_{j+1}, \alpha_\ell) = 0$ for all $j \in [1, t+1]$.
- $\Delta Y(x, \alpha_\ell), \Delta Y_0(x, \alpha_\ell)$ are of degree-t and $\Delta Y(\alpha_0, \alpha_\ell) = 0$, $\Delta Y_0(\alpha_0, \alpha_\ell) = 0$.

If any check fails and P_k has not received his shares from the dealer before terminating $\mathcal{F}_{\mathsf{ra}}$, $\mathcal{S}$ sends $(Abort, P_k)$ to $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$.

Public Reconstruction Phase

1: For each honest party P_j, for each $\langle PubRec \rangle$ message received from a corrupted party, $\mathcal{S}$ follows the protocol to check whether it is correct. If it is received from an honest party, $\mathcal{S}$ directly considers it to be correct. When P_j receives $t+1$ correct $\langle PubRec \rangle$ messages, $\mathcal{S}$ delivers the output from $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ to P_j.

2: For each corrupted party P_j, $\mathcal{S}$ first receives $f_1(\cdot), \ldots, f_L(\cdot)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$. Then for each honest party P_ℓ who accepts his row polynomials, $\mathcal{S}$ randomly samples P_ℓ's $Y(\alpha_0, \alpha_\ell), Y_0(\alpha_0, \alpha_\ell)$ based on shares of corrupted parties. Then $\mathcal{S}$ sends $f_1(\alpha_\ell), \ldots, f_L(\alpha_\ell), Y(\alpha_0, \alpha_\ell), Y_0(\alpha_0, \alpha_\ell)$ to corrupted P_j on behalf of P_ℓ. When corrupted party P_j queries the H with these inputs, $\mathcal{S}$ returns the random values $\boldsymbol{C}[k], \boldsymbol{\mathcal{C}}[k]$ he sampled in Step 3 and 4 in the Sharing phase as the output of H. If these inputs have been queried, $\mathcal{S}$ aborts the simulation.

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

$\mathbf{Hyb}_1$: In this hybrid, $\mathcal{S}$ first generates corrupted parties' row and column polynomials. Then $\mathcal{S}$ generates the whole bivariate polynomials given corrupted parties' polynomials and the secrets. We only change the generation order of bivariate polynomials, which makes no difference. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are identically distributed.

$\mathbf{Hyb}_2$: In this hybrid, we delay the generation of honest parties' row and column polynomials until the Share Interpolation Phase. For each honest party P_i, $\mathcal{S}$ randomly samples values as $\boldsymbol{C}[i], \boldsymbol{\mathcal{C}}[i], d$. If these values have been mapped to some inputs queried by $\mathcal{A}$, $\mathcal{S}$ aborts the simulation. Assuming that $\mathcal{A}$ can query for $\mathtt{poly}(\kappa)$ times, since the output space of H is at least κ bits, the probability is at most $\frac{n\mathtt{poly}(\kappa)}{2^\kappa - \mathtt{poly}(\kappa)}$ (applying the union bound on number of honest parties' commitments), which are all negligible in the security parameter κ. Therefore, the distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are computationally indistinguishable.

$\mathbf{Hyb}_3$: In this hybrid, we change the generation of $r(x)$. $\mathcal{S}$ randomly samples $r(x)$ such that $r(\alpha_j) = f_0(j) - \sum_{i=1}^{L} d^i f_i(\alpha_j)$ for each $P_j \in P_{\mathcal{C}orr}$. Then $\mathcal{S}$ re-computes $f_0(x)$ and records the map relation $\boldsymbol{\mathcal{C}}[i] = H(f_0(\alpha_j), Y_0(\alpha_0, \alpha_j))$ for each honest party P_j. Since $f_0(x) = r(x) + \sum_{i=1}^{L} d^i f_i(x)$, we only change the generation order of $f_0(x)$ and $r(x)$, which makes no difference. Thus, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ are identically distributed.

$\mathbf{Hyb}_4$: In the following, we focus on the simulation of the Share Interpolation Phase. We further delay the generation of honest party's row and column polynomials until the end of this phase.

$\mathbf{Hyb}_{4.1}$: In this hybrid, when an honest party P_j needs to sends $\langle Row \rangle$ to corrupted party P_k, $\mathcal{S}$ uses P_k's column polynomial $\{F_i(\alpha_k, y)\}_{i=1}^{L/(t+1)}, Y(\alpha_,y), Y_0(\alpha_k, y)$ to compute the $\langle Row \rangle$ message and sends it to P_k on behalf of P_j. $\mathbf{Hyb}_{4.1}$ and $\mathbf{Hyb}_3$ are identically distributed.

$\mathbf{Hyb}_{4.2}$: In this hybrid, for each honest party P_k who needs to reconstruction his column polynomials, let T be the set of first $t+1$ $\langle Row \rangle$ messages P_k received. Then $\mathcal{S}$ computes $\Delta F_i(\alpha_k, y)$ based on $\{\Delta F_i(\alpha_k, \alpha_\ell)\}_{\ell \in T}$ for all $i \in [L/(t+1)]$. $\mathcal{S}$ does the same thing to compute $\Delta Y(\alpha_k, y), \Delta Y_0(\alpha_k, y)$. These $\Delta F_i(\alpha_k, y), \Delta Y(\alpha_k, y), \Delta Y_0(\alpha_k, y)$ have not been used so far, $\mathbf{Hyb}_{4.2}$ and $\mathbf{Hyb}_{4.1}$ are identically distributed.

$\mathbf{Hyb}_{4.3}$: In this hybrid, when an honest party P_k needs to send $\langle Column \rangle$ to corrupted P_ℓ, $\mathcal{S}$ first uses P_ℓ's row polynomial $\{F_i(x, \alpha_\ell)\}_{i=1}^{L/(t+1)}, Y(x, \alpha_\ell), Y_0(x, \alpha_\ell)$ to compute P_k's $\{F_i(\alpha_k, \alpha_\ell)\}_{i=1}^{L/(t+1)}, Y(\alpha_k, \alpha_\ell), Y_0(\alpha_k, \alpha_\ell)$, then $\mathcal{S}$ computes $F'_i(\alpha_k, \alpha_\ell) = F_i(\alpha_k, \alpha_\ell) + \Delta F_i(\alpha_k, \alpha_\ell)$ for all $i \in [L/(t+1)]$. $\mathcal{S}$ also does the same thing to compute $Y'(\alpha_k, \alpha_\ell), Y'_0(\alpha_k, \alpha_\ell)$. Then $\mathcal{S}$ uses these $\{F'_i(\alpha_k, \alpha_\ell)\}_{i=1}^{L/(t+1)}, Y'(\alpha_k, \alpha_\ell), Y'_0(\alpha_k, \alpha_\ell)$ as P_k's $\langle Column \rangle$ message and sends it to P_ℓ on behalf of P_k. $\mathbf{Hyb}_{4.3}$ and $\mathbf{Hyb}_{4.2}$ are identically distributed.

$\mathbf{Hyb}_{4.4}$: In this hybrid, for each honest party P_ℓ who needs to reconstruction his row polynomials, let T' be the set of first $2t+1$ $\langle Column \rangle$ messages P_ℓ received. $\mathcal{S}$ computes $\Delta F_i(x, \alpha_\ell)$ based on $\{\Delta F_i(\alpha_k, \alpha_\ell)\}_{k \in T'}$ for all $i \in [L/(t+1)]$. $\mathcal{S}$ also does the same thing to compute $Y'(x, \alpha_\ell), Y'_0(x, \alpha_\ell)$. Then $\mathcal{S}$ only checks whether $\Delta F_i(-\alpha_{j+1}, \alpha_\ell) = 0$ for all $j \in [t+1], i \in [L/(t+1)]$, $\Delta Y(x, \alpha_\ell), \Delta Y_0(x, \alpha_\ell)$ are of degree-t and $\Delta Y(\alpha_0, \alpha_\ell) = \Delta Y_0(\alpha_0, \alpha_\ell) = 0$. If any verification fails, $\mathcal{S}$ will send $(\texttt{abort}, P_\ell)$ to $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$. Since the probability of $\mathcal{A}$ can find a second pre-image that causes the collision is negligible in the security parameter κ, therefore, $\mathbf{Hyb}_{4.4}$ and $\mathbf{Hyb}_{4.3}$ are computationally indistinguishable.

$\mathbf{Hyb}_5$: In the following, $\mathcal{S}$ no longer uses honest dealer's secrets to compute $f_0(x)$ and honest parties' row and column polynomials since they are never used. $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_{4.4}$ are identically distributed.

$\mathbf{Hyb}_6$: In this hybrid, $\mathcal{S}$ first receives honest dealer's secrets from $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$. Then $\mathcal{S}$ computes each honest party P_ℓ's $\langle PubRec \rangle$ message (randomly samples values as $Y(0, \ell), Y_0(0, \ell)$ based on shares of corrupted parties). For each honest party P_ℓ who does not abort during the Share Interpolation Phase, $\mathcal{S}$ sends the $\langle PubRec \rangle$ to the corrupted parties on behalf of P_ℓ. When corrupted parties query H with these inputs, $\mathcal{S}$ returns the random values $\boldsymbol{C}[\ell], \boldsymbol{\mathcal{C}}[\ell]$ he sampled in the sharing phase as the output of H. If these inputs have been queried, $\mathcal{S}$ aborts the simulation. $\mathcal{S}$ also aborts the simulation if shares sent by a corrupted party pass the verification, and are not equal to the shares sampled by $\mathcal{S}$.

The difference between $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ is the probability of $\mathcal{S}$ abort. This probability equals $\mathcal{A}$ correctly guesses and queries honest parties' shares and nonce before receiving. Assuming that $\mathcal{A}$ can query for $\texttt{poly}(\kappa)$ times, since the nonce is κ bits, it is at most $\frac{\texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)}$. Further, a corrupted party must find a pre-image for at least one of the commitments to pass the commitment and dZK proof checking. This probability is bounded by $\frac{n\texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)}$. Overall, the probability $\mathcal{S}$ aborts is $\leq \frac{n\texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)}$, which is negligible in the security parameter κ. Therefore, the distributions of $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ are computationally indistinguishable.

Note that $\mathbf{Hyb}_6$ corresponds to the ideal world, then $\Pi_{\mathsf{ACSS\text{-}ab}}$ securely computes $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ with error $\frac{n\texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)}$.

Simulator $\mathcal{S}$

When D is corrupted

Sharing Phase

1: **Share Recording**: For each honest party P_k, the simulator $\mathcal{S}$ records shares messages:

$$\langle Shares, (F_1(x, \alpha_k), \ldots, F_{\frac{L}{t+1}}(x, \alpha_k), Y(x, \alpha_k), Y_0(x, \alpha_k)),$$
$$(F_1(\alpha_k, y), \ldots, F_{\frac{L}{t+1}}(\alpha_k, y), Y(\alpha_k, y), Y_0(\alpha_k, y))\rangle$$

received from the D.

2: **Simulate Broadcast**: $\mathcal{S}$ simulates the $\mathcal{F}_{\mathsf{rbc}}$ functionality when D invokes it to broadcast commitments and dZK proofs.

3: **Verify dZK proofs and construct polynomials**: For each honest party P_k's $\langle Shares \rangle$ message $\mathcal{S}$ received in Step 1, $\mathcal{S}$ follows to protocol to do the verification. If true, $\mathcal{S}$ considers this P_k's message

$\langle Shares \rangle$ is correct. When $\mathcal{S}$ receives correct $\langle Shares \rangle$ messages for $t+1$ distinct honest party P_k, it interpolates degree-$(2t,t)$ bivariate polynomials $F'_1(x,y),\ldots,F'_{\frac{L}{t+1}}(x,y)$ and degree-(t,t) bivariate polynomials $Y(x,y)$ and $Y_0(x,y)$. $\mathcal{S}$ also computes share polynomials $f'_1(x),\ldots,f'_L(x)$ from these bivariate polynomials.

4: For each honest party who accepts his shares in Step 3, $\mathcal{S}$ sets his input for $\mathcal{F}_{\mathsf{ra}}$ as 1. Then $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{ra}}$. When $\mathcal{S}$ gets output 1 during the simulation of $\mathcal{F}_{\mathsf{ra}}$, it sends the polynomials $f'_1(x),\ldots,f'_L(x)$ to the functionality $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ and deliver the output from $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ to each honest party whose input for $\mathcal{F}_{\mathsf{ra}}$ is 1.

5: **Column and Row Interpolation**: $\mathcal{S}$ honestly follows the protocol to execute each honest party. For each honest party P_k who accepts his column polynomials, $\mathcal{S}$ additionally checks whether they are equal to $\{F'_i(x,y)\}_{i=1}^{L/(t+1)}$. If not, $\mathcal{S}$ aborts the simulation. For each honest party P_k who fails to verify his shares, $\mathcal{S}$ sends $(\texttt{abort}, P_k)$ to $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$.

Public Reconstruction Phase

1: For each honest party who receives his output from $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$, $\mathcal{S}$ follows the protocol to execute this party to send the $\langle PubRec \rangle$ to corrupted parties. Then for each $\langle PubRec \rangle$ received from corrupted parties, if an honest party accepts this message but the shares in this message does not lie on $f'_1(x),\ldots,f'_L(x)$, $\mathcal{S}$ aborts the simulation.

2: $\mathcal{S}$ sends the output $f'_1(x),\ldots,f'_L(x)$ to corrupted parties. For each honest party, when he succeeds in reconstructing, $\mathcal{S}$ delivers the output from $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ to him.

We prove the security of $\Pi_{\mathsf{ACSS\text{-}ab}}$ using hybrid arguments. We consider the following hybrids.

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

$\mathbf{Hyb}_1$: In this hybrid, $\mathcal{S}$ uses the first $t+1$ honest parties who accept their degree-$2t$ row polynomials to interpolate degree-$(2t,t)$ bivariate polynomial $F'_i(x,y)$ for all $i \in [L/(t+1)]$. Later for each honest party P_k who accepts his column polynomials, $\mathcal{S}$ addition checks whether his column polynomials are equal to $F'_i(k,y)$. If not, $\mathcal{S}$ aborts the simulation.

The difference between $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ is the probability of $\mathcal{S}$ aborts. Assuming that honest party P_k accepts his row polynomials $F_i(x,\alpha_k) \neq F'_i(x,\alpha_k)$ for some $i \in [L/(t+1)]$. Then we know:

$$H(f_1(\alpha_k),\ldots,f_L(\alpha_k),Y(\alpha_0,\alpha_k)) = \boldsymbol{C}[k]$$
$$H(r(\alpha_k)+\sum_{i=1}^{L} d^i f_i(\alpha_k), Y_0(\alpha_0,\alpha_k)) = \boldsymbol{\mathcal{C}}[k]$$

There are three cases $\mathcal{S}$ may abort, for case 1, $\mathcal{A}$ first samples d, computes $\boldsymbol{C},\boldsymbol{\mathcal{C}}$ accordingly and $d = H(\boldsymbol{C},\boldsymbol{\mathcal{C}})$. Assuming that $\mathcal{A}$ can query H for $\texttt{poly}(\kappa)$ times, the probability is at most:

$$\frac{\texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)}$$

Which is negligible in the security parameter κ.
For case 2, $\mathcal{A}$ first uses $F_i(x,k) \neq F'_i(x,k)$ for some P_k and $i \in [L/(t+1)]$ to compute $\boldsymbol{C},\boldsymbol{\mathcal{C}}$, but:

$$r(\alpha_k)+\sum_{i=1}^{L} d^i f_i(\alpha_k) = r(\alpha_k)+\sum_{i=1}^{L} d^i f'_i(\alpha_k)$$

This can be considered as a degree-L equation $\sum_{i=1}^{L}(f_i(\alpha_k)-f'_i(\alpha_k))\cdot x^i = 0$, where d is the root. Since not all coefficients are zero, there are at most L roots. We take the union bound for $\mathcal{O}(n)$ honest parties, and the probability is at most:

$$\frac{Ln\texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)}$$

Which is negligible in the security parameter κ. For case 3, $\mathcal{A}$ find a collision in the RO, the probability is at most:

$$\frac{(2t+1)\texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)}$$

Which is also negligible in the security parameter κ. Thus, the distributions of $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are computationally indistinguishable.

$\mathbf{Hyb}_2$: In this hybrid, in the public reconstruction phase, if an honest party accepts wrong $\langle PubRec \rangle$ from corrupted parties, $\mathcal{S}$ aborts the simulation. The probability of $\mathcal{S}$ aborting is at least as much as $\mathcal{A}$ finding a collision in the RO, which equals:

$$\frac{\mathtt{poly}(\kappa)}{2^{\kappa} - \mathtt{poly}(\kappa)}$$

which is negligible in the security parameter κ. Thus, the distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are computationally indistinguishable.

Note that $\mathbf{Hyb}_2$ corresponds to the ideal world, then $\Pi_{\mathsf{ACSS\text{-}ab}}$ securely computes $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ with error $\frac{Ln \cdot \mathtt{poly}(\kappa)}{2^{\kappa} - \mathtt{poly}(\kappa)}$.

B.2 Cost Analysis

Communication cost. The communication costs include:

- For dealer, during the Sharing phase, it requires $\mathcal{O}(Ln + n^2)$ field elements to distribute all parties' shares, $\mathcal{O}(n^2)$ field elements plus $\mathcal{O}(\kappa \cdot n^2)$ bits to broadcast the commitments.
- All parties invoke one instance of $\mathcal{F}_{\mathsf{ra}}$, which requires $\mathcal{O}(\kappa \cdot n^2)$ bits.
- For all parties, during the Share Interpolation phase, it requires $\mathcal{O}(Ln+n^2)$ field elements to exchange their shares.
- For all parties, during the Public Reconstruction phase, it requires $\mathcal{O}(Ln^2)$ field elements.

Therefore, the sharing phase and Share Interpolation phase require $\mathcal{O}(Ln + n^2)$ field elements plus $\mathcal{O}(\kappa \cdot n^2)$ bits in total. For the public reconstruction phase, it requires $\mathcal{O}(Ln^2)$ field elements.

Computation cost. We also analyze the cryptographic operation complexity of $\Pi_{\mathsf{ACSS\text{-}ab}}$. The dealer D computes a commitment vector of size $\mathcal{O}(n)$, which requires $\mathcal{O}(n)$ Hash computations. Each party verifies its shares by verifying the dZK proof, which costs $\mathcal{O}(1)$ Hash computations. Hence, this phase requires $\mathcal{O}(n)$ Hash computations.

B.3 Construction and Proof for ACSS with Identifiable Abort

We describe $\Pi_{\mathsf{ACSS\text{-}id}}$ and its security proofs in this section.

Protocol $\Pi_{\mathsf{ACSS\text{-}id}}$

Dealer D protocol

1: D encodes L share polynomials into $\frac{L}{t+1}$ bivariate polynomials $F(x, y)$. It divides them into n batches, each with $L' = \frac{L}{n(t+1)}$ polynomials.
Steps for Public Reconstruction: D constructs polynomials $g_{i,j,k}(y)$ for $i \in [1, n], j \in [1, L'], k \in [1, n]$.

$$g_{i,j,k}(y) := \sum_{\ell \in [1, t+1]} F_{i,j}(\alpha_{-\ell+1}, y) \cdot \alpha_k^{\ell-1}$$

2: **Commitments**: For each batch, D randomly samples $Y_i(x, y)$ with degree-$(2t, t)$. and $Y_k(x)$ with degree-t. Then, D computes commitments as follows.

$$C_i[x, y] = H(F_{i,1}(x, y), \ldots, F_{i,L'}(x, y), Y_i(x, y)), \text{ for } x, y \in [1, n], i \in [1, n]$$
$$\mathsf{C}_{i,k}[\ell] := H(g_{i,1,k}(\alpha_\ell), \ldots, g_{i,L',k}(\alpha_\ell), Y_{i,k}(\alpha_\ell)) \text{ for } \ell \in [n]$$

3: **dZK Proofs**.
(a) **Blinding polynomial and commitment**: For each batch, D randomly samples $F_{i,0}(x, y), Y_{i,0}(x, y)$ with degree-$(2t, t)$ and $g_{i,k,0}(x), y_{i,k,0}(x)$ with degree-t. Then, it computes commitments $C_{i,0}[x, y] := H(F_{i,0}(x, y), Y_{i,0}(x, y))$ and $\mathsf{C}_{i,k,0}[\ell] = H(g_{i,k,0}(\alpha_\ell), y_{i,k,0}(\alpha_\ell))$. Finally, it computes $d_i = H(C_i, C_{i,0})$ and $\mathsf{d}_{i,k} = H(\mathsf{C}_{i,k}, \mathsf{C}_{i,k,0})$.

(b) **dZK polynomial**: D computes $R_i(x,y)$ and $r_k(x)$ as follows.

$$R_i(x,y) := F_{i,0}(x,y) - \sum_{j\in[1,L']} d_i^j F_{i,j}(x,y) \textbf{ for } i \in [1,n]$$

$$r_{i,k}(x) := g_{i,k,0}(x) - \sum_{j\in[1,\frac{L}{t+1}]} \mathsf{d}_{i,k}^j g_{i,j,k}(x)$$

4: **Send shares and broadcast commitments**: D reliable broadcasts the following.
(a) Commitments $\langle C_1,\ldots,C_n\rangle$,$\langle \mathsf{C}_{1,1},\ldots,\mathsf{C}_{n,n}\rangle$.
(b) dZK proofs $C_{i,0}, R_i$, and $\mathsf{C}_{i,k,0}, r_{i,k}$ for $k\in[1,n], i\in[1,n]$.
Further, D sends shares $\langle Shares, \mathcal{S}_1,\ldots,\mathcal{S}_n\rangle$, where

$$\mathcal{S}_i = (\{F_{i,j}(x,\alpha_\ell), F_{i,j}(\alpha_\ell,y)\}, Y_i(x,\alpha_\ell), Y_i(\alpha_\ell,y), Y_{i,0}(x,\alpha_\ell), Y_{i,0}(\alpha_\ell,y))$$

and also nonces $\{Y_{i,k}(\alpha_\ell), y_{i,k,0}(\alpha_\ell)\}$, to all parties $P_\ell\in\mathcal{P}$.

Participant party protocol

1: **Verify shares**: Party P_ℓ receiving a $\langle Shares, \mathcal{S}'_1,\ldots,\mathcal{S}'_n\rangle$ message runs the following steps and accepts them if they succeed.
(a) **Verify commitments**: P_ℓ first computes $C'_i[x,\ell], C'_i[\ell,y], \mathsf{C}'_{i,k}[\ell]$ and checks if they match the commitments received through RBC.
(b) **Verify dZK proofs**: P_ℓ computes $d_i = H(C_i, C_{i,0})$. Then, it verifies commitments and dZK proofs.

$$H(R_i(x,\alpha_\ell) + \sum_{j\in[1,L']} d_i^j F'_{i,j}(x,\alpha_\ell), Y'_{i,0}(x,\alpha_\ell)) \stackrel{?}{=} C_{i,0}[x,\ell]$$

$$H(r_{i,k}(\alpha_\ell) + \sum_{j\in[1,L']} \mathsf{d}_{i,k}^j g'_{i,j,k}(\alpha_\ell), y'_{i,k,0}(\alpha_\ell)) \stackrel{?}{=} \mathsf{C}_{i,k,0}[\ell]$$

2: **Run Reliable Agreement**: Upon accepting its shares, P_ℓ inputs 1 to $\Pi_{\texttt{ra}}$. On terminating $\Pi_{\texttt{ra}}$ with output 1, parties conduct share interpolation.

Share Interpolation phase

3: **Send common shares on row polynomials**: P_k sends the message $\langle Shares, \{(F_{i,j}(\alpha_\ell,\alpha_k), Y_i(\alpha_\ell,\alpha_k), Y_{i,0}(\alpha_\ell,\alpha_k))\}\rangle$ to party $P_\ell\in[1,n]$.

4: **Validate shares received from parties**: A party P_ℓ receiving shares from another party P_k runs the following checks.
– **Commitments**: P_ℓ computes $C'_i[\ell,k]$ and checks if they match $C_i[\ell,k]$.
– **dZK proofs**: P_ℓ computes d_i and verifies dZK proof.

$$H(R_i(\alpha_\ell,\alpha_k) + \sum_{j\in[1,L']} d_i^j F_{i,j}(\alpha_\ell,\alpha_k), Y_{i,0}(\alpha_\ell,\alpha_k)) \stackrel{?}{=} C_{i,0}[\ell,k] \forall i\in[1,n]$$

5: **Reconstruct and verify columns**: Upon accepting $t+1$ shares, P_ℓ reconstructs the column and nonce polynomials $F'_{i,j}(\alpha_\ell,y)$ $Y'_i(\alpha_\ell,y)$,$Y'_{i,0}(\alpha_\ell,y)$. Then, P_ℓ verifies commitments and dZK proofs. If in a batch i, $C'_i[\ell,j]\neq C_i[\ell,j]$ or $H(R_i(\alpha_\ell,\alpha_j) + \sum_{k\in[1,L']} d_i^k F_i(\alpha_\ell,\alpha_j), Y_{i,0}(\alpha_\ell,\alpha_j)) \neq C_{i,0}[\ell,j]$, then P_ℓ creates an $\langle\texttt{abort}\rangle$ message using the $t+1$ accepted shares. This $\langle\texttt{abort}\rangle$ message is also an ACSS proof.

6: **Reconstruct rows**: P_ℓ forwards $\langle Shares, \{(F_{i,j}(\alpha_k,\alpha_\ell), Y_{i,j}(\alpha_k,\alpha_\ell))\}\rangle$ to $P_k\in\mathcal{P}$. P_ℓ interpolates its rows on receiving $2t+1$ valid points.
Abort verification: If P_ℓ has a proof that D is malicious, it forwards it to other parties over private channels. If P_ℓ receives an $\langle Abort\rangle$ message from P_k, it executes these steps.
– It verifies commitments and dZK proofs of the $t+1$ points on P_k's column.
– It interpolates the whole column and verifies commitments and dZK proofs for the entire column. If these checks fail, P_ℓ accepts the proof.

7: **Termination**: P_ℓ outputs its shares and terminates. If instead P_ℓ has an ACSS proof of D cheating, it forwards $\langle Abort\rangle$ to other parties and terminates with this proof.

Public Reconstruction phase

8: **Reconstruction**: Party P_ℓ computes $g'_{i,j,k}(\alpha_\ell) := \sum_{m=1}^{t+1} F'_{i,j}(\alpha_{-m+1},\alpha_\ell)\cdot\alpha_k^{m-1}$. Then, it forwards $\langle PubRec, \{\{g'_{i,j,k}(\alpha_\ell)\}, y'_{i,k}(\alpha_\ell), y'_{i,k,0}(\alpha_\ell)\}\rangle$ to P_k.
On receiving $\langle PubRec\rangle$ message from P_ℓ, P_k runs the following steps.
– It computes $\mathsf{C}'_{i,k}[\ell]$ and checks if they match $\mathsf{C}_{i,k}[\ell]$. Then, it verifies the dZK proof.

– On verifying shares from $t+1$ parties, P_k reconstructs $g_{i,j,k}(x)$, for all i, j. It then sends $\langle PubRecErr, g_{i,j,k}(\alpha_0)\rangle$ to other parties.
– P_k uses online error correction on points received through $PubRecErr$ messages to reconstruct the polynomial $\phi_{i,j}(x) := \sum_{m=1}^{t+1} F_{i,j}(\alpha_{-m+1}, \alpha_0) \cdot x^{m-1}$ for all i, j.

Accusation Phase

9: **Accusation**: If a party terminates the Share Interpolation phase with an ACSS proof and needs to accuse the dealer, it reliably broadcasts his ACSS proof.

Security proof. We construct a simulator $\mathcal{S}$, which interacts with the environment $\mathcal{Z}$ and with the ideal functionalities. $\mathcal{S}$ constructs virtual real-world honest parties and runs the real-world adversary $\mathcal{A}$. $\mathcal{S}$ communicates with $\mathcal{A}$ on behalf of the environment $\mathcal{Z}$ by forwarding every message sent by $\mathcal{Z}$. $\mathcal{S}$ also forwards the messages sent by $\mathcal{A}$ to $\mathcal{Z}$.

Lemma 6. *Protocol $\Pi_{\mathsf{ACSS\text{-}id}}$ securely computes $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ in the $\{\mathcal{F}_{\mathsf{rbc}}, \mathcal{F}_{\mathsf{ra}}\}$-hybrid model and the Random Oracle model against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t = \frac{n-1}{3}$ parties.*

Proof. **Termination**: We first show that either all honest parties terminate $\Pi_{\mathsf{ACSS\text{-}id}}$ or no honest party terminates it. Then, we show that each terminating honest party either outputs a share or $\langle Abort\rangle$. $\mathcal{F}_{\mathsf{rbc}}$ and $\mathcal{F}_{\mathsf{ra}}$ ensure totality i.e. if an honest party terminates, then eventually every honest party will terminate. $\mathcal{F}_{\mathsf{ra}}$ also requires at least $t+1$ honest parties to input 1, which ensures at least $t+1$ honest parties receive their row and column polynomials from the dealer D.

In case honest parties receive output from $\mathcal{F}_{\mathsf{ra}}$, they proceed to the share interpolation phase. In this phase, each honest party receives at least $t+1$ points on its column polynomials. This is because the honest parties that input 1 to $\mathcal{F}_{\mathsf{ra}}$ will send $\langle Row\rangle$ messages to every other honest party. Each honest party reconstructs its column polynomials and cross checks them against the commitments and dZK proofs received through $\mathcal{F}_{\mathsf{rbc}}$. In case the commitments broadcast by the dealer D are malformed, the party creates an $\langle Abort\rangle$ message with a verifiable proof and forwards it to other parties. Any party that receives this proof can verify D's malicious behavior. Therefore, each honest party will either send common points on row polynomials in the form of $\langle Column\rangle$ messages, or will send an $\langle Abort\rangle$ message with a verifiable proof that D is malicious. As there are $2t+1$ honest parties, each party will either receive sufficient points on its row polynomials or will receive an $\langle Abort\rangle$ message with a proof. Therefore, every honest party will either interpolate a row polynomial and output shares or verify the malicious behavior of D and output $\langle Abort\rangle$.

In case honest parties do not receive output from $\mathcal{F}_{\mathsf{ra}}$, no party terminates the protocol.

Next, we prove the security of $\Pi_{\mathsf{ACSS\text{-}id}}$ using a simulator $\mathcal{S}$ and hybrid arguments.

Simulator $\mathcal{S}$

When D is honest

Sharing Phase

1: The simulator $\mathcal{S}$ receives L shares of corrupted parties $\mathcal{P}_{\mathcal{C}orr}$ from the functionality $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. We denote these shares as $s_{i,j}$ for $i \in [1, L]$ and $j \in \mathcal{P}_{\mathcal{C}orr}$. In the following, when $\mathcal{S}$ randomly samples values as the output of H, if these values have been previously queried by $\mathcal{A}$, $\mathcal{S}$ aborts the simulation.

2: **Encoding shares**: $\mathcal{S}$ executes the following steps.
– **Row polynomials**. $\mathcal{S}$ randomly samples $\frac{L}{t+1}$ degree-$2t$ row polynomials $F'_{i,j}(x, \alpha_\ell)$, such that $F'_{i,j}(\alpha_{-k+1}, \alpha_\ell) = f_{(i-1)\cdot n+(j-1)\cdot(t+1)+k}(\alpha_\ell)$ for $j \in [1, L'], i \in [1, n], k \in [1, t+1]$.
– **Column polynomials**. $\mathcal{S}$ also randomly samples degree-t column polynomials $F'_{i,j}(\alpha_k, y)$: $F'_{i,j}(\alpha_k, \alpha_\ell) = F'_i(\alpha_\ell, \alpha_k)$ for $P_\ell, P_k \in \mathcal{P}_{\mathcal{C}orr}$.
– **Nonce polynomials**. $\mathcal{S}$ samples degree-$2t$ row polynomials $Y'_i(x, \alpha_\ell), Y'_{i,0}(x, \alpha_\ell)$ for $P_\ell \in \mathcal{P}_{\mathcal{C}orr}$. Further, it samples degree-t polynomials $Y'_i(\alpha_\ell, y)$: $Y'_i(\alpha_\ell, \alpha_m) = Y'_i(\alpha_m, \alpha_\ell)$, $Y'_{i,0}(\alpha_\ell, y)$: $Y'_{i,0}(\alpha_\ell, \alpha_m) = Y'_{i,0}(\alpha_m, \alpha_\ell)$, for $P_\ell, P_m \in \mathcal{P}_{\mathcal{C}orr}$.
– **Public Reconstruction**: For each $F'_{i,j}(x, \alpha_\ell)$, $\mathcal{S}$ computes n points $g_{i,j,k}(\alpha_\ell) := \sum_{m\in[1,t+1]} F'_{i,j}(\alpha_{-m+1}, \alpha_\ell) \cdot \alpha_k^{m-1}$ for $k \in [1, n]$. $\mathcal{S}$ also randomly samples nonces $y_{i,k}(\alpha_\ell), y_{i,k,0}(\alpha_\ell)$ for all i, k.

3: **Commitments**: For each batch $i \in [1, n]$, $\mathcal{S}$ executes the following steps.
– $\mathcal{S}$ randomly samples n^2 values $C'_i[x, y]$ for $x, y \in [1, n]$.
– $\mathcal{S}$ maps the inputs $(F'_{i,1}(\alpha_j, \alpha_k), \ldots, F'_{i,L'}(\alpha_j, \alpha_k), Y'_i(\alpha_j, \alpha_k))$, $(F'_{i,1}(\alpha_k, \alpha_j), \ldots, F'_{i,L'}(\alpha_k, \alpha_j), Y'_i(\alpha_k, \alpha_j))$ to outputs $C'_i[j, k]$ and $C'_i[k, j]$ for $j \in [1, n], P_k \in \mathcal{P}_{\mathcal{C}orr}$. It aborts the simulation if these inputs have already been queried.

– **PubRec commitments**: $\mathcal{S}$ randomly samples $\mathsf{C}'_{i,k}[\ell]$ from $\{0,1\}^\kappa$ for $k \in [1,n], P_\ell \in \mathcal{P}_{Corr}$. Then, $\mathcal{S}$ maps inputs $(g'_{i,1,k}(\alpha_\ell), \ldots, g'_{i,L',k}(\alpha_\ell), y'_{i,k}(\alpha_\ell))$ to output $\mathsf{C}'_{i,k}[\ell]$. It aborts the simulation if these inputs have already been queried.

If the inputs have been previously queried by $\mathcal{A}$, $\mathcal{S}$ aborts the simulation.

4: **dZK proofs**: For each batch $i \in [1,n]$, $\mathcal{S}$ executes the following steps.
– $\mathcal{S}$ next samples a matrix of n^2 random points from $\{0,1\}^\kappa$ $C'_{i,0}[x,y]$ for $x,y \in [1,n]$.
– $\mathcal{S}$ samples a random value $d_{S,i}$ from $\{0,1\}^\kappa$ and maps input $(C'_i, C'_{i,0})$ to value $d_{S,i}$. It aborts the simulation if the inputs are already mapped.
– $\mathcal{S}$ samples a random degree-$(2t,t)$ bivariate polynomial $R'_i(x,y)$. It also computes $F'_{i,0}(x,\alpha_k) := R'_i(x,\alpha_k) + \sum_{j\in[1,L']} d_S^j F'_{i,j}(x,\alpha_k)$ and $F'_{i,0}(\alpha_k,y) := R'_i(\alpha_k,y) + \sum_{j\in[1,L']} d_S^j F'_{i,j}(\alpha_k,y)$ for $P_k \in \mathcal{P}_{Corr}$.
– $\mathcal{S}$ samples t degree-$2t$ polynomials $Y_{i,0}(x,\alpha_k)$ and $Y_{i,0}(\alpha_k,y)$ for $P_k \in \mathcal{P}_{Corr}$. Then, it maps inputs $(F'_{i,0}(\alpha_j,\alpha_k), Y'_{i,0}(\alpha_j,\alpha_k))$,$(F'_{i,0}(\alpha_k,\alpha_j), Y'_{i,0}(\alpha_k,\alpha_j))$ to outputs $C'_{i,0}[j,k], C'_{i,0}[k,j]$ for $j \in [1,n], P_k \in \mathcal{P}_{Corr}$. It aborts the simulation if the inputs have already been mapped.
– **PubRec Simulation**:
• $\mathcal{S}$ randomly samples $\mathsf{C}'_{i,k,0}[\ell]$ from $\{0,1\}^\kappa$ for $k \in [1,n]$ and $P_\ell \in \mathcal{P}$, and also randomly samples $\mathsf{d}_{S,i,k}$ from $\{0,1\}^\kappa$. $\mathcal{S}$ maps inputs $(\mathsf{C}'_{i,k}, \mathsf{C}'_{i,k,0})$ to output $\mathsf{d}_{S,i,k}$.
• $\mathcal{S}$ also randomly samples degree-t polynomials $r'_{i,k}(x)$ for $k \in [1,n]$. Then, it sets $g'_{i,k,0}(\alpha_\ell) := r'_{i,k}(\alpha_\ell) + \sum_{j\in[1,L']} \mathsf{d}^j_{S,i,k} g'_{i,j,k}(\alpha_\ell)$ for $P_\ell \in \mathcal{P}_{Corr}$.
• Finally, $\mathcal{S}$ randomly samples $y'_{i,k,0}(\alpha_\ell)$ and maps input $(g'_{i,k,0}(\alpha_\ell), y'_{i,k,0}(\alpha_\ell))$ to output $\mathsf{C}'_{i,k,0}[\ell]$ for $P_\ell \in \mathcal{P}_{Corr}$.

$\mathcal{S}$ also intercepts $\mathcal{A}$'s calls to the RO H and replies with the mapped output if the input is already mapped. Otherwise, $\mathcal{S}$ picks a random value and maps the input to this value.

5: **Send shares and broadcast commitments**: $\mathcal{S}$ sends row and column polynomials $F'_{i,j}(x,\alpha_\ell), F'_{i,j}(k,\alpha_\ell)$ and the nonce values $Y'_i(x,\alpha_\ell), Y'_i(\alpha_\ell,y), Y'_{i,0}(x,\alpha_\ell), Y'_{i,0}(\alpha_\ell,y), y_{i,k}(\alpha_\ell), y_{i,k,0}(\alpha_\ell)$ to parties $k \in [1,n], j \in [1,L'], i \in [1,n], P_\ell \in \mathcal{P}_{Corr}$. Further, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{rbc}}$ to deliver $C'_i, C'_{i,0}, \mathsf{C}'_{i,k}, R'_i(x,y), r'_{i,k}(x)$ for $i \in [1,n]$ to parties in $\mathcal{P}_{Corr}$.

6: $\mathcal{S}$ also honestly simulates $\mathcal{F}_{\mathsf{ra}}$. For each honest party, $\mathcal{S}$ sets its input to $\mathcal{F}_{\mathsf{ra}}$ as 1 when this party receives a $\langle Shares \rangle$ message from dealer D.

7: **Column Interpolation**: $\mathcal{S}$ sends common shares on row polynomials $(F'_{i,j}(\alpha_k,\alpha_\ell), Y'_i(\alpha_k,\alpha_\ell), Y'_{i,0}(\alpha_k,\alpha_\ell))$ to parties P_k in $\mathcal{P}_{Corr}$, on behalf of honest parties P_ℓ. Further, $\mathcal{S}$ also notifies $\mathcal{A}$ of scheduled messages between the simulated honest parties. $\mathcal{S}$ honestly runs the protocol to verify a $\langle Rows \rangle$ message received from a corrupted party, delivered to a simulated honest party that did not input 1 to $\mathcal{F}_{\mathsf{ra}}$. If the verification succeeds and this message does not match the shares generated by $\mathcal{S}$, it aborts the simulation. Otherwise, $\mathcal{S}$ waits until the party receives $p_H + p_{\mathcal{C}orr,V} = t+1$ messages and then proceeds to the row interpolation phase.

8: **Row Interpolation Phase**: Then, $\mathcal{S}$ also sends common shares on column polynomials $\langle Col, F'_{i,j}(\alpha_\ell,\alpha_k), Y'_i(\alpha_\ell,\alpha_k), Y'_{i,0}(\alpha_\ell,\alpha_k)\rangle$ to parties in $P_j \in \mathcal{P}_{Corr}$, on behalf of honest parties P_ℓ. On receiving a $\langle Columns \rangle$ message from a corrupted party, $\mathcal{S}$ follows the protocol honestly to verify it. If the verification succeeds and this message does not match the shares generated by $\mathcal{S}$, it aborts the simulation. $\mathcal{S}$ waits until each simulated honest party receives $p_H + p_{\mathcal{C}orr,V} = 2t+1$ correct points on its row polynomials.

Public Reconstruction

1: $\mathcal{S}$ receives the dealer's secrets $f_1(\alpha_0), \ldots, f_L(\alpha_0)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. Then, it executes the following steps.
– It first samples random degree-$(2t,t)$ bivariate polynomials $F_{i,j}(x,y)$ based on corrupted parties' row and column polynomials and the dealer's secrets.
– It computes $g_{i,j,k}(y) := \sum_{m\in[1,t+1]} F_{i,j}(\alpha_{-m+1}, y) \cdot \alpha_k^m$ and samples random nonce polynomials $y_{i,k}(y), y_{i,k,0}(y)$ such that $y_{i,k}(\alpha_m) = y'_{i,k}(\alpha_m)$ for $P_m \in \mathcal{P}_{Corr}$. It then computes $g_{i,k,0}(y) := r_{i,k}(y) + \sum_{j\in[1,L']} \mathsf{d}^j_{i,k} g_{i,j,k}(y)$.
– $\mathcal{S}$ programs the RO to output $\mathsf{C}_{i,k}[\ell], \mathsf{C}_{i,k,0}[\ell]$ for inputs $(g_{i,1,k}(\alpha_\ell), \ldots, g_{i,L',k}(\alpha_\ell), y_{i,k}(\alpha_\ell)$,$(g_{i,k,0}(\alpha_\ell), y_{i,k,0}(\alpha_\ell)$ for $k \in [1,n], i \in [1,n]$. It aborts the simulation if these inputs have already been queried.

2: On completing these steps, $\mathcal{S}$ sends messages $\langle PubRec, (g_{i,1,k}(\alpha_\ell), \ldots, g_{i,L',k}(\alpha_\ell), y_{i,k}(\alpha_\ell), y_{i,k,0}(\alpha_\ell))\rangle$ to party $P_k \in \mathcal{P}_{Corr}$.

3: For each honest party P_j and for each $\langle PubRec \rangle$ message received from a corrupted party, $\mathcal{S}$ follows the protocol to check whether it is correct and aborts the simulation if the verification succeeds with shares different than what it distributed in the sharing phase. If an honest party receives it, $\mathcal{S}$ considers the message to be correct. When P_j receives $t+1$ correct $\langle PubRec \rangle$ messages, $\mathcal{S}$ delivers the output from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to P_j. Then, it sends $\langle PubRecErr, g_{i,j,\ell}(\alpha_0)\rangle$ message from P_ℓ to corrupted parties.

4: $\mathcal{S}$ uses Online Error Correction (OEC) to reconstruct the polynomial $\phi_{i,j}(x) := \sum_{m\in[1,t+1]} F_{i,j}(\alpha_{-m+1}, \alpha_0) \cdot x^{m-1}$, and secrets $F_{i,j}(\alpha_{-m+1}, \alpha_0)$. $\mathcal{S}$ waits until each simulated honest party receives enough messages such that OEC terminates. The, $\mathcal{S}$ sends `Public-Recon` to the functionality $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ and requests it to deliver the secrets $f_1(\alpha_0), \ldots, f_L(\alpha_0)$ to all honest parties.

Accusation

1: If $\mathcal{S}$ receives an $\langle Abort \rangle$ message from a corrupted party, it honestly follows the protocol to verify the message. If the verification succeeds, $\mathcal{S}$ aborts the simulation.

We prove that $\Pi_{\mathsf{ACSS\text{-}id}}$ implements $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ through a series of hybrids.

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

$\mathbf{Hyb}_1$: In this hybrid, $\mathcal{S}$ receives the corrupted parties' shares from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ and randomly generates corrupted parties' row and column polynomials $F'_{i,j}(x, \alpha_\ell)$, $F'_{i,j}(\alpha_\ell, y)$ for $P_\ell \in \mathcal{P}_{Corr}$. It also generates shares for reconstruction polynomials $g_{i,j,k}(\alpha_\ell)$. Then, $\mathcal{S}$ generates the whole bivariate polynomials given corrupted parties' polynomials and secrets. $\mathbf{Hyb}_1$ is identical to $\mathbf{Hyb}_0$ because the share polynomials of corrupted parties have been uniformly randomly sampled from the same distribution in both hybrids.

$\mathbf{Hyb}_2$: In this hybrid, we delay the generation of honest parties' shares until the share interpolation phase. For each honest party, $\mathcal{S}$ generates commitment matrices by randomly sampling $C'_i[\ell, m]$ and $\mathsf{C}'_{i,k}[\ell]$ for $k, \ell, m \in [1, n]$. Further, $\mathcal{S}$ also randomly samples row and column nonce polynomials $Y'_i(x, \alpha_\ell), Y'_i(\alpha_\ell, y)$, $y'_{i,k}(\alpha_\ell)$ for $k \in [1, n], P_\ell \in \mathcal{P}_{Corr}$. $\mathcal{S}$ aborts the simulation if any of sampled commitments have been mapped to other inputs or if the inputs have already been queried. $\mathcal{S}$ intercepts queries to the Random Oracle and returns output $C'_i[\ell, m]$ for input $(F'_{i,1}(\alpha_\ell, \alpha_m), \ldots, F'_{i,L'}(\alpha_\ell, \alpha_m), Y'_i(\alpha_\ell, \alpha_m))$ for corrupted parties' shares. $\mathcal{S}$ also maps the input $(g'_{i,1,k}(\alpha_\ell), \ldots, g'_{i,L',k}(\alpha_\ell), y'_{i,k}(\alpha_\ell))$ to output $\mathsf{C}'_{i,k}[\ell]$. Assuming $\mathcal{A}$ can query the RO for $\mathtt{poly}(\kappa)$ times, the probability that $\mathcal{S}$ will abort the simulation is $\frac{n^3\mathtt{poly}(\kappa)}{2^\kappa - \mathtt{poly}(\kappa)}$. This probability is the result of applying the union bound on the n^3 inputs to the RO. Therefore, $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are computationally indistinguishable.

$\mathbf{Hyb}_3$: In this hybrid, $\mathcal{S}$ changes the process of generation of R_i,$r_{i,k}$ polynomials. $\mathcal{S}$ first randomly samples the commitment matrix $C'_{i,0}[\ell, m]$ and the dZK polynomials $R'_i(x, y), r'_{i,k}(x)$. Then, it randomly samples $d_{S,i}, \mathsf{d}_{S,i,k}$. It computes $F'_{i,0}(x, \alpha_\ell), F_{i,0}(\alpha_\ell, y), g'_{i,k,0}(\alpha_\ell)$ from $R'_i, r'_{i,k}$. It also randomly samples $Y'_{i,0}(x, \alpha_\ell)$, $Y'_{i,0}(\alpha_\ell, y)$, $y'_{i,k,0}(\alpha_\ell)$. Then, it maps inputs $(F'_{i,0}(\alpha_\ell, \alpha_m), Y'_{i,0}(\alpha_\ell, \alpha_m))$, $(g'_{i,k,0}(\alpha_\ell), y'_{i,k,0}(\alpha_\ell))$, $(C'_i, C'_{i,0})$, $(\mathsf{C}'_{i,k}, \mathsf{C}'_{i,k,0})$ to outputs $C'_{i,0}[\ell, m]$, $\mathsf{C}'_{i,k,0}[\ell]$,$d_{S,i}$, $\mathsf{d}_{S,i,k}$, respectively. $\mathcal{S}$ aborts the simulation if these inputs are already mapped. Assuming $\mathcal{A}$ can query the RO for $\mathtt{poly}(\kappa)$ times, the probability that $\mathcal{S}$ will abort the simulation is $\frac{n^3\mathtt{poly}(\kappa)}{2^\kappa - \mathtt{poly}(\kappa)}$. Further, we only change the generation order of $F_{i,0}(x, y)$ and $R_i(x, y)$, which makes no difference. Therefore, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ only differ by the probability that $\mathcal{S}$ aborts, which is negligible in κ.

$\mathbf{Hyb}_4$: In this hybrid, $\mathcal{S}$ simulates the $\mathcal{F}_{\mathsf{ra}}$ functionality and enables the corrupted parties to terminate the protocol. $\mathbf{Hyb}_4$ is identically distributed as $\mathbf{Hyb}_3$ because Π_{ra} securely implements the $\mathcal{F}_{\mathsf{ra}}$ functionality.

$\mathbf{Hyb}_5$: In the next few sets of hybrids, we simulate the share interpolation phase.

$\mathbf{Hyb}_{5.1}$: In this hybrid, during the column interpolation phase, $\mathcal{S}$ uses corrupted party P_j's column or row polynomials $\langle Row \rangle$ it previously generated to compute these common points and send them to P_j. $\mathbf{Hyb}_{5.1}$ is identically distributed as $\mathbf{Hyb}_4$. This is because the points sampled and sent by $\mathcal{S}$ are uniformly randomly distributed over the shares of corrupted parties, which is equivalent to $\mathcal{A}$'s view in the real-world.

$\mathbf{Hyb}_{5.2}$: In this hybrid, $\mathcal{S}$ verifies each $\langle Rows \rangle$ message received from a corrupted party by honestly following the protocol. If a point $F''_{i,j}(\alpha_\ell, \alpha_m) \neq F'_{i,j}(\alpha_\ell, \alpha_m)$ passes the verification, then $\mathcal{S}$ aborts the simulation. Otherwise, it counts this message as a valid message. It waits for $t + 1$ valid messages on an honest party's column polynomial. Further, $\mathcal{S}$ also verifies an $\langle Abort \rangle$ message by following the protocol honestly. If an $\langle Abort \rangle$ message passes this verification, $\mathcal{S}$ aborts the simulation.

We show the distributions of $\mathbf{Hyb}_{5.2}$ are computationally indistinguishable from $\mathbf{Hyb}_{5.1}$ by proving that $\mathcal{S}$ aborts the simulation with negligible probability. Assuming that honest party P_ℓ accepts a set of

points $F''_{i,j}(\alpha_\ell, \alpha_m)$ for some $i \in [1, n]$. Then we know:

$$H(F''_{i,1}(\alpha_\ell, \alpha_m), \ldots, F''_{i,L'}(\alpha_\ell, \alpha_m), Y''_i(\alpha_\ell, \alpha_m)) = C_i[\ell, m]$$

$$H(R_i(\alpha_\ell, \alpha_m) + \sum_{j=1}^{\frac{L}{n(t+1)}} d^j_{S,i} F''(\alpha_\ell, \alpha_m), Y''_{i,0}(\alpha_\ell, \alpha_m)) = C_{i,0}[\ell, m]$$

As honest parties already agreed on the commitments, $\mathcal{A}$ can break either condition in only one way: It must find an alternate set of shares that evaluate the same set of commitments. However, as RO is collision-resistant for a polynomial-time $\mathcal{A}$, $\mathcal{S}$ aborts with probability $\frac{\mathtt{poly}(\kappa)}{2^\kappa - \mathtt{poly}(\kappa)}$, which is negligible in κ. Note that any corrupted party that wants to generate a proof for its Abort message must also find a collision in the RO. Hence, the distributions of $\mathbf{Hyb}_{5.2}$ and $\mathbf{Hyb}_{5.1}$ are computationally indistinguishable.

$\mathbf{Hyb}_{5.3}$: In this hybrid, P_k needs to send $\langle Column \rangle$ messages to a corrupted party P_ℓ. On receiving $t+1$ valid messages on a P_k's column polynomial, $\mathcal{S}$ sends $F'_{i,j}(\alpha_k, \alpha_\ell)$ to P_ℓ as a $\langle Column \rangle$ message. $\mathbf{Hyb}_{5.3}$ is identically distributed as $\mathbf{Hyb}_{5.2}$.

$\mathbf{Hyb}_{5.4}$: In this hybrid, $\mathcal{S}$ verifies each $\langle Column \rangle$ message received from corrupted parties by honestly running the protocol. If the received points pass the verification but do not match the shares sampled by $\mathcal{S}$, $\mathcal{S}$ aborts the simulation. Otherwise, it counts the message as valid. $\mathcal{S}$ waits until each honest party receives valid points from $2t+1$ parties. $\mathbf{Hyb}_{5.4}$ differs from $\mathbf{Hyb}_{5.3}$ only when $\mathcal{S}$ aborts the simulation. This happens only when a corrupted party finds a collision in the RO, which happens only with negligible probability. Therefore, the distributions of $\mathbf{Hyb}_{5.4}$ are computationally indistinguishable from $\mathbf{Hyb}_{5.3}$.

$\mathbf{Hyb}_6$: In this hybrid, $\mathcal{S}$ no longer uses the honest dealer's secrets to compute honest parties' row and column polynomials because they are never used. Therefore, $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_{5.5}$ are identically distributed.

$\mathbf{Hyb}_7$: In this hybrid, $\mathcal{S}$ first receives honest dealer's secrets from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. Then, $\mathcal{S}$ computes each honest party's P_ℓ's $\langle PubRec \rangle$ message by randomly sampling $y_{i,k}(\alpha_\ell)$ and incorporating corrupted parties' shares. $\mathcal{S}$ sends points $g_{i,j,k}(\alpha_\ell)$ to corrupted party P_k, along with the sampled nonces. Further, $\mathcal{S}$ intercepts calls to the RO and returns outputs $\mathsf{C}_{i,k}[\ell]$ for inputs
$(g_{i,1,k}(\alpha_\ell), \ldots, g_{i,L',k}(\alpha_\ell), y_{i,k}(\alpha_\ell))$. If $\mathcal{A}$ already queried these inputs, then $\mathcal{S}$ aborts the simulation. On receiving a $\langle PubRec \rangle$ message from a corrupted party, $\mathcal{S}$ follows the protocol honestly to validate the shares. If the verification passes and the shares do not match the ones generated by $\mathcal{S}$, $\mathcal{S}$ aborts the simulation. Further, on receiving $t+1$ valid shares, $\mathcal{S}$ interpolates $g_{i,j,\ell}(\alpha_0)$ on behalf of honest party P_ℓ.

The main difference between $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ is the probability of $\mathcal{S}$ aborting the simulation. This happens only when a corrupted party finds second pre-image to the commitment $\mathsf{C}_{i,k}[\ell]$ or $\mathsf{C}_{i,k,0}[\ell]$, which happens only with probability at most $\frac{n^3 \mathtt{poly}(\kappa)}{2^\kappa - \mathtt{poly}(\kappa)}$, which is negligible. Therefore, the distributions of $\mathbf{Hyb}_7$ are computationally indistinguishable from $\mathbf{Hyb}_6$.

$\mathbf{Hyb}_8$: In this hybrid, $\mathcal{S}$ sends values $g_{i,j,k}(\alpha_0)$ from party P_k to corrupted parties using $\langle PubRecErr \rangle$ messages. Then, $\mathcal{S}$ runs Online Error Correction (OEC) on received shares. As OEC is information-theoretically secure, $\mathbf{Hyb}_8$ is identically distributed as $\mathbf{Hyb}_7$.

$\mathbf{Hyb}_9$: In this hybrid, during the accusation phase, if $\mathcal{S}$ receives $\langle Abort \rangle$ messages from corrupted parties, $\mathcal{S}$ follows the protocol to do verification. If one of them passes this verification, $\mathcal{S}$ aborts the simulation. Note that any corrupted party that wants to generate a proof for its Abort message must also find a collision in the RO. Thus, the distributions of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ are computationally indistinguishable.

Note that $\mathbf{Hyb}_9$ corresponds to the view of $\mathcal{S}$ in the ideal world. Therefore, $\Pi_{\mathsf{ACSS\text{-}id}}$ securely computes $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ with probability error $\frac{n^3 \mathtt{poly}(\kappa)}{2^\kappa - \mathtt{poly}(\kappa)}$.

Simulator $\mathcal{S}$

When D is corrupted

1: **Share recording**: The simulator $\mathcal{S}$ records share polynomials $F_{i,j}(x, \alpha_\ell), F_{i,j}(\alpha_\ell, y)$ and nonce polynomials $Y_i(x, \alpha_\ell), Y_i(\alpha_\ell, y)$, $Y_{i,0}(x, \alpha_\ell), Y_{i,0}(\alpha_\ell, y)$, $y_{i,k}(\alpha_\ell), y_{i,k,0}(\alpha_\ell)$ for $i \in [1, n], j \in [1, L'], k \in [1, n], P_\ell \in \mathcal{H}$ received from D.

2: **Simulate Broadcast**: $\mathcal{S}$ simulates the $\mathcal{F}_{\mathsf{rbc}}$ functionality when D invokes it to broadcast commitments and dZK proofs.

3: **Verify shares and construct polynomials**: On receiving a $\langle Shares \rangle$ message from D, $\mathcal{S}$ follows the protocol to conduct verification. When $\mathcal{S}$ receives correct $\langle Shares \rangle$ messages for $t+1$ honest parties P_k,

it interpolates degree-$(2t, t)$ bivariate polynomials $F'_{i,j}(x, y)$. Then, $\mathcal{S}$ also interpolates share polynomials $f'_1(x), \ldots, f'_L(x)$. If a $\langle Shares \rangle$ message with polynomials $F_{i,j}(x, \alpha_\ell) \neq F'_{i,j}(x, \alpha_\ell)$ passes the verification for any honest party α_ℓ, $\mathcal{S}$ aborts the simulation.

4: For each honest party who accepts its shares in Step 3, $\mathcal{S}$ sets its input to $\mathcal{F}_{\mathsf{ra}}$ as 1. Then, $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{ra}}$. When $\mathcal{S}$ gets output 1 during the simulation of $\mathcal{F}_{\mathsf{ra}}$, it sends the polynomials $f'_1(x), \ldots, f'_L(x)$ to functionality $\mathcal{S}$ and deliver the output from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to each honest party whose input for $\mathcal{F}_{\mathsf{ra}}$ was 1.

5: **Commitments and dZK proofs**: $\mathcal{S}$ also computes $F'_{i,0}(x, y) := R_i(x, y) + \sum_{j \in [1, L']} d_i^j F'_{i,j}(x, y)$. Then, it interpolates nonce polynomials $Y'_i(x, y)$,$Y'_{i,0}(x, y)$, and $y_{i,k}(x), y_{i,k,0}(x)$. Then, it computes commitments $C'_i[x, y], C'_{i,0}[x, y]$ from these generated polynomials.

6: **Column Interpolation**: $\mathcal{S}$ sends common shares on row polynomials to corrupted parties as $\langle Row \rangle$ messages. Note that $\mathcal{S}$ sends shares from only those parties that received a valid $\langle Shares \rangle$ message from D. When a corrupted party sends $\langle Row \rangle$ messages to an honest party P_ℓ, then $\mathcal{S}$ follows the protocol honestly to validate the message. If this verification succeeds for a point not lie on $F'_{i,j}(x, y)$, then $\mathcal{S}$ aborts the simulation. Then, $\mathcal{S}$ checks if $C'_i[\ell, m] \neq C_i[\ell, m]$ or $C'_{i,0}[\ell, m] \neq C'_{i,0}[\ell, m]$ for any honest party $P_\ell \in \mathcal{H}$, then $\mathcal{S}$ adds party P_ℓ to the set $\mathcal{P}_{\text{proof}}$. Further, it sends $(\texttt{proof}, P_\ell)$ to the functionality $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. $\mathcal{S}$ also requests $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to change the output of P_ℓ to $(\texttt{Corrupt}, D)$.

7: **Row interpolation**: $\mathcal{S}$ sends common shares on column polynomials to corrupted parties. Note that $\mathcal{S}$ sends shares from only those parties which successfully interpolated their columns i.e. $P_\ell \in \mathcal{H} \setminus \mathcal{P}_{\text{proof}}$. For each honest party in $P_\ell \in \mathcal{P}_{\text{proof}}$, $\mathcal{S}$ sends P_ℓ's ACSS proof to all parties. $\mathcal{S}$ waits until all honest parties get their shares or proofs. For each honest party P_ℓ who receives a proof, $\mathcal{S}$ sends $(\texttt{proof}, P_\ell)$ to the functionality $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. $\mathcal{S}$ also delivers the outputs from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to each honest party.

Public Reconstruction Phase

1: For each honest party who outputs its share, $\mathcal{S}$ follows the protocol to send $\langle PubRec \rangle$ messages to corrupted parties.

2: When an honest party P_k receives a $\langle PubRec \rangle$ message from a corrupted party P_ℓ, $\mathcal{S}$ follows the protocol to verify it. If the verification succeeds and $g_{i,j,k}(\alpha_\ell) \neq g'_{i,j,k}(\alpha_\ell)$, then $\mathcal{S}$ aborts the simulation. On receiving $t+1$ correct messages on the polynomial $g'_{i,j,k}(x)$, $\mathcal{S}$ sends $g'_{i,j,k}(\alpha_0)$ in a $\langle PubRecErr \rangle$ message to corrupted parties.

3: On receiving a $\langle PubRecErr \rangle$ message from a corrupted party, $\mathcal{S}$ follows the protocol honestly to reconstruct the polynomial $\phi_{i,j}(x)$ using Online Error Correction.

4: On reconstructing the secrets, $\mathcal{S}$ sends the output $f'_1(\alpha_0), \ldots, f'_L(\alpha_0)$ to the corrupted parties. $\mathcal{S}$ also delivers the output from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to each honest party that successfully reconstructs its secrets.

Accusation

1: $\mathcal{S}$ has got the ACSS proof for each honest party in $\mathcal{P}_{\texttt{Proof}}$. When the honest parties in $\mathcal{P}_{\texttt{Proof}}$ need to broadcast the ACSS proof, $\mathcal{S}$ reliably broadcasts it on behalf of them.

We prove the security of $\Pi_{\mathsf{ACSS\text{-}id}}$ using hybrid arguments. We consider the following hybrids.

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

$\mathbf{Hyb}_1$: In this hybrid, $\mathcal{S}$ receives and records share messages $\langle Shares \rangle$ from the dealer D. So far, it does nothing with the recorded shares. Therefore, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are identically distributed.

$\mathbf{Hyb}_2$: In this hybrid, $\mathcal{S}$ simulates the $\mathcal{F}_{\mathsf{rbc}}$ functionality and enables D to terminate its broadcast. Further, $\mathcal{S}$ also records the commitments and dZK polynomials. As Π_{rbc} securely implements $\mathcal{F}_{\mathsf{rbc}}$, $\mathbf{Hyb}_1$ is identically distributed as $\mathbf{Hyb}_0$.

$\mathbf{Hyb}_3$: In this hybrid, $\mathcal{S}$ verifies the recorded shares using commitments and dZK proofs received through $\mathcal{F}_{\mathsf{rbc}}$. It waits until recording $t+1$ shares whose commitments match the ones broadcast by D. It then forms degree-$(2t, t)$ polynomials $F'_{i,j}(x, y)$. $\mathcal{S}$ verifies $\langle Shares \rangle$ messages by honestly following the protocol. It aborts the simulation only when the verification succeeds for polynomials not lie on $F'_{i,j}(x, y)$.

$\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ only differ by the probability that $\mathcal{S}$ aborts the simulation. We show $\mathbf{Hyb}_3$ is computationally identically distributed as $\mathbf{Hyb}_2$ by showing $\mathcal{S}$ will abort only with negligible probability i.e. each honest party will accept its shares only if they lie on F' polynomials, except with negligible probability. The shares must satisfy both the following equations.

$$H(F''_{i,j}(\alpha_k, \alpha_\ell), \ldots, F''_{i,j}(\alpha_k, \alpha_\ell), Y''_i(\alpha_k, \alpha_\ell)) = C_i[k, \ell] \text{ for } k \in [1, n]$$

$$F''_{i,0}(x, \alpha_\ell) - F'_{i,0}(x, \alpha_\ell)) + \sum_{j \in [1, L']} d_i^j (F'_{i,j}(x, \alpha_\ell) - F''_{i,j}(x, \alpha_\ell)) = 0$$

$\mathcal{A}$ can make $\mathcal{S}$ abort in two main ways: (a) $\mathcal{A}$ first chooses incorrect shares and d_i^j to make the above equations true, and d_i^j happens to the random oracle output of the corresponding commitments. Assuming

$\mathcal{A}$ can invoke H at most $\texttt{poly}(\kappa)$ number of times, the probability is at most:

$$\frac{\texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)}$$

This probability is negligible in κ. (b) $\mathcal{A}$ finds two $F''_{i,0}(\alpha_m, \alpha_\ell), F'_{i,0}(\alpha_m, \alpha_\ell)$ and $F''_{i,j}(\alpha_m, \alpha_\ell), F'_{i,j}(\alpha_m, \alpha_\ell)$ such that d_i is the root of the resulting degree-L polynomial $\delta(\alpha)$ with variable α. Then, $\mathcal{A}$ must choose the coefficients of term $x^k y^\ell$ in the bivariate polynomials $F''_{i,j}(\alpha_k, \alpha_\ell)$, i.e. the coefficients of $\delta(\alpha)$, such that $\alpha = d_i$ is a root of $\delta(\alpha)$. The probability that a randomly chosen α is a root of this polynomial is $\frac{L}{2^\kappa} = \text{negl}(\kappa)$. Further, d_i is computed by invoking the Random Oracle H on the commitments of polynomials dependent on $\delta(\alpha)$. This ensures that any change in $\delta(\cdot)$ requires $\mathcal{A}$ to reinvoke H to compute d_i. As $\mathcal{A}$ is a polynomial time adversary, it can make at most $\texttt{poly}(\kappa)$ queries to H. Therefore, $\mathcal{A}$ can find a $\delta(\alpha)$ satisfying these conditions with probability at most $\frac{L'}{2^\kappa - \texttt{poly}(\kappa)} = \text{negl}(\kappa)$. Taking a union bound on all coefficients in all polynomials, the probability is at most:

$$\frac{n^3 L \texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)}$$

This probability is negligible in κ. Hence, the distributions of $\mathbf{Hyb}_3$ are computationally indistinguishable from $\mathbf{Hyb}_2$.

$\mathbf{Hyb}_4$: In this hybrid, $\mathcal{S}$ simulates the reliable agreement functionality $\mathcal{F}_{\mathsf{ra}}$ and allows D to terminate the sharing protocol. Additionally, $\mathcal{S}$ invokes $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ with the interpolated polynomials $f'_1(x), \ldots, f'_L(x)$ and requests it to deliver shares to those honest parties that received their shares and participated in the Π_{ra} protocol in $\mathbf{Hyb}_3$. $\mathbf{Hyb}_4$ is identically distributed as $\mathbf{Hyb}_3$ because Π_{ra} securely implements the $\mathcal{F}_{\mathsf{ra}}$ functionality.

In the following set of hybrids, we simulate the column and row interpolation phases.

$\mathbf{Hyb}_{5.1}$: In this hybrid, $\mathcal{S}$ sends common points on row polynomials to corrupted parties in the form of $\langle Row \rangle$ messages. $\mathcal{S}$ also computes the commitments $C'_i[x, y]$ and $C'_{i,0}[x, y]$ for $x, y \in [1, n], i \in [1, n]$. $\mathbf{Hyb}_{5.1}$ is identically distributed as $\mathbf{Hyb}_4$ because $\mathcal{S}$ sends the same shares as honest parties.

$\mathbf{Hyb}_{5.2}$: In this hybrid, $\mathcal{S}$ verifies an incoming $\langle Row \rangle$ message by following the protocol. If this verification succeeds for a point not on $F'_{i,j}$ polynomials, $\mathcal{S}$ aborts the simulation. Otherwise, it waits until $\mathcal{A}$ delivers $\langle Row \rangle$ $t+1$ valid $\langle Row \rangle$ messages to each simulated honest party and then reconstructs the corresponding polynomial and commitments. If for any honest party P_ℓ and any $y \in [1, n]$ in a batch i, $C'_i[\ell, y] \neq C_i[\ell, y]$ or $C'_{i,0}[\ell, y] \neq C_{i,0}[\ell, y]$, then $\mathcal{S}$ adds P_ℓ to the set $\mathcal{P}_{\text{proof}}$. Further, $\mathcal{S}$ sends `Broadcast-Proof` to $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ on behalf of P_ℓ.

$\mathbf{Hyb}_{5.2}$ and $\mathbf{Hyb}_{5.1}$ only differ by the probability of $\mathcal{S}$ aborting the simulation. $\mathcal{S}$ aborts with negligible probability because $\mathcal{A}$ can produce an incorrect point that passes the verification with probability at most $\frac{n^2 L \texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)}$. Therefore, each honest party P_ℓ will successfully interpolate its column polynomials $F'_{i,j}(\alpha_\ell, y)$, and will construct commitments $C'_i[\ell, y], C'_{i,0}[\ell, y]$. Then, if they do not match $C_i[\ell, y], C_{i,0}[\ell, y]$, the party uses these mismatched commitments and dZK proofs as proof that D is corrupt. Hence, $\mathbf{Hyb}_{5.2}$ is computationally identically distributed as $\mathbf{Hyb}_{5.1}$.

$\mathbf{Hyb}_{5.3}$: In this hybrid,$\mathcal{S}$ sends common points on columns to parties in $\mathcal{P}_{\mathcal{C}orr}$ as $\langle Column \rangle$ messages. It sends these messages only from honest parties who are not in $\mathcal{P}_{\text{proof}}$. Further, $\mathcal{S}$ forwards the proof of dealer's malice from each party $P_\ell \in \mathcal{P}_{\text{proof}}$ to corrupted parties. $\mathbf{Hyb}_{5.3}$ is identically distributed as $\mathbf{Hyb}_{5.2}$.

$\mathbf{Hyb}_{5.4}$: In this hybrid, $\mathcal{S}$ waits until one of the conditions is true: It waits for $\mathcal{A}$ to deliver $2t+1$ messages on the interpolated bivariate polynomials $F'_{i,j}(x, y)$ (Note that $\mathcal{A}$ controls the network and delivers messages sent by honest parties), or it waits for $\mathcal{A}$ to deliver a correct `proof` message. $\mathcal{S}$ outputs shares or $(\texttt{Corrupt}, D)$ on behalf of honest parties. In $\mathbf{Hyb}_{5.2}$, honest parties detect invalid shares with overwhelming probability. Further, every honest party either interpolates its column correctly or compiles a proof that D is corrupt. Each honest party will either interpolate its row or verify that D is corrupt. Hence, $\mathbf{Hyb}_{5.4}$ is computationally identically distributed as $\mathbf{Hyb}_{5.3}$.

$\mathbf{Hyb}_6$: In this hybrid, $\mathcal{S}$ sends the message $\langle PubRec \rangle$ with appropriate shares and nonces to corrupted parties. It sends these shares from honest parties who received shares from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. $\mathbf{Hyb}_6$ is identical to $\mathbf{Hyb}_{5.4}$.

$\mathbf{Hyb}_7$: In this hybrid, $\mathcal{S}$ verifies a $\langle PubRec \rangle$ message by following the protocol. If the verification succeeds for a share $g''_{i,j,k}(\alpha_\ell)$ not on $g'_{i,j,k}(x)$, then $\mathcal{S}$ aborts the simulation.

The only difference between $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ is the probability of $\mathcal{S}$ aborting the simulation. For shares $g''_{i,j,k}(\alpha_\ell)$ to pass the verification, similar to the argument in $\mathbf{Hyb}_3$, the probability is at most $\frac{n^3 L \mathtt{poly}(\kappa)}{2^\kappa - \mathtt{poly}(\kappa)}$. Further, $\mathcal{S}$ also aborts the simulation if $\mathcal{A}$ already queried inputs that were going to be mapped by $\mathcal{S}$. Hence, the distributions of $\mathbf{Hyb}_7$ are computationally indistinguishable from $\mathbf{Hyb}_6$.

$\mathbf{Hyb}_8$: In this hybrid, $\mathcal{S}$ sends $\langle PubRecErr \rangle$ messages from each honest party P_k that successfully interpolates $g_{i,j,k}(\alpha_0)$ polynomial. Further, $\mathcal{S}$ runs the protocol honestly to interpolate $\phi_{i,j}(x)$. When an honest party receives enough messages to reconstruct $\phi_{i,j}(x)$, $\mathcal{S}$ requests $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to deliver the secrets $f'_1(\alpha_0), \ldots, f'_L(\alpha_0)$ to the honest parties in the ideal world. $\mathbf{Hyb}_8$ is identically distributed as $\mathbf{Hyb}_7$ because $\mathcal{S}$ follows the protocol.

$\mathbf{Hyb}_8$ is identically distributed as $\mathcal{S}$'s view in the ideal world. Therefore, $\Pi_{\mathsf{ACSS\text{-}id}}$ securely realizes $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ with error $\frac{n^3 L \cdot \mathtt{poly}(\kappa)}{2^\kappa - \mathtt{poly}(\kappa)}$.

B.4 Cost Analysis

Communication Cost: The communication costs include:

- For dealer, during the Sharing phase, it requires $\mathcal{O}(Ln + n^3)$ field elements to distribute shares to all parties. For each batch, it requires $\mathcal{O}(n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits to broadcast the commitment, which results in $\mathcal{O}(n^4)$ field elements plus $\mathcal{O}(\kappa \cdot n^4)$ bits in total.
- For all parties, during the Share Interpolation phase, it requires $\mathcal{O}(Ln+n^3)$ field elements to exchange their shares.
- For all parties, during the Public Reconstruction phase, it requires $\mathcal{O}(Ln + n^3)$ field elements.

Therefore, the Sharing and Share Interpolation phase require $\mathcal{O}(Ln+n^4)$ field elements plus $\mathcal{O}(\kappa \cdot n^4)$ bits. The public reconstruction requires $\mathcal{O}(Ln + n^3)$ field elements. When each party executes the Accusation phase, it requires $\mathcal{O}(L + n^2)$ field elements plus $\mathcal{O}(\kappa \cdot n^2)$ bits.

Computation cost: We evaluate the Hash computation complexity of $\Pi_{\mathsf{ACSS\text{-}id}}$. Generating commitments for each batch requires $\mathcal{O}(n^2)$ Hash computations to generate a commitment matrix of size $\mathcal{O}(n^2)$. Therefore, for $\mathcal{O}(n)$ batches, the dealer D has a computational complexity of $\mathcal{O}(n^3)$ Hash computations. Each party must verify the commitments of its shares, which requires $\mathcal{O}(n)$ Hash computations per batch. For $\mathcal{O}(n)$ batches, each party requires $\mathcal{O}(n^2)$ Hash computations. In the public reconstruction phase, each party performs $\mathcal{O}(n)$ Hash computations for verifying commitments.

C Construction and Proofs of Malicious Security with Fairness AMPC

C.1 Construction of $\Pi_{\mathtt{randSh\text{-}Weak}}$ and Security Proof

We design $\Pi_{\mathtt{randSh\text{-}Weak}}$ which realizes $\mathcal{F}_{\mathtt{randSh\text{-}Weak}}$ below. At a high level, to prepare N random degree-t Shamir sharings, we first ask each party to act as a dealer and invoke $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ to distribute $N/(t+1)$ random degree-t Shamir sharings. Then all parties agree on a set of $2t+1$ successful dealers, meaning all parties receive outputs from $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ invoked by these dealers. Finally, we follow the technique in [DN07] to extract N random sharings.

Functionality $\mathcal{F}_{\mathtt{randSh\text{-}Weak}}$

Public Input: N

$\mathcal{F}_{\mathtt{randSh\text{-}Weak}}$ proceeds as follows, running with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$.

1: For all $\ell \in [N]$, the trusted party randomly samples r_ℓ.
2: For all $\ell \in [N]$, the trusted party receives a set of shares of corrupted parties from $\mathcal{S}$ and samples a random degree-t Shamir sharing $[r_\ell]_t$ based on the shares of corrupted parties and the secret r_ℓ.
3: For all $\ell \in [N]$ and each party P_j, send a request-based delayed output of the j-th share of $[r_\ell]_t$ to P_j.
   - Upon receiving a request $(\mathtt{Fail}, P_j)$ from $\mathcal{S}$, if the output of P_j has not been delivered, change the output of P_j by $\mathtt{Fail}$.
4: All honest parties output the results received from the trusted party. Corrupted parties may output anything they want.

The construction of $\Pi_{\texttt{randSh-Weak}}$ is as follows. During the protocol, each party takes `Fail` or his shares of random degree-t Shamir sharings as output, determined by whether he can receive his shares distributed by each successful dealer in the agreement set $\mathcal{D}$.

Protocol $\Pi_{\texttt{randSh-Weak}}$

Let N be the number of random degree-t Shamir secret sharings to be prepared.

1: Each party P_i samples $N' = N/(t+1)$ random degree-t Shamir secret sharings $[s_1^{(i)}]_t, \ldots, [s_{N'}^{(i)}]_t$. Then P_i acts as the dealer D, invoking $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ to distribute the shares to all parties.

2: Each party P_i sets the property Q as P_i terminating $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ when P_j acts as a dealer. Then all parties invoke $\mathcal{F}_{\mathsf{acs}}$ with property Q to agree on a set $\mathcal{D}$ of successful dealers with size $|\mathcal{D}| = 2t+1$.

3: Each party waits to receive his output from $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ for each dealer $P_i \in \mathcal{D}$, if his output is `abort`, he outputs `Fail` and terminates; otherwise, he proceeds.

4: All parties agree on (the inverse of) a Vandermonde matrix $\boldsymbol{M}$ of size $(t+1) \times (2t+1)$. For all $\ell \in [N']$, each party computes his shares by following

$$([r_{\ell,1}]_t, \ldots, [r_{\ell,t+1}]_t) = \boldsymbol{M} \cdot ([s_\ell^{(i)}]_t)_{i \in \mathcal{D}}.$$

Finally, he outputs his shares of $\{[r_{\ell,i}]_t\}_{\ell \in [N'], i \in [t+1]}$ and terminates.

Cost Analysis. The communication costs of $\Pi_{\texttt{randSh-Weak}}$ equals to n instances of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ and an instances of $\mathcal{F}_{\mathsf{acs}}$. When we instantiate $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ by $\Pi_{\mathsf{ACSS\text{-}ab}}$ and $\mathcal{F}_{\mathsf{acs}}$ by construction in [DDL+24], the communication costs is $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits for generating N random degree-t Shamir sharings.

Lemma 7. *Protocol $\Pi_{\texttt{randSh-Weak}}$ securely computes $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$ in the $\{\mathcal{F}_{\mathsf{acs}}, \mathcal{F}_{\mathsf{ACSS\text{-}Abort}}\}$-hybrid model against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t = (n-1)/3$ parties.*

Proof. **Termination.** We first show that all honest parties will eventually terminate the protocol $\Pi_{\texttt{randSh-Weak}}$. By the definition of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$, the property Q is an ACS property. Thus in Step 2 of $\Pi_{\texttt{randSh-Weak}}$, all honest parties will eventually agree on a set $\mathcal{D}$ of successful dealers. By the definition of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ again, for each dealer in $\mathcal{D}$, all honest parties will eventually receive their shares or `abort` from $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ invoked by this dealer. Each party who receives `abort` will output `Fail` and terminate at Step 3. Since Step 4 only involves local computation, each party who receives his shares will also terminate at the end of Step 4.

Security. Now we show that the protocol $\Pi_{\texttt{randSh-Weak}}$ securely computes $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$. We start with the construction of the ideal adversary $\mathcal{S}$ as follows.

Simulator $\mathcal{S}$

1: In Step 1, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ as follows:
- For each honest party P_i, $\mathcal{S}$ generates random values as the shares of corrupted parties. For each corrupted party P_i, $\mathcal{S}$ waits to receive the sharings distributed by P_i.
- $\mathcal{S}$ sends the shares of corrupted parties to them. For each party P_j, when $\mathcal{S}$ receives $(\texttt{abort}, P_j)$ from $\mathcal{A}$, he sends requests $(\texttt{Fail}, P_j)$ to $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$.

2: In Step 2, $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{acs}}$ and learns the set $\mathcal{D}$ of size $2t+1$.

3: In Step 4, $\mathcal{S}$ follows the protocol and computes the corrupted parties' shares of $[r_{\ell,i}]_t$ for all $\ell \in [N'], i \in [t+1]$. Finally, $\mathcal{S}$ sends the shares of $[r_{\ell,i}]_t$ of corrupted parties to $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$.

4: $\mathcal{S}$ outputs the views of $\mathcal{A}$.

We show that the output in the ideal world is identically distributed to that in the real world by using the following hybrid arguments.

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

$\mathbf{Hyb}_1$: In this hybrid, we follow the protocol and compute the shares of $[r_{\ell,i}]_t$ of corrupted parties for all $\ell \in [N'], i \in [t+1]$. $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same distribution.

$\mathbf{Hyb}_2$: In this hybrid, we change the way of sampling $[s_1^{(i)}]_t, \ldots, [s_{N'}^{(i)}]_t$ for each honest party P_i. Note that in $\mathbf{Hyb}_1$, the shares of honest parties are never sent in the first two steps. Then after randomly sampling the shares of corrupted parties, we delay the generation of the whole sharings until the set $\mathcal{D}$

is determined. Let $\mathcal{H}_{\mathsf{succ}}$ denote the set of the first $t+1$ honest parties in $\mathcal{D}$. Then we generate the whole sharings of honest parties as follows:

- For honest parties not in $\mathcal{H}_{\mathsf{succ}}$: we generate the whole sharings as $\mathbf{Hyb}_1$.
- For honest parties in $\mathcal{H}_{\mathsf{succ}}$: Since $\boldsymbol{M}$ is a Vandermonde matrix, any $(t+1)\times(t+1)$ sub-matrix of $\boldsymbol{M}$ is invertible, then for all $\ell \in [N']$, given the sharings $\{[s_\ell^{(i)}]_t\}_{i\notin\mathcal{H}_{\mathsf{succ}}}$, there is a one-to-one map between $\{[r_{\ell,i}]_t\}_{i=1}^{t+1}$ and $\{[s_\ell^{(i)}]_t\}_{i\in\mathcal{H}_{\mathsf{succ}}}$. We first randomly sample $\{[r_{\ell,i}]_t\}_{i=1}^{t+1}$ based on the shares of corrupted parties and then compute the random sharings of honest parties in $\mathcal{H}_{\mathsf{succ}}$.

This does not change the distribution of the random sharings prepared by honest parties. Thus, $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ have the same distribution.

$\mathbf{Hyb}_3$: In this hybrid, we no longer prepare the shares of $[s_1^{(i)}]_t,\ldots,[s_{N'}^{(i)}]_t$ of honest parties not in $\mathcal{H}_{\mathsf{succ}}$ since they are not used in generating the output of $\mathbf{Hyb}_2$. $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ have the same distribution.

$\mathbf{Hyb}_4$: In this hybrid, when $\mathcal{S}$ simulates each $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ and receives $(\mathtt{abort}, P_i)$ from $\mathcal{A}$, instead of let $\mathcal{S}$ send $\mathtt{Fail}$ to P_i, $\mathcal{S}$ will send $(\mathtt{Fail}, P_i)$ to $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$. As a result, P_i can receive $\mathtt{Fail}$ from $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$, which makes no difference. $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ have the same distribution.

$\mathbf{Hyb}_5$: In the last hybrid, we ask $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$ to generate $\{[r_{\ell,i}]_t\}_{\ell\in[N'],i\in[t+1]}$ based on the shares of corrupted parties. Note that the way of generating $\{[r_{\ell,i}]_t\}_{\ell\in[N'],i\in[t+1]}$ remains unchanged. $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ have the same distribution.

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

C.2 Construction of $\Pi_{\mathtt{pubRec\text{-}Weak}}$ and Security Proof

We design $\Pi_{\mathtt{pubRec\text{-}Weak}}$ which realizes $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ as below. This corresponds to the weak public reconstruction introduced in section 2.2. We assume that for each degree-t Shamir sharing, the shares of honest parties that are not equal to $\perp$ lie on a valid degree-t Shamir sharing. This will be the case when we invoke $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ in our construction.

Functionality $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$

Public Input: N

$\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ proceeds as follows, running with parties $\mathcal{P}=\{P_1,\ldots,P_n\}$ and an adversary $\mathcal{S}$.

1: Wait to receive the number N of degree-t Shamir sharings to be reconstructed from all parties.
2: For all $i\in[N]$, upon receiving the shares of $[s_i]_t$ from all honest parties (including $\perp$), send the shares of $[s_i]_t$ of honest parties to $\mathcal{S}$. If there are at least $t+1$ shares from honest parties, compute the secret s_i by using the shares of honest parties. Otherwise, set $s_i=\perp$.
3: For each party P_j, send a request-based delayed output $\{s_i\}_{i=1}^N$ to P_j.
   – Upon receiving a request $(\mathtt{Fail}, P_j)$ from $\mathcal{S}$, if the output of P_j has not been delivered, change the output of P_j by $\mathtt{Fail}$.
4: All honest parties output the results received from the trusted party. Corrupted parties may output anything they want.

The communication complexity of $\Pi_{\mathtt{pubRec\text{-}Weak}}$ is $\mathcal{O}(N\cdot n+n^2)$ field elements plus $\mathcal{O}(n^2)$ bits for reconstructing N degree-t Shamir sharings.

Protocol $\Pi_{\mathtt{pubRec\text{-}Weak}}$

All parties start with N degree-t Shamir sharings and the goal is to reconstruct the secrets. If a party holds $\perp$ as input, he sends a failure symbol $\perp$ to all other parties and outputs $\mathtt{Fail}$.

1: The Shamir sharings are divided into $N/(t+1)$ groups where each group contains $t+1$ degree-t Shamir sharings. For each group, all parties execute the following steps.
2: Suppose the degree-t Shamir sharings of this group are $[s_0]_t,[s_1]_t,\ldots,[s_t]_t$. All parties locally set a degree-t polynomial $f(\cdot)\in\mathbb{F}[X]$ by using $s_0,\ldots,s_t$ as coefficients, i.e.,

$$f(X)=s_0+s_1\cdot X+\ldots+s_t\cdot X^t.$$

3: Each party whose input sharings are not $\perp$ locally compute $[f(\alpha_1)]_t,\ldots[f(\alpha_n)]_t$.

4: For all $i \in [N]$, all parties send their shares of $[f(\alpha_i)]_t$ to P_i. Then P_i waits to receive $2t+1 = n-t$ shares. If the shares do not lie on a degree-t polynomial, P_i sends a failure symbol $\perp$ to all parties and outputs `Fail`. Otherwise P_i reconstructs the secret $f(\alpha_i)$ and sends $f(\alpha_i)$ to all parties.
5: Each party P_i waits to receive messages from all parties:
- When P_i first receives $f(\alpha_j)$ from $2t+1 = n-t$ different parties P_j, if the received values do not lie on a degree-t polynomial, P_i outputs a failure symbol `Fail`. Otherwise, P_i reconstructs the polynomial $f(\cdot)$ and outputs the coefficients of $f(\cdot)$.
- When P_i first receives a failure symbol $\perp$, P_i outputs `Fail`.

Lemma 8. *Protocol $\Pi_{\texttt{pubRec-Weak}}$ securely computes $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t = (n-1)/3$ parties.*

Proof. **Termination.** We first show that all honest parties will eventually terminate the protocol $\Pi_{\texttt{pubRec-Weak}}$. The first four steps of $\Pi_{\texttt{pubRec-Weak}}$ only involve local computation. In Step 5, each party P_i waits to receive messages from all parties. Since there are at most t corrupted parties, all honest parties' messages will be eventually delivered, then each honest party P_i will eventually receive either a failure symbol $\perp$ or at least $2t+1$ shares. In Step 6, each party P_i again waits to receive messages from all parties. Following the same argument, P_i will eventually receive either $\perp$ or $2t+1$ values of $f(\alpha_j)$ and terminate.

Security. Now we show that the protocol $\Pi_{\texttt{pubRec-Weak}}$ securely computes $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$. We start with the construction of the ideal adversary $\mathcal{S}$ as follows.

Simulator $\mathcal{S}$

1: $\mathcal{S}$ first invokes $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ and receives the shares of each $[s_i]_t$ of honest parties. With honest parties' inputs, $\mathcal{S}$ simply follows the protocol honestly on behalf of each honest party.
2: For each part honest P_j, if $\mathcal{S}$ learns P_j fail to do reconstruction, $\mathcal{S}$ sends $(\texttt{Fail}, P_j)$ to $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$.
3: $\mathcal{S}$ outputs the views of $\mathcal{A}$.

Since $\mathcal{S}$ obtains the inputs of honest parties in the beginning, the simulated output of $\mathcal{A}$ is identically distributed to that of $\mathcal{A}$. It is sufficient to argue that given the output of $\mathcal{A}$, the output of honest parties in the ideal world is identical to that in the real world. We consider two cases.

- Case 1: At most t honest parties hold shares that are not equal to $\perp$. In this case, $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ will always send `Fail` to all honest parties. We argue that this is also the case in the real world. In Step 5, since each honest party P_i waits to receive $2t+1$ shares before continuing, P_i must receive at least one $\perp$ from other parties. Thus, all honest parties will terminate with `Fail` in the real world.
- Case 2: At least $t+1$ honest parties hold shares that are not equal to $\perp$. In this case, we show that all honest parties that do not output `Fail` would receive $\{s_i\}_{i=1}^N$ in the real world. For each group of $t+1$ degree-t Shamir sharings $[s_0]_t, [s_1]_t, \ldots, [s_t]_t$, by assumption, the shares of honest parties that are not $\perp$ form valid degree-t Shamir sharings. Then, for honest parties that do not output `Fail`, their shares of $[f(\alpha_1)]_t, \ldots [f(\alpha_n)]_t$ also form valid degree-t Shamir sharings. Thus for each honest party P_i that does output `Fail` in Step 5, he must reconstruct the correct $f(\alpha_i)$.
  Since f is of degree t and each honest party P_i that does not output `Fail` has distributed the correct $f(\alpha_i)$ to all parties, in Step 6, all honest parties that do not output `Fail` must reconstruct the correct polynomial $f(\cdot)$ and learn $s_0, \ldots, s_t$.

C.3 Construction of $\Pi_{\texttt{randShareZero-Weak}}$

Protocol $\Pi_{\texttt{randShareZero-Weak}}$

Let N be the number of total degree-$2t$ Shamir sharings of 0.
1: Each party P_i samples $N' = N/(t+1)$ random degree-$2t$ Shamir sharings of 0, $([o_1^{(i)}]_{2t}, \ldots, [o_{N'}^{(i)}]_{2t})$. Then P_i acts as the dealer D and runs $\Pi_{\texttt{Sh2tZero-Weak}}$ to distribute the sharings to all parties.
2: Each party P_i sets the property Q as P_i terminating $\Pi_{\texttt{Sh2tZero-Weak}}$ when P_j acts as a dealer, then all parties invoke $\mathcal{F}_{\mathsf{acs}}$ with property Q to agree on a set $\mathcal{D}$ of size $2t+1$ which includes successful dealers.

3: All parties agree on (the inverse of) a Vandermonde matrix $\boldsymbol{M}$ of size $(t+1)\times(2t+1)$. For all $\ell\in[N']$, all parties locally compute

$$([o_{\ell,1}]_{2t},\ldots,[o_{\ell,t+1}]_{2t}) = \boldsymbol{M}\cdot([o_\ell^{(i)}]_{2t})_{i\in\mathcal{D}}.$$

Finally, all parties output $\{[o_{\ell,k}]_{2t}\}_{\ell\in[N'],k\in[t+1]}$.

Costs Analysis. The communication costs of $\Pi_{\texttt{randShareZero-Weak}}$ include n instances of $\Pi_{\texttt{Sh2tZero-Weak}}$ and one instances of $\mathcal{F}_{\mathsf{acs}}$. For each instance of $\Pi_{\texttt{Sh2tZero-Weak}}$, it requires $\mathcal{O}(N\cdot n+n^2)$ field elements to prepare N degree-$2t$ Shamir sharings of zero. Therefore, the total costs of $\Pi_{\texttt{randShareZero-Weak}}$ are $\mathcal{O}(N\cdot n+n^3)$ field elements plus $\mathcal{O}(\kappa\cdot n^3)$ bits for preparing N random degree-t Shamir sharings of zero.

C.4 Construction of $\Pi_{\texttt{tripleExt-Weak}}$

We give the construction of $\Pi_{\texttt{tripleExt-Weak}}$ as follows.

Process $\Pi_{\texttt{tripleExt-Weak}}$

Let N be the number of Beaver triples to be prepared.

1: **Distribution:**
Let $N' = 4N/(t+1), L = 2t+(t-1)/2$, all parties agree on distinct field elements $\beta_1,\ldots,\beta_{(L+1)/2-t},\alpha_0,\ldots,\alpha_{2N'}$.
Each party P_i samples two random degree-$(N'-1)$ polynomials f_i, g_i and computes $h_i = f_i\cdot g_i$. Then P_i samples $3N'$ random degree-t Shamir secret sharings:

$$\{[f_i(\alpha_\ell)]_t\}_{\ell=1}^{N'},\{[g_i(\alpha_\ell)]_t\}_{\ell=1}^{N'},\{[h_i(\alpha_\ell)]_t\}_{\ell=1}^{N'}$$

Finally, P_i acts as a dealer and distributes these $3N'$ degree-t Shamir secret sharings using $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$.

2: **Determine the Set of Successful Dealers:**
All parties set the property Q as terminating $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ invoked by each dealer, then they invoke the modified Π_{acs}^{Q} to agree on a set $\mathcal{D}$ of size L which contains the successful dealers.

3: **Extracting Random Triples:**
For all $\ell\in[N']$, pick the first unused Beaver triple from each dealer in $\mathcal{D}$ and denote them by $\{[a_i^{(\ell)}]_t,[b_i^{(\ell)}]_t,[c_i^{(\ell)}]_t\}_{i=1}^{L}$. Then execute the following steps to extract N random Beaver triples:

(1). For all $i\in[L'+1]$ and $\ell\in[N']$, all parties set two polynomials of $f^{(\ell)}$, $g^{(\ell)}$ of degree $L'=(L-1)/2$ such that $[f^{(\ell)}(\alpha_i)]_t=[a_i^{(\ell)}]_t$ and $[g^{(\ell)}(\alpha_i)]_t=[b_i^{(\ell)}]_t$.

(2). For all $i\in[L'+2,L]$ and $\ell\in[N']$, all parties do the following things:

1). Locally compute $[f^{(\ell)}(\alpha_i)+a_i^{(\ell)}]_t,[g^{(\ell)}(\alpha_i)+b_i^{(\ell)}]_t$.

2). Send input sharings $[f^{(\ell)}(\alpha_i)+a_i^{(\ell)}]_t,[g^{(\ell)}(\alpha_i)+b_i^{(\ell)}]_t$ to $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ to reconstruct $f^{(\ell)}(\alpha_i)+a_i^{(\ell)},g^{(\ell)}(\alpha_i)+b_i^{(\ell)}$. Then locally compute:

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

(3). For each $\ell\in[N']$, all parties set a polynomial $h^{(\ell)}$ of degree $L-1$ such that $[h^{(\ell)}(\alpha_i)]_t=[c_i^{(\ell)}]_t$ for all $i\in[L'+1]$ and $[h^{(\ell)}(\alpha_i)]_t=[f^{(\ell)}(\alpha_i)\cdot g^{(\ell)}(\alpha_i)]_t$ for all $i\in[L'+2,L]$.

(4). Each party finishes the computation in step (3) will output $([f^{(\ell)}(\beta_i)]_t,[g^{(\ell)}(\beta_i)]_t,[h^{(\ell)}(\beta_i)]_t)$ for all $i\in[(L+1)/2-t],\ell\in[N']$. Each party who receives `abort` from $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ for each dealer in $\mathcal{D}$ in Step 2 or receives `Fail` from $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ in Step 3-(2) will output `Fail`.

C.5 Proof of Lemma 2 and Costs Analysis

Proof. **Termination.** We show that all honest parties will eventually terminate the protocol $\Pi_{\texttt{triple-Add-Weak}}$. It is sufficient to show:

1. Each honest party will eventually terminate $\Pi_{\texttt{tripleExt-Weak}}$ or $\Pi_{\texttt{tripleDN-Weak}}$.

2. If an honest party terminates one process, all honest parties will eventually terminate this process.

For process 1, the $\Pi_{\texttt{tripleExt-Weak}}$:

- In the first step, all parties invoke $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ to distribute their shares.
- In the second step, all parties invoke Π^{Q}_{acs} to agree on a set $\mathcal{D}$ of size $L = 2t + (t-1)/2$ which contains the successful dealers. Since there are only $2t+1$ honest parties, we need that at least $(t-3)/2$ corrupted parties acting as dealers also terminate their $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ and then all parties can proceed.
- In the third step, all parties extract the triples and the only interactive step is to invoke $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$, which is guaranteed to terminate.

To summarize, when at least $(t-3)/2$ corrupted parties terminate their $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$, all parties will eventually terminate process 1.

For process 2, the $\Pi_{\texttt{tripleDN-Weak}}$:

- All parties invoke $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$ and $\Pi_{\texttt{randShareZero-Weak}}$ to prepare random degree-t and $2t$ Shamir secret sharings. For $\Pi_{\texttt{randShareZero-Weak}}$:
  - Each party acts as the dealer and invokes $\Pi_{\texttt{Sh2tZero-Weak}}$ to distribute $2t$ shares of zero. Each dealer will encode $(t+1)/2$ shares of zero into a degree-$(2t, t+(t-1)/2)$ bivariate polynomial. To promise that each party can eventually reconstruct his column polynomial, at least $t+(t+1)/2$ honest parties should receive their row polynomial from the dealer. We let each party who receives his row polynomial send `support` to all parties, but since each party can only expect to receive $2t+1$ `support` from all parties, then among these $2t+1$ `support`:
    - If at most $(t+1)/2$ of them come from corrupted parties, that means at least $t+(t+1)/2$ honest parties have received their row polynomial and can help all parties reconstruct their column polynomial.
    - Otherwise, if more than $(t+1)/2$ of them come from corrupted parties, it is not guaranteed that all parties can eventually terminate the $\Pi_{\texttt{Sh2tZero-Weak}}$ when the dealer is corrupted.
  - All parties invoke $\mathcal{F}_{\mathsf{acs}}$ to agree on a set of size $2t+1$ which contains successful dealers. This is guaranteed to be terminated, but for each corrupted dealer in this set, even if an honest party receives his shares, according to the above analysis, when there are more than $(t+1)/2$ corrupted parties sends `support` to this honest party, he can not guarantee all parties can get their shares of zero.
- Each party acts as P_{king} and leads an instances of $\Pi_{\texttt{tripleKingDN}}$. During the $\Pi_{\texttt{tripleKingDN}}$, the honest king can always expect to receive $2t+1$ shares of $[z]_{2t}$ or $\perp$ from $2t+1$ honest parties. Therefore, the honest king will eventually reliably broadcast the secret z or $\perp$ as the response. Then all parties invoke $\mathcal{F}_{\mathsf{acs}}$ to agree on a set $\mathcal{D}$ of size $2t+1$ which contains successful kings, that will be guaranteed to terminate.

To summarize, when more than $(t+1)/2$ corrupted parties send `support` to honest parties, all parties are not guaranteed to get their shares of zero. Otherwise, process 2 is guaranteed to terminate.

Therefore, we find that the termination conditions of processes 1 and 2 are mutually exclusive, we can combine them and require that during process 2, only when each party terminates his $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ in process 1, then the message `support` he send to all parties in process 2 is valid. As a result, if more than $(t+1)/2$ corrupted parties send `support` to honest parties, process 2 may not terminate but process 1 will eventually terminate. Vice versa, if less than $(t-3)/2$ corrupted sends `support` to honest parties in process 2, process 1 may never terminate but process 2 will eventually terminate. All parties will terminate at least one of the processes.

Then in the second step, all parties invoke $\mathcal{F}_{\mathsf{ba}}$ to agree on which process they eventually terminate, this is guaranteed to terminate since each party will take 0 or 1 as his input for $\mathcal{F}_{\mathsf{ba}}$. Assuming that the output of $\mathcal{F}_{\mathsf{ba}}$ is b, we know that at least one honest party's input is b, then we need to prove that if this honest party terminates the corresponding process, the rest of the honest parties can also terminate the corresponding process.

- If $b = 0$: that means at least one honest party accepts a set $\mathcal{D}$ of size L which contains the successful dealers, according to the termination property of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$, the rest of honest parties will also receive their output. Then all parties will eventually terminate the first process $\Pi_{\texttt{tripleExt-Weak}}$.

– If $b = 1$: that means at least one honest party accepts a set $\mathcal{D}$ of size $2t+1$ which contains the successful kings, and for each king, he has received the message broadcast by the king. According to the termination property of reliably broadcast and $\mathcal{F}_{\mathsf{acs}}$, the rest of honest parties will also receive the message from these kings in $\mathcal{D}$. Since $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$ is guaranteed to terminate, then all parties can locally compute their shares of Beaver triples and terminate.

Security. Now we show that the protocol $\Pi_{\mathsf{pubRec\text{-}Weak}}$ securely computes $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$. We start with the construction of the ideal adversary $\mathcal{S}$ as follows.

Simulator $\mathcal{S}_0$

Simulation of $\Pi_{\mathtt{tripleExt\text{-}Weak}}$

1: In the first step, $\mathcal{S}$ simulates each $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ as follows:
   – For each honest dealer P_i, $\mathcal{S}$ randomly samples corrupted parties' shares of $\{[f_i(\alpha_\ell)]_t, [g_i(\alpha_\ell)]_t, [h_i(\alpha_\ell)]_t\}$ for all ℓ and sends them to the corrupted parties.
   – For each corrupted dealer P_i, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ and waits to receive degree-t Shamir secret sharings.

   During the simulation of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$, for each honest party whose output is `abort`, $\mathcal{S}$ sets this honest party's input for $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ as $\perp$ and the output of $\Pi_{\mathtt{tripleExt\text{-}Weak}}$ as `Fail`.

2: In the second step, $\mathcal{S}$ honestly simulates Π^Q_{acs} and whenever an honest party terminates Π^Q_{acs} with a set $\mathcal{D}$ of size L, $\mathcal{S}$ continues to simulate the behavior of this party.

3: In the third step, $\mathcal{S}$ does the following things:
   (1). Compute the corrupted parties' shares of $[f^{(\ell)}(\alpha_i)]_t, [g^{(\ell)}(\alpha_i)]_t$ for all $i \in [L'+1], \ell \in [N']$ and $[f^{(\ell)}(\alpha_i) + a_i^{(\ell)}]_t, [g^{(\ell)}(\alpha_i) + b_i^{(\ell)}]_t$ for all $i \in [L'+2, L], \ell \in [N']$.
   (2). For each $i \in [L'+2, L], \ell \in [N']$, randomly samples degree-t Shamir secret sharings $[f^{(\ell)}(\alpha_i) + a_i^{(\ell)}]_t, [g^{(\ell)}(\alpha_i) + b_i^{(\ell)}]_t$ based on the corrupted parties' shares.
   (3). Compute each honest party's shares of $[f^{(\ell)}(\alpha_i) + a_i^{(\ell)}]_t, [g^{(\ell)}(\alpha_i) + b_i^{(\ell)}]_t$, replace it with $\perp$ if this honest party's output of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ is $\perp$.
   (4). Honestly simulate $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$. For each honest party whose output of $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ is `Fail`, $\mathcal{S}$ sets this honest party's output of $\Pi_{\mathtt{tripleExt\text{-}Weak}}$ as `Fail`. Then compute corrupted parties' shares of $([f^{(\ell)}(\beta_i)]_t, [g^{(\ell)}(\beta_i)]_t, [h^{(\ell)}(\beta_i)]_t)$ for all $i \in [(L+1)/2 - t], \ell \in [N']$.

4: For each corrupted party $P_i \in \mathcal{D}$, $\mathcal{S}$ has received the whole $\{[f_i(\alpha_\ell)]_t, [g_i(\alpha_\ell)]_t, [h_i(\alpha_\ell)]_t\}$ for all ℓ when $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$. Then $\mathcal{S}$ computes a degree-L polynomial $d_i(\cdot)$ such that:
   – For each honest $P_i \in \mathcal{D}$, $\mathcal{S}$ sets $d_i(\alpha_\ell) = 0$.
   – For each corrupted $P_i \in \mathcal{D}$, $\mathcal{S}$ computes $d_i(\alpha_\ell) = h_i(\alpha_\ell) - f_i(\alpha_\ell) \cdot g_i(\alpha_\ell)$.

   Finally, $\mathcal{S}$ computes the additive errors $d(\beta_i)$ in each Beaver triple $([f^{(\ell)}(\beta_i)]_t, [g^{(\ell)}(\beta_i)]_t, [h^{(\ell)}(\beta_i)]_t)$.

Simulator $\mathcal{S}_1$

Simulation of $\Pi_{\mathtt{tripleDN\text{-}Weak}}$

Preparation Phase:

1: $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$, receives shares of corrupted parties from $\mathcal{A}$ and sends them to the corrupted parties.

2: $\mathcal{S}$ simulates $\Pi_{\mathtt{randShareZero\text{-}Weak}}$ as follows:
   (1). In the first step, let $\{[\tilde{o}_k]_{2t}\}_{k=1}^{(t+1)/2}$ be the degree-$2t$ sharings of zero shared by each honest dealer. Then $\mathcal{S}$ randomly samples corrupted parties' shares of $\{[\tilde{o}_k]_{2t}\}_{k=1}^{(t+1)/2}$. $\mathcal{S}$ simulates $\Pi_{\mathtt{Sh2tZero\text{-}Weak}}$ for each dealer as follows:
   - If the dealer is corrupted, $\mathcal{S}$ follows the protocol to execute each honest party. For all $k \in [(t+1)/2]$ $\mathcal{S}$ defines a vector $\Delta[o_k]_{2t}$ to the all-0 vector.
   - If the dealer is honest, $\mathcal{S}$ does the following things:
     1). Randomly sample degree-$2t$ row polynomial $f_\ell^{(i)}(x)$ and degree-$(t+(t-1)/2)$ column polynomial $g_\ell^{(i)}(y)$ based on shares of corrupted parties. Then send row polynomial $f_\ell^{(i)}(x)$ to each corrupted party P_i on behalf of the dealer.
     2). For each honest party P_i, when his row polynomials are delivered, send $f_\ell^{(i)}(\alpha_j) = g_\ell^{(j)}(\alpha_i)$ to corrupted P_j on behalf of P_i. Then send $(\mathtt{support}, P_i, D)$ to all parties.
     3). When each honest party P_i receives $\bar{f}_\ell^{(j)}(\alpha_i)$ from $t+(t+1)/2$ distinct parties P_j, consider that P_i has reconstructed his column polynomials. $\mathcal{S}$ computes a degree-$(t+(t-1)/2)$ polynomial $\Delta g_\ell^{(i)}(y)$ such that, among these $t+(t+1)/2$ distinct parties:

– For each honest P_j, $\Delta g_\ell^{(i)}(\alpha_j) = 0$.
– For each corrupted P_j, $\Delta g_\ell^{(i)}(\alpha_j) = \bar{f}_\ell^{(j)}(\alpha_i) - f_\ell^{(j)}(\alpha_i)$.

4). For each ℓ and all $k \in [(t+1)/2]$, $\mathcal{S}$ defines a vector $\Delta[o_k]_{2t}$ where the j-th entry is $\Delta g_\ell^{(j)}(\beta_k)$ if P_j is honest, and 0 otherwise.

(2). In the second step, $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{acs}}$ and learns a set $\mathcal{D}$ of size $2t+1$ which contains successful dealers.

(3). In the third step, for each output sharing $[o_{\ell,k}]_{2t}$, note that it is a linear combination of the sharings $[o_\ell^{(i)}]_{2t}$ distributed by parties in $\mathcal{D}$. Let $[o_{\ell,k}^{\mathcal{H}}]_{2t}$ be the linear combination of the sharings distributed by honest parties in $\mathcal{D}$ and $[o_{\ell,k}^{\mathcal{C}orr}]_{2t}$ be the linear combination of the sharings distributed by corrupted parties in $\mathcal{D}$. Then $[o_{\ell,k}]_{2t} = [o_{\ell,k}^{\mathcal{H}}]_{2t} + [o_{\ell,k}^{\mathcal{C}orr}]_{2t}$. For each $[o_{\ell,k}]_{2t}$, $\mathcal{S}$ also computes $\Delta[o_{\ell,k}^{\mathcal{H}}]_{2t}$ by using $\Delta[o_\ell^{(i)}]_{2t}$ accordingly.
Note that for $[o_{\ell,k}^{\mathcal{H}}]_{2t} = [\tilde{o}_{\ell,k}^{\mathcal{H}}]_{2t} + \Delta[o_{\ell,k}^{\mathcal{H}}]_{2t}$, $\mathcal{S}$ can compute corrupted parties' shares of $[o_{\ell,k}^{\mathcal{H}}]_{2t}$. For $[o_{\ell,k}^{\mathcal{C}orr}]_{2t}$, $\mathcal{S}$ only learns the shares of honest parties that successfully terminate all executions of $\Pi_{\texttt{Sh2tZero-Weak}}$ led by corrupted parties in $\mathcal{D}$.

Generation Phase:

1: In the first step, rewrite $[z_\ell]_{2t} = [a_\ell]_t \cdot [b_\ell]_t + [r_\ell]_t + [o_\ell^{\mathcal{H}}]_{2t} + [o_\ell^{\mathcal{C}orr}]_{2t}$. Let $[z'_\ell]_{2t} = [a_\ell]_t \cdot [b_\ell]_t + [r_\ell]_t + [o_\ell^{\mathcal{H}}]_{2t} - \Delta[o_\ell^{\mathcal{H}}]_{2t}$. Note that $\mathcal{S}$ can compute the shares of $[z'_\ell]_{2t}$ of corrupted parties. On the other hand, for each honest party P_j that terminates $\Pi_{\texttt{randShareZero-Weak}}$, $\mathcal{S}$ learns the j-th share of $[o_\ell^{\mathcal{C}orr}]_{2t}$.
Then $\mathcal{S}$ simulates each $\Pi_{\texttt{tripleKingDN}}$ as follows:

(1). When each party needs to send his share of $[z_\ell]_{2t}$ to P_{king}, randomly sample the whole $[z'_\ell]_{2t}$ based on corrupted parties' shares, then for each honest party P_j who terminates $\Pi_{\texttt{randShareZero-Weak}}$, compute his share of $[z_\ell]_{2t} = [z'_\ell]_{2t} + \Delta[o_\ell^{\mathcal{H}}]_{2t} + [o_\ell^{\mathcal{C}orr}]_{2t}$.

(2). For each P_{king}:
- If P_{king} is corrupted, send each honest party's share of $[z_\ell]_{2t}$ to P_{king}. Wait to receive z_ℓ or $\perp$ from P_{king}. If first receive z_ℓ, compute the additive error $z_\ell - z'_\ell$. If first receive $\perp$, learn that all honest parties' outputs are `Fail`.
- If P_{king} is honest, honestly follows the protocol. If first receive $2t+1$ shares of $[z_\ell]_{2t}$, compute z_ℓ, reliably broadcast it and extract the additive error $z_\ell - z'_\ell$. If first receive $\perp$, reliably broadcast $\perp$ and learn that all honest parties' outputs are `Fail`.

2: In the second step, $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{acs}}$ and learns a set $\mathcal{D}'$ of size $2t+1$ which contains successful kings. Then $\mathcal{S}$ computes corrupted parties' shares of Beaver triples.

Simulator $\mathcal{S}$

Simulation of $\Pi_{\texttt{triple-Add-Weak}}$

1: $\mathcal{S}$ invokes $\mathcal{S}_0$ and $\mathcal{S}_1$ in parallel.
2: $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{ba}}$ and learns the output b. Then $\mathcal{S}$ sends corrupted parties' shares as well as the additive errors he gets in $\mathcal{S}_b$ to $\mathcal{F}_{\texttt{triple-add-Weak}}$. For each honest party P_i whose output is `Fail` in $\mathcal{S}_b$, $\mathcal{S}$ sends $(\texttt{Fail}, P_i)$ to $\mathcal{F}_{\texttt{triple-add-Weak}}$.
3: $\mathcal{S}$ outputs the views of $\mathcal{A}$.

We show that the output in the ideal world is identically distributed to that in the real world by using the following hybrid arguments.

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

$\mathbf{Hyb}_1$: In the following hybrid, we focus on the simulation $\mathcal{S}_0$ of $\Pi_{\texttt{tripleExt-Weak}}$.

$\mathbf{Hyb}_{1.1}$: In this hybrid, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ as follows:

– If the dealer is corrupted, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ and receives the whole sharings.
– If the dealer is honest, $\mathcal{S}$ randomly samples shares of corrupted parties, then samples the random Beaver triples based on shares of corrupted parties. $\mathbf{Hyb}_{1.1}$ and $\mathbf{Hyb}_0$ have the same distribution.

$\mathbf{Hyb}_{1.2}$: In this hybrid, for each honest dealer, we delay the generation of random Beaver triples until the set $\mathcal{D}$ is terminated. $\mathbf{Hyb}_{1.2}$ and $\mathbf{Hyb}_{1.1}$ have the same distribution.

$\mathbf{Hyb}_{1.3}$: In this hybrid, for each honest party $P_{j_i} \in \mathcal{D}$, we further change the way of determining the first two $([a_i^{(\ell)}]_t, [b_i^{(\ell)}]_t)$ for all ℓ. 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 honest parties based on $[f(\cdot)]_t, [g(\cdot)]_t$.

Assuming that $\mathcal{D} = \{P_{j_1}, \ldots, P_{j_L}\}$ and $\mathcal{D}' = \{P_{j_1}, \ldots, P_{j_{L'+1}}\}$, for each corrupted party P_{j_i}:

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

Note that for each $[f^{(\ell)}(\beta_i)]_t$, it is a linear combination of $\{[f^{(\ell)}(\alpha_i)]_t\}_{i\in\mathcal{D}'}$. Then for all $i \in [(L+1)/2-t]$, $\mathcal{S}$ computes corrupted parties' shares of $[f^{(\ell)}(\beta_i)]_t$ and randomly samples a degree-t Shamir secret sharing as $[f^{(\ell)}(\beta_i)]_t$ based on the shares of corrupted parties.

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

Finally, for each honest party $P_{j_i} \in \mathcal{D}$, note that $[c_i^{(\ell)}]_t = [a_i^{(\ell)} \cdot b_i^{(\ell)}]_t$ is also a random degree-t Shamir sharing given t shares of corrupted parties.

To show that $\mathbf{Hyb}_{1.3}$ and $\mathbf{Hyb}_{1.2}$ have the same distribution, it is sufficient to show that the degree-t Shamir sharings of honest parties generated in the above approach are identically distributed to those in $\mathbf{Hyb}_{1.2}$. To this end, it is sufficient to show that the distribution of the shared polynomial $[f^{(\ell)}(\cdot)]_t$ in both hybrids is identical. In $\mathbf{Hyb}_{1.2}$, $[f^{(\ell)}(\cdot)]_t$ is a random shared polynomial given shares of $[f^{(\ell)}(\alpha_i)]_t$ for all corrupted $P_{j_i} \in \mathcal{D}'$. In $\mathbf{Hyb}_{1.3}$, the only difference is that we additionally fix $[f^{(\ell)}(\alpha_i)]_t$ for all corrupted party $P_{j_i} \in \mathcal{D} \setminus \mathcal{D}'$ and $[f^{(\ell)}(\beta_i)]_t$ for all $i \in [(L+1)/2-t]$. However, those degree-t Shamir sharings are randomly sampled. Therefore, the obtained shared polynomial $[f^{(\ell)}(\cdot)]_t$ has the same distribution as that in $\mathbf{Hyb}_{1.2}$.

$\mathbf{Hyb}_{1.4}$: In this hybrid, for each corrupted party P_{j_i} where $i \in [L'+2, L]$, $\mathcal{S}$ first randomly samples values as $f^{(\ell)}(\alpha_i)+a_i^{(\ell)}, g^{(\ell)}(\alpha_i)+b_i^{(\ell)}$, then computes $f^{(\ell)}(\alpha_i) = f^{(\ell)}(\alpha_i)+a_i^{(\ell)}-a_i^{(\ell)}, g^{(\ell)}(\alpha_i) = g^{(\ell)}(\alpha_i)+b_i^{(\ell)}-b_i^{(\ell)}$. Finally, $\mathcal{S}$ randomly samples $[f^{(\ell)}(\alpha_i)]_t, [g^{(\ell)}(\alpha_i)]_t$ based on secrets $f^{(\ell)}(\alpha_i), g^{(\ell)}(\alpha_i)$ and shares of corrupted parties. $\mathbf{Hyb}_{1.4}$ and $\mathbf{Hyb}_{1.3}$ have the same distribution.

$\mathbf{Hyb}_{1.5}$: In this hybrid, for each honest party P_{j_i} where $i \in [L'+2, L]$, instead of first randomly sampling degree-t Shamir sharings $[a_i^{(\ell)}]_t, [b_i^{(\ell)}]_t$, $\mathcal{S}$ first randomly samples the whole $[f^{(\ell)}(\alpha_i)+a_i^{(\ell)}]_t, [g^{(\ell)}(\alpha_i)+b_i^{(\ell)}]_t$, then compute $[a_i^{(\ell)}]_t = [f^{(\ell)}(\alpha_i)+a_i^{(\ell)}]_t - [f^{(\ell)}(\alpha_i)]_t, [b_i^{(\ell)}]_t = [g^{(\ell)}(\alpha_i)+b_i^{(\ell)}]_t - [g^{(\ell)}(\alpha_i)]_t$. $\mathbf{Hyb}_{1.5}$ and $\mathbf{Hyb}_{1.4}$ have the same distribution.

$\mathbf{Hyb}_{1.6}$: In this hybrid, for each honest party $P_{j_i} \in \mathcal{D}$, we change the way of determining the third sharing $[c_i^{(\ell)}]_t$ 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 party P_{j_i}'s shares of $[h^{(\ell)}(\cdot)]_t$ as follows:

- If $P_{j_i} \in \mathcal{D}'$, set $[h^{(\ell)}(\alpha_i)]_t = [c_i^{(\ell)}]_t$.
- If $P_{j_i} \in \mathcal{D} \setminus \mathcal{D}'$, follows the protocol to compute $[h^{(\ell)}(\alpha_i)]_t$.

Then $\mathcal{S}$ can compute the linear combination of $\{[h^{(\ell)}(\alpha_i)]_t\}_{i\in\mathcal{D}}$ to get corrupted parties' shares of $[h^{(\ell)}(\beta_i)]_t$ for all $i \in [(L+1)/2-t]$. $\mathcal{S}$ computes a degree-$(L-1)$ polynomial $d^{(\ell)}(\cdot)$ such that $d^{(\ell)}(\alpha_i) = c_i^{(\ell)} - a_i^{(\ell)} \cdot b_i^{(\ell)}$ for corrupted $P_{j_i} \in \mathcal{D}$ and $d^{(\ell)}(\alpha_i) = 0$ for honest $P_{j_i} \in \mathcal{D}$. Then $\mathcal{S}$ computes a degree-t Shamir sharing $[h^{(\ell)}(\beta_i)]_t$ based on shares of corrupted parties and the secret $h^{(\ell)}(\beta_i) = f^{(\ell)}(\beta_i) \cdot g^{(\ell)}(\beta_i) + d^{(\ell)}(\beta_i)$.

Now we have fixed $(L+1)/2-t+t'$ degree-t Shamir sharing in $[h^{(\ell)}(\cdot)]_t$, $\mathcal{S}$ computes $h^{(\ell)}(\alpha_i) = f^{(\ell)}(\alpha_i)g^{(\ell)}(\alpha_i)+d^{(\ell)}(\alpha_i)$ and randomly samples $[h^{(\ell)}(\alpha_i)]_t$ based on the $h^{(\ell)}(\alpha_i)$ and shares of corrupted parties for the first $(L-1)/2+t-t'$ honest dealers $P_{j_i} \in \mathcal{D}$. With these L degree-t Shamir sharings, $\mathcal{S}$ interpolates $[h^{(\ell)}(\alpha_i)]_t$ for the rest of honest dealers.

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

To show that $\mathbf{Hyb}_{1.6}$ and $\mathbf{Hyb}_{1.5}$ have the same distribution, it is sufficient to show that the degree-t Shamir sharings of honest parties generated in the above approach are identically distributed to those in $\mathbf{Hyb}_{1.5}$. To this end, it is sufficient to show that the distribution of the shared polynomial $[h^{(\ell)}(\cdot)]_t$ in both hybrids is identical. In $\mathbf{Hyb}_{1.5}$, let $d^{(\ell)}(\cdot)$ be the degree-$(L-1)$ polynomial defined above. Note that for all

$i \in [L]$, we have $d^{(\ell)}(\alpha_i) = h^{(\ell)}(\alpha_i) - f^{(\ell)}(\alpha_i) \cdot g^{(\ell)}(\alpha_i)$. Therefore, $[h^{(\ell)}(\cdot)]_t$ is a random shared polynomial given $h(\cdot) = f(\cdot)g(\cdot)+d(\cdot)$, $[h(\alpha_i)]_t$ for all corrupted party $P_{j_i} \in \mathcal{D}$, and the shares of corrupted parties. In $\mathbf{Hyb}_{1.5}$, the only difference is that we additionally choose $[h(\beta_i)]_t$ for all $i \in [(L+1)/2-t]$. However, those degree-t Shamir sharings are randomly sampled. Therefore, the obtained shared polynomial $[h^{(\ell)}(\cdot)]_t$ has the same distribution as that in $\mathbf{Hyb}_{1.5}$.

$\mathbf{Hyb}_{1.7}$: In this hybrid, we no longer generate the whole random Beaver triples for each honest party. Note that these are never used in the simulation. For each output triple $([f^{(\ell)}(\beta_i)]_t, [g^{(\ell)}(\beta_i)]_t, [h^{(\ell)}(\beta_i)]_t)$, it is a random multiplication triple given the shares of corrupted parties and the additive error $d^{(\ell)}(\beta_i) = h^{(\ell)}(\beta_i) - f^{(\ell)}(\beta_i) \cdot g^{(\ell)}(\beta_i)$. $\mathbf{Hyb}_{1.7}$ and $\mathbf{Hyb}_{1.6}$ have the same distribution.

$\mathbf{Hyb}_2$: In the following hybrid, we focus on the simulation $\mathcal{S}_1$ of $\Pi_{\texttt{tripleDN-Weak}}$.

$\mathbf{Hyb}_{2.1}$: In this hybrid, we delay the generation of honest parties' shares of degree-t Shamir sharings until the Generation phase. $\mathcal{S}$ receives corrupted parties' shares of degree-t Shamir sharings and learns which honest party's output si `Fail` when he simulates $\mathcal{F}_{\texttt{randSh-Weak}}$. Since honest parties' shares of degree-t Shamir sharings are never used in the Preparation phase, $\mathbf{Hyb}_{2.1}$ and $\mathbf{Hyb}_{1.7}$ have the same distribution.

$\mathbf{Hyb}_{2.2}$: In this hybrid, during the $\Pi_{\texttt{randShareZero-Weak}}$, for each $\Pi_{\texttt{Sh2tZero-Weak}}$ led by the honest dealer, $\mathcal{S}$ first randomly samples degree-$2t$ row polynomial $f_\ell^{(i)}(x)$ and degree-$(t+(t-1)/2)$ column polynomial $g_\ell^{(i)}(y)$ for each corrupted party P_i. Then $\mathcal{S}$ computes the whole bivariate polynomial $F_\ell(x, y)$ such that $F_\ell(x, \alpha_i) = f_\ell^{(i)}(x), F_\ell(\alpha_i, y) = g_\ell^{(i)}(y)$ for each corrupted party and $F_\ell(x, \beta_k) = [o_k]_{2t}$ for all $k \in [(t+1)/2]$. We just change the generation method of bivariate polynomial, which does not affect the distribution. $\mathbf{Hyb}_{2.2}$ and $\mathbf{Hyb}_{2.1}$ have the same distribution.

$\mathbf{Hyb}_{2.3}$: In this hybrid, during the $\Pi_{\texttt{randShareZero-Weak}}$, for each $\Pi_{\texttt{Sh2tZero-Weak}}$ led by the honest dealer, we do not generate honest parties' row polynomials.

- When honest party P_i needs to send $f_\ell^{(i)}(\alpha_j)$ to corrupted party P_j, $\mathcal{S}$ sends $f_\ell^{(i)}(\alpha_j) = g_\ell^{(j)}(\alpha_i)$ to P_j on behalf of P_i.
- When honest party P_i needs to send $f_\ell^{(i)}(\alpha_j)$ to honest party P_j, $\mathcal{S}$ first compute P_j's column polynomial $g_\ell^{(j)}(y) = F(\alpha_j, y)$, then $\mathcal{S}$ sends $f_\ell^{(i)}(\alpha_j) = g_\ell^{(j)}(\alpha_i)$ to P_j on behalf of P_i.

Since $F_\ell(\alpha_j, \alpha_i) = g_\ell^{(j)}(\alpha_i) = f_\ell^{(i)}(\alpha_j)$, we do not need to generate $f_\ell^{(i)}(x)$. $\mathbf{Hyb}_{2.3}$ and $\mathbf{Hyb}_{2.2}$ have the same distribution.

$\mathbf{Hyb}_{2.4}$: In this hybrid, we do not generate the whole bivariate polynomial when $\mathcal{S}$ simulates $\Pi_{\texttt{Sh2tZero-Weak}}$ for honest dealer. Let $\{[\tilde{o}_k]_{2t}\}_{k=1}^{(t+1)/2}$ be the sharings that should be distributed by the dealer, and let $\Delta[o_k]_{2t}$ defined as above, we set the output of honest parties to be their shares of $[\tilde{o}_k]_{2t} + \Delta[o_k]_{2t}$ for all $k \in [(t+1)/2]$.

To show that $\mathbf{Hyb}_{2.4}$ and $\mathbf{Hyb}_{2.3}$ have the same distribution, it is sufficient to show $[o_k]_{2t} = [\tilde{o}_k]_{2t} + \Delta[o_k]_{2t}$. During the $\Pi_{\texttt{Sh2tZero-Weak}}$, each honest party P_i's shares of $[o_k]_{2t}$ is determined by $\bar{f}_\ell^{(i)}(\alpha_j)$ received from $t+(t+1)/2$ distinct P_j. Here we assume that the real point distributed by the dealer is $f_\ell^{(i)}(\alpha_j)$ and therefore if P_j is honest, $\bar{f}_\ell^{(i)}(\alpha_j) = f_\ell^{(i)}(\alpha_j)$; otherwise, it can be arbitrary value chosen by corrupted P_j. Then we rewrite $\bar{f}_\ell^{(i)}(\alpha_j) = f_\ell^{(i)}(\alpha_j) + \bar{f}_\ell^{(i)}(\alpha_j) - f_\ell^{(i)}(\alpha_j)$. Note that $[\tilde{o}_k]_{2t}$ is determined by $t + (t+1)/2$ values of $f_\ell^{(i)}(\alpha_j)$. For $\bar{f}_\ell^{(i)}(\alpha_j) - f_\ell^{(i)}(\alpha_j)$, we define a degree-$(t+(t-1)/2)$ polynomial $\Delta g_\ell^{(i)}(y)$ determined by $t + (t+1)/2$ values of $\bar{f}_\ell^{(i)}(\alpha_j) - f_\ell^{(i)}(\alpha_j)$ and the i-th entry of $\Delta[o_k]_{2t}$ as $\Delta g_\ell^{(i)}(\beta_k)$. Therefore, we prove that $[o_k]_{2t} = [\tilde{o}_k]_{2t} + \Delta[o_k]_{2t}$.

$\mathbf{Hyb}_{2.5}$: In this hybrid, let $\mathcal{H}$ be the set of honest parties in $\mathcal{D}$ and $\mathcal{H}'$ be the set of first $t+1$ honest parties in $\mathcal{H}$, we change the generation method of the $[\tilde{o}_\ell^{(i)}]_{2t}$ for the honest dealer $P_i \in \mathcal{H}'$. After randomly sampling the shares of corrupted parties, we delay the generation of the whole sharings until the set $\mathcal{D}$ is determined. For each $[o_{\ell,k}]_{2t}$, we rewrite it as $[o_{\ell,k}]_{2t} = [o_{\ell,k}^{\mathcal{H}}]_{2t} + [o_{\ell,k}^{\mathcal{D}\setminus\mathcal{H}}]_{2t} = [\tilde{o}_{\ell,k}^{\mathcal{H}}]_{2t} + \Delta[o_{\ell,k}^{\mathcal{H}}]_{2t} + [o_{\ell,k}^{\mathcal{D}\setminus\mathcal{H}}]_{2t}$. Since $\boldsymbol{M}$ is a Vandermonde matrix, there is a one-to-one map between $\{[\tilde{o}_{\ell,k}^{\mathcal{H}}]_{2t}\}_{k=1}^{t+1}$ and $\{[\tilde{o}_\ell^{(i)}]_{2t}\}_{i\in\mathcal{H}'}$. For each honest $P_i \notin \mathcal{H}'$, we prepare $[\tilde{o}_\ell^{(i)}]_{2t}$ in the same way as that in $\mathbf{Hyb}_{2.4}$. Then we randomly samples $\{[\tilde{o}_{\ell,k}^{\mathcal{H}}]_{2t}\}_{k=1}^{t+1}$ based on the shares of corrupted parties and compute $\{[\tilde{o}_\ell^{(i)}]_{2t}\}_{i\in\mathcal{H}'}$ from $\{[\tilde{o}_{\ell,k}^{\mathcal{H}}]_{2t}\}_{k=1}^{t+1}$. This does not change the distribution of the random sharings prepared by honest parties. Thus, $\mathbf{Hyb}_{2.5}$ and $\mathbf{Hyb}_{2.4}$ have the same distribution.

$\mathbf{Hyb}_{2.6}$: In this hybrid, we no longer prepare the shares of $[\tilde{o}_\ell^{(i)}]_{2t}$ for each honest $P_i \in \mathcal{H}'$ since they are not used in generating the output of $\mathbf{Hyb}_{2.5}$. Thus, $\mathbf{Hyb}_{2.6}$ and $\mathbf{Hyb}_{2.5}$ have the same distribution.

$\mathbf{Hyb}_{2.7}$: In this hybrid, during each $\Pi_{\texttt{tripleKingDN}}$, each secret r_ℓ is generated by first sampling a random value as $a_\ell \cdot b_\ell + r_\ell$ and then computing $r_\ell = (a_\ell \cdot b_\ell + r_\ell) - a_\ell \cdot b_\ell$. This does not change the distribution of $\{[a_\ell]_t, [b_\ell]_t, [r_\ell]_t\}$. $\mathbf{Hyb}_{2.7}$ and $\mathbf{Hyb}_{2.6}$ have the same distribution.

$\mathbf{Hyb}_{2.8}$: In this hybrid, during each $\Pi_{\texttt{tripleKingDN}}$, recall that each degree-$2t$ Shamir secret sharing $[o_\ell]_{2t} = [o_\ell^{\mathcal{H}}]_{2t} + [o_\ell^{\mathcal{C}orr}]_{2t} = [\tilde{o}_\ell^{\mathcal{H}}]_{2t} + \Delta[o_\ell^{\mathcal{H}}]_{2t} + [o_\ell^{\mathcal{C}orr}]_{2t}$. In particular, in $\mathbf{Hyb}_{2.6}$, $[\tilde{o}_\ell^{\mathcal{H}}]_{2t}$ is sampled as a random degree-$2t$ Shamir sharing of 0 based on the shares of corrupted parties. We change the way of sampling $[\tilde{o}_\ell^{\mathcal{H}}]_{2t}$ as follows. $\mathcal{S}$ computes the shares of $[z'_\ell]_{2t} = [a_\ell]_t \cdot [b_\ell]_t + [r_\ell]_t + [\tilde{o}_\ell^{\mathcal{H}}]_{2t}$ of corrupted parties, and randomly samples a degree-$2t$ Shamir sharing of $a_\ell \cdot b_\ell + r_\ell$ as $[z'_\ell]_{2t}$ based on the shares of corrupted parties. Finally, $\mathcal{S}$ computes $[\tilde{o}_\ell^{\mathcal{H}}]_{2t} = [z'_\ell]_{2t} - ([a_\ell]_t \cdot [b_\ell]_t + [r_\ell]_t)$. $\mathbf{Hyb}_{2.8}$ and $\mathbf{Hyb}_{2.7}$ have the same distribution.

$\mathbf{Hyb}_{2.9}$: In this hybrid, we change the way of generating $[r_\ell]_t$. During each $\Pi_{\texttt{tripleKingDN}}$, after P_{king} successfully broadcast z_ℓ, $\mathcal{S}$ first follows the protocol and computes the shares of $[c_\ell]$ of corrupted parties. Then $\mathcal{S}$ randomly samples a degree-t Shamir sharing based on the secret $c_\ell = a_\ell \cdot b_\ell + z_\ell - z'_\ell$ and shares of corrupted parties. Then $\mathcal{S}$ computes $[r_\ell]_t = z_\ell - [c_\ell]_t$. $\mathbf{Hyb}_{2.9}$ and $\mathbf{Hyb}_{2.8}$ have the same distribution.

$\mathbf{Hyb}_{2.10}$: In this hybrid, we no longer generate $[o_\ell^{\mathcal{H}}]_{2t}$ and $[r_\ell]_t$ since they are never used in the simulation. Note that for each output triple $([a_\ell]_t, [b_\ell]_t, [c_\ell]_t)$, it is a random multiplication triple given the shares of corrupted parties and the additive error $c_\ell - a_\ell \cdot b_\ell = z_\ell - z'_\ell$. $\mathbf{Hyb}_{2.10}$ and $\mathbf{Hyb}_{2.9}$ have the same distribution.

$\mathbf{Hyb}_3$: In the following hybrid, we focus on the simulation $\mathcal{S}$ of $\Pi_{\texttt{tripleExt-Weak}}$. After invoking $\mathcal{S}_0$ and $\mathcal{S}_1$, $\mathcal{S}$ honestly simulates Step 2 of the Generation phase in $\Pi_{\texttt{triple-Add-Weak}}$ and therefore learns which process success. Assuming that $\mathcal{S}$ gets output b when he simulates $\mathcal{F}_{\mathsf{ba}}$, then $\mathcal{S}$ no longer generates honest parties' shares of Beaver triples but sends corrupted parties' shares of Beaver triples, the additive errors and $(\texttt{Fail}, P_i)$ for all honest party P_i whose output is $\texttt{Fail}$ during the simulation to $\mathcal{F}_{\mathsf{triple\text{-}add\text{-}Weak}}$. Since those triples are generated in the same way. $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_{2.10}$ have the same distribution.

Note that $\mathbf{Hyb}_3$ corresponds to the ideal world, then $\Pi_{\texttt{triple-Add-Weak}}$ securely computes $\mathcal{F}_{\mathsf{triple\text{-}add\text{-}Weak}}$.

Cost Analysis. We start by analyzing the communication complexity of each sub-protocol. $\Pi_{\texttt{triple-Weak}}$ includes one instance of $\Pi_{\texttt{tripleExt-Weak}}$, $\Pi_{\texttt{tripleDN-Weak}}$ and $\mathcal{F}_{\mathsf{ba}}$.

For $\Pi_{\texttt{tripleExt-Weak}}$, to prepare N random Beaver triples:

- In the first step, we require n instances of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$, after instantiating them by $\Pi_{\mathsf{ACSS\text{-}ab}}$, that requires $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits.
- In the second step, we require an instance of Π_{acs}, which requires $\mathcal{O}(\kappa \cdot n^3)$ bits.
- In the third step, we require an instance of $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$, after instantiating it by $\Pi_{\mathsf{pubRec\text{-}Weak}}$, that requires $\mathcal{O}(N \cdot n + n^2)$ field elements.

To summarize, $\Pi_{\texttt{tripleExt-Weak}}$ requires $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits for preparing N random Beaver triples.

For $\Pi_{\texttt{tripleDN-Weak}}$, to prepare N Beaver triples:

- In the Preparation phase, all parties invoke an instance of $\mathcal{F}_{\mathsf{randSh\text{-}Weak}}$ to prepare $\mathcal{O}(N)$ random degree-t Shamir sharings, after instantiating it by $\Pi_{\texttt{randSh-Weak}}$, that requires $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits. All parties invoke an instance of $\Pi_{\texttt{randShareZero-Weak}}$ to prepare $\mathcal{O}(N)$ degree-$2t$ Shamir sharings, which contains n instances of $\Pi_{\texttt{Sh2tZero-Weak}}$ and an instance of $\mathcal{F}_{\mathsf{acs}}$. Since for each $\Pi_{\texttt{Sh2tZero-Weak}}$, it requires $\mathcal{O}(N' \cdot n + n^2)$ field elements for sharing N' sharings. Then the whole $\Pi_{\texttt{randShareZero-Weak}}$ requires $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits.
- In the first step of the Generation phase, all parties invoke n instances of $\Pi_{\texttt{tripleKingDN}}$. For each $\Pi_{\texttt{tripleKingDN}}$, it requires $\mathcal{O}(N' \cdot n)$ field elements plus $\mathcal{O}(\kappa \cdot n^2)$ bits for preparing N' random Beaver triples. Then it requires $\mathcal{O}(N \cdot n)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits in total.
- In the second step of the Generation phase, all parties invoke an instance of $\mathcal{F}_{\mathsf{acs}}$, which requires $\mathcal{O}(\kappa \cdot n^3)$ bits.

To summarize, $\Pi_{\texttt{tripleDN-Weak}}$ requires $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits for preparing N random Beaver triples.

The $\mathcal{F}_{\mathsf{ba}}$ here can be instantiated by a binary BA protocol, which requires $\mathcal{O}(n^3)$ bits, therefore, the whole communication complexity of $\Pi_{\texttt{triple-Add-Weak}}$ is $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits.

C.6 Construction of $\Pi_{\texttt{tripleVerify-Weak}}$

We give the construction of $\Pi_{\texttt{tripleVerify-Weak}}$ as follows. At a high level, to prepare N random Beaver triples, let all parties first prepare $2N+1$ random Beaver triples with additive errors, denoted by $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=0}^{2N}$.

The verification process is very similar to the triple extraction process. Let $\alpha_0, \ldots, \alpha_{2N}$ be $2N+1$ distinct field elements. All parties set two polynomials f, g of degree N such that $[f(\alpha_i)]_t = [a_i]_t$ and $[g(\alpha_i)]_t = [b_i]_t$ for all $i \in [0, N]$. Then for all $i \in [N+1, 2N]$, all parties locally compute $[f(\alpha_i)]_t, [g(\alpha_i)]_t$ and use the i-th triple $([a_i]_t, [b_i]_t, [c_i]_t)$ to compute $[f(\alpha_i) \cdot g(\alpha_i)]_t$. Now all parties set a degree-$2N$ polynomial h such that $[h(\alpha_i)] = [c_i]_t$ for all $i \in [0, N]$, and $[h(\alpha_i)]_t = [f(\alpha_i) \cdot g(\alpha_i)]_t$.

The main observation is that, if all random Beaver triples are correct, then we have $h = f \cdot g$ and vice versa. Therefore, to check whether all Beaver triples are correct, it is sufficient to check whether $h = f \cdot g$. By Schwartz-Zippel lemma, it is sufficient to test a random evaluation point. Each party locally computes and sends his share of $([f(r)]_t, [g(r)]_t, [h(r)]_t)$ to all parties. All parties will use online error correction to reconstruct the secrets $f(r), g(r), h(r)$ and check whether $h(r) = f(r) \cdot g(r)$.

Protocol $\Pi_{\texttt{tripleVerify-Weak}}$

Let N be the number of Beaver triples that need to be verified, all parties agree on $2N+1$ distinct field elements $\alpha_0, \ldots, \alpha_{2N}$ and use their shares of $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=0}^{2N+1}$ or $\perp$ as inputs.

1: **Build Polynomials:**

(1). All parties set two polynomials of f, g of degree N such that $[f(\alpha_i)]_t = [a_i]_t$ and $[g(\alpha_i)]_t = [b_i]_t$ for all $i \in [0, N]$.

(2). For all $i \in [N+1, 2N]$, all parties locally compute $[f(\alpha_i)]_t$, $[g(\alpha_i)]_t$. Then they execute $\mathcal{F}_{\text{pubRec-Weak}}$ with input sharings $[f(\alpha_i) + a_i]_t$, $[g(\alpha_i) + b_i]_t$ or $\perp$ to reconstruct $f(\alpha_i) + a_i, g(\alpha_i) + b_i$. Each party who has all shares of $([a_i]_t, [b_i]_t, [c_i]_t)$ and receives $f(\alpha_i) + a_i, g(\alpha_i) + b_i$ from $\mathcal{F}_{\text{pubRec-Weak}}$ locally computes:

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

Otherwise, he sets his output as a failure symbol $\perp$.

(3). All parties set a polynomial h of degree $2N$ such that $[h(\alpha_i)]_t = [c_i]_t$ for all $i \in [0, N]$ and $[h(\alpha_i)]_t = [f(\alpha_i) \cdot g(\alpha_i)]_t$ for all $i \in [N+1, 2N]$.

2: **Verification and Output Phase:**

All parties locally set $[r]_t := [a_{2N+1}]_t$ and execute $\mathcal{F}_{\text{pubRec-Weak}}$ with input sharing $[r]_t$ or $\perp$ to reconstruct r. Each party who receives `Fail` or $r \in \{\alpha_1, \ldots, \alpha_N\}$ from $\mathcal{F}_{\text{pubRec-Weak}}$ will set his output as $\perp$.

(1). Each party locally computes his share of $([f(r)]_t, [g(r)]_t, [h(r)]_t)$ (if he can). Then all parties execute $\mathcal{F}_{\text{pubRec-Weak}}$ with input sharings $([f(r)]_t, [g(r)]_t, [h(r)]_t)$ or $\perp$ to reconstruct the secrets $f(r), g(r), h(r)$. Similarly, each party who receives `abort` from $\mathcal{F}_{\text{pubRec-Weak}}$ sets his output as $\perp$.

(2). Each party checks whether $h(r) = f(r) \cdot g(r)$. If true and he has his shares of $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=1}^{N}$, he uses them as output. Otherwise, he must have set his output as $\perp$ in the previous steps and will use `Fail` as output.

C.7 Proof of Lemma 3 and Costs Analysis

Proof. **Termination.** We first show that all honest parties will eventually terminate the protocol $\Pi_{\texttt{triple-Weak}}$. In Step 1, all parties invoke $\mathcal{F}_{\text{triple-add-Weak}}$ to prepare Beaver triples, each party will eventually get his shares of Beaver triples or output `Fail`. Then in Step 2, when all parties invoke $\Pi_{\texttt{tripleVerify-Weak}}$ to check their Beaver triples, since all parties only invoke $\mathcal{F}_{\text{pubRec-Weak}}$ to do public reconstruction and the rest of the steps are just local computation, each party will eventually terminate with `Fail` or get his shares of Beaver triples (without additive errors).

Security. Now we show that the protocol $\Pi_{\texttt{pubRec-Weak}}$ securely computes $\mathcal{F}_{\text{pubRec-Weak}}$. We start with the construction of the ideal adversary $\mathcal{S}$ as follows.

Simulator $\mathcal{S}$

Denote N as the number of Beaver triples.

1: In Step 1, $\mathcal{S}$ simulates $\mathcal{F}_{\text{triple-add-Weak}}$ and learns which honest party's output is `Fail`, corrupted parties' shares of Beaver triples and the additive errors d_i for all $i \in [0, 2N+1]$.

2: In Step 2, during the $\Pi_{\texttt{tripleVerify-Weak}}$:

– **In Step 1 of $\Pi_{\texttt{tripleVerify-Weak}}$:**

(1). For each corrupted party, $\mathcal{S}$ locally computes his shares of $[f(\alpha_i)]_t, [g(\alpha_i)]_t$ for all $i \in [0, 2N]$. Then for all $i \in [N+1, 2N]$, $\mathcal{S}$ computes the corrupted parties' shares of $[f(\alpha_i) + a_i]_t, [g(\alpha_i) + b_i]_t$ and randomly sample two degree-t Shamir secret sharings based on corrupted parties' shares.

(2). $\mathcal{S}$ computes each honest party's shares of $[f(\alpha_i) + a_i]_t, [g(\alpha_i) + b_i]_t$ and replaces it with $\perp$ if this honest party's output of $\mathcal{F}_{\text{triple-add-Weak}}$ is `Fail`.

(3). $\mathcal{S}$ honestly simulates $\mathcal{F}_{\text{pubRec-Weak}}$ and learns which honest party's output is `Fail`.

(4). $\mathcal{S}$ defines a polynomial $d(\cdot)$ of degree-$2N$ such that for all $i \in [0, 2N]$, $d(\alpha_i) = d_i$. $\mathcal{S}$ also follows the protocol and computes corrupted parties' shares of $[h(\alpha_i)]_t$ for all $i \in [0, 2N]$.

– **In Step 2 of $\Pi_{\texttt{tripleVerify-Weak}}$:**

(1). $\mathcal{S}$ randomly samples a degree-t Shamir sharings as $[r]_t := [a_{2N+1}]_t$ based on the shares of corrupted parties. Then $\mathcal{S}$ computes each honest party's shares of $[r]_t$ and replaces it with $\perp$ if this honest party's output of $\mathcal{F}_{\text{triple-add-Weak}}$ is `Fail`.

(2). $\mathcal{S}$ honestly simulates $\mathcal{F}_{\text{pubRec-Weak}}$ and learns which honest party's output is `Fail`. If $r \in \{\alpha_1, \ldots, \alpha_N\}$, $\mathcal{S}$ sets all honest parties' outputs as $\perp$. Otherwise, $\mathcal{S}$ computes $d(r)$, samples two random values as $f(r), g(r)$ and computes $h(r) = f(r) \cdot g(r) + d(r)$.

(3). $\mathcal{S}$ randomly samples three degree-t Shamir secret sharings $[f(r)]_t, [g(r)]_t, [h(r)]_t$ based on the shares of corrupted parties and the secrets $f(r), g(r), h(r)$.

(4). $\mathcal{S}$ computes each honest party's shares of $[f(r)]_t, [g(r)]_t, [h(r)]_t$ and replaces it with $\perp$ if this honest party's output has been set to $\perp$.

(5). $\mathcal{S}$ honestly simulates $\mathcal{F}_{\text{pubRec-Weak}}$ and learns which honest party's output is `Fail`.

(6). $\mathcal{S}$ checks whether $h(r) = f(r) \cdot g(r)$, if true and $d(\cdot) \equiv 0$, $\mathcal{S}$ sends corrupted parties' shares of $([a_i]_t, [b_i]_t, [c_i]_t)_{i=1}^N$ to $\mathcal{F}_{\text{triple-Weak}}$. If $h(r) = f(r) \cdot g(r)$ but $d(\cdot) \not\equiv 0$, $\mathcal{S}$ aborts the simulation. If $h(r) \neq f(r) \cdot g(r)$, $\mathcal{S}$ sets all honest parties' output as $\perp$.

3: For each honest party whose output has been set to $\perp$, $\mathcal{S}$ sends $(\texttt{Fail}, P_i)$ to $\mathcal{F}_{\text{triple-Weak}}$. $\mathcal{S}$ also outputs the views of $\mathcal{A}$.

We show that the output in the ideal world is identically distributed to that in the real world by using the following hybrid arguments.

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

$\mathbf{Hyb}_1$: In this hybrid, let $d(\cdot)$ be defined as above. Then $d = h - f \cdot g$. If $r \notin \{\alpha_1, \ldots, \alpha_N\}$, $d(\cdot) \not\equiv 0$ and $d(r) = 0$, $\mathcal{S}$ aborts the simulation. By the Schwartz-Zipple lemma, the probability is at most $\frac{2N}{2^\kappa - N}$, which is negligible in the security parameter κ. Thus, the distributions of $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are statistically close.

$\mathbf{Hyb}_2$: In this hybrid, we delay the generation of $([a_i]_t, [b_i]_t, [c_i]_t)$ for $i \in [N+1, 2N]$:

In the first step of $\Pi_{\texttt{tripleVerify-Weak}}$, For $i \in [N+1, 2N]$ $\mathcal{S}$ randomly samples two degree-t Shamir secret sharings $[f(\alpha_i) + a_i]_t, [g(\alpha_i) + b_i]_t$ based on corrupted parties shares. Then $\mathcal{S}$ computes each honest party's shares of $[f(\alpha_i) + a_i]_t, [g(\alpha_i) + b_i]_t$ and $[a_i]_t = [f(\alpha_i) + a_i]_t - [f(\alpha_i)]_t, [b_i]_t = [g(\alpha_i) + b_i]_t - [g(\alpha_i)]_t$ for all $i \in [N+1, 2N]$. Since for $i \in [N+1, 2N]$, $[a_i]_t, [b_i]_t$ are random values, we just change the generation order, which does not influence the distribution of $[a_i]_t, [b_i]_t$.

For $i \in [N+1, 2N]$, $\mathcal{S}$ computes degree-t Shamir secret sharing $[h(\alpha_i)]_t$ based on t shares of corrupted parties and the secret $h(\alpha_i) = f(\alpha_i) \cdot g(\alpha_i) + d_i$. Then $\mathcal{S}$ computes $[c_i]_t = [h(\alpha_i)]_t - (f(\alpha_i) + a_i) \cdot (g(\alpha_i) + b_i) + (g(\alpha_i) + b_i)[a_i]_t + (f(\alpha_i) + a_i)[b_i]_t$. Note that $c_i = a_i \cdot b_i + d_i$ and $[c_i]_t$ is a random degree-t Shamir secret sharing of c_i given the shares of corrupted parties.

Finally, $\mathcal{S}$ honestly simulates $\mathcal{F}_{\text{pubRec-Weak}}$ and computes corrupted parties' shares of $[h(\alpha_i)]_t$ for all $i \in [0, 2N]$. $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ have the same distribution.

$\mathbf{Hyb}_3$: In this hybrid, we delay the generation of $([a_0]_t, [b_0]_t, [c_0]_t)$. When $r \notin [\alpha_1, \ldots, \alpha_N]$, we have $f(r)$ is a linear combination of $f(\alpha_0) = a_0, \ldots, f(\alpha_N)$ and the coefficient of $f(\alpha_0)$ is non-zero. $\mathcal{S}$ first randomly samples $[f(r)]_t$ based on the shares of corrupted parties and then computes $[f(\alpha_0)]_t$ by using $[f(\alpha_1)]_t, \ldots, [f(\alpha_N)]_t$. $\mathcal{S}$ also generates $[g(\alpha_0)]_t$ in a similar way. Finally, $\mathcal{S}$ generates $[c_0]_t$ accordingly. $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ have the same distribution.

Hyb$_4$: In this hybrid, we change the way to generate $[c_0]_t$ and compute $[h(r)]_t$ when $r \notin \{\alpha_0, \ldots, \alpha_{2N}\}$. Recall that $h = f \cdot g + d$, $\mathcal{S}$ first randomly samples a degree-t Shamir secret sharing $[h(r)]_t$ based on t corrupted parties' shares and the secret $h(r) = f(r) \cdot g(r) + d(r)$. Finally, $\mathcal{S}$ computes $[c_0]_t = [h(\alpha_0)]_t$ as a linear combination of $[h(r)]_t, [h(\alpha_1)]_t, \ldots, [h(\alpha_{2N})]_t$. In **Hyb**$_3$, $[h(r)]_t$ is the linear combination of $[h(\alpha_0)]_t, \ldots, [h(\alpha_{2N})]_t$. When $r \notin \{\alpha_0, \ldots, \alpha_{2N}\}$, the coefficient of $[h(\alpha_0)]_t$ is not zero. Since $[c_0]_t$ is sampled as a random degree-t Shamir sharing of $h(\alpha_0)$ given the shares of corrupted parties in **Hyb**$_3$, $[c_0]_t$ generated in **Hyb**$_4$ has the same distribution as that in **Hyb**$_3$. **Hyb**$_4$ and **Hyb**$_3$ have the same distribution.

Hyb$_5$: In this hybrid, $\mathcal{S}$ does not generate honest parties' shares of $\{[a_0]_t, [b_0]_t, [c_0]_t\} \cup \{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=N+1}^{2N}$ since they are never used. **Hyb**$_5$ and **Hyb**$_4$ have the same distribution.

Hyb$_6$: In this hybrid, $\mathcal{S}$ does not generate honest parties' shares of $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=1}^{N}$, just provides shares of $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=1}^{N}$ of corrupted parties to $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$. For each honest party whose output is `Fail`, $\mathcal{S}$ sends $(\texttt{Fail}, P_i)$ to $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$. **Hyb**$_6$ and **Hyb**$_5$ have the same distribution.

Note that **Hyb**$_6$ corresponds to the ideal world, then $\Pi_{\texttt{triple-Weak}}$ securely computes $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$.

Cost Analysis. $\Pi_{\texttt{triple-Weak}}$ contains one instance of $\mathcal{F}_{\mathsf{triple\text{-}add\text{-}Weak}}$ and $\Pi_{\texttt{tripleVerify-Weak}}$. For $\mathcal{F}_{\mathsf{triple\text{-}add\text{-}Weak}}$, it requires $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits for preparing $\mathcal{O}(N)$ random Beaver triples. For $\Pi_{\texttt{tripleVerify-Weak}}$:

- In the first step, all parties invoke one instance of $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ to reconstruct $\mathcal{O}(N)$ field elements, which requires $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(n^2)$ bits.
- In the second step, all parties invoke one instance of $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ to reconstruct a field element, which requires $\mathcal{O}(n^2)$ field elements plus $\mathcal{O}(n^2)$ bits.

Therefore, $\Pi_{\texttt{tripleVerify-Weak}}$ requires $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(n^2)$ bits. The total communication complexity of $\Pi_{\texttt{triple-Weak}}$ is $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits.

C.8 Construction of Main protocol

We first define the functionality $\mathcal{F}_{\mathsf{AMPC\text{-}Fair}}$ we want to achieve below.

Functionality $\mathcal{F}_{\mathsf{AMPC\text{-}Fair}}$

$\mathcal{F}_{\mathsf{AMPC\text{-}Fair}}$ proceeds as follows, running with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, an adversary $\mathcal{S}$ and a n-party function $f : (\{0,1\}^* \cup \{\perp\})^n \to \{0,1\}^* \cup \{\perp\}$. For each party P_i, initialize an input value $x^{(i)} = \perp$.

1: Upon receiving an input v from $P_i \in \mathcal{P}$, if CoreSet has not been recorded yet or if $P_i \in$ CoreSet, set $x^{(i)} = v$.
2: Upon receiving an input CoreSet from $\mathcal{S}$, verify that CoreSet is a subset of $\mathcal{P}$ of size at least $n - t$, else ignore the message. If CoreSet has not been recorded yet, then record CoreSet and for every $P_i \notin$ CoreSet, set $x^{(i)} = 0$.
3: If the CoreSet has been recorded and the value $x^{(i)}$ has been set to a value different from $\perp$ for every $P_i \in$ CoreSet, compute $y = f(x^{(1)}, \ldots, x^{(n)})$. Then for each $P_i \in \mathcal{P}$, send a requested-based delayed output y to P_i.
   – Upon receiving the request `Fail`, if the output has not been delivered to any party, change the output of all parties by `Fail`. Otherwise, ignore this request.
4: All honest parties output the results received from the trusted party. Corrupted parties may output anything they want.

Then we recall the brief outline of our AMPC with fairness and give the detailed construction as follows.

Step 1: Preparing Random Beaver Triples. In the offline phase, all parties invoke $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$ to prepare a sufficient number of random Beaver triples.

Step 2: Distributing Input t-Sharings. In the input phase, each party invokes $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ to distribute his input as well as a random mask. Then all parties execute an ACS protocol to agree on a set of $n - t$ successful dealers. Later all parties will sum up the mask shared by the first $t + 1$ successful dealers as a random mask of output.

Step 3: Evaluation of the Circuit. In the computation phase, all parties compute each addition gate locally and jointly use Beaver triples to compute each multiplication gate. They will invoke $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ to do public reconstruction.
Step 4: Output of the Circuit. Here we assume that all parties' outputs are the same. In the output phase, each party adds his share of the random mask to his shares of output and sends the result to all parties. Then he waits to receive $2t+1$ output shares and checks whether they lie on a degree-t polynomial. If true, he sets his output as the reconstruction result; otherwise, he sets his output as $\perp$.
Step 5: Termination Process. In the termination phase, to achieve fairness, we first let all parties invoke the byzantine agreement to agree on a message that is either $\perp$ or the correct output plus a mask. If all parties agree on $\perp$, they will terminate with `Fail`. Otherwise, all parties jointly reconstruct the mask and recover the output.

Protocol $\Pi_{\texttt{main-Fair}}$

Let C_O be the output size.

Offline Phase

1: Let C denote the circuit to be computed. All parties invoke $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$ to prepare $|C|$ random Beaver triples and assign one random triple with each multiplication gate in the circuit. In the following, if a party receives `Fail` from $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$, this party sets his output as $\perp$ and proceeds.

Input Phase

1: Each party P_i acts as a dealer invokes two instances of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ to distribute his shares x_i and C_O random value r_i respectively.
2: Each party P_i sets the property Q as P_i terminating $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ where the dealer is P_j. Then all parties invoke $\mathcal{F}_{\mathsf{acs}}$ with property Q to agree on a set $\mathcal{D}$ of size $2t+1$ and each party in this set has successfully shared their inputs. For every $P_i \notin \mathcal{D}$, all parties set their shares of P_i's input as 0.
3: If party P_i terminates $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ for any dealer in $\mathcal{D}$ with `abort`, he sets his output as $\perp$ and proceeds.

Computation Phase

Each party who has set his output as $\perp$ will send $\perp$ to $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ and do not compute $[z_i]_t$.
1: For every addition gate with input sharings $[x]_t, [y]_t$, all parties locally compute $[z]_t = [x]_t + [y]_t$.
2: For a group of at most $n - 2t = t + 1$ multiplication gates, suppose the input degree-t Shamir sharings are denoted by $([x_i]_t, [y_i]_t)_{i=1}^{t+1}$. 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.
   1. All parties locally compute $[x_i + a_i]_t = [x_i]_t + [a_i]_t$ and $[y_i + b_i]_t = [y_i]_t + [b_i]_t$ for all $i \in [t+1]$.
   2. All parties invoke $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ to reconstruct $\{x_i + a_i, y_i + b_i\}_{i=1}^{t+1}$.
   3. For all $i \in [t+1]$, all parties locally compute:

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

Output Phase

1: For each output wire $[y]_t$, we assign the random masks $\{[r_i]_t\}_{i\in\mathcal{D}}$. For the first $t+1$ successful dealer $P_i \in \mathcal{D}$, each party checks whether he has shares of $[r_i]_t$ for each P_i. If true, he computes $[r]_t = \sum_{i=1}^{t+1}[r_i]_t$ and proceeds; otherwise, he sets his output as $\perp$ and proceeds.
2: For output $[y]_t$, each party who has shares of $[y]_t, [r]_t$ will send his share of $[y+r]_t = [y]_t + [r]_t$ to all parties. Each party who has set his output as $\perp$ will send $\perp$ to all parties.
3: For each party, if he first receives $2t+1$ shares of $[y+r]_t$ and these shares lie on a degree-t polynomial, he reconstructs and outputs the secret $y+r$. Otherwise, if he first receives $\perp$ or fails to do reconstruction, he sets his output as $\perp$ and proceeds.

Termination Phase.

1: All parties invoke $\mathcal{F}_{\mathsf{ba}}$ to agree on the output. For each party P_i, if his output is $y' = y + r$, he sets $y_i = 1|y'$. Otherwise, if his output has been set to $\perp$, he sets y_i as a zero string of size $|y'| + 1$. Then he sends y_i to $\mathcal{F}_{\mathsf{ba}}$.
2: Upon receiving $\bar{y}$ from $\mathcal{F}_{\mathsf{ba}}$, if the first bit of $\bar{y}$ is 1, all parties proceed. Otherwise, all parties terminate with output `Fail`.
3: Each party P_i checks whether $\bar{y} = y_i$. If true, P_i sets $b_i = 1$; otherwise, he sets $b_i = 0$. Then all parties invoke another $\mathcal{F}_{\mathsf{ba}}$ to agree on whether the output is valid (meaning that the output comes from an honest party). Each party P_i sends b_i to $\mathcal{F}_{\mathsf{ba}}$.

4: Upon receiving b from $\mathcal{F}_{\mathsf{ba}}$, if $b = 0$, all parties terminate with output `Fail`. Otherwise, each party sends `Public-Recon` to $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ invoked by the first $t+1$ dealer in $\mathcal{D}$ and therefore learns secret $r_1, \ldots, r_{t+1}$. Then all parties locally compute $r = r_1 + \cdots + r_{t+1}$ and terminate with output $y = \bar{y} - r$.

Cost Analysis. For the communication costs of $\Pi_{\texttt{main-Fair}}$:

- In the offline phase, all parties invoke $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$ to prepare $|C|$ random Beaver triples, which requires $\mathcal{O}(|C| \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits.
- In the input phase, all parties invoke n instances of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ to shares their inputs and an instance of $\mathcal{F}_{\mathsf{acs}}$ to agree on a set, let C_I, C_O be the input and output size, that requires $\mathcal{O}((C_I + C_O) \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits.
- In the computation phase, for each layer, all parties invoke $\Pi_{\texttt{pubRec-Weak}}$ to do public reconstruction. Assuming that there are C_k multiplication gates in the k-th layer, it requires $\mathcal{O}(C_k \cdot n + n^2)$ field elements plus $\mathcal{O}(n^2)$ bits. Then for all $k \in [D]$, where D is the depth of the circuit and $\sum_{k=1}^{D} C_k = |C|$, the whole communication complexity is $\mathcal{O}(|C| \cdot n + D \cdot n^2)$ field elements plus $\mathcal{O}(n^2)$ bits. Note that the overhead of $\mathcal{O}(n^2)$ bits is the cost of all parties sending $\perp$ to each other, each party will only send $\perp$ to all parties once and this term is independent of the circuit depth.
- In the output phase, all parties send shares of $[y + r]_t$ to each other, which requires $\mathcal{O}(C_O \cdot n^2)$ field elements.
- In the termination phase, all parties invoke two instances of $\mathcal{F}_{\mathsf{ba}}$ to agree on the output. Here we require a binary BA and a multi-valued BA, let C_O be the output size, that requires $\mathcal{O}(C_O \cdot n)$ field elements plus $\mathcal{O}(\kappa \cdot n^2 \log n + n^3)$ bits in total. For the reconstruction of all masks, it requires $\mathcal{O}(C_O \cdot n^3)$ field elements.

Therefore, the whole communication complexity of $\Pi_{\texttt{main-Fair}}$ is $\mathcal{O}((|C| + C_I) \cdot n + D \cdot n^2 + C_O \cdot n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits.

For the computation costs of $\Pi_{\texttt{main-Fair}}$:

- For $\Pi_{\mathsf{ACSS\text{-}ab}}$: each dealer invokes $\mathcal{O}(n)$ hash to compute the commitments, each party performs $\mathcal{O}(1)$ hash to do verification. Therefore, for all dealers, each party requires $\mathcal{O}(n)$ hash.
- We require $\mathcal{O}(1)$ instance of $\mathcal{F}_{\mathsf{ba}}$ and Π_{acs}, which requires $\mathcal{O}(n^3)$ hash in total.

Therefore, the whole computation costs of $\Pi_{\texttt{main-Fair}}$ are $\mathcal{O}(n^3)$ hash per party.

Lemma 9. *Protocol $\Pi_{\texttt{main-Fair}}$ securely computes $\mathcal{F}_{\mathsf{AMPC\text{-}Fair}}$ in the $\{\mathcal{F}_{\mathsf{acs}}, \mathcal{F}_{\mathsf{ba}}, \mathcal{F}_{\mathsf{ACSS\text{-}Abort}}, \mathcal{F}_{\mathsf{pubRec\text{-}Weak}}, \mathcal{F}_{\mathsf{triple\text{-}Weak}}\}$-hybrid model against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t < n/3$ parties.*

Proof. **Termination.** We first show that all honest parties will eventually terminate the protocol $\Pi_{\texttt{main-Fair}}$.

- In the offline phase, all parties are guaranteed to terminate $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$.
- In the input phase, the $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ invoked by honest dealers will eventually terminate. Since there are at least $n - t = 2t + 1$ honest parties, by the termination property of $\mathcal{F}_{\mathsf{acs}}$ when the agreement set is $2t + 1$, all parties will eventually agree on the set $\mathcal{D}$. Each party will also eventually receive his output from $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ invoked by all dealers in set $\mathcal{D}$.
- In the computation phase, all parties are guaranteed to finish $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$, which is the only interactive step.
- In the output phase, each honest party will send his share of $[y]_t$ or $\perp$ to all parties. Since there are at least $2t + 1$ honest parties, then each party is guaranteed to receive $2t + 1$ messages. If he first receives $2t + 1$ shares, he can check whether these shares lie on a degree-t polynomial and decide whether output $\perp$. If he first receives $\perp$, he will output $\perp$.
- In the termination phase, each party will either use $1|y'$ or zero string as his input for the first $\mathcal{F}_{\mathsf{ba}}$. When all honest parties have inputs, by the termination property of $\mathcal{F}_{\mathsf{ba}}$, each party will eventually agree on the same output $\bar{y}$. Then each party locally checks whether $\bar{y}$ equals his input for the first $\mathcal{F}_{\mathsf{ba}}$ and decides whether the input is 1 or 0 for the second $\mathcal{F}_{\mathsf{ba}}$. Similarly, the second $\mathcal{F}_{\mathsf{ba}}$ will eventually finish and each party can get the output b. If $b = 0$, all parties will terminate with output $\perp$. Otherwise, all parties send requests to $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ invoked by the first $t + 1$ successful dealers in $\mathcal{D}$ and therefore eventually learn the secret $r_1, \ldots, r_{t+1}$. Then all parties can compute the output y accordingly.

Security. Finally, we show that the protocol $\Pi_{\texttt{main-Fair}}$ securely computes $\mathcal{F}_{\mathsf{AMPC\text{-}Fair}}$. We start with constructing the ideal adversary $\mathcal{S}$ as follows.

Simulator $\mathcal{S}$

Denote $|C|$ as the number of Beaver triples. Let $\mathcal{C}orr$ denote the set of corrupted parties, then $|\mathcal{C}orr| = t' \leq t$. Let $\mathcal{C}orr'$ be the set of all corrupted parties together with the first $t - t'$ honest parties, then $|\mathcal{C}orr'| = t$

1: In the offline phase, during the simulation of $\mathcal{F}_{\mathsf{triple\text{-}Weak}}$, $\mathcal{S}$ receives shares of corrupted parties and learns which honest party's output is `Fail`. For each honest party in $\mathcal{C}orr' \setminus \mathcal{C}orr$, $\mathcal{S}$ samples random values as his shares of Beaver triples.

2: In the input phase, $\mathcal{S}$ simulates each $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ as follows:
- For each honest party, for all parties in $\mathcal{C}orr'$, $\mathcal{S}$ randomly samples values as their shares. Then $\mathcal{S}$ sends corrupted parties' shares to parties in $\mathcal{C}orr$ on behalf of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$.
- For each corrupted party, $\mathcal{S}$ waits to receive degree-t Shamir secret sharings from him and honestly follows $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$. For each honest party in $\mathcal{C}orr' \setminus \mathcal{C}orr$, $\mathcal{S}$ computes his input shares.

Then $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{acs}}$ and learn a set $\mathcal{D}$ of size $2t+1$. For each honest party, if his output of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ for any dealer in $\mathcal{D}$ is `abort`, $\mathcal{S}$ sets this honest party's output as $\perp$.

3: In the computation phase:
- For each addition gate, $\mathcal{S}$ follows the protocol and computes the shares of parties in $\mathcal{C}orr'$.
- For each multiplication gate, $\mathcal{S}$ follows the protocol and computes the shares of $[x+a]_t, [y+b]_t$ of parties in $\mathcal{C}orr'$. Then $\mathcal{S}$ randomly samples two degree-t Shamir secret sharings as $[x+a]_t, [y+b]_t$ based on the shares of parties in $\mathcal{C}orr'$. Finally, $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ and learns which honest party's output is $\perp$. After terminating $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$, $\mathcal{S}$ follows the protocol to compute shares of $[z]_t$ for each party in $\mathcal{C}orr'$.

4: In the output phase, for the first $t+1$ successful dealer $P_i \in \mathcal{D}$ and all parties in $\mathcal{C}orr'$, $\mathcal{S}$ first computes their shares of $[r]_t = [r_1]_t + \cdots + [r_{t+1}]_t$ and $[\bar{y}]_t = [y]_t + [r]_t$, then randomly samples the whole $[\bar{y}]_t$ based on shares of parties in $\mathcal{C}orr'$. Finally, for each honest party not in $\mathcal{C}orr' \setminus \mathcal{C}orr$, $\mathcal{S}$ computes his shares of $[\bar{y}]_t$ and replaces it with $\perp$ if this honest party's output is $\perp$. Then $\mathcal{S}$ sends it to all corrupted parties on behalf of each honest party. $\mathcal{S}$ waits to receive messages from all parties, for each honest party:
- If first receive $2t+1$ shares of $[\bar{y}]_t$, check whether these shares lie on a degree-t polynomial. If true, $\mathcal{S}$ continues to do the simulation of this party. Otherwise, $\mathcal{S}$ sets this honest party's output as $\perp$.
- If first receive $\perp$, $\mathcal{S}$ sets this honest party's output as $\perp$.

5: In the termination phase, $\mathcal{S}$ simulates these two $\mathcal{F}_{\mathsf{ba}}$ and follows the protocol to determine all honest parties' output is $\bar{y}$ or `Fail`. If the output is `Fail`, $\mathcal{S}$ sends `Fail` to $\mathcal{F}_{\mathsf{AMPC\text{-}Fair}}$. Otherwise, if the output is $\bar{y}$, $\mathcal{S}$ sends the inputs of corrupted parties and the set $\mathcal{D}$ to $\mathcal{F}_{\mathsf{AMPC\text{-}Fair}}$ and receives the output y. Let $\mathcal{D}'$ be the set of the first $t+1$ dealers in $\mathcal{D}'$ and $\mathcal{C}orr^*$ be the set of corrupted parties in $\mathcal{D}'$. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ invoked by dealers in $\mathcal{D}$ as follows:
- For each dealer $P_i \in \mathcal{C}orr^*$, $\mathcal{S}$ sends $[r_i]_t$ to corrupted parties.
- For all dealer $P_i \in \mathcal{D}' \setminus \mathcal{C}orr^*$, $\mathcal{S}$ randomly samples r_i such that $\bar{y} - y = \sum_{i \in \mathcal{D}' \setminus \mathcal{C}orr^*} r_i + \sum_{j \in \mathcal{C}orr^*} r_j$. Then $\mathcal{S}$ computes $[r_i]_t$ given the shares of parties in $\mathcal{C}orr'$ and the secret r_i and sends $[r_i]_t$ to all corrupted parties.

When each honest party receives $[r_1]_t, \ldots, [r_{t+1}]_t$, $\mathcal{S}$ delivers the output from $\mathcal{F}_{\mathsf{AMPC\text{-}Fair}}$ to this party.

6: $\mathcal{S}$ outputs the views of $\mathcal{A}$.

We show that the output in the ideal world is identically distributed to that in the real world by using the following hybrid arguments.

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

$\mathbf{Hyb}_1$: In this hybrid, when $\mathcal{S}$ receives degree-t Shamir sharings from a corrupted dealer, $\mathcal{S}$ records the first $t - t'$ honest parties' shares. $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same distribution.

$\mathbf{Hyb}_2$: In this hybrid, in the input phase, for each honest dealer, after randomly sampling shares of parties in $\mathcal{C}orr'$, $\mathcal{S}$ delays the generation of shares of the rest of honest parties until the set $\mathcal{D}$ is determined. Since these honest parties' shares are not used in the input phase, $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ have the same distribution.

$\mathbf{Hyb}_3$: In this hybrid, in the input phase, for each honest dealer not in $\mathcal{D}$, $\mathcal{S}$ does not generate shares of honest parties. Since these honest parties' sharings are never used, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ have the same distribution.

$\mathbf{Hyb}_4$: In this hybrid, in the computation phase, for every multiplication gate and each honest party not in $\mathcal{C}orr' \setminus \mathcal{C}orr$, $\mathcal{S}$ delays the generation of his shares of $[a]_t, [b]_t, [c]_t$. $\mathcal{S}$ first randomly samples two

degree-t Shamir secret sharings $[x+a]_t, [y+b]_t$ based on shares of parties in $\mathcal{C}orr'$, then computes the rest of honest parties' shares of $[a]_t = [x+a]_t - [x]_t, [b]_t = [y+b]_t - [y]_t$. In $\mathbf{Hyb}_3$ we first sample random $[a]_t, [b]_t$ then computing $[x+a]_t, [y+b]_t$, while in $\mathbf{Hyb}_4$ we change the order and first sample random $[x+a]_t, [y+b]_t$ then computing $[a]_t, [b]_t$, which makes no difference.

Then $\mathcal{S}$ randomly samples a degree-t Shamir secret sharing $[z]_t$ based on shares of parties in $\mathcal{C}orr'$ and the secret $z = x \cdot y$. Finally, $\mathcal{S}$ computes $[c]_t = [z_t] - (x+a)(y+b) + (y+b)[a]_t + (x+a)[b]_t$. Note that $c = a \cdot b$ and $[c]_t$ is a random degree-t Shamir secret sharing given the shares of corrupted parties. Thus, $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ have the same distribution.

$\mathbf{Hyb}_5$: In this hybrid, in the computation phase, for every multiplication gate, $\mathcal{S}$ follows the protocol to compute shares of $[z]_t$ of parties in $\mathcal{C}orr'$. Then $\mathcal{S}$ determines the rest of honest parties' shares of $[z]_t$ by using the secret $z = x \cdot y$ and shares of parties in $\mathcal{C}orr'$. Since a degree-t Shamir sharing is fully determined by these t shares and secret, this does not change the distribution of the shares of honest parties. $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ have the same distribution.

$\mathbf{Hyb}_6$: In this hybrid, in the computation phase, for each honest party not in $\mathcal{C}orr' \setminus \mathcal{C}orr$, $\mathcal{S}$ does not generate their shares of $([a]_t, [b]_t, [c]_t)$ since they are never used. $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ have the same distribution.

$\mathbf{Hyb}_7$: In this hybrid, in the output phase, let $\mathcal{C}orr^* = \{P_{j_1}, \ldots, P_{j_L}\}$ and $\mathcal{D}' \backslash \mathcal{C}orr^* = \{P_{i_1}, \ldots, P_{i_{L'}}\}$, $\mathcal{S}$ first randomly samples $[r]_t, [r_{i_2}]_t, \ldots, [r_{i_{L'}]_t}$ based on shares of corrupted parties, then compute $[r_{i_1}]_t = [r]_t - \sum_{k=2}^{L'} [r_{i_k}]_t - \sum_{k=1}^{L} [r_{j_k}]_t$. In both $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_7$, $\mathcal{S}$ samples $[r_{i_2}]_t, \ldots, [r_{i_{L'}}]_t$ in the same way and $[r]_t, [r_{i_1}]_t$ are both uniformly distributed, then $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ have the same distribution.

$\mathbf{Hyb}_8$: In this hybrid, in the output phase, we change the generation of $[r]_t$. For each party in $\mathcal{C}orr'$, $\mathcal{S}$ first computes their shares of $[y+r]_t = [y]_t + [r]_t$, then randomly samples the whole $[y+r]_t$ based on shares of parties in $\mathcal{C}orr'$. Finally, $\mathcal{S}$ computes $[r]_t = [y+r]_t - [y]_t$. $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ have the same distribution.

$\mathbf{Hyb}_9$: In this hybrid, we delay the generation of $[y]_t$ until all parties agree on the output $\bar{y}$ in termination phase. During the termination phase, if all parties agree on `Fail`, then all parties will not ask $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ to distribute the secrets $r_1, \ldots, r_{t+1}$, then these secrets sampled by honest dealer P_i are never used and there is no need to use $[y]_t$ to compute $[r]_t$. Thus, $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ have the same distribution.

$\mathbf{Hyb}_{10}$: In this hybrid, in the termination phase, when all parties agree on the output $\bar{y}$, $\mathcal{S}$ uses inputs for parties in $\mathcal{D}$ to compute the output y. Then $\mathcal{S}$ computes the whole $[y]_t$ based on secret y and shares of parties in $\mathcal{C}orr'$. Finally $\mathcal{S}$ computes $[r]_t = [\bar{y}]_t - [y]_t$. With $[r]_t$, $\mathcal{S}$ gets all $[r_i]_t$ distributed by dealers in $\mathcal{D}'$ and can send $[r_i]_t$ to all parties on behalf of $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$. Each party can locally computes output $y = \bar{y} - r_1 - \cdots - r_{t+1}$. $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ have the same distribution.

$\mathbf{Hyb}_{11}$: In this hybrid, $\mathcal{S}$ no longer computes the whole sharings in the input phase and computation phase except the shares of parties in $\mathcal{C}orr'$. Note that they are not needed in producing the output in $\mathbf{Hyb}_{10}$. $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ have the same distribution.

$\mathbf{Hyb}_{12}$: In this hybrid, if $\mathcal{S}$ learns that all parties agree on `Fail` during the termination phase, $\mathcal{S}$ sends `Fail` to $\mathcal{F}_{\mathsf{AMPC\text{-}Fair}}$. Otherwise, $\mathcal{S}$ sends corrupted parties' inputs and the set $\mathcal{D}$ to $\mathcal{F}_{\mathsf{AMPC\text{-}Fair}}$ to learn the output y, and $\mathcal{S}$ delivers the output y to all parties. Since $\mathcal{F}_{\mathsf{AMPC\text{-}Fair}}$ computes the function in the same way as $\mathcal{S}$ does in $\mathbf{Hyb}_{11}$, $\mathbf{Hyb}_{12}$ and $\mathbf{Hyb}_{11}$ have the same distribution.

Note that $\mathbf{Hyb}_{12}$ corresponds to the ideal world, then $\Pi_{\texttt{main-Fair}}$ securely computes $\mathcal{F}_{\mathsf{AMPC\text{-}Fair}}$.

D Construction and Proofs of Malicious Security with GOD AMPC

D.1 Construction of Public Reconstruction and Agreement

Batch Reconstruction. Given a parameter L, we batch L secrets for public reconstruction to achieve amortized linear communication costs.

Protocol $\Pi_{\texttt{BatchPubRec}}$

For each party P_i:

1: Initialize a vector $\boldsymbol{m}_i$ of size L and divide it into $L/(t+1)$ sub-vectors, each of size $t+1$, denoted by $\boldsymbol{m}_i = (\boldsymbol{m}_{i,1}, \ldots, \boldsymbol{m}_{i,L/(t+1)})$. Divide $[x^{(1)}]_t, \ldots, [x^{(L)}]_t$ into $L/(t+1)$ groups, each of size $t+1$. For all $k \in [L/(t+1)]$, P_i does the following.

(1). Let $[s^{(0)}]_t, \ldots, [s^{(t)}]_t$ denote the k-th group of degree-t Shamir secret sharings. Define $f(X) = s^{(0)} + s^{(1)} \cdot X + \cdots + s^{(t)} \cdot X^t$. P_i sends his share of $[f(\alpha_j)]_t = [s^{(0)}]_t + [s^{(1)}]_t \cdot \alpha_j + \cdots + [s^{(t)}]_t \cdot \alpha_j^t$ to each P_j.

(2). To reconstruct $f(\alpha_i)$, P_i waits to receive messages:
- Upon receiving new shares of $[f(\alpha_i)]_t$, P_i uses online error correction on all received shares of $[f(\alpha_i)]_t$ to reconstruct $f(\alpha_i)$. If succeeds, P_i sends $f(\alpha_i)$ to all parties and moves to Step 1.(3). Otherwise, P_i keeps waiting for more messages.
- Upon receiving $(\texttt{Proof}, D)$ from a party and $(\texttt{Corrupt}, D)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$, P_i sets $\boldsymbol{m}_i = \perp$ and moves to Step 2.

(3). To reconstruct $f(X)$, P_i waits to receive messages from all parties:
- Upon receiving $f(\alpha_j)$ from P_j, P_i uses online error correction on all received $f(\alpha_j)$ to reconstruct $f(X)$. If succeeds, P_i sets the k-th sub-vector $\boldsymbol{m}_{i,k}$ as $(s^{(0)}, \ldots, s^{(t+1)})$ (the coefficients of $f(x)$). Otherwise, P_i keeps waiting for more messages.
- Upon receiving $(\texttt{Proof}, D)$ from a party and $(\texttt{Corrupt}, D)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$, P_i sets $\boldsymbol{m}_i = \perp$ and moves to Step 2.

2: Output $\boldsymbol{m}_i$.

Agreement on Public Reconstruction Result. The construction of the agreement step is as follows. Actually, we invoke two $\mathcal{F}_{\mathsf{ba}}$ to realize the non-intrusion BA protocol [MR17] with linear costs.

Protocol $\Pi_{\texttt{Agreement}}$

1: All parties invoke $\mathcal{F}_{\mathsf{ba}}$ and party P_i uses m_i as his input. Upon receiving result m from $\mathcal{F}_{\mathsf{ba}}$, P_i checks whether $m = m_i$. If true, he sets $b_i = 1$. Otherwise, he sets $b_i = 0$.

2: All parties invoke $\mathcal{F}_{\mathsf{ba}}$ and party P_i uses b_i as his input. Upon receiving the result b from $\mathcal{F}_{\mathsf{ba}}$, if $b = 1$, P_i outputs m. Otherwise, P_i uses $\perp$ as his output.

3: When the output is not $\perp$, all parties terminate. Otherwise, all parties follow the steps to agree on the identity of a corrupted party.

(1). All parties invoke n instances of $\mathcal{F}_{\mathsf{ba}}$. Upon receiving $(\texttt{Proof}, P_i)$ from a party and $(\texttt{Corrupt}, P_i)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$, each party sets his input of the i-th $\mathcal{F}_{\mathsf{ba}}$ as 1.

(2). If a party receives output 1 from $\mathcal{F}_{\mathsf{ba}}$, he sets his input to 0 for the rest of $\mathcal{F}_{\mathsf{ba}}$ (unless already set to 1) and waits for outputs from all instances of $\mathcal{F}_{\mathsf{ba}}$.

(3). Let j be the smallest index such that the j-th $\mathcal{F}_{\mathsf{ba}}$ outputs 1. All parties output the identity of P_j.

D.2 Construction and Analysis of $\Pi_{\texttt{SubCktEval}}$

We give the construction of $\Pi_{\texttt{SubCktEval}}$ as follows.

Protocol $\Pi_{\texttt{SubCktEval}}$

1: **Check Shares.**
Given a circuit C' of depth D' with $|C'|$ multiplication gates and C'_O output gates, all parties hold degree-t Shamir sharings of the inputs of C' and $|C'|$ random Beaver triples. Each party P_i checks:
- Whether he has all shares of the Beaver triples,
- And whether he has all shares of the input degree-t Shamir sharings for the circuit C'.

If true, he moves to Step 2. Otherwise, he sets the output $\boldsymbol{m}_i = \perp$, reliably broadcasts $(\texttt{Proof}, D)$ to all parties and sends `Broadcast-Proof` to $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by an active corrupted dealer D know by him, and moves to Step 4.

2: **Circuit Evaluation.**
From $k = 1$ to D', for the k-th layer in the circuit C':
- For every addition gate with input sharings $[x]_t, [y]_t$, locally compute

$$[z]_t = [x]_t + [y]_t.$$

- Let L be the number of multiplication gates in the k-th layer. Suppose the input degree-t Shamir sharings are denoted by $([x_i]_t, [y_i]_t)_{i=1}^L$. Let $([a_i]_t, [b_i]_t, [c_i]_t)_{i=1}^L$ denote the random Beaver triples assigned to these L gates. Each party P_j executes $\Pi_{\texttt{BatchPubRec}}$ with his shares of $([x_i + a_i]_t, [y_i + b_i]_t)_{i=1}^L$, and gets output $\boldsymbol{m}_j^{(k)}$ after terminating $\Pi_{\texttt{BatchPubRec}}$.
  - If $\boldsymbol{m}_j^{(k)} = \perp$, P_j moves to Step 4.

- Otherwise, for all $i \in [L]$, P_j parses $\boldsymbol{m}_j^{(k)}$ to get $\{x_i + a_i, y_i + b_i\}_{i=1}^{L}$ and locally compute

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

3: **Output Reconstruction.**
For the output layer in the circuit C' (if have):
- Each party P_j executes $\Pi_{\texttt{BatchPubRec}}$ with his output sharings and gets output $\boldsymbol{m}_j^{(D'+1)}$ after terminating $\Pi_{\texttt{BatchPubRec}}$.

4: **Agreement on Output.**
Each party P_j checks whether $\exists k \in [D'+1], \boldsymbol{m}_j^{(k)} = \perp$. If true, P_j sets $\boldsymbol{m}_j = \perp$. Otherwise, P_j sets $\boldsymbol{m}_j = (\boldsymbol{m}_j^{(1)}, \ldots, \boldsymbol{m}_j^{(D'+1)})$ and executes $\Pi_{\texttt{Agreement}}$ with input $\boldsymbol{m}_j$ and gets the output after terminating $\Pi_{\texttt{Agreement}}$:
- If the output is the identity of a corrupted dealer, all parties output this identity.
- Otherwise, all parties terminate $\Pi_{\texttt{SubCktEval}}$ with the output.

Termination of $\Pi_{\texttt{SubCktEval}}$. Here we prove that each instance of $\Pi_{\texttt{SubCktEval}}$ will eventually terminate, and all parties will either agree on the output of this sub-circuit or the identity of a corrupted party.

- In Step 1, each party with his degree-t sharings just does some local computation, and each party who learns a corrupted dealer will invoke $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to let all parties learn a corrupted dealer.
- In Step 2, for each layer in the sub-circuit, the only thing that all parties need to interact with is using $\Pi_{\texttt{BatchPubRec}}$ to do public reconstruction. During the $\Pi_{\texttt{BatchPubRec}}$, for party P_i:
  - For the reconstruction of $f(\alpha_i)$, if all honest parties have sent their shares of $[f(\alpha_i)]_t$ to P_i, then P_i will eventually reconstruct $f(\alpha_i)$ and proceed. If at least one honest party requests $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$, P_i can eventually learn a corrupted party from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ and proceed. Note that in this step, the corrupt party may help P_i to reconstruct $f(\alpha_i)$ when not all honest parties have sent their shares of $[f(\alpha_i)]_t$ to P_i.
  - For the reconstruct of $f(X)$, this is similar to the reconstruction of $f(\alpha_i)$ and P_i will eventually get $f(X)$ or learn a corrupted party.

  Therefore, all honest parties will eventually terminate Step 2.
- In Step 3, all parties invoke $\Pi_{\texttt{BatchPubRec}}$ to reconstruct the output of the sub-circuit, this is similar to Step 2 and all parties will eventually terminate Step 3.
- In Step 4, since each party will eventually terminate Step 3, each party P_j will either set $\boldsymbol{m}_j$ to $\perp$ or a meaningful vector. Then all parties invoke $\Pi_{\texttt{Agreement}}$:
  - For the first $\mathcal{F}_{\mathsf{ba}}$, since all parties have inputs, then $\mathcal{F}_{\mathsf{ba}}$ will eventually terminate and send output m to all parties.
  - Upon receiving m from $\mathcal{F}_{\mathsf{ba}}$, each party will locally check whether m equals to his input for the first $\mathcal{F}_{\mathsf{ba}}$. Therefore, each party honest P_i will eventually get his input b_i for the second $\mathcal{F}_{\mathsf{ba}}$. Then all honest parties can also terminate the second $\mathcal{F}_{\mathsf{ba}}$ with output b. Then each honest party can follow the protocol to determine his output.
  - When the output is $\perp$, all parties invoke n instances of $\mathcal{F}_{\mathsf{ba}}$ to agree on a corrupted party. In this case, we can ensure that at least one honest party's input for the first $\mathcal{F}_{\mathsf{ba}}$ is $\perp$. Otherwise, the output of the first $\mathcal{F}_{\mathsf{ba}}$ should equal all honest parties' common inputs and therefore the second $\mathcal{F}_{\mathsf{ba}}$ will also output $b = 1$. If one honest party's input for the first $\mathcal{F}_{\mathsf{ba}}$ is $\perp$, he must receive the identity of a corrupted party from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ or request $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to let all parties learn a corrupted party. Then all honest parties will eventually receive the identity of this corrupted party from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. Therefore, all parties will set their input for the corresponding $\mathcal{F}_{\mathsf{ba}}$ as 1, and that $\mathcal{F}_{\mathsf{ba}}$ will eventually terminate with output 1. For each party who first terminates any instances of $\mathcal{F}_{\mathsf{ba}}$ with 1, he will set his input for the rest of $\mathcal{F}_{\mathsf{ba}}$ as 0. Then all instances of $\mathcal{F}_{\mathsf{ba}}$ will eventually terminate and all parties will agree on the $\mathcal{F}_{\mathsf{ba}}$ with the smallest index and output 1. Note that if the output of $\mathcal{F}_{\mathsf{ba}}$ is 1, then at least one honest party's input is 1 and this honest party learns the corresponding dealer is corrupted.

  Therefore, all honest parties will eventually terminate Step 4 with the output of the sub-circuit or the identity of a corrupted party.

Costs Analysis. The communication costs include:

– In Step 1, after we instantiate $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ with our construction, it requires $\mathcal{O}(n)$ instances of RBC of ACSS proofs. The overhead of RBA is $\mathcal{O}(\kappa \cdot n^3)$ bits.

– In Step 2 and Step 3, $D'+1$ instances of $\Pi_{\texttt{BatchPubRec}}$, which requires $\mathcal{O}((|C'|+C'_O)) \cdot n + D' \cdot n^2)$ field elements.

– In Step 4, one instance of $\mathcal{F}_{\mathsf{ba}}$ on $(|C'|+C'_O)$ field elements, which requires $\mathcal{O}((|C'|+C'_O) \cdot n)$ field elements plus $\mathcal{O}(\kappa \cdot n^2 \log(n) + n^3)$ bits. $n+1$ instances of $\mathcal{F}_{\mathsf{ba}}$ on 1 bit, which requires $\mathcal{O}(n^4)$ bits.

In addition to the cost of broadcasting $\mathcal{O}(n)$ ACSS proof, the total communication costs is $\mathcal{O}((|C'|+C'_O) \cdot n + D' \cdot n^2)$ field elements plus $\mathcal{O}(\kappa \cdot n^3 + n^4)$ bits.

D.3 Construction of $\Pi_{\texttt{randSh}}$ and $\Pi_{\texttt{randShareZero}}$

The construction of $\Pi_{\texttt{randSh}}$ is as follows which is the same as $\Pi_{\texttt{randSh-Weak}}$ except that we replace $\mathcal{F}_{\mathsf{ACSS\text{-}Abort}}$ by $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. For $\Pi_{\texttt{randShareZero}}$, we let each party record their column polynomials during the $\Pi_{\texttt{Sh2tZero}}$, then they locally divide them into $n(n+t)$ groups and do extraction for each group. We require $n(n+t)$ groups because we use the party elimination framework which may fail for t times and there are n kings in each segment.

Protocol $\Pi_{\texttt{randSh}}$

Let N be the number of degree-t random Shamir sharings to be prepared.

1: Each party P_i samples $N' = N/(t+1)$ random degree-t Shamir secret sharings $[s_1^{(i)}]_t, \ldots, [s_{N'}^{(i)}]_t$. Then P_i acts as the dealer D and invokes $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to distribute the shares to all parties.

2: Each party P_i sets the property Q as P_i terminating $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ when P_j acts as a dealer. Then all parties invoke $\mathcal{F}_{\mathsf{acs}}$ with property Q to agree on a set $\mathcal{D}$ of successful dealers with size $|\mathcal{D}| = 2t+1$.

3: All parties agree on (the inverse of) a Vandermonde matrix $\boldsymbol{M}$ of size $(t+1) \times (2t+1)$. For all $\ell \in \{1, \ldots, N'\}$, all parties locally compute

$$([r_{\ell,1}]_t, \ldots, [r_{\ell,t+1}]_t) = \boldsymbol{M} \cdot ([s_\ell^{(i)}]_t)_{i \in \mathcal{D}}.$$

Finally, all parties output $\{[r_{\ell,i}]_t\}_{\ell \in \{1,\ldots,N'\}, i \in \{1,\ldots,t+1\}}$.

Protocol $\Pi_{\texttt{randShareZero}}$

Let N be the number of degree-$2t$ Shamir sharings of 0 to be prepared, $\beta_1, \ldots, \beta_{(t+1)/2}$ be distinct field elements.

1: Each party P_i samples $N' = N/(t+1)$ random degree-$2t$ Shamir sharings of 0. Then P_i acts as the dealer D and runs $\Pi_{\texttt{Sh2tZero}}$ to distribute the sharings to all parties.

2: Each party P_i sets the property Q as P_i terminating $\Pi_{\texttt{Sh2tZero}}$ when P_j acts as a dealer, then all parties invoke $\mathcal{F}_{\mathsf{acs}}$ with property Q to agree on a set $\mathcal{D}$ of size $2t+1$ which includes successful dealers.

3: Assume that for each dealer $P_i \in \mathcal{D}$, each party P_j's output of $\Pi_{\texttt{Sh2tZero}}$ are degree-$(t+(t-1)/2)$ column polynomials $\{g_{\ell,j}^{(i)}\}_{\ell \in [2N'/(t+1)]}$. For each dealer P_i, P_j locally divide these $2N'/(t+1)$ column polynomials into $n(n+t)$ groups, each of size $2N'/(n(n+t)(t+1))$. For each group, each party evaluates the column polynomial at $\beta_1, \ldots, \beta_{(t+1)/2}$ to get degree-$2t$ Shamir sharings of zero $[o_1^{(i)}]_{2t}, \ldots, [o_{N'/(n(n+t))}^{(i)}]_{2t}$.

4: All parties agree on (the inverse of) a Vandermonde matrix $\boldsymbol{M}$ of size $(t+1) \times (2t+1)$. For each group and for all $\ell' \in [N'/(n(n+t))]$, all parties locally compute

$$([o_{\ell',1}^{(i)}]_{2t}, \ldots, [o_{\ell',t+1}^{(i)}]_{2t}) = \boldsymbol{M} \cdot ([o_{\ell'}^{(i)}]_{2t})_{i \in \mathcal{D}}.$$

Finally, all parties output $\{[o_{\ell',k}]_{2t}\}_{\ell' \in [N'/(n(n+t))], k \in [t+1]}$ for all $n(n+t)$ groups.

D.4 Construction of $\Pi_{\texttt{tripleExt}}$

Process $\Pi_{\texttt{tripleExt}}$

When all parties invoke $\Pi_{\texttt{SubCktEval}}$ to do public reconstruction but the output is the identity of an active corrupted dealer. All parties send `Public-Recon` to the $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by this corrupted dealer and wait to receive the secrets from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. Then all parties locally update their shares by these secrets and invoke $\Pi_{\texttt{SubCktEval}}$ again until they succeed in reconstruction.

1: **Distribution:**
Let $N' = 4N/(t+1), L = 2t + (t-1)/2$, all parties agree on distinct field elements $\beta_1, \ldots, \beta_{(L+1)/2-t}, \alpha_0, \ldots, \alpha_{2N'}$.
Each party P_i samples two random degree-N' polynomials f_i, g_i and computes $h_i = f_i \cdot g_i$. Then P_i samples $4N'+3$ random degree-t Shamir secret sharings:

$$\{[f_i(\alpha_\ell)]_t\}_{\ell=0}^{N'}, \{[g_i(\alpha_\ell)]_t\}_{\ell=0}^{N'}, \{[h_i(\alpha_\ell)]_t\}_{\ell=0}^{2N'}$$

Finally, P_i acts as a dealer and invokes $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to distribute these $4N'+3$ degree-t Shamir sharings.

2: **Determine the Set of Successful Dealers:**
All parties set the property Q as terminating $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by each dealer, then they invoke the modified Π_{acs}^{Q} to agree on a set $\mathcal{D}$ of size L which contains the successful dealers.

3: **Verify Triples:**
(1). Upon agreeing on set $\mathcal{D}$, for each P_i in $\mathcal{D}$, each party P_j waits for the termination of $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ where P_i acts as the dealer. Then P_j sends request to $\mathcal{F}_{\mathsf{coin}}$.
(2.) Upon receiving r from $\mathcal{F}_{\mathsf{coin}}$, if $r \in \{\alpha_1, \ldots, \alpha_{N'}\}$, all parties request $\mathcal{F}_{\mathsf{coin}}$ again. Otherwise, all parties locally compute $\{[f_i(r)]_t, [g_i(r)]_t, [h_i(r)]_t\}_{P_i \in \mathcal{D}}$ and invoke $\Pi_{\texttt{SubCktEval}}$ with input sharings $[f_i(r)]_t, [g_i(r)]_t, [h_i(r)]_t$ to reconstruct $\{f_i(r), g_i(r), h_i(r)\}_{P_i \in \mathcal{D}}$.
(3). For each $P_i \in \mathcal{D}$, if $f_i(r) \cdot g_i(r) = h_i(r)$, all parties set $([a_\ell^{(i)}]_t, [b_\ell^{(i)}]_t, [c_\ell^{(i)}]_t)_{\ell=1}^{N'} = ([f_i(\alpha_\ell)]_t, [g_i(\alpha_\ell)]_t, [h_i(\alpha_\ell)]_t)_{\ell=1}^{N'}$. Otherwise, all parties set $([a_\ell^{(i)}]_t, [b_\ell^{(i)}]_t, [c_\ell^{(i)}]_t)_{\ell=1}^{N'}$ to be all-0 sharings.

4: **Extracting Random Triples:**
For all $k \in [N']$, pick the first unused Beaver triple from each dealer in $\mathcal{D}$ and denote them by $\{[a_i^{(k)}]_t, [b_i^{(k)}]_t, [c_i^{(k)}]_t\}_{i=1}^{L}$. Then execute the following steps to extract N random Beaver triples:
(1). For each $k \in [N']$, all parties set two polynomials of $f^{(k)}$, $g^{(k)}$ of degree $L' = \frac{L-1}{2}$ such that $[f^{(k)}(\alpha_i)]_t = [a_i^{(k)}]_t$ and $[g^{(k)}(\alpha_i)]_t = [b_i^{(k)}]_t$ for all $i \in [L'+1]$.
(2). From $i = L'+2$ to L, all parties do the following things:
1). For all $k \in [N']$, locally compute $[f^{(k)}(\alpha_i)]_t, [g^{(k)}(\alpha_i)]_t$.
2). Executing $\Pi_{\texttt{SubCktEval}}$ with input sharings $[f^{(k)}(\alpha_i) + a_i^{(k)}]_t, [g^{(k)}(\alpha_i) + b_i^{(k)}]_t$ to do public reconstruction. Upon terminating with output $\{f^{(k)}(\alpha_i) + a_i^{(k)}, g^{(k)}(\alpha_i) + b_i^{(k)}\}_{k=1}^{N'}$, locally compute:

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

Finally, set $i = i + 1$.
(3). For each $k \in [N']$, all parties set a polynomial $h^{(k)}$ of degree $L-1$ such that $[h^{(k)}(\alpha_i)]_t = [c_i^{(k)}]_t$ for all $i \in [L'+1]$ and $[h^{(k)}(\alpha_i)]_t = [f^{(k)}(\alpha_i) \cdot g^{(k)}(\alpha_i)]_t$ for all $i \in [L'+2, L]$.
(4). All parties output $([f^{(k)}(\beta_i)]_t, [g^{(k)}(\beta_i)]_t, [h^{(k)}(\beta_i)]_t)$ for all $i \in [(L+1)/2 - t], k \in [N']$.

Communication Cost of $\Pi_{\texttt{tripleExt}}$. The communication costs include:

- In Step 1, n instances of $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$, which requires $\mathcal{O}(N \cdot n + n^5)$ field elements plus $\mathcal{O}(\kappa \cdot n^5)$ bits.
- In Step 2, n instances of $\mathcal{F}_{\mathsf{ba}}$, which requires $\mathcal{O}(n^4)$ bits.
- In Step 3, one instance of $\mathcal{F}_{\mathsf{coin}}$, which requires $\mathcal{O}(n^3)$ bits.
- In Step 3 and 4, at most $t+1$ instances of $\Pi_{\texttt{SubCktEval}}$, in addition to the costs of ACSS proofs, it requires $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^4 + n^5)$ bits. For $\mathcal{O}(n^2)$ broadcast of ACSS proofs, it requires $\mathcal{O}(N \cdot n)$ field elements.

Therefore, the whole communication costs of $\Pi_{\texttt{tripleExt}}$ is $\mathcal{O}(N \cdot n + n^5)$ field elements plus $\mathcal{O}(\kappa \cdot n^5)$ bits.

D.5 Construction of Triples Verification

The verification method of triples is the same at $\Pi_{\texttt{tripleVerify-Weak}}$ introduced in 4.4. The communication complexity of $\Pi_{\texttt{tripleVerify}}$ equals the number of times $\Pi_{\texttt{SubCktEval}}$ is executed plus one instance of $\mathcal{F}_{\mathsf{coin}}$ for verifying $\mathcal{O}(N)$ Beaver triples.

Protocol $\Pi_{\texttt{tripleVerify}}$

When all parties invoke $\Pi_{\texttt{SubCktEval}}$ to do public reconstruction, they may fail to get the reconstruction result and output the identity of an active corrupted dealer. In this case, all parties will send `Public-Recon` to the $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by this corrupted dealer and wait to receive his secrets from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. Then all parties locally update their shares by these secrets and invoke $\Pi_{\texttt{SubCktEval}}$ again until they succeed in reconstructing.

1: **Build Polynomials:**
Let N be the number of Beaver triples that need to be verified and $N' = (N/(2t+1)-1)/2$, all parties agree on $2N'+1$ distinct field elements $\alpha_0, \ldots, \alpha_{2N'}$. Then all parties run the following steps:

(1). All parties have $2t+1$ groups of Beaver triples, for the k-th group, it contains $2N'+1$ Beaver triples and denoted it by $\{[a_i^{(k)}]_t, [b_i^{(k)}]_t, [c_i^{(k)}]_t\}_{i=0}^{2N'}$.

(2). All parties set two polynomials of $f^{(k)}$, $g^{(k)}$ of degree N' such that $[f^{(k)}(\alpha_i)]_t = [a_i^{(k)}]_t$ and $[g^{(k)}(\alpha_i)]_t = [b_i^{(k)}]_t$ for all $i \in [N'], k \in [2t+1]$.

(3). For all $i \in [N'+1, 2N'], k \in [2t+1]$, all parties locally compute $[f^{(k)}(\alpha_i)]_t$, $[g^{(k)}(\alpha_i)]_t$. Then they execute $\Pi_{\texttt{SubCktEval}}$ with input sharings $[f^{(k)}(\alpha_i)+a_i^{(k)}]_t, [g^{(k)}(\alpha_i)+b_i^{(k)}]_t$ to do public reconstruction, upon terminating with output $\{f^{(k)}(\alpha_i)+a_i^{(k)}, g^{(k)}(\alpha_i)+b_i^{(k)}\}_{i\in[N'+1,2N'],k\in[2t+1]}$, locally compute:

$$\begin{aligned}[f^{(k)}(\alpha_i)\cdot g^{(k)}(\alpha_i)]_t = (f^{(k)}(\alpha_i)+a_i^{(k)})\cdot(g^{(k)}(\alpha_i)+b_i^{(k)}) + [c_i^{(k)}]_t \\ - (f^{(k)}(\alpha_i)+a_i^{(k)})[b_i^{(k)}]_t - (g^{(k)}(\alpha_i)+b_i^{(k)})[a_i^{(k)}]_t\end{aligned}$$

(4). All parties set a polynomial $h^{(k)}$ of degree $2N$ such that $[h^{(k)}(\alpha_i)]_t = [c_i^{(k)}]_t$ for all $i \in [N']$ and $[h^{(k)}(\alpha_i)]_t = [f^{(k)}(\alpha_i)\cdot g^{(k)}(\alpha_i)]_t$ for all $i \in [N'+1, 2N'], k \in [2t+1]$.

2: **Verification Phase:**

– All parties send a request to $\mathcal{F}_{\mathsf{coin}}$. Upon receiving r from $\mathcal{F}_{\mathsf{coin}}$, if $r \in \{\alpha_1, \ldots, \alpha_{N'}\}$, he sets $v_i^{(k)} = 0$ for all $k \in [2t+1]$. Otherwise:

(1). For all $k \in [2t+1]$, each party locally compute his share of $([f^{(k)}(r)]_t, [g^{(k)}(r)]_t, [h^{(k)}(r)]_t)$. Then all parties execute $\Pi_{\texttt{SubCktEval}}$ with input sharings $\{([f^{(k)}(r)]_t, [g^{(k)}(r)]_t, [h^{(k)}(r)]_t)\}_{k=1}^{2t+1}$ to reconstruct the secrets $\{f^{(k)}(r), g^{(k)}(r), h^{(k)}(r)\}_{k=1}^{2t+1}$.

(2). Each party checks whether $h^{(k)}(r) = f^{(k)}(r)\cdot g^{(k)}(r)$ for each $k \in [2t+1]$. If true, he sets $v_i^{(k)} = 1$. Otherwise, he sets $v_i^{(k)} = 0$.

3: **Output Phase:**
For each party P_i and $k \in [2t+1]$, if $v_i^{(k)} = 1$, he outputs his shares of $\{[a_i^{(k)}]_t, [b_i^{(k)}]_t, [c_i^{(k)}]_t\}_{i=1}^{N'}$. If $v_i^{(k)} = 0$, P_i outputs `Fail`.

D.6 Construction of Detecting Corruptions

Protocol $\Pi_{\texttt{faultLoc}}$

Let $\mathcal{D}$ be the set generated during the $\Pi_{\texttt{randShareZero}}$, N be the number of Beaver triples prepared by each P_{king}.

Each Party P_j

All parties help P_{king} to detect the corrupted party.

1. For each dealer in $\mathcal{D}$, if P_i has accepted his column polynomials, send $\{g_\ell^{(i)}(y), \bar{g}_\ell^{(i)}(y)\}_{\ell\in[2N/(t+1)^2]}$ to P_{king}. Otherwise, send a `ShareProof` to P_{king}.
2. If P_j has his shares of $\{([a_\ell]_t, [b_\ell]_t, [r_\ell]_t)\}_{\ell=1}^{N}$, sends them to P_{king}.

Party P_{king}

P_{king} detects the corrupted party and sends proof to convince all parties. In the following, when P_{king} receives $(\texttt{Proof}, P_c)$ from a party P_i and $(\texttt{Corrupt}, P_c)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by dealer P_c, P_{king} reliably broadcasts

the identity of P_i, lets all parties wait to receive the message $(\texttt{Corrupt}, P_c)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ and terminates. P_{king} also reliably broadcasts $(\texttt{Corrupt}, 1, P_c)$.

(a). **Invalid $2t$ Shares of Zero.**

(1). For each dealer in $\mathcal{D}$: wait to receive messages from all parties:

* Upon receiving $\{g_\ell^{(i)}(y), \bar{g}_\ell^{(i)}(y)\}_{\ell\in[2N/(t+1)^2]}$ from P_i and $\{\mathsf{cm\text{-}row}_j^{(i)}\}_{j\in[n]}$ from the dealer, verify that:

$$\mathsf{cm\text{-}col}_\ell^{(i)} \stackrel{?}{=} H(g_\ell^{(i)}(\beta_*), \bar{g}_\ell^{(i)}(\beta_*)) \quad \forall \ell \in [2N/(t+1)^2]$$

If true, accept P_i's $\{g_\ell^{(i)}(y)\}_{\ell\in[2N/(t+1)^2]}$.

* Upon receiving $\texttt{ShareProof}$ from party, record it.

(2). If P_{king} first receives a $\texttt{ShareProof}$ for a dealer $P_c \in \mathcal{D}$, he sends this proof to all parties, reliably broadcasts $(\texttt{Corrupt}, 0, P_c)$ and terminates. Otherwise, for each dealer, P_{king} first accept $2t+1$ distinct P_i's column polynomials, he reconstructs dealer's bivariate polynomials $\{F_\ell(x,y), \bar{F}_\ell(x,y)\}_{\ell\in[2N/(t+1)^2]}$. Then P_{king} checks whether the secrets encoded in each $F_\ell(x,y)$ are all zero. If not, P_{king} moves to Step (4).

(3). For each dealer, P_{king} computes the rest of t parties' $\{g_\ell^{(i)}(y), \bar{g}_\ell^{(i)}(y)\}_{\ell\in[2N/(t+1)^2]}$ and verify that:

$$\mathsf{cm\text{-}col}_\ell^{(i)} \stackrel{?}{=} H(g_\ell^{(i)}(\beta_*), \bar{g}_\ell^{(i)}(\beta_*)) \quad \forall \ell \in [2N/(t+1)^2]$$

If true, P_{king} computes all parties' shares of zero $\{[o_k^{(j)}]_{2t}\}_{k\in[N], P_j\in\mathcal{D}}$ and moves to Step (b). Otherwise, P_{king} moves to Step (4).

(4). P_{king} generates the $\texttt{SecretProof}$ which includes the $2t+1$ distinct parties' column polynomials he accepted in Step (2). Each party can follow Steps (2)-(3) to verify this proof. Then P_{king} sends this proof to all parties, reliably broadcasts $(\texttt{Corrupt}, 0, P_c)$ and terminates.

(b). **Wrong Shares of $[z_\ell]_{2t}$.**

(1). P_{king} waits to receive messages from all parties, when P_{king} succeeds in using online error correction to reconstruct all degree-t sharings $\{([a_\ell]_t, [b_\ell]_t, [r_\ell]_t)\}_{\ell=1}^N$, he proceeds.

(2). With $\{[o_k^{(j)}]_{2t}\}_{k\in[N]}$ distributed by each $P_j \in \mathcal{D}$, P_{king} locally compute each party's shares of $[\tilde{z}_\ell]_{2t}$ for all $\ell \in [N]$. For all shares of $[z_\ell]_{2t}$ received from $2t+1$ distinct parties during the $\Pi_{\texttt{tripleKingDN}}$, P_{king} records the identity of the first party who provided shares of $[z_\ell]_{2t} \neq [\tilde{z}_\ell]_{2t}$. Let the index of first $[z_\ell]_{2t} \neq [\tilde{z}_\ell]_{2t}$ be idx and this corrupted party's share of $[o_{\mathsf{idx}}]_{2t} := [\tilde{z}_{\mathsf{idx}}]_{2t} - [a_{\mathsf{idx}}]_t \cdot [b_{\mathsf{idx}}]_t - [r_{\mathsf{idx}}]_t$.

(3). Assuming that the identity of this corrupted party is P_c, P_{king} generates $\texttt{ShareRecProof}$: including P_c's column polynomials which are used to extract $[o_{\mathsf{idx}}]_{2t}$ and distributed by each dealer in $\mathcal{D}$. Each party receiving this proof can first verify that the shares in these columns match each dealer's commitment. Then, each party can reconstruct P_c's extracted shares of zero.

(4). P_{king} sends the idx, the whole sharings $\{([a_{\mathsf{idx}}]_t, [b_{\mathsf{idx}}]_t, [r_{\mathsf{idx}}]_t)\}$, P_c's shares $\{z_\ell^{(c)}\}_{\ell=1}^N$, P_c's opening $\nu^{(c)}$ (corresponding to the commitment of $\{z_\ell^{(c)}\}_{\ell=1}^N$), and the $\texttt{ShareRecProof}$ to all parties. P_{king} also reliably broadcasts $(\texttt{Corrupt}, 0, P_c)$.

Each Party

Each party waits for valid proof against an active corrupted party from P_{king}. Each party may receive a proof for case (a) and (b) from P_{king} or notified by P_{king} to wait to receive $(\texttt{Corrupt}, P_c)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. For the former, each party P_i does the following verification and sets his input $b_i = 0$ if he accepts the proof. For the latter, P_i will set $b_i = 1$.

- For case (a): Upon receiving $\texttt{ShareProof}$ or $\texttt{SecretProof}$ and $(\texttt{Corrupt}, 0, P_c)$. from P_{king}, each party checks whether it is valid. If true, he records the identity of the corresponding corrupted dealer P_c.
- For case (b):
  (1). Upon receiving the idx and $\{([a_{\mathsf{idx}}]_t, [b_{\mathsf{idx}}]_t, [r_{\mathsf{idx}}]_t)\}$ from P_{king}, each party checks whether his shares of $\{([a_{\mathsf{idx}}]_t, [b_{\mathsf{idx}}]_t, [r_{\mathsf{idx}}]_t)\}$ is correct. If true, he reliably broadcast $\texttt{OK}$ to all parties. Each party proceeds if he receives $\texttt{OK}$ from $2t+1$ distinct parties.
  (2). Upon receiving $(\texttt{Corrupt}, 0, P_c)$, P_c's shares $\{z_\ell^{(c)}\}_{\ell\in[N]}$, opening $\nu^{(c)}$ from P_{king} and τ_c^{king} which is the P_c's commitment for P_{king}, each party checks whether $\tau_c^{\mathsf{king}} = H(z_1^{(c)}||\cdots||z_L^{(c)}, \nu^{(c)})$. If true, he proceeds.
  (3). Upon receiving the valid $\texttt{ShareRecProof}$ from P_{king}, each party locally computes P_c's share of $[o_{\mathsf{idx}}]_{2t}$. Then he checks whether $[z_{\mathsf{idx}}]_{2t} = [a_{\mathsf{idx}}]_t \cdot [b_{\mathsf{idx}}]_t + [r_{\mathsf{idx}}]_t + [o_{\mathsf{idx}}]_{2t}$. If not, he records the identity of the corrupted party P_c.

All parties jointly invoke $\mathcal{F}_{\mathsf{ba}}$, each party P_i sends b_i to $\mathcal{F}_{\mathsf{ba}}$. Let the output of $\mathcal{F}_{\mathsf{ba}}$ be b, then all parties wait to receive $(\texttt{Corrupt}, b, P_c)$ from P_{king} and output the identity of P_c.

Communication Cost of $\Pi_{\texttt{faultLoc}}$. The communication costs include:

- For all parties P_j, it requires $\mathcal{O}(N \cdot n + n^2 \log(N/n^2))$ field elements.
- For P_{king}, in case (a), it requires $\mathcal{O}(N + n^2 \log(N/n^2))$ field elements. In case (b), assuming that the size of idx is N elements, since the proof size of `ShareRecProof` (P_c's real column polynomial received from $2t+1$ dealers) is $\mathcal{O}(n^2)$ field elements, it requires $\mathcal{O}(N \cdot n + n^3)$ field elements in total.
- For all parties, they send OK to each other, which requires $\mathcal{O}(n^2)$ bits. P_{king} broadcasts the identity of corrupted parties, which requires $\mathcal{O}(n^2)$ bits.

Since $N \cdot n + n^3 = n^2(N/n + n) \geq 2n^2\sqrt{N} > n^2 \log(N/n^2)$, the whole costs of $\Pi_{\texttt{faultLoc}}$ is $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(n^2)$ bits.

D.7 Cost Analysis of $\Pi_{\texttt{tripleDN}}$

We first analyse the communication costs of $\Pi_{\texttt{Sh2tZero}}$ and $\Pi_{\texttt{tripleKingDN}}$ as follows:
For $\Pi_{\texttt{Sh2tZero}}$:

- For dealer, in Step 1, it requires $\mathcal{O}(N \cdot n + n^2)$ field elements. In Step 2, it requires one instance of RBC, which requires $\mathcal{O}(N \cdot n\kappa + \kappa \cdot n^3)$ bits.
- For the parties, in Step 1, it requires $\mathcal{O}(N \cdot n + n^2)$ field elements.

The total communication costs of $\Pi_{\texttt{Sh2tZero}}$ are $\mathcal{O}(N \cdot n + n^2)$ field elements plus $\mathcal{O}(N \cdot n\kappa + \kappa \cdot n^3)$ bits.
For $\Pi_{\texttt{tripleKingDN}}$:

- In Step 1, it requires $\mathcal{O}(N \cdot n)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits.
- In Step 2, it requires $\mathcal{O}(N \cdot n)$ field elements plus $\mathcal{O}(\kappa \cdot n^2)$ bits.

The total communication costs of $\Pi_{\texttt{tripleKingDN}}$ are $\mathcal{O}(N \cdot n)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits.
Then the whole communication costs of $\Pi_{\texttt{tripleDN}}$ include:

- In the Preparation, one instance of $\Pi_{\texttt{randSh}}$ costs $\mathcal{O}(N \cdot n + n^5)$ field elements plus $\mathcal{O}(\kappa \cdot n^5)$ bits, one instance of $\Pi_{\texttt{randShareZero}}$ costs $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(N \cdot n \cdot \kappa + \kappa \cdot n^4)$ bits.
- In the Generation, in each segment, it requires:
  - All parties broadcast ACSS proofs require $\mathcal{O}(N + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3)$ bits.
  - n instance of $\Pi_{\texttt{tripleKingDN}}$: $\mathcal{O}(N + n^2)$ field elements plus $\mathcal{O}(\kappa \cdot n^4)$ bits.
  - One instance of $\mathcal{F}_{\mathsf{acs}}$: $\mathcal{O}(\kappa \cdot n^3)$ bits.
  - One instance of $\Pi_{\texttt{tripleVerify}}$: one instance of $\mathcal{F}_{\mathsf{coin}}$ plus the times of $\Pi_{\texttt{SubCktEval}}$.
  - $t+1$ instances of $\Pi_{\texttt{faultLoc}}$: $\mathcal{O}(N + n^4)$ field elements plus $\mathcal{O}(n^3)$ bits.
  - n instances of $\mathcal{F}_{\mathsf{ba}}$: $\mathcal{O}(n^4)$ bits.

  For n segments, there are at most $\mathcal{O}(n)$ times of $\Pi_{\texttt{SubCktEval}}$, which requires $\mathcal{O}(N \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^3 + n^4)$ bits. Also, there are at most t times of reconstruction of corrupted parties' secrets, which requires $\mathcal{O}(N \cdot n + n^4)$ field elements. In total, for n segments, it requires $\mathcal{O}(N \cdot n + n^5)$ field elements plus $\mathcal{O}(\kappa \cdot n^5)$ bits.

Therefore, the whole communication costs of $\Pi_{\texttt{tripleDN}}$ are $\mathcal{O}(N \cdot n + n^5)$ field elements plus $\mathcal{O}(N \cdot n \cdot \kappa + \kappa \cdot n^5)$ bits.

D.8 Construction of $\Pi_{\texttt{triple}}$

Based on $\Pi_{\texttt{triple-Add-Weak}}$, after replacing $\Pi_{\texttt{tripleExt-Weak}}$, $\Pi_{\texttt{tripleDN-Weak}}$ with $\Pi_{\texttt{tripleExt}}$, $\Pi_{\texttt{tripleDN}}$, we get the construction of $\Pi_{\texttt{triple}}$.

Protocol $\Pi_{\texttt{triple}}$

Let N be the number of Beaver triples to be prepared.

1: **Run Process 1 and Process 2:**
   All parties execute $\Pi_{\texttt{tripleExt}}$ and $\Pi_{\texttt{tripleDN}}$ in parallel to prepare N random Beaver triples.
2: **Agree on Successful Process:**

For each party P_i, if the first process first succeeds, he sets $b_i = 0$; otherwise, he sets $b_i = 1$. Then P_i sends b_i to $\mathcal{F}_{\mathsf{ba}}$. Upon receiving b from $\mathcal{F}_{\mathsf{ba}}$, if $b = 0$, he takes the output of the first process as the final output; otherwise, he takes the output of the second process as the final output.

D.9 Construction and Security Proof of Main Protocol

The functionality for AMPC with GOD [CP23] is below and we design $\Pi_{\texttt{main-GOD}}$ to realize it.

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 $f : (\{0,1\}^* \cup \{\perp\})^n \to \{0,1\}^* \cup \{\perp\}$. For each party P_i, initialize an input value $x^{(i)} = \perp$ and output value $y^{(i)} = \perp$.

1: Upon receiving an input v from $P_i \in \mathcal{P}$, if $\mathsf{CoreSet}$ has not been recorded yet or if $P_i \in \mathsf{CoreSet}$, set $x^{(i)} = v$.
2: Upon receiving an input $\mathsf{CoreSet}$ from $\mathcal{S}$, verify that $\mathsf{CoreSet}$ is a subset of $\mathcal{P}$ of size at least $n - t$, else ignore the message. If $\mathsf{CoreSet}$ has not been recorded yet, then record $\mathsf{CoreSet}$ and for every $P_i \notin \mathsf{CoreSet}$, set $x^{(i)} = 0$.
3: If the $\mathsf{CoreSet}$ has been recorded and the value $x^{(i)}$ has been set to a value different from $\perp$ for every $P_i \in \mathsf{CoreSet}$, then compute $y = f(x^{(1)}, \ldots, x^{(n)})$ and generate a request-based delayed output $y^{(i)} = y$ for every $P_i \in \mathcal{P}$.
4: All honest parties output the results received from the trusted party. Corrupted parties may output anything they want.

Protocol $\Pi_{\texttt{main-GOD}}$

Offline Phase

Let C denote the circuit to be computed. All parties invoke $\Pi_{\texttt{triple}}$ to prepare $|C|$ random Beaver triples and assign one random triple with each multiplication gate in the circuit.

Online Phase

1: **Distributing Inputs.** Each party P_i invokes $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to shares his secret x_i.
2: Each party P_i sets the property Q as P_i terminates $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ where P_j acts as the dealer. 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 0.
3: Divide the circuit C into t disjoint sub-circuits $C_1, \ldots, C_t$ (sorted by topology) such that each sub-circuit contains $|C|/t$ multiplication gates.
4: **Circuit Evaluation.** From $k = 1$ to t, for each sub-circuit C_k, let $([a_i]_t, [b_i]_t, [c_i]_t)_{i=1}^{|C|/t}$ denote the random Beaver triples assigned to multiplication gates in C_k, all parties execute $\Pi_{\texttt{SubCktEval}}$ with shares of inputs for C_k and these $|C|/t$ random Beaver triples to evaluate C_k.
   – If the output of $\Pi_{\texttt{SubCktEval}}$ is the identity of a corrupted party, all parties send `Public-Recon` to the $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by this corrupted party and wait to receive his secrets from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. Then they locally update their sharings by these secrets and invoke $\Pi_{\texttt{SubCktEval}}$ for the current C_k again.
   – Otherwise, all parties set $k = k + 1$.
5: **Output Reconstruction.** Upon all parties terminating the last sub-circuit C_k and learning their output shares, they divide C_O output wires into t segments and invoke $\Pi_{\texttt{SubCktEval}}$ as above for each segment to do public reconstruction in order. As a result, all parties get all C_O outputs.

Costs Analysis of $\Pi_{\texttt{main-GOD}}$. The communication costs include:

Assuming that the input and output size are C_I and C_O respectively, during the online phase, the circuit C is divided into $C_1, \ldots, C_t$, the depth of each C_k is D_k:

– For $\Pi_{\texttt{triple}}$, it requires $\mathcal{O}(|C| \cdot n + n^5)$ field elements plus $\mathcal{O}(|C| \cdot n\kappa + \kappa \cdot n^5)$ bits.
– For the input phase, there are n instances of $\Pi_{\mathsf{ACSS\text{-}id}}$, which costs $\mathcal{O}(C_I \cdot n + n^5)$ field elements plus $\mathcal{O}(\kappa \cdot n^5)$ bits.
– One instance of $\mathcal{F}_{\mathsf{acs}}$, which costs $\mathcal{O}(\kappa \cdot n^3)$ bits.

- During the circuit evaluation, at most $2t$ instance of $\Pi_{\texttt{SubCktEval}}$, since $\sum_{k=1}^{t} |C_i| = |C|, \sum_{k=1}^{t} D_k = \mathcal{O}(D+n)$, the communication costs are $\mathcal{O}(|C| \cdot n + D \cdot n^2 + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^4)$ bits.
- During the output reconstruction, at most $2t$ times of $\Pi_{\texttt{SubCktEval}}$, which requires $\mathcal{O}(C_O \cdot n + n^3)$ field elements plus $\mathcal{O}(\kappa \cdot n^4)$ bits.
- Public Reconstruction for at most t corrupted parties' secrets, which costs $\mathcal{O}(|C| \cdot n + n^5)$ field elements plus $\mathcal{O}(\kappa \cdot n^5)$ bits.

To see why $\sum_{k=1}^{t} D_k = \mathcal{O}(D+n)$, Let $L(C_i)$ and $L(C_{i+1})$ denote the layers of circuits C_i and C_{i+1}, respectively. The number of layers at the intersection of C_i and C_{i+1} can be expressed as $L(C_i) \cap L(C_{i+1}) = \{0, 1\}$, then

$$\sum_{k=1}^{t} D_k = D + \sum_{i=1}^{t-1} L(C_i) \cap L(C_{i+1}) < D + t.$$

Therefore, the total communication costs of $\Pi_{\texttt{main-GOD}}$ is $\mathcal{O}((|C| + C_I + C_O) \cdot n + D \cdot n^2 + n^5)$ field elements plus $\mathcal{O}(|C| \cdot n\kappa + \kappa \cdot n^5)$ bits.

For the computation costs of $\Pi_{\texttt{main-GOD}}$:

- For $\Pi_{\mathsf{ACSS\text{-}id}}$: the dealer invokes $\mathcal{O}(n^3)$ hash to compute the commitments, each party performs $\mathcal{O}(n^2)$ hash to do verification. Therefore, for all dealers, each party requires $\mathcal{O}(n^3)$ hash.
- For $\Pi_{\texttt{Sh2tZero}}$: to prepare $|C|$ triples, the size of L is $|C|/n^2$, the dealers invokes $\mathcal{O}(|C|)$ hash to compute the commitments, each party performs $\mathcal{O}|C|/n$ hash to verify his row polynomials and $\mathcal{O}|C|/n$ hash to interpolate and verify column polynomials. Therefore, for all dealers, each party requires $\mathcal{O}(|C|n)$ hash. Note that to verify the `ShareProof`, it requires $\mathcal{O}(n \log(|C|/n^2))$ hash.
- For $\Pi_{\texttt{tripleKingDN}}$, for each P_{king} in each segment, each party requires $\mathcal{O}(1)$ hash. Therefore, there is $\mathcal{O}(n^2)$ hash in total.
- For $\Pi_{\texttt{faultLoc}}$, each P_{king} in each segment may verify $\mathcal{O}(n)$ distinct `ShareProof`, which requires $\mathcal{O}(n^2 \log(|C|/n^2))$ hash. For the `SecretProof`, `ShareRecProof` generated by P_{king}, each party requires $\mathcal{O}(n)$ hash for verification. Therefore, there is $\mathcal{O}(n^4 \log(|C|/n^2))$ hash in total.
- There are $\mathcal{O}(n^2)$ instances of $\mathcal{F}_{\mathsf{ba}}$ and $\mathcal{O}(n)$ instances of $\mathcal{F}_{\mathsf{acs}}$, which requires $\mathcal{O}(n^4)$ hash.
- For the verification of ACSS proofs, each time it requires $\mathcal{O}(n)$ hash to verify. Since each party sends ACSS proofs to all parties with at most $\mathcal{O}(n)$ times, each party will receive at most $\mathcal{O}(n^2)$ ACSS proofs and require $\mathcal{O}(n^3)$ hash for verification in total.

Therefore, the whole computation costs of $\Pi_{\texttt{main-GOD}}$ are $\mathcal{O}(|C|n + n^4 \log(|C|/n^2))$ hash per party.

Lemma 10. *Let κ denote the security parameter, for a finite field $\mathbb{F}$ of size $2^{\Omega(\kappa)}$, protocol $\Pi_{\texttt{main-GOD}}$ securely computes $\mathcal{F}_{\mathsf{AMPC}}$ against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t < n/3$ parties.*

Proof. **Termination.** We first show that all honest parties will eventually terminate the protocol $\Pi_{\texttt{main-GOD}}$. We first analyze the termination of the offline phase and then the online phase.

All parties invoke $\Pi_{\texttt{triple}}$ during the offline phase to prepare random Beaver triples. It contains one instance of $\Pi_{\texttt{tripleExt}}$, $\Pi_{\texttt{tripleDN}}$ and $\mathcal{F}_{\mathsf{ba}}$. In the following, we only analyze why each party will eventually terminate one of the processes, the reason why when an honest party terminates one process, all parties will eventually terminate that process is similar to the analysis in Appendix C.5.

For process 1, the $\Pi_{\texttt{tripleExt}}$:

- In Step 1, all parties invoke $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to distribute degree-t Shamir sharings.
- In Step 2, all parties invoke $\mathcal{F}_{\mathsf{acs}}$ to agree on a set $\mathcal{D}$ of size L which contains successful dealers. When at least $L - (2t+1) = (t-3)/2$ corrupted dealers also terminate their $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$, all parties can agree on such a set.
- In Step 3, all parties invoke $\mathcal{F}_{\mathsf{coin}}$ and eventually get a random value r. Then they do local computation and invoke $\Pi_{\texttt{SubCktEval}}$ to do public reconstruction. We have analyzed the termination of $\Pi_{\texttt{SubCktEval}}$ in Appendix D.2, then all parties will either succeed in reconstructing $\{f_i(r), g_i(r), h_i(r)\}_{P_i \in \mathcal{D}}$ or agree on a corrupted party:
  - For the former case, all parties can proceed to do verification and move to Step 4.
  - For the latter case, all parties will reconstruct the secret distributed by this corrupted dealer and the next reconstruction will not fail again due to this corrupted party. Therefore, after at most t times $\Pi_{\texttt{SubCktEval}}$, all parties will eventually get the reconstruction results and proceed.

Therefore, at the end of Step 3, all parties will terminate with their shares of Beaver triples distributed by each dealer in $\mathcal{D}$. Note that some honest parties may still not know their shares, but they will know the public reconstruction succeeds and can proceed.

– In Step 4, all parties only invoke $\Pi_{\texttt{SubCktEval}}$ to do public reconstruction. After at most t times $\Pi_{\texttt{SubCktEval}}$, they will eventually get the reconstruction results and proceed. Similarly, some honest parties may not know their shares of Beaver triples, but they can terminate after getting public reconstruction results.

To summarize, if all parties can agree on the set of size L in Step 2, they will eventually terminate $\Pi_{\texttt{tripleExt}}$.

For process 2, the $\Pi_{\texttt{tripleDN}}$:

– In the preparation phase, all parties invoke $\Pi_{\texttt{randSh}}$ and $\Pi_{\texttt{randShareZero}}$ to prepare degree-t and $2t$ Shamir sharings. Similar to the analysis in Appendix C.5, all parties will eventually terminate $\Pi_{\texttt{randSh}}$. For $\Pi_{\texttt{randShareZero}}$, if less than $(t-1)/2$ corrupted parties participate, all honest parties will eventually get their degree-$2t$ Shares of zero distributed by each dealer.

– In the following, we prove that during the generation phase, in each segment, all parties will eventually output shares of Beaver triples or agree on a corrupted party.

- In Step 1, each party who does not have degree-t sharings will request $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ to let all parties learn a corrupted dealer.
- In Step 2, each party acts P_{king} and leads an instance of $\Pi_{\texttt{tripleKingDN}}$:
  * Each honest party who can compute shares of $[z_\ell]_{2t}$ will send them to P_{king}, each party who can not compute the sharings has requested $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ in Step 1 and sent $(\texttt{Proof}, D)$ to P_{king}. Therefore, P_{king} can always expect to receive messages from honest parties. If he first receives $2t+1$ shares of $[z_\ell]_{2t}$, he can reconstruct z_ℓ and proceed. If he first receives $(\texttt{Proof}, D)$ from a party and the identity of a corrupted party from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by D, he can notify all parties to wait for the same output $(\texttt{Corrupt}, D)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$.
  * When P_{king} is honest, they can eventually receive the secrets reconstructed by P_{king} or the message $(\texttt{Corrupt}, D)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$.
- In Step 3, since the instances of $\Pi_{\texttt{tripleKingDN}}$ invoked by honest P_{king} will eventually terminate, all parties can agree on such a set of size $2t+1$. For each king in this set, if the output they get from this king is the identity of a corrupted party, all parties will reconstruct the secret distributed by this corrupted party and invoke the current segment again. Otherwise, they can proceed.
- In Step 4, all parties invoke $\Pi_{\texttt{tripleVerify}}$ to check whether there exists additive errors. Similar to $\Pi_{\texttt{tripleVerify-Weak}}$, after replacing $\mathcal{F}_{\mathsf{pubRec\text{-}Weak}}$ by $\Pi_{\texttt{SubCktEval}}$, all parties will eventually terminate $\Pi_{\texttt{tripleVerify}}$ with valid shares of Beaver triples or $\texttt{Fail}$ for each P_{king} in the set.
- In Step 4, after terminating $\Pi_{\texttt{tripleVerify}}$, if at least $t+1$ distinct P_{king} provides valid shares of Beaver triples, all parties can output their shares and terminate. Otherwise, if the prepared triples provided by at least $t+1$ distinct P_{king} are incorrect, all parties will invoke $\Pi_{\texttt{faultLoc}}$ to help these $t+1$ distinct P_{king} to find a corrupted party. Since there is at least one honest P_{king} among these $t+1$ corrupted parties, we focus on the reason why honest P_{king} can find a corrupted party. Here honest P_{king} may identify a corrupted party in two different ways, one is the corrupted party that does not honestly distribute degree-$2t$ Shamir sharings or sends incorrect shares of $[z_\ell]_{2t}$ to him. The other is the corrupted party received from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. If P_{king} identifies a corrupted party P_k in the former case, he will reliably broadcasts $(\texttt{Corrupt}, 0, P_k)$. If he learns a corrupted party P_k from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$, he will reliably broadcasts $(\texttt{Corrupt}, 1, P_k)$. P_{king} may reliably broadcast both of these two messages. P_{king} does the following things to identify a corrupted party P_k:

  (1). First, each party sends their degree-$2t$ shares of zero distributed by each dealer to P_{king}, and each party who does not have his shares of zero will send a $\texttt{ShareProof}$ to P_{king} (we consider the case when all parties terminate $\Pi_{\texttt{randShareZero}}$). As a result, P_{king} can eventually receive $2t+1$ honest parties' messages and either reconstruct the secrets based on $2t+1$ shares of zero or get a valid proof $\texttt{ShareProof}$:
  · For the former case, P_{king} can proceed to do checks for each dealer.
  · For the latter case, P_{king} will use this proof to accuse a corrupted party.
  If P_{king} finds that all dealers behave honestly, he moves to the next step.

  (2). Second, all parties send their degree-t Shamir Sharings to P_{king}, each party who does not have his sharings has requested $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ and sends $(\texttt{Proof}, P_k)$ to all parties in Step 1. Therefore,

P_{king} can eventually receive all honest parties' messages and either succeed in reconstructing the secret or learn a corrupted dealer P_k from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$:

- For the former case, P_{king} proceeds.
- For the latter case, P_{king} notifies all parties to wait for the messages $(\texttt{Proof}, P_k)$ and $(\texttt{Corrupt}, P_k)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ and terminate.

After reconstructing the degree-t Shamir sharings, P_{king} will follow the protocol to check which party provides incorrect shares of $[z_\ell]_{2t}$ and send the corresponding proof to all parties.

To prevent corrupted P_{king} provides wrong degree-t Shamir sharings, each party will locally check whether his share is correct (if have). If true, each party will send $\texttt{OK}$ to all parties and when he receives $\texttt{OK}$ from $2t+1$ distinct parties, he can believe that P_{king} provides correct degree-t Shamir sharings. Note that when P_{king} is honest, the problem here is that maybe not all honest parties have sharings and each party can not expect to receive $\texttt{OK}$ from $2t+1$. But in this case, each honest party who does not send $\texttt{OK}$ to all parties must have sent $(\texttt{Proof}, P_k)$ to all parties and request $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ in Step 1. Since honest P_{king} will eventually receive $(\texttt{Corrupt}, P_k)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ and broadcast $(\texttt{Corrupt}, 1, P_k)$, each party can wait to receive $(\texttt{Corrupt}, 1, P_k)$ from P_{king} and wait until he receives $(\texttt{Corrupt}, P_k)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. Then we ensure that for honest P_{king}, each honest party will eventually receives $(\texttt{Corrupt}, 0, P_k)$ from P_{king} and agree on the proof provided by P_{king} or receives $(\texttt{Corrupt}, 1, P_k)$ from P_{king} and agree on it after he receives $(\texttt{Corrupt}, P_k)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. We let each party P_i set his input b_i to be 0 or 1 determined by which line he first agrees. Then we let all parties invoke an instance of $\mathcal{F}_{\mathsf{ba}}$ to agree on which line they eventually agree and therefore all parties will eventually agree on a corrupted party.

- In Step 4, finally, all parties invoke n instances of $\mathcal{F}_{\mathsf{ba}}$ to agree on one corrupted party accused by P_{king}, similar to the previous analysis, all parties will eventually terminate and agree on the same corrupted party.

– Now we prove that all parties will eventually get their shares of Beaver triples or agree on a corrupted party. If they agree on a corrupted party, they can eliminate this corrupted party and execute the current segment again. If this corrupted party had previously distributed degree-t Shamir sharings, all parties would have jointly reconstructed his secrets. If this corrupted party had previously distributed degree-$2t$ Shamir sharings, all parties would change their shares by zero. The next time all parties will not fail due to this corrupted party.

To summarize, if all parties can terminate $\Pi_{\texttt{randShareZero}}$, they will eventually terminate $\Pi_{\texttt{tripleExt}}$.

In Appendix C.5, we have proved that at least one process will eventually terminate, and therefore all parties will terminate $\Pi_{\texttt{triple}}$.

Then in the online phase, we divide the circuit into t sub-circuit. All parties invoke $\Pi_{\texttt{SubCktEval}}$ to evaluate each sub-circuit in sequence. Each time when they fail to get the result in $\Pi_{\texttt{SubCktEval}}$, they will agree on a corrupted party and reconstruct all degree-t Shamir sharings distributed by this corrupted party. Then all parties will not fail in $\Pi_{\texttt{SubCktEval}}$ due to this corrupted party next time. As a result, all parties will eventually terminate all instances of $\Pi_{\texttt{SubCktEval}}$ and get the final results.

Security. Then we show that the protocol $\Pi_{\texttt{main-GOD}}$ securely computes $\mathcal{F}_{\mathsf{AMPC}}$. We start with constructing the ideal adversary $\mathcal{S}$ as follows.

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

Let $\mathcal{C}orr'$ be a set of size at most t and $\mathcal{S}$ learns all shares for parties in $\mathcal{C}orr'$. For public reconstruction, $\mathcal{S}$ takes the whole degree-t output Shamir sharings as inputs and skips Step 2. For circuit evaluation, $\mathcal{S}$ takes shares of degree-t Shamir sharings $([x^{(i)}]_t, [y^{(i)}]_t)$ and $([a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t$ for parties in $\mathcal{C}orr'$ as inputs.

Simulation of $\Pi_{\texttt{SubCktEval}}$

1: In Step 1, $\mathcal{S}$ honestly follows the protocol to execute each honest party to check their shares. When each party P_i reliably broadcasts $(\texttt{Proof}, D)$ and sends $\texttt{Broadcast-Proof}$ to $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$, $\mathcal{S}$ reliably broadcasts $(\texttt{Proof}, D)$ on behalf of P_i. Then $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by D and sends the secrets he received from D to all parties.

2: In Step 2, in each layer, for every addition gate, $\mathcal{S}$ computes shares of $[z^{(i)}]_t$ for each party in $\mathcal{C}orr'$. For a group of L multiplication gates, $\mathcal{S}$ first computes shares of $[x^{(i)} + a^{(i)}]_t, [y^{(i)} + b^{(i)}]_t$ for parties in $\mathcal{C}orr'$, then randomly samples the whole $[x^{(i)} + a^{(i)}]_t, [y^{(i)} + b^{(i)}]_t$ based on shares of parties in $\mathcal{C}orr'$ and simulates $\Pi_{\texttt{BatchPubRec}}$ with honest parties' shares. Then $\mathcal{S}$ computes shares of $[z^{(i)}]_t$ for parties in $\mathcal{C}orr'$.

3: In Step 3, $\mathcal{S}$ honestly simulates $\Pi_{\texttt{BatchPubRec}}$ with honest parties' output sharings.
4: In Step 4, $\mathcal{S}$ follows the protocol to compute each honest party P_j's input $\boldsymbol{m}_j$. Then $\mathcal{S}$ honestly simulates $\Pi_{\texttt{Agreement}}$ and learns each honest party's output.

Simulator $\mathcal{S}_0$

Simulation of $\Pi_{\texttt{tripleExt}}$

1: In Step 1, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ as follows:
   – For each honest dealer, $\mathcal{S}$ randomly samples shares of corrupted parties. For each corrupted dealer, $\mathcal{S}$ waits to receive degree-t Shamir sharings.
   – For corrupted parties, $\mathcal{S}$ sends their sharings to them on behalf of $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ and learns the identity of honest party whose output is $(\texttt{Corrupt}, D)$.
2: In Step 2, $\mathcal{S}$ honestly simulates Π^{Q}_{acs} and whenever an honest party terminates Π^{Q}_{acs} with a set $\mathcal{D}$ of size L, $\mathcal{S}$ continues to simulate the behavior of this party.
3: In Step 3, $\mathcal{S}$ first sends request to $\mathcal{F}_{\mathsf{coin}}$ on behalf of each honest party. Then $\mathcal{S}$ randomly samples a value r and sends it to all parties on behalf of $\mathcal{F}_{\mathsf{coin}}$. If $r \in [N']$, $\mathcal{S}$ aborts the simulation. Otherwise, $\mathcal{S}$ does the following things:
   – For each honest dealer in $\mathcal{D}$, $\mathcal{S}$ randomly samples $f_i(r), g_i(r)$ and computes $h_i(r) = f_i(r) \cdot g_i(r)$. Then $\mathcal{S}$ randomly samples the whole $([f_i(r)]_t, [g_i(r)]_t, [h_i(r)]_t)$ based on the secrets $(f_i(r), g_i(r), h_i(r))$ and shares of corrupted parties. For each corrupted dealer in $\mathcal{D}$, $\mathcal{S}$ computes the whole $([f_i(r)]_t, [g_i(r)]_t, [h_i(r)]_t)$.
   – $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{SubCktEval}}$ with each dealer P_i's $([f_i(r)]_t, [g_i(r)]_t, [h_i(r)]_t)$:
     • If $\mathcal{S}$ learns that all parties agree on a corrupted dealer during $\mathcal{S}_{\texttt{SubCktEval}}$, $\mathcal{S}$ sends $\texttt{Public-Recon}$ to $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by this dealer on behalf of each honest party. When $\mathcal{S}$ receives $t+1$ requests, $\mathcal{S}$ sends this corrupted dealer's secrets to all parties on behalf of $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ and locally updates each honest party's shares. Then $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{SubCktEval}}$ again.
     • Otherwise, $\mathcal{S}$ proceeds.
   – For each dealer in $\mathcal{D}$, $\mathcal{S}$ checks whether $f_i(r) \cdot g_i(r) = h_i(r)$ and if not, $\mathcal{S}$ changes all honest parties' degree-t sharings distributed by this dealer with 0. For corrupted dealer P_i, if $f_i(r) \cdot g_i(r) = h_i(r)$ but $h_i \neq f_i \cdot g_i$, $\mathcal{S}$ aborts the simulation.
4: In Step 4, for $i \in [L' + 2, L]$:
   – If P_i is honest, $\mathcal{S}$ first randomly samples the whole $[f^{(k)}(\alpha_i) + a_i^{(k)}]_t$ based on shares of corrupted parties.
   – If P_i is corrupted, $\mathcal{S}$ first randomly samples a value as $f^{(k)}(\alpha_i) + a_i^{(k)}$, then he computes $f^{(k)}(\alpha_i) = f^{(k)}(\alpha_i) + a_i^{(k)} - a_i^{(k)}$. Finally $\mathcal{S}$ randomly samples the whole $[f^{(k)}(\alpha_i)]_t$ based on shares of corrupted parties and computes $[f^{(k)}(\alpha_i) + a_i^{(k)}]_t = [f^{(k)}(\alpha_i)]_t + [a_i^{(k)}]_t$.

   $\mathcal{S}$ does the same thing for $[g^{(k)}(\alpha_i) + b_i^{(k)}]_t$ and invokes $\mathcal{S}_{\texttt{SubCktEval}}$ with the whole sharings $([f^{(k)}(\alpha_i) + a_i^{(k)}]_t, [g^{(k)}(\alpha_i) + b_i^{(k)}]_t)$ to do public reconstruction. Upon terminating $\mathcal{S}_{\texttt{SubCktEval}}$, $\mathcal{S}$ computes corrupted parties' shares of Beaver triples.

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

Simulation of $\Pi_{\texttt{randSh}}$

1: In Step 1, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ as follows:
   – For each honest dealer, $\mathcal{S}$ randomly samples shares of corrupted parties. For each corrupted dealer, $\mathcal{S}$ waits to receive degree-t Shamir sharings.
   – For corrupted parties, $\mathcal{S}$ sends their sharings to them on behalf of $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ and learns the identity of honest party whose output is $(\texttt{Corrupt}, D)$.
2: In Step 2, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{acs}}$ and learns the set $\mathcal{D}$ of size $2t + 1$.
3: In Step 3, $\mathcal{S}$ follows the protocol and computes corrupted parties' output shares.

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

Simulation of $\Pi_{\texttt{randShareZero}}$

1: In Step 1, for each corrupted dealer, $\mathcal{S}$ honestly executes each honest party. When an honest party terminates $\Pi_{\texttt{Sh2tZero}}$, $\mathcal{S}$ will either learn this honest party's degree-$2t$ shares of zero distributed by this dealer or the `ShareProof` against this dealer.

2: For each honest dealer D, $\mathcal{S}$ samples corrupted parties' degree-$2t$ shares of zero and simulates $\Pi_{\texttt{Sh2tZero}}$ as follows:

(1). $\mathcal{S}$ randomly samples degree-$2t$ row polynomials $f_\ell^{(i)}(x), \bar{f}_\ell^{(i)}(x)$ and degree-$t+(t-1)/2$ column polynomials $g_\ell^{(i)}(y), \bar{g}_\ell^{(i)}(y)$ for each corrupted party P_i based on his shares. Then $\mathcal{S}$ sends $f_\ell^{(i)}(x), \bar{f}_\ell^{(i)}(x)$ for all $\ell \in [L]$ to P_i on behalf of D.

(2). For each corrupted party P_i, $\mathcal{S}$ honestly simulates the random oracle $H(\cdot)$ to get P_i's $\{\mathsf{cm\text{-}row}_j^{(i)}\}_{j=1}^n, \{\mathsf{cm\text{-}col}_\ell^{(i)}\}_{\ell=1}^L$.

(3). For each honest party P_i and the index j where P_j is corrupted, $\mathcal{S}$ computes $\mathsf{cm\text{-}row}_j^{(i)}$ as follows: $\mathcal{S}$ sets:

$$\boldsymbol{f}_*^{(i)}(\alpha_j) = (H(g_1^{(j)}(\alpha_i), \bar{g}_1^{(j)}(\alpha_i)), \ldots, H(g_L^{(j)}(\alpha_i), \bar{g}_L^{(j)}(\alpha_i)))$$

Then $\mathcal{S}$ follows the protocol to compute $\mathsf{cm\text{-}row}_j^{(i)}$. For the rest of index j when P_j is honest, $\mathcal{S}$ randomly samples a vector of values as $\boldsymbol{f}_*^{(i)}(\alpha_j)$ and computes $\mathsf{cm\text{-}row}_j^{(i)}$ accordingly. $\mathcal{S}$ also randomly samples values as each honest party P_i's $\{\mathsf{cm\text{-}col}_\ell^{(i)}\}_{\ell=1}^L$. For each node in the Merkle trees and the commitments, if any value has been mapped to some inputs queried by $\mathcal{A}$ previously, $\mathcal{S}$ aborts the simulation. Then $\mathcal{S}$ reliably broadcasts these commitments on behalf of D.

(4). For each corrupted party P_j, $\mathcal{S}$ computes $(f_\ell^{(i)}(\alpha_j), \bar{f}_\ell^{(i)}(\alpha_j)) = (g_\ell^{(j)}(\alpha_i), \bar{g}_\ell^{(j)}(\alpha_i))$ and sends them to P_j on behalf of each honest party P_i.

(5). For each honest party P_i, when his row polynomials are delivered, $\mathcal{S}$ considers P_i accepts his row polynomials and sends `support` to all parties.

(6). For each honest party P_i, for $f_\ell^{(j)}(\alpha_i), \bar{f}_\ell^{(j)}(\alpha_i)$ which P_i received from corrupted P_j, $\mathcal{S}$ follows the protocol to check whether it is correct. For the sharings P_i received from honest P_j, $\mathcal{S}$ directly considers it to be correct. When P_i receives $t+(t+1)/2$ distinct P_j's correct $f_\ell^{(j)}(\alpha_i), \bar{f}_\ell^{(j)}(\alpha_i)$, $\mathcal{S}$ considers P_i has reconstructed the column polynomial.

(7). $\mathcal{S}$ follows the protocol for each honest party for the termination procedure.

3: In Step 2, $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{acs}}$ and learns a set $\mathcal{D}$ of size $2t+1$.

4: In Step 4, according to the analysis in Appendix C.5, for each output wire $[o_{\ell,k}]_{2t} = [o_{\ell,k}^{\mathcal{H}}]_{2t} + [o_{\ell,k}^{\mathcal{C}orr}]_{2t}$, $\mathcal{S}$ can compute corrupted parties' shares of $[o_{\ell,k}^{\mathcal{H}}]_{2t}$. For $[o_{\ell,k}^{\mathcal{C}orr}]_{2t}$, $\mathcal{S}$ only learns the shares of honest parties that successfully terminate all executions of $\Pi_{\texttt{Sh2tZero}}$ led by corrupted parties in $\mathcal{D}$.

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

Simulation of $\Pi_{\texttt{faultLoc}}$

For each corrupted P_{king}, $\mathcal{S}$ does the following things:

(1). For each honest dealer, $\mathcal{S}$ randomly samples a degree-$(2t, t+(t-1)/2)$ bivariate polynomial $F_\ell(x,y)$ based on corrupted parties' row and column polynomials and degree-$2t$ Shamir sharings of zero. $\mathcal{S}$ also randomly samples degree-$(2t, t+(t-1)/2)$ bivariate polynomials $\bar{F}_\ell(x,y)$ based on corrupted parties' row and column polynomials. Then $\mathcal{S}$ computes each honest party P_i's degree-$(t+(t-1)/2)$ column polynomials $g_\ell^{(i)}(y), \bar{g}_\ell^{(i)}(y)$. For each corrupted dealer, $\mathcal{S}$ uses $g_\ell^{(i)}(y), \bar{g}_\ell^{(i)}(y)$ he received from the dealer. $\mathcal{S}$ checks whether these shares have been queried by $\mathcal{A}$ during the simulation of random oracle, if true, $\mathcal{S}$ aborts the simulation. Otherwise, $\mathcal{S}$ follows the protocol to send $g_\ell^{(i)}(y), \bar{g}_\ell^{(i)}(y)$ or `ShareProof` to P_{king} on behalf of each honest party P_i.

(2). For the degree-t Shamir sharings, $\mathcal{S}$ randomly samples the whole $([a_\ell]_t, [b_\ell]_t, [r_\ell]_t)$ based on shares of corrupted parties. For each honest party, if his output of $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ is not $(\texttt{Corrupt}, D)$, $\mathcal{S}$ computes and sends this honest party's degree-t Shamir sharings to P_{king}.

(3). For each honest dealer, when corrupted P_{king} asks Random Oracle $H(\cdot)$ to verify the commitment $\mathsf{cm\text{-}col}_\ell^{(i)}$ on honest party P_i's degree-$2t$ Shamir sharings $g_\ell^{(i)}(\beta_*), \bar{g}_\ell^{(i)}(\beta_*)$, $\mathcal{S}$ sends the commitment $\mathsf{cm\text{-}col}_\ell^{(i)}$ he randomly sampled during $\Pi_{\texttt{Sh2tZero}}$ to P_{king} as the output of Random Oracle $H(\cdot)$.

(4). $\mathcal{S}$ honestly executes each honest party and waits to receive the identity of the corrupted party and valid proof from P_{king}.

For each honest P_{king}, $\mathcal{S}$ does the following things:

(1). $\mathcal{S}$ executes the following two things in parallel:

- For each corrupted dealer P_c who distributes degree-$2t$ Shamir sharings during $\mathcal{S}_{\texttt{RandShareZero}}$, $\mathcal{S}$ has learnt the output of each honest party who terminates the $\Pi_{\texttt{Sh2tZero}}$ invoked by P_c. If at least one honest party's output is the `ShareProof` and this messages first delivered to P_{king}, $\mathcal{S}$ will send

this proof to all parties on behalf of P_{king}. Otherwise, $\mathcal{S}$ waits until P_{king} receives enough correct column polynomials to generate `SecretProof`. According to the analysis of Termination property, if the simulation of $\Pi_{\texttt{tripleExt}}$ can never terminate due to corrupted dealers not terminating their $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$, then all honest parties will eventually receive their degree-$2t$ shares in $\mathcal{S}_{\texttt{RandShareZero}}$. Then $\mathcal{S}$ can eventually use these $2t+1$ honest parties' shares to generate the `SecretProof` and send it to all parties. Finally, $\mathcal{S}$ reliably broadcasts $(\texttt{Corrupt}, 0, P_c)$ on behalf of P_{king}. Otherwise, $\mathcal{S}$ follows the protocol to check which corrupted party P_c provides incorrect shares of $[z_{\mathsf{idx}}]_{2t}$. Then $\mathcal{S}$ randomly samples the whole $([a_{\mathsf{idx}}]_t, [b_{\mathsf{idx}}]_t, [r_{\mathsf{idx}}]_t)$ based on shares of corrupted parties, sends them as well as the rest of proof to all parties and reliably broadcasts $(\texttt{Corrupt}, 0, P_c)$. For each honest party who receives these degree-t Shamir sharings, $\mathcal{S}$ sends OK to all parties on behalf of this honest party.

- When $\mathcal{S}$ receives $(\texttt{Proof}, P_c)$ from a party P_i and $(\texttt{Corrupt}, P_c)$ from $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$, he reliably broadcasts the identity of P_i and $(\texttt{Corrupt}, 1, P_c)$ on behalf of P_{king}. $\mathcal{S}$ also sends $(\texttt{Corrupt}, P_c)$ to all parties on behalf of $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$.

(2). $\mathcal{S}$ follows the protocol, honestly simulates $\mathcal{F}_{\mathsf{ba}}$ with each honest party P_i's input b_i and learns the corrupted party all parties agree on.

Simulator $\mathcal{S}_1$

Simulation of $\Pi_{\texttt{tripleDN}}$

1: In the Preparation phase, $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{RandShare}}$ and $\mathcal{S}_{\texttt{RandShareZero}}$.
2: In each segment of the Generation phase:

– In Step 1, for each honest party, if $\mathcal{S}$ learns his output of $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ is $(\texttt{Corrupt}, D)$, $\mathcal{S}$ reliably broadcasts $(\texttt{Proof}, D)$ on behalf of this honest party. $\mathcal{S}$ also sends $(\texttt{Corrupt}, D)$ to all parties on behalf of the corresponding $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$.

– In Step 2, $\mathcal{S}$ simulates each $\Pi_{\texttt{tripleKingDN}}$ as follows:

(1). When each party P_i needs to send his share of $[z_\ell]_{2t}$ to P_{king}, $\mathcal{S}$ first randomly samples the whole $[z'_\ell]_{2t} = [a_\ell]_t \cdot [b_\ell]_t + [r_\ell]_t + [o_\ell^{\mathcal{H}}]_{2t}$ based on shares of corrupted parties. Then for each honest party who has terminated $\Pi_{\texttt{randShareZero}}$, $\mathcal{S}$ computes his shares of $[z_\ell]_{2t} = [z'_\ell]_{2t} + [o_\ell^{Corr}]_{2t}$ and sends it to P_{king}. Finally, $\mathcal{S}$ honestly executes honest party P_i to sample $\nu^{(i)}$ and simulates the random oracle to sample $\tau^{(i)}$.

(2). For each P_{king}:

∗ If P_{king} is honest and he first receives $2t+1$ shares of $[z_\ell]_{2t}$, $\mathcal{S}$ reconstructs the secret z_ℓ and reliably broadcasts it on behalf of P_{king}. Otherwise, $\mathcal{S}$ follows the protocol to broadcast the identity of a party.

∗ If P_{king} is corrupted, $\mathcal{S}$ waits to receive z_ℓ or the identity of a party from P_{king}.

(3). If $\mathcal{S}$ receives the secret z_ℓ on behalf of each party, he locally computes corrupted parties' shares of Beaver triples and records the additive error $d_\ell = z_\ell - z'_\ell$. Otherwise, $\mathcal{S}$ learns the identity of a party and waits to receive $(\texttt{Corrupt}, D)$.

– In Step 3, $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{acs}}$ and learns a set $\mathcal{D}$ of size $2t+1$. Then $\mathcal{S}$ follows the protocol to check whether the output of $\Pi_{\texttt{tripleKingDN}}$ led by each $P_{\mathsf{king}} \in \mathcal{D}$ is the identity of a corrupted party:

- If true, $\mathcal{S}$ sends `Public-Recon` to $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ on behalf of each honest party. Upon receiving $t+1$ requests from all parties, $\mathcal{S}$ sends the secrets to all parties on behalf of $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. Then $\mathcal{S}$ follows the protocol to update each honest party's shares and simulates the current segment again.
- Otherwise, $\mathcal{S}$ proceeds.

– In Step 4, $\mathcal{S}$ first simulates $\Pi_{\texttt{tripleVerify}}$ as follows:

- In Step 1, $\mathcal{S}$ first randomly samples the whole $([f^{(k)}(\alpha_i) + a_i^{(k)}]_t, [g^{(k)}(\alpha_i) + b_i^{(k)}]_t)$ based on shares of corrupted parties. Then $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{SubCktEval}}$ with the whole sharings $\{[f^{(k)}(\alpha_i) + a_i^{(k)}]_t, [g^{(k)}(\alpha_i) + b_i^{(k)}]_t\}_{i \in [N'+1, 2N'], k \in [2t+1]}$ to do public reconstruction:
  ∗ If $\mathcal{S}$ learns that all parties agree on a corrupted dealer during $\mathcal{S}_{\texttt{SubCktEval}}$, $\mathcal{S}$ sends `Public-Recon` to $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by this dealer on behalf of each honest party. When $\mathcal{S}$ receives $t+1$ requests, $\mathcal{S}$ sends this corrupted dealer's secrets to all parties on behalf of $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ and locally updates each honest party's shares. Then $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{SubCktEval}}$ again.
  ∗ Otherwise, $\mathcal{S}$ proceeds.

  Finally, for all $i \in [0, 2N'], k \in [2t+1]$, $\mathcal{S}$ follows the protocol to compute corrupted parties' shares of $[h^{(k)}(\alpha_i)]_t$. $\mathcal{S}$ computes a degree-$2N'$ polynomial $d^{(k)}(\cdot)$ such that $d^{(k)}(\alpha_i) = d_i^{(k)}$.
- In Step 2, $\mathcal{S}$ sends requests to $\mathcal{F}_{\mathsf{coin}}$ on behalf of each honest party. Then $\mathcal{S}$ randomly samples a value r and sends it to all parties on behalf of $\mathcal{F}_{\mathsf{coin}}$. If $r \in [\alpha_1, \ldots, \alpha_{N'}]$, $\mathcal{S}$ aborts the simulation. Then for all $k \in [2t+1]$, $\mathcal{S}$ first randomly samples value $f^{(k)}(r), g^{(k)}(r)$, then computes $h^{(k)}(r) =$

$f^{(k)}(r) \cdot g^{(k)}(r) + d^{(k)}(r)$ and randomly samples the whole $([f^{(k)}(r)]_t, [g^{(k)}(r)]_t, [h^{(k)}(r)]_t)$ based on the secrets $(f^{(k)}(r), g^{(k)}(r), h^{(k)}(r))$ and shares of corrupted parties. Finally, $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{SubCktEval}}$ with the whole $([f^{(k)}(r)]_t, [g^{(k)}(r)]_t, [h^{(k)}(r)]_t)$ to do public reconstruct as above.

- In Step 3, $\mathcal{S}$ follows the protocol to determine each honest party's output. For each $k \in [2t+1]$, if $f^{(k)}(r) \cdot g^{(k)}(r) = h^{(k)}(r)$ but $d^{(k)}(\cdot) \not\equiv 0$, $\mathcal{S}$ aborts the simulation. Otherwise, $\mathcal{S}$ computes corrupted parties' shares of Beaver triples.

– In Step 4, after the simulation of $\Pi_{\texttt{tripleVerify}}$. If $\mathcal{S}$ learns that at least $t+1$ distinct kings in $\mathcal{D}$ provides correct Beaver triples, $\mathcal{S}$ follows the protocol and proceeds. Otherwise, $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{FaultLoc}}$ for the first $t+1$ kings in $\mathcal{D}$ who generate incorrect Beaver triples. Then $\mathcal{S}$ follows the protocol to simulates n instances of $\mathcal{F}_{\mathsf{ba}}$ and eventually agree on a corrupted party P_k:

- If P_k has distributed degree-t Shamir sharings, $\mathcal{S}$ sends `Pubic-Recon` to $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by P_k on behalf of each honest party. Then $\mathcal{S}$ sends P_k's secrets to all parties on behalf of $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. Then $\mathcal{S}$ locally updates each honest party's shares.
- If P_k has distributed degree-$2t$ Shamir sharings of zero, $\mathcal{S}$ locally changes each honest parties' degree-$2t$ shares of zero with 0.

Then $\mathcal{S}$ simulates the current segment again.

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

Simulation of $\Pi_{\texttt{triple}}$

1: $\mathcal{S}$ invokes $\mathcal{S}_0$ and $\mathcal{S}_1$ in parallel.
2: $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{ba}}$ and learns the output b. Then $\mathcal{S}$ outputs shares of corrupted parties he computed during $\mathcal{S}_b$.

Simulator $\mathcal{S}$

Let $\mathcal{C}orr$ denote the set of corrupted parties, then $|\mathcal{C}orr| = t' \leq t$. Let $\mathcal{C}orr'$ be the set of all corrupted parties together with the first $t - t'$ honest parties, then $|\mathcal{C}orr'| = t$.

Simulation of $\Pi_{\texttt{main-GOD}}$

1: In the offline phase, $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{Triple}}$ and learns corrupted parties' shares of Beaver triples. For each honest party in $\mathcal{C}orr' \setminus \mathcal{C}orr$, $\mathcal{S}$ randomly samples values as his shares of Beaver triples. $\mathcal{S}$ also learns which honest party does not get his shares but an ACSS proof during $\mathcal{S}_{\texttt{Triple}}$.
2: In the online phase, $\mathcal{S}$ does the following things:
– In Step 1, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ as follows:
- For each honest dealer, $\mathcal{S}$ randomly samples shares of parties in $\mathcal{C}orr'$.
- For each corrupted dealer, $\mathcal{S}$ waits to receive degree-t Shamir sharings and learns the identity of the honest party whose output is $(\texttt{Corrupt}, D)$. $\mathcal{S}$ uses these degree-t Shamir sharing to compute the shares of honest parties in $\mathcal{C}orr'$.

– In Step 2, $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{acs}}$ and learns a set $\mathcal{D}$ of size $2t+1$.
– In Steps 3 and 4, for each sub-circuit C_k and $k \in [t]$, $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{SubCktEval}}$ with shares of parties in $\mathcal{C}orr'$ for C_k:
- If $\mathcal{S}$ learns that all parties agree on a corrupted dealer during $\mathcal{S}_{\texttt{SubCktEval}}$, $\mathcal{S}$ sends `Public-Recon` to $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ invoked by this dealer on behalf of each honest party. When $\mathcal{S}$ receives $t+1$ requests, $\mathcal{S}$ sends this corrupted dealer's secrets to all parties on behalf of $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$. Then $\mathcal{S}$ updates each honest party's shares distributed by this corrupted dealer with secrets. Finally, $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{SubCktEval}}$ again.
- Otherwise, $\mathcal{S}$ proceeds.

– In Step 5, For all parties in $\mathcal{C}orr'$ and output wires $[y]_t$, after getting their shares of output $[y]_t$, $\mathcal{S}$ sends the inputs of corrupted parties and the set $\mathcal{D}$ to $\mathcal{F}_{\mathsf{AMPC}}$ and receives the output y. Then $\mathcal{S}$ computes the whole $[y]_t$ based on the secret y and shares of parties in $\mathcal{C}orr'$ and honestly follows the protocol to invoke $\mathcal{S}_{\texttt{SubCktEval}}$.
3: $\mathcal{S}$ outputs the views of $\mathcal{A}$.

We use the following hybrid arguments to show that the output distribution in the ideal and real world is computationally indistinguishable.

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

$\mathbf{Hyb}_1$: In this hybrid, when $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS\text{-}id}}$ on behalf of honest dealers, $\mathcal{S}$ first randomly samples corrupted parties' shares and then generates the whole sharings based on shares of corrupted parties and

the secrets. According to the property of Shamir sharing, the distribution of the whole sharing is the same. Therefore, the distributions of $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are identical.

$\mathbf{Hyb}_2$: In the following, we focus on the simulation of $\Pi_{\texttt{tripleExt}}$:

$\mathbf{Hyb}_{2.1}$: In this hybrid, we delay the generation of honest parties' degree-t Shamir sharings until the set $\mathcal{D}$ is determined. After $\mathcal{S}$ learns the set $\mathcal{D}$ of size L, $\mathcal{S}$ does not generate honest parties' Shamir sharings distributed by each honest dealer not in $\mathcal{D}$. Since these Shamir sharings are never used, the distributions of $\mathbf{Hyb}_{2.1}$ and $\mathbf{Hyb}_1$ are identical.

$\mathbf{Hyb}_{2.2}$: In this hybrid, for each honest dealer $P_i \in \mathcal{D}$, we change the generation of $\{[f_i(\alpha_0)]_t, [g_i(\alpha_0)]_t, [h_i(\alpha_0)]_t\}$. In $\mathbf{Hyb}_{2.1}$, $[f_i(r)]_t$ is the linear combination of $\{[f_i(\alpha_\ell)]_t\}_{\ell=0}^{N'}$. If $r \notin \{\alpha_1, \ldots, \alpha_{N'}\}$, the coefficient of $[f_i(\alpha_0)]_t$ is non-zero. Then $[f_i(\alpha_0)]_t$ is also the linear combination of $\{[f_i(\alpha_\ell)]_t\}_{\ell=1}^{N'}$ and $[f_i(r)]_t$. So we first randomly sample $[f_i(r)]_t$ and then recompute $[f_i(\alpha_0)]_t$. We do the same thing for $[g_i(\alpha_0)]_t$. Then $\mathcal{S}$ computes the secret $h_i(\alpha_0) = f_i(\alpha_0) \cdot g_i(\alpha_0)$ and randomly samples the whole $[h_i(\alpha_0)]_t$ based on the secret and shares of corrupted parties. The distributions of $\{[f_i(\alpha_0)]_t, [g_i(\alpha_0)]_t, [h_i(\alpha_0)]_t\}$ remain unchanged, so the distributions of $\mathbf{Hyb}_{2.2}$ and $\mathbf{Hyb}_{2.1}$ are identical.

$\mathbf{Hyb}_{2.3}$: In this hybrid, for each honest dealer $P_i \in \mathcal{D}$, we change the generation of $[h_i(r)]_t$. In $\mathbf{Hyb}_{2.2}$, $[h_i(r)]_t$ is the linear combination of $\{[h_i(\alpha_\ell)]_t\}_{\ell=0}^{2N'}$. If $r \notin \{\alpha_1, \ldots, \alpha_{N'}\}$, at least one coefficient of $[h_i(\alpha_0)]_t$ and $\{[h_i(\alpha_\ell)]_t\}_{\ell=N'+1}^{2N'}$ is non-zero. To be more concrete, if $r = \alpha_k$, where $k \in \{0\} \cup \{N'+1, \ldots, 2N'\}$, then $[h_i(r)]_t = [h_i(\alpha_k)]_t$ and $\mathcal{S}$ can randomly samples $[h_i(r)]_t$ based on the secret $h_i(r) = f_i(r) \cdot g_i(r)$ and shares of corrupted parties. If $r \notin \{\alpha_0, \ldots, \alpha_{2N'}\}$, then we know the coefficient of $[h_i(\alpha_0)]_t$ is non-zero and $\mathcal{S}$ can first randomly sample $[h_i(r)]_t$ as above and recompute $[h_i(\alpha_0)]_t$ by the linear combination of $[h_i(r)]_t$ and $\{[h_i(\alpha_\ell)]_t\}_{\ell=1}^{2N'}$. The distribution of $[h_i(r)]_t$ and $\{[h_i(\alpha_\ell)]_t\}_{\ell=0}^{2N'}$ remain unchanged, so the distributions of $\mathbf{Hyb}_{2.3}$ and $\mathbf{Hyb}_{2.2}$ are identical.

$\mathbf{Hyb}_{2.4}$: In this hybrid, for each corrupted dealer $P_i \in \mathcal{D}$. If $r \notin \{\alpha_1, \ldots, \alpha_{N'}\}$, $h_i \neq f_i \cdot g_i$ but $h_i(r) = f_i(r) \cdot g_i(r)$, $\mathcal{S}$ aborts the simulation. By the Schwartz-Zipple lemma, the probability is at most $\frac{2N'n}{2^\kappa - N'}$, which is negligible in the security parameter κ. Thus, the distributions of $\mathbf{Hyb}_{2.4}$ and $\mathbf{Hyb}_{2.3}$ are statistically close.

$\mathbf{Hyb}_{2.5}$: In this hybrid, for each honest dealer in $\mathcal{D}$, we change the generation of honest parties' degree-t Shamir sharings $[a_i^{(k)}]_t, [b_i^{(k)}]_t$ as follows. At a high level, $\mathcal{S}$ will first randomly samples $[f^{(k)}(\cdot)]_t, [g^{(k)}(\cdot)]_t$ and then computes $[a_i^{(k)}]_t, [b_i^{(k)}]_t$. To be more concrete,

Let $\mathcal{D} = \{P_{j_1}, \ldots, P_{j_L}\}$ and $\mathcal{D}' = \{P_{j_1}, \ldots, P_{j_{L'+1}}\}$, then for each corrupted dealer $P_{j_i} \in \mathcal{D}$:

- If $P_{j_i} \in \mathcal{D}'$, set $[f^{(k)}(\alpha_i)]_t = [a_i^{(k)}]_t$.
- If $P_{j_i} \notin \mathcal{D}'$, $\mathcal{S}$ first uses corrupted parties' shares of $\{[f^{(k)}(\alpha_i)]_t\}_{i \in \mathcal{D}'}$ to compute their shares of $\{[f^{(k)}(\alpha_i)]_t\}_{i \in \mathcal{D} \setminus \mathcal{D}'}$, then randomly samples $[f^{(k)}(\alpha_i)]_t$ based on shares of corrupted parties.

$\mathcal{S}$ computes corrupted parties' shares of $[f^{(k)}(\beta_j)]_t$ for all $j \in [(L+1)/2 - t], k \in [N']$ by the linear combination of $\{[f^{(k)}(\alpha_i)]_t\}_{i \in \mathcal{D}'}$. Then $\mathcal{S}$ randomly samples the whole $[f^{(k)}(\beta_j)]_t$ based on shares of corrupted parties. Finally, $\mathcal{S}$ randomly samples a degree-L' polynomial $f^{(k)}(\cdot)$ based on $(L+1)/2 - t$ points $f^{(k)}(\beta_j)$ and corrupted party P_{j_i}'s $f^{(k)}(\alpha_i)$.

With $f^{(k)}(\cdot)$, for each honest dealer $P_{j_i} \in \mathcal{D}$:

- If $P_{j_i} \in \mathcal{D}'$, set $[a_i^{(k)}]_t = [f^{(k)}(\alpha_i)]_t$.
- If $P_{j_i} \notin \mathcal{D}'$, randomly sample $[a_i^{(k)}]_t$ based on shares of corrupted parties.

We do the same thing for $[b_i^{(k)}]_t$. Then $\mathcal{S}$ computes $c_i^{(k)} = a_i^{(k)} \cdot b_i^{(k)}$ and randomly samples the whole $[c_i^{(k)}]_t$ based on the secret $c_i^{(k)}$ and shares of corrupted parties. Similar to the previous analysis in Appendix C.5, the distributions of $\mathbf{Hyb}_{2.5}$ and $\mathbf{Hyb}_{2.4}$ are identical.

$\mathbf{Hyb}_{2.6}$: In this hybrid, for all $i \in [L'+2, L]$ where P_{j_i} is honest dealer, $\mathcal{S}$ first randomly samples $[f^{(k)}(\alpha_i) + a_i^{(k)}]_t, [g^{(k)}(\alpha_i) + b_i^{(k)}]_t$ based on shares of corrupted parties. Then $\mathcal{S}$ computes $[a_i^{(k)}]_t = [f^{(k)}(\alpha_i) + a_i^{(k)}]_t - [f^{(k)}(\alpha_i)]_t, [b_i^{(k)}]_t = [g^{(k)}(\alpha_i) + b_i^{(k)}]_t - [g^{(k)}(\alpha_i)]_t$. The distributions of $\mathbf{Hyb}_{2.6}$ and $\mathbf{Hyb}_{2.5}$ are identical.

$\mathbf{Hyb}_{2.7}$: In this hybrid, for all $i \in [L'+2, L]$ where P_{j_i} is corrupted dealer, $\mathcal{S}$ first randomly samples values as $f^{(k)}(\alpha_i) + a_i^{(k)}, g^{(k)}(\alpha_i) + b_i^{(k)}$, then computes $f^{(k)}(\alpha_i) = f^{(k)}(\alpha_i) + a_i^{(k)} - a_i^{(k)}, g^{(k)}(\alpha_i) = g^{(k)}(\alpha_i) + b_i^{(k)} - b_i^{(k)}$ and randomly samples the whole $[f^{(k)}(\alpha_i)]_t, [g^{(k)}(\alpha_i)]_t$ based on the secrets $f^{(k)}(\alpha_i), g^{(k)}(\alpha_i)$ and shares of corrupted parties. The distributions of $\mathbf{Hyb}_{2.7}$ and $\mathbf{Hyb}_{2.6}$ are identical.

$\mathbf{Hyb}_{2.8}$: In this hybrid, we further change the generation of honest parties' degree-t Shamir sharings $[c_i^{(k)}]_t$ as follows. At a high level, $\mathcal{S}$ will first randomly samples $[h^{(k)}(\cdot)]_t$ and then computes $[c_i^{(k)}]_t$. To be more concrete,

For each corrupted dealer $P_{j_i} \in \mathcal{D}$:

- If $P_{j_i} \in \mathcal{D}'$, set $[h^{(k)}(\alpha_i)]_t = [c_i^{(k)}]_t$.
- If $P_{j_i} \notin \mathcal{D}'$, $\mathcal{S}$ computes $h^{(k)}(\alpha_i) = f^{(k)}(\alpha_i) \cdot g^{(k)}(\alpha_i)$ and randomly samples the whole $[h^{(k)}(\alpha_i)]_t$ based on the secret $h^{(k)}(\alpha_i)$ and shares of corrupted parties.

$\mathcal{S}$ computes corrupted parties' shares of $\{[h^{(k)}(\beta_j)]_t\}_{j=1}^{(L+1)/2-t}$ by the linear combination of $\{h^{(k)}(\alpha_i)\}_{i\in\mathcal{D}}$. Then $\mathcal{S}$ randomly samples the whole $[h^{(k)}(\beta_j)]_t$ for all $j \in [(L+1)/2-t]$ based on share of corrupted parties and secrets $h^{(k)}(\beta_j) = f^{(k)}(\beta_j) \cdot g^{(k)}(\beta_j)$. Finally, $\mathcal{S}$ computes $h^{(k)}(\alpha_i) = f^{(k)}(\alpha_i) \cdot g^{(k)}(\alpha_i)$ and randomly samples $[h^{(k)}(\alpha_i)]_t$ based on $h^{(k)}(\alpha_i)$ and shares of corrupted parties for the first $(L-1)/2+t-t'$ honest dealers $P_{j_i} \in \mathcal{D}$. With these L degree-t Shamir sharings, $\mathcal{S}$ interpolates the whole $[h^{(k)}(\cdot)]$ and computes each honest dealer P_{j_i}'s $[c_i^{(k)}]_t$ as follows:

- If $P_{j_i} \in \mathcal{D}'$, set $[c_i^{(k)}]_t = [h^{(k)}(\alpha_i)]_t$.
- If $P_{j_i} \notin \mathcal{D}'$, $\mathcal{S}$ computes $[c_i^{(k)}]_t = [h^{(k)}(\alpha_i)]_t - (f^{(k)}(\alpha_i)+a_i^{(k)})(g^{(k)}(\alpha_i)+b_i^{(k)}) + (f^{(k)}(\alpha_i)+a_i^{(k)})[b_i^{(k)}]_t + (g^{(k)}(\alpha_i)+b_i^{(k)})[a_i^{(k)}]_t$.

Similar to the previous analysis in Appendix C.5, the distributions of $\mathbf{Hyb}_{2.8}$ and $\mathbf{Hyb}_{2.7}$ are identical.

$\mathbf{Hyb}_{2.9}$: In this hybrid, for each honest dealer in $\mathcal{D}$, we no longer generate honest parties' sharings since they are never used. The distributions of $\mathbf{Hyb}_{2.9}$ and $\mathbf{Hyb}_{2.8}$ are identical.

$\mathbf{Hyb}_3$: In the following, we focus on the simulation of $\Pi_{\texttt{randShareZero}}$:

$\mathbf{Hyb}_{3.1}$: In this hybrid, during the $\Pi_{\texttt{Sh2tZero}}$, for each honest dealer, $\mathcal{S}$ first randomly samples corrupted parties' degree-$2t$ row and degree-$(t+(t-1)/2)$ column polynomials, then randomly samples the whole degree-$(2t, t+(t-1)/2)$ bivariate polynomials based on corrupted parties' row, column polynomials and the input degree-$2t$ Shamir sharings of zero. According to the property of Shamir sharings, the distributions of $\mathbf{Hyb}_{3.1}$ and $\mathbf{Hyb}_{2.9}$ are identical.

$\mathbf{Hyb}_{3.2}$: In this hybrid, during the $\Pi_{\texttt{Sh2tZero}}$, when the dealer is honest, we delay the generation of honest parties' row and column polynomials until the commitment is computed. For each honest party P_i:

- For each party P_j, if P_j is honest party, $\mathcal{S}$ randomly samples a vector of values as $\boldsymbol{f}_*^{(i)}(\alpha_j)$ and computes $\mathsf{cm\text{-}row}_j^{(i)}$ accordingly. Otherwise, $\mathcal{S}$ uses corrupted party P_j's column polynomial to compute P_i's $f_\ell^{(i)}(\alpha_j), \bar{f}_\ell^{(i)}(\alpha_j)$ for all $\ell \in [L]$, then $\mathcal{S}$ follows the protocol to get $\mathsf{cm\text{-}row}_j^{(i)}$.
- $\mathcal{S}$ randomly samples values as P_i's $\mathsf{cm\text{-}col}_\ell^{(i)}$.

Upon generating all $\mathsf{cm\text{-}row}_j^{(i)}, \mathsf{cm\text{-}col}_\ell^{(i)}$ for all honest party P_i, $\mathcal{S}$ additionally checks whether these values have been mapped to some inputs queried by $\mathcal{A}$, if true, $\mathcal{S}$ aborts the simulation. For honest dealer and each honest P_i's $\mathsf{cm\text{-}row}_j^{(i)}$, $\mathcal{S}$ samples $2L-1$ randomly values for the Merkle tree. Then there are $(2L-1)(2t+1)^2$ values in total. For $\mathsf{cm\text{-}col}_\ell^{(i)}$, there are $L(2t+1)^2$ values in total. Assuming that $\mathcal{A}$ can query $\texttt{poly}(\kappa)$ times, the probability of collision is at most:

$$\frac{(3L-1)(2t+1)^2}{2^\kappa - \texttt{poly}(\kappa) - (3L-1)(2t+1)^2} \leq \frac{\texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)}$$

which is negligible in the security parameter. The distributions of $\mathbf{Hyb}_{3.2}$ and $\mathbf{Hyb}_{3.1}$ are computationally indistinguishable.

$\mathbf{Hyb}_{3.3}$: In this hybrid, during the $\Pi_{\texttt{Sh2tZero}}$, when the dealer is honest, we further delay the generation of honest parties' row and column polynomials until the termination procedure. When an honest party P_i needs to send $\{f_\ell^{(i)}(\alpha_j), \bar{f}_\ell^{(i)}(\alpha_j)\}$ to corrupted party P_j, $\mathcal{S}$ computes $\{f_\ell^{(i)}(\alpha_j), \bar{f}_\ell^{(i)}(\alpha_j)\} = \{g_\ell^{(j)}(\alpha_i), \bar{g}_\ell^{(j)}(\alpha_i)\}$ and sends them to P_j on behalf of P_i. For each honest party P_i, when he receives $\{g_\ell^{(j)}(\alpha_i), \bar{g}_\ell^{(j)}(\alpha_i)\}$ from honest P_j, $\mathcal{S}$ directly consider it is correct. When he receives them from corrupted party P_j, $\mathcal{S}$ follows the protocol to check whether it is correct. When P_i receives $t+(t+1)/2$ correct

$\{g_\ell^{(j)}(\alpha_i), \bar{g}_\ell^{(j)}(\alpha_i)\}$, $\mathcal{S}$ considers P_i has reconstructed his column polynomials. Note that when corrupted parties send incorrect shares to an honest party, since the probability that $\mathcal{A}$ can find a second pre-image which can cause collision is negligible in the security parameter κ, $\mathcal{S}$ can detect it with overwhelming probability. Thus, the distributions of $\mathbf{Hyb}_{3.3}$ and $\mathbf{Hyb}_{3.2}$ are computational indistinguishable.

$\mathbf{Hyb}_{3.4}$: In this hybrid, during the $\Pi_{\texttt{randShareZero}}$, for each honest dealer in $\mathcal{D}$, $\mathcal{S}$ delays the generation of honest parties' row and column polynomials until the set $\mathcal{D}$ is determined. For each honest dealer not in $\mathcal{D}$, $\mathcal{S}$ does not generate honest parties' row and column polynomials. Since these honest parties' column and row polynomials have not been used so far, the distributions of $\mathbf{Hyb}_{3.4}$ and $\mathbf{Hyb}_{3.3}$ are identical.

$\mathbf{Hyb}_{3.5}$: In this hybrid, let $\mathcal{H}$ be the set of the honest parties in $\mathcal{D}$ and $\mathcal{H}'$ be the set of the first $t+1$ parties in $\mathcal{H}$, we change the generation of honest parties' sharings for dealers in $\mathcal{H}'$. Recall that $\{[o_{\ell,j}]_{2t}\}_{j=1}^{t+1} = [o_\ell^{\mathcal{H}}]_{2t} + [o_\ell^{\mathcal{C}orr}]_{2t}$, since $\boldsymbol{M}$ is a Vandermonde matrix, there is a one to one map between $[o_\ell^{\mathcal{H}}]_{2t}$ and $\{[o_\ell^{(i)}]_{2t}\}_{i\in\mathcal{H}'}$. We first randomly sample $[o_\ell^{\mathcal{H}}]_{2t}$ based on shares of corrupted parties and then compute $\{[o_\ell^{(i)}]_{2t}\}_{i\in\mathcal{H}'}$. This does not change the distribution of $[o_\ell^{\mathcal{H}}]_{2t}, \{[o_\ell^{(i)}]_{2t}\}_{i\in\mathcal{H}'}$, so the distributions of $\mathbf{Hyb}_{3.5}$ and $\mathbf{Hyb}_{3.4}$ are identical.

$\mathbf{Hyb}_4$: In the following, we focus on the simulation of $\Pi_{\texttt{randSh}}$:

$\mathbf{Hyb}_{4.1}$: In this hybrid, for each honest dealer, $\mathcal{S}$ first randomly samples shares of corrupted parties, then randomly samples the whole degree-t Shamir sharings based on shares of corrupted parties. According to the property of Shamir sharings, the distributions of $\mathbf{Hyb}_{4.1}$ and $\mathbf{Hyb}_{3.5}$ are identical.

$\mathbf{Hyb}_{4.2}$: In this hybrid, for each honest dealer in $\mathcal{D}$, $\mathcal{S}$ delays the generation of honest parties' sharings until the set $\mathcal{D}$ is determined. For each honest dealer not in $\mathcal{D}$, $\mathcal{S}$ does not generate honest parties' sharings. Since these honest parties' sharings have not been used so far, the distributions of $\mathbf{Hyb}_{4.2}$ and $\mathbf{Hyb}_{4.1}$ are identical.

$\mathbf{Hyb}_{4.3}$: In this hybrid, let $\mathcal{H}$ be the set of the first $t+1$ honest parties in $\mathcal{D}$, we change the generation of honest parties' sharings for dealers in $\mathcal{H}$. Since $\boldsymbol{M}$ is a Vandermonde matrix, any $(t+1)\times(t+1)$ sub-matrix of $\boldsymbol{M}$ is invertible, then for all $\ell \in [N']$, given shares of $\{[s_\ell^{(i)}]_t\}_{i\notin\mathcal{H}}$, there is a one to one map between $\{[r_{\ell,i}]_t\}_{i=1}^{t+1}$ and $\{[s_\ell^{(i)}]_t\}_{i\in\mathcal{H}}$. We first randomly sample $\{[r_{\ell,i}]_t\}_{i=1}^{t+1}$ based on shares of corrupted parties and then compute $\{[s_\ell^{(i)}]_t\}_{i\in\mathcal{H}}$. Since this does not change the distribution of $\{[r_{\ell,i}]_t\}_{i=1}^{t+1}$ and $\{[s_\ell^{(i)}]_t\}_{i\in\mathcal{H}}$, the distributions of $\mathbf{Hyb}_{4.3}$ and $\mathbf{Hyb}_{4.2}$ are identical.

$\mathbf{Hyb}_{4.4}$: In this hybrid, for each honest dealer in $\mathcal{D}$, we no longer generate honest parties' degree-t Shamir sharings. Since they are never used in the simulation, the distributions of $\mathbf{Hyb}_{4.4}$ and $\mathbf{Hyb}_{4.3}$ are identical.

$\mathbf{Hyb}_5$: In the following, we focus on the simulation of $\Pi_{\texttt{tripleDN}}$:

$\mathbf{Hyb}_{5.1}$: In this hybrid, we change the generation of each secret $r_\ell^{(i,j)}$ by letting $\mathcal{S}$ first randomly sample a random value as $a_\ell^{(i,j)}\cdot b_\ell^{(i,j)} + r_\ell^{(i,j)}$, then compute $r_\ell^{(i,j)} = a_\ell^{(i,j)}\cdot b_\ell^{(i,j)} + r_\ell^{(i,j)} - a_\ell^{(i,j)}\cdot b_\ell^{(i,j)}$. This does not change the distribution of $\{[a_\ell^{(i,j)}]_t, [b_\ell^{(i,j)}]_t, [r_\ell^{(i,j)}]_t\}$, the distributions of $\mathbf{Hyb}_{5.1}$ and $\mathbf{Hyb}_{4.4}$ are identical.

$\mathbf{Hyb}_{5.2}$: In this hybrid, during the $\Pi_{\texttt{tripleKingDN}}$, recall that each $[o_\ell]_{2t}$ can be rewritten as $[o_\ell]_{2t} = [o_\ell^{\mathcal{H}}]_{2t} + [o_\ell^{\mathcal{C}orr}]_{2t}$, where $[o_\ell^{\mathcal{H}}]_{2t}$ is randomly sampled based on the secret 0 and shares of corrupted parties in $\mathbf{Hyb}_{5.1}$. Here we change the generation method of $[o_\ell^{\mathcal{H}}]_{2t}$. $\mathcal{S}$ first computes corrupted parties' shares of $[z'_\ell]_{2t} = [a_\ell]_t\cdot[b_\ell]_t + [r_\ell]_t + [o_\ell^{\mathcal{H}}]_{2t}$, then randomly samples the whole $[z'_\ell]_{2t}$ based on the secret $a_\ell\cdot b_\ell + r_\ell$ and shares of corrupted parties. Finally, $\mathcal{S}$ computes $[o_\ell^{\mathcal{H}}]_{2t} = [z'_\ell]_{2t} - [a_\ell]_t\cdot[b_\ell]_t - [r_\ell]_t$. This does not change the distribution of $[o_\ell^{\mathcal{H}}]_{2t}$, so the distributions of $\mathbf{Hyb}_{5.2}$ and $\mathbf{Hyb}_{5.1}$ are identical.

$\mathbf{Hyb}_{5.3}$: In this hybrid, we delay the generation of honest parties' shares of $([a_\ell]_t, [b_\ell]_t, [r_\ell]_t)$ until the set $\mathcal{D}$ is determined. When an honest party who terminates $\Pi_{\texttt{randShareZero}}$ needs to send his shares of $[z_\ell]_t$ to P_{king}, $\mathcal{S}$ computes this party's shares of $[z_\ell]_t$ by $[z'_\ell]_t + [o_\ell^{\mathcal{C}orr}]_{2t}$. The distributions of $\mathbf{Hyb}_{5.3}$ and $\mathbf{Hyb}_{5.2}$ are identical.

$\mathbf{Hyb}_{5.4}$: In this hybrid, we change the way of sampling $[r_\ell]_t$. If P_{king} first receives $2t+1$ shares of $[z_\ell]_{2t}$ and succeeds in broadcasting the secret z_ℓ, $\mathcal{S}$ follows the protocol to compute corrupted parties' shares of $[c_\ell]_t$, then randomly samples the whole $[c_\ell]_t$ based on the shares of corrupted parties and secret $c_\ell = a_\ell\cdot b_\ell + z_\ell - z'_\ell$. Finally, $\mathcal{S}$ computes $[r_\ell]_t = z_\ell - [c_\ell]_t$. This does not change the distribution of $[r_\ell]_t$, the distributions of $\mathbf{Hyb}_{5.4}$ and $\mathbf{Hyb}_{5.3}$ are identical.

$\mathbf{Hyb}_{5.5}$: In this hybrid, we no longer generate honest parties' shares of $[o_\ell]_{2t}$ since they are never used. The distributions of $\mathbf{Hyb}_{5.5}$ and $\mathbf{Hyb}_{5.4}$ are identical.

$\mathbf{Hyb}_6$: In the following, we focus on the simulation of each $\Pi_{\texttt{tripleVerify}}$ in $\Pi_{\texttt{tripleDN}}$:

$\mathbf{Hyb}_{6.1}$: In this hybrid, let $d^{(k)}(\cdot)$ be defined as above. Then $d = h - f \cdot g$. If $r \notin \{\alpha_1, \ldots, \alpha_N\}, d \not\equiv 0$ and $d(r) = 0$, $\mathcal{S}$ aborts the simulation. By the Schwartz-Zipple lemma, the probability is at most $\frac{2Nn}{2^\kappa - N}$, which is negligible in the security parameter κ. Thus, the distributions of $\mathbf{Hyb}_{6.1}$ and $\mathbf{Hyb}_{5.5}$ are statistically close.

$\mathbf{Hyb}_{6.2}$: In this hybrid, we delay the generation of $([a_i^{(k)}]_t, [b_i^{(k)}]_t)$ for all $k \in [2t+1], i \in [N'+1, 2N']$:

For $[a_i^{(k)}]_t, [b_i^{(k)}]_t$, $\mathcal{S}$ first follows the protocol to compute corrupted parties' shares of $[f^{(k)}(\alpha_i) + a_i^{(k)}]_t, [g^{(k)}(\alpha_i) + b_i^{(k)}]_t$ for all $k \in [2t+1], i \in [N'+1, 2N']$. Then $\mathcal{S}$ randomly samples values as $f^{(k)}(\alpha_i) + a_i^{(k)}, g^{(k)}(\alpha_i) + b_i^{(k)}$ and recomputes $a_i^{(k)}, b_i^{(k)}$. Finally $\mathcal{S}$ randomly samples the whole $[a_i^{(k)}]_t, [b_i^{(k)}]_t$ based on the secrets $a_i^{(k)}, b_i^{(k)}$ and shares of corrupted parties.

To compute $[c_i^{(k)}]_t$, $\mathcal{S}$ first computes $c_i^{(k)} = a_i^{(k)} \cdot b_i^{(k)} + d_i^{(k)}$ and then samples the whole $[c_i^{(k)}]_t$ based on the secret $c_i^{(k)}$ and shares of corrupted parties. The distributions of $([a_i^{(k)}]_t, [b_i^{(k)}]_t, [c_i^{(k)}]_t)$ remain unchanged, so the distributions of $\mathbf{Hyb}_{6.2}$ and $\mathbf{Hyb}_{6.1}$ are identical.

$\mathbf{Hyb}_{6.3}$: In this hybrid, we delay the generation of $([c_i^{(k)}]_t)$ for all $k \in [2t+1], i \in [N'+1, 2N']$. $\mathcal{S}$ first follows the protocol to compute corrupted parties' shares of $[h^{(k)}(\alpha_i)]_t$, then randomly samples the whole $[h^{(k)}(\alpha_i)]_t$ based on the secret $h^{(k)}(\alpha_i) = f^{(k)}(\alpha_i) \cdot g^{(k)}(\alpha_i) + d^{(k)}(\alpha_i)$ and shares of corrupted parties. Finally $\mathcal{S}$ computes $[c_i^{(k)}]_t = [h^{(k)}(\alpha_i)]_t - (f^{(k)}(\alpha_i) + a_i^{(k)}) \cdot (g^{(k)}(\alpha_i) + b_i^{(k)}) + (f^{(k)}(\alpha_i) + a_i^{(k)})[b^{(k)}]_t + (g^{(k)}(\alpha_i) + b_i^{(k)})[a^{(k)}]_t$. The distributions of $\mathbf{Hyb}_{6.3}$ and $\mathbf{Hyb}_{6.2}$ are identical.

$\mathbf{Hyb}_{6.4}$: In this hybrid, we change the generation of $([a_0^{(k)}]_t, [b_0^{(k)}]_t, [c_0^{(k)}]_t)$. If $r \notin \{\alpha_1, \ldots, \alpha_{N'}\}$, $f^{(k)}(r)$ is a linear combination of $\{f^{(k)}(\alpha_i)\}_{i=0}^{N'}$ and the coefficient of $f(\alpha_0) = a_0$ is non-zero. Then $f(\alpha_0)$ also can be computed by the linear combination of $\{f^{(k)}(\alpha_i)\}_{i=1}^{N'}$ and $f^{(k)}(r)$. We let $\mathcal{S}$ first compute corrupted parties' shares of $[f^{(k)}(r)]_t$ by the linear combination of $\{[f^{(k)}(\alpha_i)]_t\}_{i=0}^{N'}$, then randomly samples the whole $[f^{(k)}(r)]_t$ based on shares of corrupted parties. $\mathcal{S}$ does the same thing to generate $[g^{(k)}(r)]_t$. Finally, $\mathcal{S}$ computes $[a_0^{(k)}]_t, [b_0^{(k)}]_t$ by the linear combination of $\{[f^{(k)}(\alpha_i)]_t, [g^{(k)}(\alpha_i)]_t\}_{i=1}^{N}$ and $\{[f^{(k)}(r)]_t, [g^{(k)}(r)]_t\}$. $\mathcal{S}$ also computes $c_0^{(k)} = a_0^{(k)} \cdot b_0^{(k)} + d_0^{(k)}$ and randomly samples the whole $[c_0^{(k)}]_t$ based on the secret $c_0^{(k)}$ and shares of corrupted parties. The distributions of $\mathbf{Hyb}_{6.4}$ and $\mathbf{Hyb}_{6.3}$ are identical.

$\mathbf{Hyb}_{6.5}$: In this hybrid, we change the generation of $[h^{(k)}(r)]_t$. When $r \notin \{\alpha_0, \ldots, \alpha_{2N'}\}$, we let $\mathcal{S}$ randomly sample $[h^{(k)}(r)]_t$ based on the secret $h^{(k)}(r) = f^{(k)}(r) \cdot g^{(k)}(r) + d^{(k)}(r)$ and shares of corrupted parties. Since $[h^{(k)}(r)]_t$ is a linear combination of $\{[h^{(k)}(\alpha_i)]_t\}_{i=0}^{2N'}$ and when $r \notin \{\alpha_0, \ldots, \alpha_{2N'}\}$, the coefficient of $[h^{(k)}(\alpha_0)]_t$ is non-zero. $\mathcal{S}$ computes $[h^{(k)}(\alpha_0)]_t$ by a proper linear combination of $\{[h^{(k)}(\alpha_i)]_t\}_{i=1}^{2N'}$ and $[h^{(k)}(r)]_t$. Note that in $\mathbf{Hyb}_{6.4}$, $[h^{(k)}(r)]_t$ is a random degree-t Shamir sharings given shares of corrupted parties and the secret $h^{(k)}(r)$. The distributions of $\mathbf{Hyb}_{6.5}$ and $\mathbf{Hyb}_{6.4}$ are identical.

$\mathbf{Hyb}_{6.6}$: In this hybrid, we no longer generate honest parties' shares of $\{[a_0^{(k)}]_t, [b_0^{(k)}]_t, [c_0^{(k)}]_t\} \cup \{[a_i^{(k)}]_t, [b_i^{(k)}]_t, [c_i^{(k)}]_t\}_{i=N'+1}^{2N'}$ since they are never used. The distributions of $\mathbf{Hyb}_{6.6}$ and $\mathbf{Hyb}_{6.5}$ are identical.

$\mathbf{Hyb}_7$: In the following, we focus on the simulation of each $\Pi_{\texttt{faultLoc}}$ in $\Pi_{\texttt{tripleDN}}$:

$\mathbf{Hyb}_{7.1}$: In this hybrid, when P_{king} is corrupted, for each honest dealer D who distributes degree-$2t$ Shamir sharings of zero, we delay the generation of honest parties' row and column polynomials until now. For each honest party P_i who terminates $\Pi_{\texttt{randShareZero}}$, $\mathcal{S}$ computes his columns $g_\ell^{(i)}(y), \bar{g}_\ell^{(i)}(y)$ distributed by D and sends them to P_{king}. When P_{king} queries the random oracle with honest party P_i's $g_\ell^{(i)}(\beta_*), \bar{g}_\ell^{(i)}(\beta_*)$, $\mathcal{S}$ returns the random value $\mathsf{cm\text{-}col}_\ell^i$ he sampled during $\Pi_{\texttt{Sh2tZero}}$. If these values have been previously queried by the corrupted parties, $\mathcal{S}$ aborts the simulation. Assuming that corrupted parties can query the random oracle for $\texttt{poly}(\kappa)$ times, for all honest dealers and all parties, the probability of abortion is at most:

$$(2t+1) \cdot n \cdot \frac{N}{n} \cdot \frac{\texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)} \leq \frac{N \cdot n \cdot \texttt{poly}(\kappa)}{2^\kappa - \texttt{poly}(\kappa)}$$

which is negligible in the security parameter κ. Thus, the distributions of $\mathbf{Hyb}_{7.1}$ and $\mathbf{Hyb}_{6.6}$ are computationally indistinguishable.

$\mathbf{Hyb}_{7.2}$: In this hybrid, we delay the generation of honest parties' shares of $([a_\ell]_t, [b_\ell]_t, [r_\ell]_t)$. When honest parties need to send their sharings to corrupted P_{king}, $\mathcal{S}$ randomly the whole $([a_\ell]_t, [b_\ell]_t, [r_\ell]_t)$ based on shares of corrupted parties. Then $\mathcal{S}$ computes honest parties' shares of $([a_\ell]_t, [b_\ell]_t, [r_\ell]_t)$ and sends

them to corrupted P_{king}. Since $([a_\ell]_t, [b_\ell]_t, [r_\ell]_t)$ are random degree-t Shamir sharings, the distributions of $\mathbf{Hyb}_{7.2}$ and $\mathbf{Hyb}_{7.1}$ are identical.

$\mathbf{Hyb}_{7.3}$: In this hybrid, for honest P_{king}, $\mathcal{S}$ only samples $([a_{\mathsf{idx}}]_t, [b_{\mathsf{idx}}]_t, [r_{\mathsf{idx}}]_t)$ when he needs to broadcast them on behalf of P_{king}. The distributions of $\mathbf{Hyb}_{7.3}$ and $\mathbf{Hyb}_{7.2}$ are identical.

$\mathbf{Hyb}_8$: In the following, we focus on the simulation of $\Pi_{\texttt{main-GOD}}$:

$\mathbf{Hyb}_{8.1}$: In this hybrid, in the input phase, for each honest dealer, after randomly sampling shares of parties in $\mathcal{C}orr'$, $\mathcal{S}$ delays the generation of the rest of honest parties' shares until the set $\mathcal{D}$ is determined. Since these honest parties' shares are not used in the input phase, the distributions of $\mathbf{Hyb}_{8.1}$ and $\mathbf{Hyb}_{7.3}$ are identical.

$\mathbf{Hyb}_{8.2}$: In this hybrid, in the input phase, for each honest dealer not in $\mathcal{D}$, $\mathcal{S}$ does not generate shares of honest parties. Since these honest dealers' sharings are never used, the distributions of $\mathbf{Hyb}_{8.2}$ and $\mathbf{Hyb}_{8.1}$ are identical.

$\mathbf{Hyb}_{8.3}$: In this hybrid, in the computation phase, $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{SubCktEval}}$ to simulate each $\Pi_{\texttt{SubCktEval}}$. Similar to the analysis in Appendix C.8, the distributions of $\mathbf{Hyb}_{8.3}$ and $\mathbf{Hyb}_{8.2}$ are identical.

$\mathbf{Hyb}_{8.4}$: In this hybrid, $\mathcal{S}$ first computes $y = f(x_1, \ldots, x_n)$, then computes the whole $[y]_t$ based on secret y and shares of parties in $\mathcal{C}orr'$. For honest parties not in $\mathcal{C}orr' \setminus \mathcal{C}orr$, we also no longer generate their shares during each $\Pi_{\texttt{SubCktEval}}$ since they are never used. The distributions of $\mathbf{Hyb}_{8.4}$ and $\mathbf{Hyb}_{8.3}$ are identical.

$\mathbf{Hyb}_9$: In this hybrid, for honest parties not in $\mathcal{C}orr' \setminus \mathcal{C}orr$, we no longer generate their shares of Beaver triples since they are never used. The distributions of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_{8.4}$ are identical.

$\mathbf{Hyb}_{10}$: In this hybrid, $\mathcal{S}$ sends the inputs of corrupted parties and the set $\mathcal{D}$ to $\mathcal{F}_{\mathsf{AMPC}}$ and receives the output y. Since $\mathcal{F}_{\mathsf{AMPC}}$ computes y in the same way as $\mathcal{S}$, $\mathcal{S}$ no longer needs honest parties' inputs. The distributions of $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ are identical.

Note that $\mathbf{Hyb}_{10}$ corresponds to the ideal world, then $\Pi_{\texttt{main-GOD}}$ securely computes $\mathcal{F}_{\mathsf{AMPC}}$.