Towards Achieving Asynchronous MPC with Linear Communication and Optimal Resilience

Vipul Goyal[1], Chen-Da Liu-Zhang[2], and Yifan Song[3]

[1] vipul@cmu.edu, NTT Research and Carnegie Mellon University
[2] chen-da.liuzhang@hslu.ch, Luzern University of Applied Sciences and Arts & Web3 Foundation
[3] yfsong@mail.tsinghua.edu.cn, Tsinghua University and Shanghai Qi Zhi Institute

Abstract. Secure multi-party computation (MPC) allows a set of n parties to jointly compute a function over their private inputs. The seminal works of Ben-Or, Canetti and Goldreich [STOC '93] and Ben-Or, Kelmer and Rabin [PODC '94] settled the feasibility of MPC over asynchronous networks. Despite the significant line of work devoted to improving the communication complexity, current protocols with information-theoretic security and optimal resilience $t < n/3$ communicate $\Omega(n^4C)$ field elements for a circuit with C multiplication gates. In contrast, synchronous MPC protocols with $\Omega(nC)$ communication have long been known.

In this work we make progress towards closing this gap. We provide a novel MPC protocol that makes black-box use of an asynchronous complete secret-sharing (ACSS) protocol, where the cost per multiplication reduces to the cost of distributing a constant number of sharings via ACSS, improving a linear factor over the state of the art by Choudhury and Patra [IEEE Trans. Inf. Theory '17].

Instantiating ACSS with the protocol by Choudhury and Patra [J. Crypto '23] we achieve an MPC protocol with $\mathcal{O}(n^3C)$ communication. Moreover, with a recent concurrent work achieving ACSS with linear cost per sharing, we achieve an MPC with $\mathcal{O}(nC)$ communication.

1 Introduction

In the problem of secure multi-party computation (MPC), a set of n parties aim to compute a function over their private inputs, in such a way that the parties' inputs remain secret, and the computed output is correct, even when a subset of the parties are dishonest.

Secure multi-party computation [Yao82, GMW87, BGW88, CCD88, RB89] has been extensively studied in the so-called *synchronous* model, where parties have access to synchronized clocks and there is an upper bound on the network communication delay. This model is theoretically interesting and allows to provide clean protocols, but fails to capture real-world network behaviors such as the Internet, which typically have an unstable delay and are asynchronous. This gave rise to the *asynchronous* network model, which allows messages sent to be arbitrarily delayed and delivered out of order, and protocols in this setting do not need to rely on any timing assumptions.

One of the main challenges in asynchronous MPC protocols is that one cannot distinguish between a dishonest party not sending a message, or an honest party that sent a message that is being delayed by the adversary. As a result, such protocols are message-driven, and parties need to make progress after seeing enough messages from other parties: typically an honest party can only afford to receive messages from at most $n - t$ parties (where t is the corruption threshold) to avoid getting stuck, since the other missing messages may come from dishonest parties and have never been sent. In turn, it could also be that the missing messages are from honest parties, and the protocol needs to continue. Due to this, synchronous protocols completely fail when executed over an asynchronous network, given that the security of these protocols require receiving the messages from all honest parties.

Due to these challenges, asynchronous protocols require designing further techniques. For example, the classic Fischer, Lynch and Patterson [FLP85] rule out the possibility of deterministic protocols in the asynchronous setting, even for basic tasks such as Byzantine agreement which do not have any privacy requirements. Turned around, asynchronous protocols also inherently achieve weaker security guarantees. To give another example, one can show that the optimal achievable corruption tolerance for MPC protocols (and guaranteed output delivery) in the asynchronous setting is $t < n/3$ [BKR94, ADS20], even assuming correlated randomness setup, in both the cryptographic and information-theoretic setting; and perfect security is possible if and only if $t < n/4$ [BCG93]. This is in contrast to the synchronous setting where MPC protocols with guaranteed output delivery can be achieved for $t < n/2$ [RB89, CDD$^+$99] with setup (and negligible error), and $t < n/3$ with perfect security [BGW88].

1.1 Communication-Complexity of Asynchronous MPC

The communication complexity in MPC has been the subject of a very significant line of works. While efficient synchronous MPC solutions with optimal resilience have been known for a long time, even with linear $O(n)$ field elements per multiplication gate (see e.g. [HN06, DI06, BTH08, BFO12, GLS19, GSZ20]), asynchronous MPC protocols still feature higher asymptotic communication complexities.

In the information-theoretic setting, the first protocol with optimal resilience $t < n/3$ was provided by Ben-Or, Kelmer and Rabin [BKR94], and later improved by Patra, Choudhury and Rangan [PCR10, PCR08] to $O(n^5)$ field elements per multiplication, and recently further improved by Choudhury and Patra [CP23] to $O(n^4)$ field elements per multiplication.

In the cryptographic setting, there are several communication-efficient protocols with optimal resilience $t < n/3$ under different assumptions. The works by Hirt, Nielsen and Przydatek [HNP05, HNP08] make use of an additive homomorphic encryption, with the protocol in [HNP08] communicating $O(n^2)$ field elements per multiplication. The work by Choudhury and Patra [CP15] achieves $O(n)$ field elements per multiplication at the cost of using somewhat-homomorphic encryption, and the work by Cohen [Coh16] achieves a communication independent of the circuit size using fully-homomorphic encryption.

Other efficient solutions have been provided for sub-optimal resilience $t < n/4$ setting. Notable works include the protocols in [SR00, PSR02, CHP13, PCR15], achieving information-theoretic security with linear communication complexity.

1.2 Contributions

In this work, we target for maliciously secure information-theoretic asynchronous MPC with guaranteed output delivery. It is known that optimal achievable corruption tolerance in this setting is $t < n/3$ [BKR94, ADS20]. As we have mentioned above, the best-known result [CP23] in this setting requires $O(n^4)$ field elements of communication per multiplication gate. On the other hand, in the synchronous setting, it has long been known that linear communication complexity $O(n)$ field elements per multiplication gate can be achieved with perfect security against 1/3 corruption [BH08] and statistical security against 1/2 corruption [BSFO12]. This leads to our question:

"Can we construct information-theoretic asynchronous MPC with linear communication and optimal tolerance?"

To study this question, we have to first understand where this additional communication overhead in the asynchronous setting comes from. The standard paradigm of constructing IT asynchronous MPC protocols is to achieve the following three steps.

Step 1: Asynchronous Complete Secret Sharings(ACSS). The first step is to build a protocol which allows a dealer to share degree-t Shamir sharings to all parties. It satisfies that:

- If the dealer is honest, then all honest parties will eventually terminate the protocol and obtain correct shares distributed by the dealer.
- If the dealer is corrupted, then either no honest party terminates, or all honest parties terminate. In particular, if all honest parties terminate, their shares lie on valid degree-t polynomials.

In the synchronous setting, the best known results [BH08, BSFO12] can achieve $O(n)$ field elements of communication per sharing. On the other hand, in the asynchronous setting, the best-known construction [CP23] requires to communicate $O(n^3)$ field elements per sharing.

Step 2: Beaver Triples. With an ACSS protocol, all parties can efficiently prepare random degree-t Shamir sharings following the known techniques in the synchronous setting [DN07] with constant overhead. The second step is to prepare random Beaver triples shared by degree-t Shamir sharings by using an ACSS protocol in a black-box way. In the synchronous setting, the best known result [BH08, BSFO12] can achieve $O(n)$ field elements of communication plus sharing $O(1)$ degree-t Shamir sharings per triple. On the other hand, in the asynchronous setting, the best-known construction [CP17] requires to communicate $O(n^2)$ field elements plus sharing $O(n)$ degree-t Shamir sharings per triple.

Step 3: Online MPC from Beaver Triples. The last step is to evaluate the circuit by using random Beaver triples. Relying on the error-correction property of the Shamir secret sharings, this step can be easily achieved [CP17] following essentially the same technique as that in the synchronous setting [BH08]. In fact, the resulting online protocol can even achieve perfect security. In both the synchronous setting [BH08, BSFO12] and the asynchronous setting [CP17], to evaluate a circuit of size C, we need to consume $O(|C|)$ random Beaver triples.

We can see that in the synchronous setting (either the perfect security setting with $t < n/3$ or the statistical security setting with $t < n/2$), the first step can be realized with linear overhead in the number of parties and the rest of two steps can achieve constant overhead. As a result, known results with $O(n)$ field elements of communication per multiplication are known in [BH08] with perfect security and in [BSFO12] with statistical security. In the asynchronous setting, however, the best known result is [CP23] which only achieves $O(n^4)$ field elements of communication per multiplication gate. To achieve linear communication complexity following the above paradigm, there are two difficulties one needs to address: (1) constructing an ACSS protocol with linear communication overhead in the number of parties, and (2) preparing random Beaver triples with $O(n)$ field elements of communication plus sharing $O(1)$ degree-t Shamir sharings per triple.

In this work, we give an solution to the second difficulty.

Theorem 1. *Let $n = 3t + 1$. For 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 in the $\mathcal{F}_{\mathsf{ACSS}}$-hybrid model. The achieved communication complexity is $\mathcal{O}(|C|\cdot n+D\cdot n^2+n^8)$ elements plus $\mathcal{O}(n^2)$ invocations of $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(|C|)$ degree-t Shamir sharings.*

To achieve our result, we first extend the techniques in [CP17] and construct two different protocols for preparing random Beaver triples. Both protocols can achieve constant overhead but neither of them guarantees the success of the execution. In particular, for some constant $\epsilon \in (0, 1)$,

- The first protocol would eventually succeed if at least ϵt corrupted parties participate;
- The second protocol would eventually succeed if at most ϵt corrupted parties participate.

We introduce a novel technique that allows us to run these two protocols in parallel and ensure that at least one approach succeeds. In a nutshell, we manage to force that a party can only participate in the second protocol if he has participated in the first protocol. In this way, we can ensure that either at least ϵt corrupted parties participate in the first protocol and the first protocol would eventually succeed, or at most ϵt corrupted parties participate in the second protocol and the second protocol would eventually succeed.

Plugging in Known Results of ACSS. From [CP23], we have the following theorem about realizing $\mathcal{F}_{\mathsf{ACSS}}$.

Theorem 2 ([CP23]). *There exists a protocol that securely realizes $\mathcal{F}_{\mathsf{ACSS}}$ against a fully malicious adversary who corrupts at most $t < n/3$ parties. The achieved communication complexity is $O(N \cdot n^3 + n^4 \cdot \kappa + n^5)$ field elements to share N degree-t Shamir sharings, where κ is the security parameter.*

When instantiating $\mathcal{F}_{\mathsf{ACSS}}$ by the construction from [CP23], we obtain the following corollary.

Corollary 1. *Let $n = 3t + 1$. For any circuit C of size $|C|$ and depth D, there is a fully malicious information-theoretic asynchronous MPC protocol that is secure against at most t corrupted parties with guaranteed output delivery. The total communication complexity is $O(|C|\cdot n^3+n^6\cdot\kappa+n^8)$ field elements.*

We note that a concurrent work [JLS24] addresses the first difficulty and gives the first construction of an ACSS protocol that achieves $O(n)$ field elements per sharing:

Theorem 3 ([JLS24]). *There exists a protocol that securely realizes $\mathcal{F}_{\mathsf{ACSS}}$ against a fully malicious adversary who corrupts at most $t < n/3$ parties. The achieved communication complexity is $O(N \cdot n + n^{12} \cdot \kappa)$ field elements to share N degree-t Shamir sharings, where κ is the security parameter.*

When instantiating $\mathcal{F}_{\mathsf{ACSS}}$ by the construction from [JLS24], we obtain the first construction of asynchronous MPC that achieves $O(n)$ field elements of communication per multiplication gate.

Corollary 2. *Let $n = 3t + 1$. For any circuit C of size $|C|$ and depth D, there is a fully malicious information-theoretic asynchronous MPC protocol that is secure against at most t corrupted parties with guaranteed output delivery. The total communication complexity is $O(|C| \cdot n + D \cdot n^2 + n^{14} \cdot \kappa)$ field elements.*

2 Technical Overview

We give a high-level overview of our main techniques. In our setting, parties have access to a complete network of point-to-point asynchronous and secure channels. Asynchronous channels only guarantee that messages sent by honest parties are eventually delivered, and the adversary can control the message scheduling; in particular, the order in which messages are delivered.

In the following, we will use $[x]_t$ to denote a degree-t Shamir sharing of x. We assume the existence of an ACSS protocol that allows a dealer to share a batch of degree-t Shamir sharings to all the parties. We refer the readers to $\mathcal{F}_{\mathsf{ACSS}}$ (Appendix B.2) for the formal description of the security.

Our goal is to prepare random Beaver triples with $O(n)$ field elements of communication plus sharing $O(1)$ degree-t Shamir sharings per triple. Recall that a random Beaver triple consists of $([a]_t, [b]_t, [c]_t)$ where a, b are random field elements and $c = a \cdot b$. With $\mathcal{F}_{\mathsf{ACSS}}$, random degree-$t$ Shamir sharings can be efficiently prepared with constant overhead following the same techniques in the synchronous setting [DN07]. I.e., generating each random degree-t Shamir sharing only requires sharing $O(1)$ degree-t Shamir sharings using $\mathcal{F}_{\mathsf{ACSS}}$. To prepare a random Beaver triple, we first prepare two random degree-t Shamir sharings $[a]_t, [b]_t$. Then the main task is to allow all parties to obtain $[c]_t = [a \cdot b]_t$.

2.1 Overview of Previous Techniques

Why Techniques in the Synchronous Setting Do not Work. In the synchronous setting, the generic approach [DN07, BSFO12] is to first locally multiply these two random sharings and obtain $[c]_{2t} = [a]_t \cdot [b]_t$. Then by utilizing a pair of random double sharings $([r]_t, [r]_{2t})$, all parties interactively transform $[c]_{2t}$ to $[c]_t$. In the synchronous setting, both of preparing random double sharings $([r]_t, [r]_{2t})$ and transforming $[c]_{2t}$ to $[c]_t$ can be done with linear communication in the number of parties. In [BSFO12],

- To achieve malicious security, the authors design an efficient verification protocol to check the correctness of the Beaver triples.
- To achieve guaranteed output delivery, the framework of dispute control [BTH06] is used. Very informally, each time the verification fails, all parties reveal their views and find out the cheater. Then the cheater is kicked out and all parties retry the preparation.

When try to adapt the above approach to the asynchronous setting, the immediate difficulty is to efficiently prepare random double sharings. In the synchronous setting, the generic approach of preparing random linear sharings [DN07] is to let each party prepare and distribute one such random sharing and all parties apply a Vandermonde matrix on the distributed random sharings to extract $n-t$ random sharings that are not known to any party. Note that in the synchronous setting, if a message is not received from some party, then this party must be corrupted. However, in the asynchronous setting, it may also be the case that this party is honest but his message is delayed by the adversary. To achieve liveness, we cannot wait for shares from all parties. As a result, when using this approach in the asynchronous setting, some honest parties may not be able to obtain their shares. This also partially explains why designing an efficient ACSS protocol is not trivial.

Even with random double sharings prepared, a more severe issue is that the above approach of achieving guaranteed output delivery does not work either. To catch the cheater, we need the views from all parties that participate in the preparation of Beaver triples. Again since we cannot hope that all parties provide their views in the asynchronous setting, it is not clear how to find the cheater and use the framework of dispute control to achieve guaranteed output delivery.

Techniques in [CP17]. In [CP17], the authors take an entirely different approach that only needs to share and reconstruct degree-t Shamir sharings, which avoids the above difficulties. Here sharing degree-t Shamir sharings can be done by $\mathcal{F}_{\mathsf{ACSS}}$ and reconstructing degree-t Shamir sharings can achieve linear communication with guaranteed output delivery relying on the error-correction property of Shamir sharings. However, their techniques introduce a factor of $O(n)$ overhead in the communication cost. To be more explicit, the amortized cost per random Beaver triple is $O(n^2)$ field elements of communication plus sharing $O(n)$ degree-t Shamir sharings using $\mathcal{F}_{\mathsf{ACSS}}$.

At a high level, the idea is to first ask each party to distribute random Beaver triples by using $\mathcal{F}_{\mathsf{ACSS}}$ and then extract random Beaver triples that are not known to any party. The extraction process only involves reconstructions of degree-t Shamir sharings and local computation. Recall that in the asynchronous setting, one cannot wait for all parties successfully distributing their random Beaver triples since corrupted parties may never respond. Instead, the best one can hope is that $L = n - t = 2t + 1$ parties successfully distribute their random Beaver triples to other parties. On the other hand, in the worst case t out of the L successful dealers can be corrupted. The extraction process will sacrifice $(L-1)/2$ triples and therefore only $(L+1)/2$ triples remain. Since t out of the remaining triples may be generated by corrupted parties, it can only extract $(L + 1)/2 - t = 1$ random Beaver triple.

Potential Ways of Achieving Constant Overhead. In [CP17], the extraction process only outputs one random Beaver triple, which leads to a factor of $O(n)$ overhead. To remove this overhead, our hope is to obtain $O(n)$ random Beaver triples each time. We note the following two potential ways that allow us to obtain more Beaver triples each time.

The first way is to try to wait for more parties that successfully distribute their random Beaver triples. To be more concrete, if $L = (2 + \epsilon)t + 1$ for some constant $\epsilon \in (0, 1)$, then the extraction process can produce $(L + 1)/2 - t > \epsilon t/2 = O(n)$ random Beaver triples, thus achieving constant overhead. However, as we discussed above, corrupted parties may never respond and parties may wait forever and never terminate.

The second way is to extend the techniques in [CP17] to packed Shamir secret sharings. At a high level, the idea of packed Shamir sharings is to store multiple secrets within a single sharing. In general, to store k secrets and achieve t-privacy, we have to use a degree-$(t + k - 1)$ packed Shamir sharing. We use $\boldsymbol{x} \in \mathbb{F}^k$ to denote a vector and $[\boldsymbol{x}]_{t+k-1}$ to denote a degree-$(t + k - 1)$ packed Shamir sharing of $\boldsymbol{x}$. Now applying the techniques in [CP17] over packed Shamir sharings, we obtain a single packed Beaver triple each time. Then, we depack a packed Beaver triple to k standard Beaver triples. When $k = O(n)$, this also allows us to obtain $O(n)$ random Beaver triples each time, thus achieving constant overhead. However, the issue is that we cannot rely on the error-correction property anymore and the reconstruction of a degree-$(t + k - 1)$ packed Shamir sharing may fail. Besides, it is not clear how to efficiently prepare random degree-$(t + k - 1)$ packed Shamir sharings since we cannot use $\mathcal{F}_{\mathsf{ACSS}}$, which is only for degree-t Shamir sharings.

2.2 Our Solution

We made the following observations for our two attempts above.

- The first process would eventually succeed if at least $L = (2 + \epsilon)t + 1$ parties participate. Note that if at least ϵt corrupted participate, since honest parties will eventually participate, the first process would also succeed.
- In the second process, the reason that the error-correction property does not work is because we may receive t incorrect shares in the worst case while for a degree-$(t + k - 1)$ packed Shamir sharing with $2t + 1$ correct shares, we may hope to correct at most $t + 1 - k$ incorrect shares. On the other hand, if at most $\epsilon t \leq t + 1 - k$ corrupted parties participate in the second process, we can continue to rely on the error-correction property. As we will show later, in this case we can also prepare degree-$(t + k - 1)$ packed Shamir sharings efficiently due to the smaller number of corrupted parties.

As we can see, the failure conditions for these two processes are contradictory. However, the adversary may choose to let less than ϵt corrupted parties participate in the first process while letting more than ϵt corrupted parties participate in the second process, making both processes fail.

Our idea is to run these two processes in parallel and force that each party can only participate in the second process if he has participated in the first process. Now if an adversary wants to make the first process fail by letting less than ϵt corrupted parties participate in the first process, then there are also less than ϵt corrupted parties in the second process and the second process will eventually succeed. On the other hand, if an adversary wants to make the second process fail by letting more than ϵt corrupted parties participate in the second process, then there are also more than ϵt corrupted parties in the first process and the first process will eventually succeed. Therefore, an adversary cannot make both processes fail.

We note that in the first process, we simply use the techniques in [CP17] and wait for more parties. In [CP17], each party distributes his random Beaver triples by using $\mathcal{F}_{\mathsf{ACSS}}$, which guarantees that either all honest parties terminate or no honest party terminates. Thus, we just need to add the following requirement for the second process: A party P_i accepts P_j's messages for the second process only if P_i terminates the sharing step led by P_j in the first process. To be more concrete, during the execution of the second process, when P_i receives a message from P_j, P_i locally stores this message. Only when P_i has terminated the sharing step led by P_j in the first process, P_i starts to handle all stored messages (and future messages) from P_j following the second process. In this way, if a corrupted party P_j does not participate in the first process, then every honest party P_i will ignore his messages for the second process, which is equivalent to that P_j does not participate in the second process.

Following the above idea, we give more details about our constructions for these two different processes below. In particular,

- For the first process, it would eventually succeed if at least $L = (2 + \epsilon)t + 1$ parties or at least ϵt corrupted parties participate.
- For the second process, it would eventually succeed if at most $L = (2 + \epsilon)t + 1$ parties *and* at most ϵt corrupted parties participate.

We point out that having at most ϵt corrupted parties participate does not imply that there are at most $(2 + \epsilon)t + 1$ parties since corrupted parties may corrupt less than t parties and there may be more than $2t + 1$ honest parties. Our construction sets $\epsilon = 0.1$ for which the reason will be clear later.

We note that only achieving the above requirements for these two processes are not sufficient. This is because during the protocol execution, parties cannot distinguish which case happens and they have to try both processes. We need to ensure that for each of these two processes, if the success requirement is not met, parties should not accept incorrect or insecure Beaver triples.

We first assume that there is a trusted P_{king}. We will remove this assumption later.

Process 1 with Trusted P_{king}. In the first process, we essentially follow the same steps as in [CP17]:

1. Each party uses $\mathcal{F}_{\mathsf{ACSS}}$ to share a batch of random Beaver triples. Each party also uses $\mathcal{F}_{\mathsf{ACSS}}$ to share data that are used to check the correctness of his Beaver triples.
2. All parties agree on a set of L successful dealers.
3. For each successful dealer, all parties check the correctness of the Beaver triples dealt by this party.
4. All parties run the extraction process to obtain random Beaver triples.

For simplicity, we omit the details about the triple verification in Step 1 and Step 3 since they are the same as [CP17]. We refer the readers to Section 4.1 for more details.

The only difference is that in the second step, we wait for $L = (2 + \epsilon)t + 1$ successful dealers whereas in [CP17], they only wait for $2t + 1$ successful dealers. In their case, Step 2 can be achieved by running an ACS (Agreement on a Common Subset) protocol. However, this does not work in our case since an ACS protocol only allows all parties to agree on a subset of size $2t + 1$.

Our solution is to let P_{king} decide this subset. For each dealer D, if a party P_i terminates the sharing step led by D, P_i sends $(\mathtt{support}, P_i, D)$ to P_{king}. After receiving $t + 1$ supporting messages for D, P_{king} counts D as a successful dealer. Note that at least one of these $t + 1$ supporting messages come from honest parties. By the property of $\mathcal{F}_{\mathsf{ACSS}}$, all honest parties will eventually terminate the sharing step led by D. P_{king} waits for L successful dealers and then reliably broadcasts the set $\mathcal{D}$ of these L successful dealers to all parties. Each party accepts $\mathcal{D}$ if he terminates the sharing step led by each party in $\mathcal{D}$.

Following the same argument as in [CP17], if P_{king} is honest and at least $L = (2+\epsilon)t+1$ parties or at least ϵt corrupted parties participate in the first process, then all parties can obtain correct and random Beaver triples with overwhelming probability.

Security Analysis When Success Requirements are not Met. As we mentioned above, we have to also consider the case when the success requirements are not met. I.e., either P_{king} is corrupted or less than $L = (2 + \epsilon)t + 1$ parties including less than ϵt corrupted parties participate in the first process. In this case, we do not require the protocol to succeed or terminate but we need to ensure that if the protocol terminates, parties will not accept incorrect or insecure Beaver triples.

When P_{king} is honest but less than L parties including less than ϵt corrupted participate in the first process, P_{king} will wait forever for L successful dealers. In this case, the protocol simply does not terminate. Thus, parties will not end up with incorrect or insecure Beaver triples.

When P_{king} is corrupted and no matter how many parties or corrupted parties participate in the first process, there are two possibilities:

– P_{king} never broadcasts the set of successful dealers or the set broadcast by P_{king} is of size less than L. In this case, all (honest) parties will wait forever and never terminate.
– P_{king} broadcast a set $\mathcal{D}$ of L dealers.
  - If there is an honest party that accepts this set, then by the property of $\mathcal{F}_{\mathsf{ACSS}}$, all honest parties will accept this set, which means that each dealer in $\mathcal{D}$ indeed shares his random Beaver triples. In this case, P_{king} just performs as an honest P_{king} and as a result, all parties will obtain correct and random Beaver triples.
  - If no honest party accepts this set, then all honest parties will wait forever and never terminate.

As seen, when the success requirements are not met, either all (honest) parties do not terminate, or they will still obtain correct and random Beaver triples.

Process 2 with Trusted P_{king}. Recall that the high-level idea of the second process is to (1) follow the techniques in [CP17] by using packed Shamir sharings, and (2) transform the obtained random packed Beaver triples to standard Beaver triples. The construction of the second process is more involved due to the following difficulties:

– We have to design an efficient sharing protocol for packed Shamir sharings to allow parties to share their random packed Beaver triples.
– After using the techniques in [CP17] and obtaining random packed Beaver triples, we have to depack them to standard random Beaver triples efficiently.
– We need to ensure that the above sub-steps succeed when at most $(2+\epsilon)t+1$ parties including at most ϵt corrupted parties participate in the second process. And more importantly, if the success requirement is not met, parties should not end up with incorrect or insecure Beaver triples. This means that when designing the protocols, we still need to maintain the security against t corrupted parties.

We elaborate our techniques for each sub-step. Recall that $\epsilon = 0.1$ is a constant. Let $d = (1+\epsilon)t - 1$. We will use a degree-d packed Shamir sharing which can store $d-t+1 = \epsilon t$ secrets while ensuring privacy against t corrupted parties. In the following discussion, we first assume that the success requirement is met. I.e., at most $(2+\epsilon)t+1$ parties including at most ϵt corrupted parties participate in the second process. In this case, we need to ensure the success of each sub-step. Later on, we will discuss the case when the success requirement is not met.

Distributing Degree-d Packed Shamir Sharings. The first difficulty is to allow a dealer to distribute degree-d packed Shamir sharings such that all honest parties can eventually receive their shares. Our goal is to achieve linear communication complexity in the number of parties per packed Shamir sharing.

Let $\alpha_1, \ldots, \alpha_n$ be distinct field elements. Following previous constructions for distributing Shamir sharings in the asynchronous setting [CP17, AAPP22], our starting point is to let the dealer share a degree-d bivariate polynomial $F(x,y)$ where each party P_i should receive $F(x,\alpha_i)$ and $F(\alpha_i,y)$. Consider the following steps.

1. The dealer D sends $F(x,\alpha_i)$ and $F(\alpha_i,y)$ to P_i.
2. Each party P_i, upon receiving the polynomials from D, reliably broadcasts $(\mathsf{support}, P_i, D)$ to all parties, and sends $F(\alpha_i,\alpha_j)$ and $F(\alpha_j,\alpha_i)$ to each party P_j.
3. Each party P_j, upon receiving $(2-\epsilon)t+1$ shares from all parties, interpolates $F(x,\alpha_j)$ and $F(\alpha_j,y)$.
4. If P_i has obtained $F(x,\alpha_i)$ and $F(\alpha_i,y)$ and received $2t+1$ parties supporting D, P_i terminates.

Termination Condition. First note that when D is honest, all honest parties will eventually receive their shares and support D. In this case, all parties will eventually terminate.

When D is corrupted, if an honest party terminates, then he has received $2t+1$ parties supporting D. Since there are at most ϵt corrupted parties by assumption, at least $(2-\epsilon)t+1$ honest parties support D. Thus, every honest party will eventually receive his shares, either from D or interpolated from $(2-\epsilon)t+1$ shares received from other parties. Therefore, if an honest party receives his shares and terminates, all honest parties will eventually receive their shares and terminate.

Correctness. Now we want to ensure that if D is honest, then every honest party P_i will always receive the correct shares. Note that if P_i directly receives shares from D, then he must obtain correct shares. Consider the case where a party P_j interpolates $F(x,\alpha_j)$ and $F(\alpha_j,y)$ from $(2-\epsilon)t+1$ shares received from other parties. Our observation is that since there are at most ϵt corrupted parties, he must receive at least $(2-2\epsilon)t+1$ shares from honest parties, which are correct shares. Given that ϵ is a small constant, P_j can use the error correction of the Reed-Solomon codes to recover the correct polynomials. Thus, when D is honest, every honest party P_i will eventually receive the correct shares.

We would also want to ensure that if D is corrupted, all honest parties should hold valid degree-d bivariate polynomials. For this, our attempt is to let all parties check a random linear combination of all bivariate polynomials distributed by D. Usually this kind of check is useless in the asynchronous setting because when we generate a random challenge, up to t honest parties may have not received their shares from D (since we cannot wait for all parties receiving their shares in the asynchronous setting). If D knows the challenge, D can cheat by choosing shares for those honest parties such that the shares are incorrect but still pass the check. In our case, however, since there are only ϵt corrupted parties by assumption, we can at least ensure that most honest parties have received their shares before generating the challenge.

In more details, the challenge is generated when $2t+1$ parties have received their shares. At this moment, at least $(2-\epsilon)t+1$ honest parties have received their shares. All parties compute a random linear combination of all bivariate polynomials distributed by D and reliably broadcast their shares. Then all parties run an ACS protocol to agree on a set of $2t+1$ parties that have broadcast their shares and check whether the shares of these $2t+1$ parties lie on a valid degree-d bivariate polynomial. We note that, however, this check may fail even if D is honest since corrupted parties may send incorrect shares. Since there are at most ϵt corrupted parties, we relax the requirement by checking whether there are $(2-\epsilon)t+1$ parties' shares lie on a valid degree-d bivariate polynomial. This check can be done efficiently relying on the error-correction algorithm of the Reed-Solomon Code. In this way, the check will always succeed when D is honest. When D is corrupted and the check passes, note that

- At least $(2-\epsilon)t+1$ honest parties have received their shares before the challenge is generated;
- The shares of $(2-\epsilon)t+1$ parties lie on a valid degree-d bivariate polynomial;
- By assumption, there are at most $(2+\epsilon)t+1$ parties (including at most ϵt corrupted parties).

According to the inclusion-and-exclusion principle, at least $(2-3\epsilon)t+1$ honest parties who have received their shares before the challenge is generated, and their shares lie on a valid degree-d bivariate polynomial. Thus, the shares of *most* honest parties lie on valid degree-d bivariate polynomials.

While this protocol does not ensure that all honest parties receive correct shares, we can use it as a commitment. That is, once all parties receive the shares from D, corrupted parties can no longer change the degree-d bivariate polynomial anymore, and an honest party P_i can always reconstruct the correct degree-d bivariate polynomial by using error correction: To reconstruct such a bivariate polynomial, P_i waits to receive $2t+1$ shares from all parties. Since at least $(2-3\epsilon)t+1$ honest parties hold correct shares and there are at most $(2+\epsilon)t+1$ parties, by the inclusion-and-exclusion principle again, P_i receives at least $(2-4\epsilon)t+1$ correct shares, which means that there are at most $4\epsilon t$ incorrect shares. In this case, the error-correction property of the Reed-Solomon Code allows us to reconstruct a polynomial of degree $(2-8\epsilon)t$. Recall that we set $d=(1+\epsilon)t-1$. We choose $\epsilon=0.1$ so that we have $d<(2-8\epsilon)t$ and it is sufficient for P_i to reconstruct the correct degree-d bivariate polynomial.

Now, to share degree-d packed Shamir sharings,

1. The dealer first uses the above protocol to commit the shares of each party.
2. Then all parties together verify that the committed shares indeed form valid degree-d packed Shamir sharings.
3. Finally, the commitments are opened by letting all parties send their polynomials to each receiver. And the receiver can always reconstruct the correct shares by using error correction.

We remind the readers that all the above security guarantees hold only when the success requirement is met, i.e., there are at most $(2+\epsilon)t$ parties including ϵt corrupted parties. As we have discussed in the beginning, only satisfying these guarantees are not sufficient. We still need to ensure that if the success requirement is not met, all parties should not accept incorrect or insecure Beaver triples. We will discuss this scenario later.

Cost Analysis. To share a degree-d bivariate polynomial, the above protocol requires $O(n^2)$ communication. Note that a degree-d bivariate polynomial can store $(d-t+1)^2=O(n^2)$ secrets. Therefore, the amortized cost per secret is constant. To share degree-d packed Shamir sharings, the dealer needs to commit the shares of each party. Thus, the amortized cost per packed Shamir sharing is $O(n)$.

Transforming Packed Beaver Triples to Standard Beaver Triples. Now all parties follow the approach in [CP17] to generate random packed Beaver triples. Recall that the approach in [CP17] requires $\mathcal{O}(n^2)$ communication to generate a single Beaver triple. However, since we extract a degree-d packed Beaver triple, the amortized communication per secret remains linear.

After obtaining random packed Beaver triples, we have to transform them to standard Beaver triples. This task can be abstracted as follows. All parties start with a degree-d packed Shamir sharing $[\boldsymbol{x}]_d$ and they want to obtain $[x_1]_t, \ldots, [x_{\epsilon t}]_t$ (since $\boldsymbol{x}$ is of length $d - t + 1 = \epsilon t$). This is done by preparing a tuple of random sharings $([\boldsymbol{r}]_d, [r_1]_t, \ldots, [r_{\epsilon t}]_t)$. Then all parties reconstruct $\boldsymbol{x} + \boldsymbol{r}$ and compute $[x_i]_t = (\boldsymbol{x} + \boldsymbol{r})_i - [r_i]_t$.

At a high level, we rely on the observation in [EGPS22] to transform the preparation of $([\boldsymbol{r}]_d, [r_1]_t, \ldots, [r_{\epsilon t}]_t)$ to prepare correlated degree-t Shamir sharings $([r_i|_i]_t)_{i=1}^{\epsilon t}, ([r_i]_t)_{i=1}^{\epsilon t}$. Here $[r_i|_i]_t$ is a degree-t Shamir sharing with the secret r_i stored at the i-th position. Then $[\boldsymbol{r}]_d$ can be computed by

$$\sum_{i=1}^{\epsilon t} [\boldsymbol{e}_i]_{d-t} \cdot [r_i|_i]_t,$$

where $\boldsymbol{e}_i$ is the i-th unit vector of size ϵt and $[\boldsymbol{e}_i]_{d-t}$ is the degree-$(d-t)$ packed Shamir sharing that is fully determined by $\boldsymbol{e}_i$[4]. To see why this is the case, note that $[\boldsymbol{e}_i]_{d-t} \cdot [r_i|_i]_t$ is a degree-d packed Shamir sharing where the i-th secret is r_i and all other secrets are 0. Now since all correlated sharings are of degree t, we make use of $\mathcal{F}_{\mathsf{ACSS}}$ to prepare these sharings.

To reconstruct $\boldsymbol{x} + \boldsymbol{r}$, we rely on an honest P_{king}. All parties send their shares of $[\boldsymbol{x} + \boldsymbol{r}]_d$ to P_{king}. Then P_{king} waits for $2t+1$ shares. Note that by assumption, there are at most ϵt corrupted parties. Thus P_{king} will receive at least $(2-\epsilon)t + 1$ shares from honest parties, which are correct. Therefore, P_{king} uses error correction to construct $\boldsymbol{x} + \boldsymbol{r}$ and broadcast the secrets to all parties.

In summary, the above ideas allow us to prepare Beaver triples with linear communication complexity when P_{king} is honest and there are at most $(2+\epsilon)t + 1$ parties including at most ϵt corrupted parties.

Security Analysis When Success Requirements are not Met. As we mentioned in the beginning, we have to also consider the case when the success requirements are not met. I.e., either P_{king} is corrupted or there are more than $(2+\epsilon)t+1$ parties or there are more than ϵt corrupted parties. In this case, we do not require the protocol to succeed or terminate but we need to ensure that if the protocol terminates, parties will not accept incorrect or insecure Beaver triples.

Unfortunately, the current construction for Process 2 can easily go wrong when the success requirements are not met. For example, when distributing a degree-d packed Shamir sharing, if there are more than ϵt corrupted parties, then we cannot rely on the error-correction property of the Reed-Solomon Code as described above. As a result, even for an honest dealer, honest parties may not be able to receive correct shares. When transforming packed Beaver triples to standard Beaver triples, a corrupted P_{king} can simply broadcast incorrect reconstruction results to all parties.

Our solution is to add the multiplication verification step from [BSFO12] at the end of Process 2 to check whether the obtained Beaver triples are correct. If not, then Process 2 fails. We show that this is sufficient to achieve both correctness and secrecy of the obtained Beaver triples.

Our key observation is that, for most of our protocols, a malicious adversary can only add additive errors to the shares of honest parties. Intuitively, this is because for each value sent from a corrupted party to an honest party, the adversary knows the difference between the actual value and the value it should be. Since each party just performs linear operations locally, this difference eventually translates to additive errors to the shares of honest parties.

However, when running the triple extraction protocol in [CP17], for each obtained packed Beaver triple $([\boldsymbol{a}]_d, [\boldsymbol{b}]_d, [\boldsymbol{c}]_d)$, the adversary may insert an error to $\boldsymbol{c}$ such that it is linear in $\boldsymbol{a}$ and $\boldsymbol{b}$. This is because the protocol in [CP17] requires to do multiplications by using Beaver triples (and packed Beaver triples in our case) and each party needs to multiply a potentially incorrect value from P_{king} with his local share, resulting in linear errors in $\boldsymbol{a}$ and $\boldsymbol{b}$. Fortunately, we note that such an error will cause $\boldsymbol{c} \neq \boldsymbol{a} * \boldsymbol{b}$ with overwhelming probability when the underlying field is large enough since $\boldsymbol{a}, \boldsymbol{b}$ are random values,

[4] Note that a degree-$(d-t)$ packed Shamir sharing corresponds to a degree-$(d-t)$ polynomial which is determined by $d - t + 1 = \epsilon t$ evaluation points. Here $\boldsymbol{e}_i$ is a vector of size ϵt, thus fully determining $[\boldsymbol{e}_i]_{d-t}$. Note that we do not require privacy for $[\boldsymbol{e}_i]_{d-t}$ since $\boldsymbol{e}_i$ is a public vector.

as also used in [RS22]. In summary, an adversary can only insert additive errors and linear errors to the triples in the second process. Both errors will be caught by the final triple-verification.

Removing the Assumption of Trusted P_{king}. As we have discussed in each process, when P_{king} is honest, either Process 1 or Process 2 will eventually succeed. When P_{king} is corrupted, he can only cause the processes to fail or not terminate. Thus, to remove the assumption of a trusted P_{king}, we simply let each party act as the P_{king} and lead one session of generating random Beaver triples. Note that at least $2t+1$ out of n sessions are led by an honest party, which are guaranteed to succeed. All parties will run an ACS protocol to agree on a set of $2t+1$ successful P_{king}'s and using the Beaver triples generated from sessions led by these $2t+1$ successful P_{king}'s in the online MPC protocol. This allows us to remove the assumption of a trusted P_{king} while maintaining linear communication complexity.

3 Preliminaries

We denote the security parameter by κ. In this work, we assume that field elements are of size $\Theta(\kappa)$ (so the field size is $2^{\Theta(\kappa)}$).

3.1 Security Model

The UC Framework. We follow the UC framework introduced by Canetti [Can01], based on the real and ideal world paradigm [Can00]. This means that one compares what an adversary can do in a real execution of the protocol with an ideal execution where a trusted party (the ideal functionality) interacts with the parties. A protocol is then secure if whatever an adversary can do in the real protocol, can be also achieved in the ideal execution. We recap the model in Appendix A. The standard UC framework does not model eventual delivery guarantees, but to model those, we follow the models in [CGHZ16, Coh16, CP23]. In particular, to model that the adversary can decide when each honest party learns the output from an ideal functionality, we model time via activations. When the functionality $\mathcal{F}$ has an output for some party, the party requests $\mathcal{F}$ for the output, and the adversary can instruct $\mathcal{F}$ to delay the output for each party. The party will then eventually receive the output when the environment activates the party sufficiently many times. As in [Coh16, CP23], we say that $\mathcal{F}$ sends a *request-based delayed output* to P_i to describe such behavior.

Ideal Functionality for Asynchronous MPC. We recall the ideal functionality for asynchronous MPC with guaranteed output delivery [CGHZ16, Coh16].[5]

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

The functionality runs with parties $P_1, \ldots, P_n$ and adversary $\mathcal{S}$. It is parameterized by a function $f\colon (\{0,1\}^* \cup \{\perp\})^n \to (\{0,1\}^*)^n$. Each party P_i has an initial input value $x_i = \perp$ and output value $y_i = \perp$.

1: Upon receiving an input v from party P_i, set $x_i = v$ and notify $\mathcal{S}$.
2: Upon receiving the core-set input $S \subseteq \mathcal{P}$ of size at least $n-t$ from the adversary $\mathcal{S}$ for the first time, record it and for every party $P_i \notin S$, set its input to $x_i = \perp$.
3: Upon receiving inputs from all parties in the core-set, the functionality evaluates the function f on the given inputs and obtains output $\boldsymbol{y} = (y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$.
4: The functionality generates a request-based delayed output to send P_i the output y_i.

3.2 Shamir Secret Sharing Scheme

In this work, we will use the standard Shamir Secret Sharing Scheme [Sha79]. Let n be the number of parties and $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2n$. Let $\alpha_1, \ldots, \alpha_n$ be n distinct non-zero elements in $\mathbb{F}$. A

[5] We choose to follow the formalization of [CGHZ16, Coh16], but there are different design choices one can make. See [CFG+23] for a discussion.

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 yields a degree-$2d$ sharing. The secret of the new sharing is the product of the original 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.

3.3 Building Blocks

We give definitions of the following primitives in Section B.

Agreement Primitives. Our construction makes use of the following agreement primitives.

- **Reliable Broadcast** $\mathcal{F}_{\mathsf{rbc}}$: It allows the parties to agree on the value of a sender without requiring termination if the sender is corrupted. It is known from [Bra84] that when $t < n/3$, there is a t-resilient broadcast protocol with communication complexity $\mathcal{O}(n^2\ell)$, where ℓ is the size of the sender's input. In addition, Patra [Pat11] introduced a protocol with $\mathcal{O}(n\ell + n^4\log(n))$ communication.
- **Byzantine Agreement** $\mathcal{F}_{\mathsf{ba}}$: It allows parties to agree on one of the honest parties' input values. For $t < n/3$, t-resilient asynchronous Byzantine agreement with communication complexity $\mathcal{O}(n^2\ell)$ can be achieved (see e.g. [MMR15]), where ℓ is the size of the input values.
- **Agreement on a Common Subset** $\mathcal{F}_{\mathsf{acs}}$: It allows the parties to agree on a set of at least $n-t$ parties that satisfy a certain property. Given an ACS property, there exists a t-resilient ACS protocol [BKR94] with communication complexity $\mathcal{O}(n^3)$ bits, for $t < n/3$ active corruptions.

Further Primitives. Some of our constructions rely on the following functionalities. The instantiations we give below all consider active security against $t < n/3$ corruptions with information-theoretic security.

- **Asynchronous Complete Secret Sharing** $\mathcal{F}_{\mathsf{ACSS}}$: It allows a dealer to specify a degree-t Shamir sharing $[x]_t$ and sends the shares to all parties. From [CP23], $\mathcal{F}_{\mathsf{ACSS}}$ can be realized with $\mathcal{O}(N \cdot n^3 + n^4 \cdot \kappa + n^5)$ elements of communication to share N degree-t Shamir sharings.
- **Random Sharing** $\mathcal{F}_{\mathsf{randShare}}$: It allows all parties to prepare N random degree-t Shamir sharings. Relying on known techniques [DN07] in the synchronous setting, $\mathcal{F}_{\mathsf{randShare}}$ can be realized with n invocations of $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(N)$ degree-t Shamir sharings.
- **Public Reconstruction** $\mathcal{F}_{\mathsf{pubRec}}$: It reconstructs a batch of degree-t Shamir sharings and sends the secrets to all parties. We assume that $t < n/3$ and the shares of honest parties lie on valid degree-t polynomials before invoking this functionality. From [CP17], $\mathcal{F}_{\mathsf{pubRec}}$ can be realized with $\mathcal{O}(N \cdot n + n^2)$ elements of communication to reconstruct $\mathcal{O}(N)$ degree-t Shamir sharings.
- **Random Coin** $\mathcal{F}_{\mathsf{coin}}$: It samples a random value r and distributes r to all parties. $\mathcal{F}_{\mathsf{coin}}$ can be realized by first preparing random degree-t Shamir sharings by $\mathcal{F}_{\mathsf{randShare}}$. Then each time a random value is requested, all parties invoke $\mathcal{F}_{\mathsf{pubRec}}$ to reconstruct a random degree-t Shamir sharings. When instantiating $\mathcal{F}_{\mathsf{ACSS}}$ (which is sued in $\mathcal{F}_{\mathsf{randShare}}$) by [CP23], $\mathcal{F}_{\mathsf{coin}}$ can be realized with amortized communication complexity of $\mathcal{O}(n^3)$ field elements per random value.

4 Beaver Triple Generation

We show how to prepare random Beaver triples with linear communication complexity. Recall that our idea is to run two different processes in parallel while ensuring that a party can only participate in the second process if he has participated in the first process. These two processes satisfy that for $\epsilon = 0.1$,

- if at least $(2+\epsilon)t+1$ parties *or* at least ϵt corrupted parties participate in the first process, then the first process will eventually succeed;
- if at most $(2+\epsilon)t+1$ parties *and* at most ϵt corrupted parties participate in the second process, then the second process will eventually succeed.

Then any adversary cannot make both processes fail. As we have discussed in Section 2.2, only satisfying the above requirements is not sufficient. We need to ensure that for each of these two processes, if the success requirement is not met, parties will not accept incorrect or insecure random Beaver triples.

We first describe a scenario where there is a trusted party P_{king}. Later we show how to remove this assumption by letting each party play the role of P_{king}.

4.1 Construction of Process 1

In the first process, we follow the techniques in [CP17] except that we wait for $(2+\epsilon)t+1$ successful dealers. At a very high level, each party first distributes random Beaver triples by using $\mathcal{F}_{\mathsf{ACSS}}$. Then all parties wait for at least $L = 2t + (t-1)/2$ successful dealers. Since $L > 2t+1$, we cannot use an ACS protocol to reach an agreement on the successful dealers. Instead, we rely on the trusted party P_{king} to propose the set of successful dealers. After this step, all parties verify the triples generated by each successful dealer and if the triples are incorrect (which indicates that the dealer is corrupted), those triples are replaced by all-0 triples. Finally, all parties extract random Beaver triples from those shared by successful dealers. The communication complexity of Process 1 is $\mathcal{O}(N \cdot n + n^3)$ elements plus n invocations of $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(N)$ degree-t Shamir sharings in total.

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

1: **Distribution:**
Let $\epsilon = 0.1$, $L = (2+\epsilon)t - 1$, and $N' = 2N/(\epsilon t)$. All parties agree on $2N'+1$ distinct field elements $\alpha_0, \ldots, \alpha_{2N'}$.
Each party P_i samples two random degree-N' polynomials f_i, g_i and sets $h_i = f_i \cdot g_i$. Then P_i samples $4N'+3$ random degree-t Shamir 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 distributes these $4N'+3$ degree-t Shamir sharings using $\mathcal{F}_{\mathsf{ACSS}}$. For each P_j that terminates $\mathcal{F}_{\mathsf{ACSS}}$ when P_i is the dealer, P_j sends $(\mathsf{support}, P_j, P_i)$ to all parties.

2: **Determine the Set of Successful Dealers:**
For each P_i, if P_{king} receives $(\mathsf{support}, P_j, P_i)$ from at least $t+1$ parties, P_{king} adds P_i to its list. P_{king} waits for L successful dealers. Let $\mathcal{D}$ be the set of L successful dealers. P_{king} reliably broadcasts the set $\mathcal{D}$.

3: **Verifying Triples:**
1. For each party P_j, after receiving $\mathcal{D}$, for each P_i in $\mathcal{D}$, P_j waits for the termination of $\mathcal{F}_{\mathsf{ACSS}}$ where P_i acts as the dealer. Then P_j sends a request to $\mathcal{F}_{\mathsf{coin}}$.
2. Upon receiving r from $\mathcal{F}_{\mathsf{coin}}$, if $r \in \{1, \ldots, N'\}$, all parties output `fail`. For each $P_i \in \mathcal{D}$, all parties locally compute $([f_i(r)]_t, [g_i(r)]_t, [h_i(r)]_t)$. Then all parties invoke $\mathcal{F}_{\mathsf{pubRec}}$ 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 $\ell \in \{1, \ldots, N'\}$, pick the first unused Beaver triple from each dealer in $\mathcal{D}$ and denote them by $\{([a_i]_t, [b_i]_t, [c_i]_t)\}_{i=1}^{L}$. Then run the following steps to extract $\frac{L+1}{2} - t = \epsilon t/2$ random Beaver triples:
1. All parties set two polynomials f, g of degree $L' = \frac{L-1}{2}$ such that $[f(\alpha_i)]_t = [a_i]_t$ and $[g(\alpha_i)]_t = [b_i]_t$ for all $i \in \{1, \ldots, L'+1\}$.
2. All parties locally compute $[f(\alpha_i)]_t$, $[g(\alpha_i)]_t$ for all $i \in \{L'+2, \ldots, L\}$.

3. All parties use the Beaver triple $([a_i]_t, [b_i]_t, [c_i]_t)$ to compute $[f(\alpha_i) \cdot g(\alpha_i)]_t$ for all $i \in \{L'+2, \ldots, L\}$ as follows.
   (a) For all $i \in \{L'+2, \ldots, L\}$, all parties locally compute $[f(\alpha_i) + a_i]_t, [g(\alpha_i) + b_i]_t$.
   (b) All parties invoke $\mathcal{F}_{\mathsf{pubRec}}$ to reconstruct the secrets $f(\alpha_i)+a_i, g(\beta_i)+b_i$ for all $i \in \{L'+2, \ldots, L\}$.
   (c) All parties locally compute

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

4. All parties set a polynomial h of degree $L-1$ such that $[h(\alpha_i)]_t = [f(\alpha_i) \cdot g(\alpha_i)]_t$ for all $i \in \{1, \ldots, L\}$.
5. Let $\beta_1, \ldots, \beta_{(L+1)/2-t}$ be distinct non-zero field elements that are different from $\alpha_1, \ldots, \alpha_n$. All parties output $([f(\beta_i)]_t, [g(\beta_i)]_t, [h(\beta_i)]_t)$ for all $i \in \{1, 2, ..., (L+1)/2 - t\}$.

4.2 Construction of Process 2

We describe our construction of Process 2 step by step. In the first step, we show how to let a party distribute packed Shamir sharings. In the second step, we show how to adapt the approach in [CP17] for packed Shamir sharings. In the third step, we show how to transform packed Beaver triples to random triples.

Distributing Packed Shamir Sharings. Let $\epsilon = 0.1$ and $d = t + \epsilon t - 1$. Our goal is to let a dealer distribute degree-d packed Shamir sharings such that if there are at most $(2+\epsilon)t+1$ parties including at most ϵt corrupted parties, then all honest parties will eventually receive their shares and the shares of honest parties lie on valid degree-d polynomials.

We first consider the following protocol $\mathit{\Pi}_{\mathtt{ShBi}}$ that allows a dealer to distribute N degree-d bivariate polynomials. The communication of $\mathit{\Pi}_{\mathtt{ShBi}}$ is $\mathcal{O}(N \cdot n^2 + n^4)$ elements in $\mathbb{F}$.

Protocol $\mathit{\Pi}_{\mathtt{ShBi}}$

Dealer D

1: Let $F_1, F_2, \ldots, F_N$ be the N bivariate polynomials that D wants to share.
2: D samples a random degree-d bivariate polynomial F_0.
3: For all $\ell \in \{0, 1, \ldots, N\}$, send to each P_i the polynomials $f_{\ell,i}(x) = F_\ell(x, \alpha_i)$ and $g_{\ell,i}(y) = F_\ell(\alpha_i, y)$.

Party P_i

1: Waiting for Shares: P_i keeps receiving messages until one of the following conditions is satisfied.
- Upon receiving $\{f_{\ell,i}(x), g_{\ell,i}(y)\}_{\ell=0}^N$ from D, broadcast $(\mathsf{support}, P_i, D)$ to all parties. Then send to each P_j the points $\{f_{\ell,i}(\alpha_j), g_{\ell,i}(\alpha_j)\}_{\ell=0}^N$ and wait to receive $(\mathsf{support}, P_j, D)$ from $2t+1$ distinct parties P_j.
- Upon receiving $(\mathsf{support}, P_j, D)$ from $2t+1$ distinct parties P_j, P_i waits to receive $\{f_{\ell,i}(\alpha_j), g_{\ell,i}(\alpha_j)\}_{\ell=0}^N$ from $(2-\epsilon)t+1$ parties. Then for all $\ell \in \{0, \ldots, N\}$, try to find two degree-d polynomials $f_{\ell,j}(x), g_{\ell,j}(y)$ such that $f_{\ell,j}(\alpha_i) = g_{\ell,i}(\alpha_j)$ and $g_{\ell,j}(\alpha_i) = f_{\ell,i}(\alpha_j)$ for at least $(2-2\epsilon)t+1$ different indices i. This step is done by running the error-correction algorithm of the Reed-Solomon Code. If such polynomials do not exist, interpolate $f_{\ell,j}(x), g_{\ell,j}(y)$ by using the first $d+1$ points.

2: Verification:
1. After completing $\{f_{\ell,i}(x), g_{\ell,i}(y)\}_{\ell=0}^N$ and receiving $(\mathsf{support}, P_j, D)$ from $2t+1$ distinct parties P_j, P_i sends a request to $\mathcal{F}_{\mathsf{coin}}$ and waits to receive a challenge r.
2. P_i computes $f_i(x) = \sum_{\ell=0}^N r^i \cdot f_{\ell,i}(x)$ and $g_i(x) = \sum_{\ell=0}^N r^i \cdot g_{\ell,i}(x)$.
3. P_i broadcasts $f_i(x), g_i(y)$.
4. P_i sets the property Q as P_i terminating the broadcast protocol led by P_j. Then all parties run $\mathcal{F}_{\mathsf{acs}}$ to agree on a set $\mathcal{B}$ of party P_j that broadcasts $f_j(x), g_j(y)$.
5. P_i checks whether exists a subset of $(2-\epsilon)t+1$ parties in $\mathcal{B}$ such that their polynomials lie on a degree-d bivariate polynomial. This is done by the following checks.
   - For all $j_1 \in \{1, 2, \ldots, n\}$, try to find $\tilde{f}_{j_1}(x)$ such that $\tilde{f}_{j_1}(\alpha_{j_2}) = g_{j_2}(\alpha_{j_1})$ for at least $(2-\epsilon)t+1$ party P_{j_2} in $\mathcal{B}$. This step is done by running the error-correction algorithm of the Reed-Solomon Code. If any $\tilde{f}_{j_1}(x)$ does not exist, the check fails.
   - Check whether $\{\tilde{f}_{j_1}(x)\}_{j_1=1}^n$ lie on a valid degree-d bivariate polynomial. If not, the check fails. Otherwise, let $\tilde{F}(x, y)$ denote this degree-d bivariate polynomial.

– Check whether for at least $(2-\epsilon)t+1$ parties in $\mathcal{B}$, the polynomials they broadcast lie on $\tilde{F}(x,y)$. If not, the check fails.

3: Termination procedure: If the check passes, P_i takes $\{f_{\ell,i}(x), g_{\ell,i}(y)\}_{\ell=1}^{N}$ as output. Otherwise, P_i takes `fail` as output and terminates.

Lemma 1. *Let $\epsilon = 0.1$. Suppose there are at most $(2+\epsilon)t+1$ parties including at most ϵt corrupted parties. The protocol $\Pi_{\mathtt{ShBi}}$ satisfies that*

- *If the dealer D is honest, then all honest parties eventually receive the correct shares.*
- *If the dealer D is corrupted, then if one honest party terminate, all honest parties eventually terminate. Moreover, either all honest parties take* `fail` *as output or there exists a set of $(2-3\epsilon)t+1$ honest parties whose shares lie on valid degree-d bivariate polynomials with probability $1 - N \cdot \binom{(2+\epsilon)t+1}{4\epsilon t}/|\mathbb{F}|$.*

We refer the readers to Section D.1 for the proof of Lemma 1.

Remark 1. We note that the failure error is $N \cdot \binom{(2+\epsilon)t+1}{4\epsilon t}/|\mathbb{F}| \approx N \cdot 2^{\mathcal{O}(n)}/|\mathbb{F}|$. To obtain a negligible failure error, we need the size of an element in $|\mathbb{F}|$ grows linearly in n. This can be addressed by doing the verification over a large extension field of $\mathbb{F}$. Taking this into consideration, the communication complexity of $\Pi_{\mathtt{ShBi}}$ becomes $\mathcal{O}(N \cdot n^2 + n^5)$ elements in $\mathbb{F}$.

We note that $\Pi_{\mathtt{ShBi}}$ does not guarantee that all honest parties' shares are correct when the dealer is corrupted. To address this issue, we use $\Pi_{\mathtt{ShBi}}$ as a tool to allow the dealer to commit a batch of shares. After the commitment, all parties check whether the committed shares form valid degree-d packed Shamir sharings. Then all parties reconstruct the shares to each party. When there are at most $(2+\epsilon)t+1$ parties, all honest parties can successfully reconstruct their shares. Consider the following protocol $\Pi_{\mathtt{ShPack}}$. The communication of $\Pi_{\mathtt{ShPack}}$ is $\mathcal{O}(N \cdot n + n^6)$ elements in $\mathbb{F}$.

Protocol $\Pi_{\mathtt{ShPack}}$

Dealer D

1: Let $[\boldsymbol{s}_1]_d, \ldots, [\boldsymbol{s}_N]_d$ be the N degree-d packed Shamir sharings that D wants to share.

2: D samples $B = (\epsilon t)^2$ random degree-d packed Shamir sharings $[\boldsymbol{s}_1^{(0)}]_d, \ldots, [\boldsymbol{s}_B^{(0)}]_d$. Then D samples n random degree-d bivariate polynomials $F_1^{(0)}(x,y), \ldots, F_n^{(0)}(x,y)$ such that $F_i^{(0)}(\beta_{j_1}, \beta_{j_2})$ is the i-th share of $[\boldsymbol{s}_{(j_1-1)\epsilon t+j_2}^{(0)}]_d$ for all $i \in \{1, 2\ldots, n\}$, $j_1, j_2 \in \{1, \ldots, \epsilon t\}$, where $\{\beta_j\}_{j=1}^{\epsilon t}$ are distinct field elements other than $\{\alpha_j\}_{j=1}^{n}$.

3: Let $N' = N/B$. For all $\ell \in \{1, \ldots, N'\}$,
  – D samples n random degree-d bivariate polynomials $F_1^{(\ell)}(x,y), \ldots, F_n^{(\ell)}(x,y)$ such that $F_i^{(\ell)}(\beta_{j_1}, \beta_{j_2})$ is the i-th share of $[\boldsymbol{s}_{(\ell-1)B+(j_1-1)\epsilon t+j_2}]_d$ for all $i \in \{1, 2\ldots, n\}$, $j_1, j_2 \in \{1, \ldots, \epsilon t\}$.

4: D invokes $\Pi_{\mathtt{ShBi}}$ to distributes $\{F_i^{(\ell)}(x,y)\}_{i\in\{1,\ldots,n\},\ell\in\{0,\ldots,N'\}}$.

Party P_i

1: P_i waits to receive either `fail` or $\{F_j^{(\ell)}(x,\alpha_i), F_j^{(\ell)}(\alpha_i,y)\}_{j\in\{1,\ldots,n\},\ell\in\{0,\ldots,N'\}}$. If `fail` is received, P_i outputs `fail` and terminates.

2: P_i sends a request to $\mathcal{F}_{\mathsf{coin}}$ and waits to receive a challenge r.

3: For all $j \in \{1, \ldots, n\}$, P_i computes $F_j(x,\alpha_i) = \sum_{\ell=0}^{N'} r^i F_j^{(\ell)}(x,\alpha_i)$ and $F_j(\alpha_i,y) = \sum_{\ell=0}^{N'} r^i F_j^{(\ell)}(\alpha_i,y)$.

4: P_i broadcasts $\{F_j(x,\alpha_i), F_j(\alpha_i,y)\}_{j=1}^{n}$.

5: P_i sets the property Q as P_i terminating the broadcast protocol led by P_j. Then all parties run $\mathcal{F}_{\mathsf{acs}}$ to agree on a set $\mathcal{B}$ of party P_{j_2} that broadcasts $\{(f_{j_1,j_2}(x), g_{j_1,j_2}(y))\}_{j_1=1}^{n}$.

6: For all $j_1 \in \{1, \ldots, n\}$, P_i checks whether there exists a subset of $(2-4\epsilon)t+1$ parties in $\mathcal{B}$ such that their polynomials lie on a degree-d bivariate polynomial $\tilde{F}_{j_1}(x,y)$. This is done by the following checks.
  – For all $j_2 \in \{1, 2, \ldots, n\}$, try to find $\tilde{f}_{j_1,j_2}(x)$ such that $\tilde{f}_{j_1,j_2}(\alpha_{j_3}) = g_{j_1,j_3}(\alpha_{j_2})$ for at least $(2-4\epsilon)t+1$ party P_{j_3}. This step is done by running the error-correction algorithm of the Reed-Solomon Code. If any $\tilde{f}_{j_1,j_2}(x)$ does not exist, the check fails.
  – Check whether $\{\tilde{f}_{j_1,j_2}(x)\}_{j_2=1}^{n}$ lie on a valid degree-d bivariate polynomial. If not, the check fails. Otherwise, let $\tilde{F}_{j_1}(x,y)$ denote this degree-d bivariate polynomial.
  – Check whether for at least $(2-4\epsilon)t+1$ parties, the received polynomials lie on $\tilde{F}_{j_1}(x,y)$. If not, the check fails.

7: Let $\{\tilde{F}_{j_1}(x,y)\}_{j_1=1}^n$ denote the degree-d bivariate polynomials constructed above. For all $j_2, j_3 \in \{1,\dots,\epsilon t\}$, P_i checks whether $\{\tilde{F}_{j_1}(\beta_{j_2},\beta_{j_3})\}_{j_1=1}^n$ is a valid degree-d packed Shamir sharing. If not, the check fails.
8: If any of the above check fails, P_i takes `fail` as output and terminates. Otherwise, P_i sends to each P_j the values $\{F_j^{(\ell)}(x,\alpha_i), F_j^{(\ell)}(\alpha_i,y)\}_{\ell\in\{1,\dots,N'\}}$.
9: For all $\ell \in \{1,\dots,N'\}$, upon receiving $(f_{i,j}^{(\ell)}(x), g_{i,j}^{(\ell)}(y))$ from $2t+1$ party P_j, P_i checks whether there exists a subset of $(2-4\epsilon)t+1$ parties such that their polynomials lie on a degree-d bivariate polynomial $\tilde{F}_i^{(\ell)}(x,y)$. This step is done in the same way as Step 5 above. If true, P_i computes $\tilde{F}_i^{(\ell)}(x,y)$. Otherwise P_i interpolates $\tilde{F}_i^{(\ell)}(x,y)$ by using the first $d+1$ received polynomials $f_{i,j}^{(\ell)}(x)$ from P_j. Then P_i takes $\{\tilde{F}_i^{(\ell)}(\beta_{j_1},\beta_{j_2}) \mid j_1,j_2 \in \{1,\dots,\epsilon t\}\}_{\ell=1}^{N'}$ as output.

Lemma 2. *Let $\epsilon = 0.1$. Suppose there are at most $(2+\epsilon)t+1$ parties including at most ϵt corrupted parties. The protocol $\Pi_{\texttt{ShPack}}$ satisfies that*

- *If the dealer D is honest, then all honest parties eventually receive the correct shares.*
- *If the dealer D is corrupted, then if one honest party terminate, all honest parties eventually terminate. Moreover, either all honest parties take* `fail` *as output or the shares of all honest parties lie on valid degree-d packed Shamir sharings with overwhelming probability.*

We refer the readers to Section D.2 for the proof of Lemma 2.

Distributing Packed Beaver Triples. We use $\Pi_{\texttt{ShPack}}$ to let a dealer distribute degree-d packed Beaver triples. The description of $\Pi_{\texttt{ShTriple}}$ appears below. The communication complexity of $\Pi_{\texttt{ShTriple}}$ is $\mathcal{O}(N \cdot n + n^6)$ elements.

Protocol $\Pi_{\texttt{ShTriple}}$

<u>**Dealer D**</u>

1: Let $([\boldsymbol{a}_\ell]_d, [\boldsymbol{b}_\ell]_d, [\boldsymbol{c}_\ell]_d)_{\ell=1}^N$ be the N degree-d packed Beaver triples that D wants to share. All parties agree on $2N+1$ distinct field elements $\alpha_0,\dots,\alpha_{2N}$.
2: D samples a random packed Beaver triple $([\boldsymbol{a}_0]_d, [\boldsymbol{b}_0]_d, [\boldsymbol{c}_0]_d)$. Then D computes two vectors of shared polynomials $[\boldsymbol{f}]_d, [\boldsymbol{g}]_d$ of degree N such that $[\boldsymbol{f}(\alpha_\ell)]_d = [\boldsymbol{a}_\ell]_d$ and $[\boldsymbol{g}(\alpha_\ell)]_d = [\boldsymbol{b}_\ell]_d$ for all $\ell \in \{0,\dots,N\}$. Finally, D computes a vector of shared polynomials $[\boldsymbol{h}]_d$ of degree $2N$ such that $[\boldsymbol{h}(\alpha_\ell)]_d = [\boldsymbol{c}_\ell]_d$ for all $\ell \in \{0,\dots,N\}$ and $\boldsymbol{h} = \boldsymbol{f} * \boldsymbol{g}$, where $*$ denotes the coordinate-wise multiplication.
3: D invokes $\Pi_{\texttt{ShPack}}$ to distribute $\{[\boldsymbol{f}(\alpha_\ell)]_d, [\boldsymbol{g}(\alpha_\ell)]_d\}_{\ell=0}^N$ and $\{[\boldsymbol{h}(\alpha_\ell)]_d\}_{\ell=0}^{2N}$.

<u>**All Parties**</u>

1: Each P_i waits to receive either `fail` or his shares of $\{[\boldsymbol{f}(\alpha_\ell)]_d, [\boldsymbol{g}(\alpha_\ell)]_d\}_{\ell=0}^N$ and $\{[\boldsymbol{h}(\alpha_\ell)]_d\}_{\ell=0}^{2N}$. If `fail` is received, P_i outputs `fail` and terminates.
2: Each P_i sends a request to $\mathcal{F}_{\mathsf{coin}}$ and waits to receive a challenge r.
3: Upon receiving r from $\mathcal{F}_{\mathsf{coin}}$, if $r \in \{1,\dots,N\}$, all parties output `fail` and terminate. Otherwise, all parties locally compute $([\boldsymbol{f}(r)]_d, [\boldsymbol{g}(r)]_d, [\boldsymbol{h}(r)]_d)$.
4: Each P_i broadcasts his shares of $([\boldsymbol{f}(r)]_d, [\boldsymbol{g}(r)]_d, [\boldsymbol{h}(r)]_d)$.
5: Each P_i sets the property Q as P_i terminating the broadcast protocol led by P_j. Then all parties run $\mathcal{F}_{\mathsf{acs}}$ to agree on a set $\mathcal{B}$ of party P_j that broadcasts his shares of $([\boldsymbol{f}(r)]_d, [\boldsymbol{g}(r)]_d, [\boldsymbol{h}(r)]_d)$.
6: Each P_i checks whether there exists a subset of $(2-\epsilon)t+1$ parties in $\mathcal{B}$ such that their shares form valid degree-d packed Shamir sharings $[\boldsymbol{x}]_d, [\boldsymbol{y}]_d, [\boldsymbol{z}]_d$. This is done by running the error-correction algorithm of the Reed-Solomon Code. If it does not hold for any of $[\boldsymbol{x}]_d, [\boldsymbol{y}]_d, [\boldsymbol{z}]_d$, the check fails.
7: Each P_i checks whether the secrets satisfy that $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$. If not, the check fails.
8: If any of the above check fails, P_i takes `fail` as output and terminates. Otherwise, P_i takes his shares of $\{[\boldsymbol{f}(\alpha_\ell)]_d, [\boldsymbol{g}(\alpha_\ell)]_d, [\boldsymbol{h}(\alpha_\ell)]_d\}_{\ell=1}^N$ as output.

Lemma 3. *Let $\epsilon = 0.1$. Suppose there are at most $(2+\epsilon)t+1$ parties including at most ϵt corrupted parties. The protocol $\Pi_{\texttt{ShTriple}}$ satisfies that, with overwhelming probability,*

- *If the dealer D is honest, then all honest parties eventually receive the correct shares.*
- *If the dealer D is corrupted, then if one honest party terminate, all honest parties eventually terminate. Moreover, either all honest parties take* `fail` *as output or all honest parties receive valid degree-d packed Beaver triples.*

We refer the readers to Section D.3 for the proof of Lemma 3.

Adapting the Approach in [CP17] for Packed Secret Sharings. We follow the standard approach in [CP17] and replace degree-t sharings by degree-d packed sharings. For the technique of Beaver triples, we use the packing technique from [GPS22]. When applying the technique of packed Beaver triples, all parties need to reduce some randomized degree-d packed Shamir sharings to degree-$(d-t)$. This is done by relying on an honest P_{king}. We describe the protocol $\Pi_{\mathtt{tripleExtPack}}$ below. The communication of $\Pi_{\mathtt{tripleExtPack}}$ is $\mathcal{O}(N \cdot n^2 + n^7)$ elements plus one invocation of $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(N \cdot n)$ degree-$(d-t)$ Shamir sharings.

Protocol $\Pi_{\mathtt{tripleExtPack}}$

1: **Distribution:**
Each party P_i samples N random degree-d packed Beaver triples $([a_\ell^{(i)}]_t, [b_\ell^{(i)}]_t, [c_\ell^{(i)}]_t)_{\ell=1}^N$. Then P_i acts as a dealer and distributes these N packed Beaver triples using $\Pi_{\mathtt{ShTriple}}$. For each P_j that terminates $\Pi_{\mathtt{ShTriple}}$ and does not output `fail` when P_i is the dealer, P_j sends $(\mathsf{support}, P_j, P_i)$ to all parties.

2: **Determine the Set of Successful Dealers:**
For each P_i, if P_{king} receives $(\mathsf{support}, P_j, P_i)$ from at least $t+1$ parties, P_{king} adds P_i to its list. P_{king} waits for $2t+1$ successful dealers. Let $\mathcal{D}$ be the set of successful dealers. P_{king} reliably broadcasts the set $\mathcal{D}$.

3: **Extracting Random Triples:**
For each party P_j, after receiving $\mathcal{D}$, for each $P_i \in \mathcal{D}$, P_j watis for the termination of $\Pi_{\mathtt{ShTriple}}$ where P_i acts as the dealer. If the output is `fail` for any $P_i \in \mathcal{D}$, P_j outputs `fail` and terminate. Otherwise, for all $\ell \in \{1, \ldots, N\}$, pick the first unused packed Beaver triple from each dealer in $\mathcal{D}$ and denote them by $\{([\boldsymbol{a}_i]_d, [\boldsymbol{b}_i]_d, [\boldsymbol{c}_i]_d)\}_{i=1}^{2t+1}$. Then run the following steps to extract a random packed Beaver triple:

1. All parties set two vector of polynomials of $\boldsymbol{f}, \boldsymbol{g}$ of degree t such that $[\boldsymbol{f}(\alpha_i)]_d = [\boldsymbol{a}_i]_d$ and $[\boldsymbol{g}(\alpha_i)]_d = [\boldsymbol{b}_i]_d$ for all $i \in \{1, \ldots, t+1\}$.
2. All parties locally compute $[\boldsymbol{f}(\alpha_i)]_d$, $[\boldsymbol{g}(\alpha_i)]_d$ for all $i \in \{t+2, \ldots, 2t+1\}$.
3. All parties use the packed Beaver triple $([\boldsymbol{a}_i]_d, [\boldsymbol{b}_i]_d, [\boldsymbol{c}_i]_d)$ to compute $[\boldsymbol{f}(\alpha_i) * \boldsymbol{g}(\alpha_i)]_d$ for all $i \in \{t+2, \ldots, 2t+1\}$ as follows.
   (a) For all $i \in \{t+2, \ldots, 2t+1\}$, all parties locally compute $[\boldsymbol{f}(\alpha_i) + \boldsymbol{a}_i]_d, [\boldsymbol{g}(\alpha_i) + \boldsymbol{b}_i]_d$.
   (b) All parties send their shares of $[\boldsymbol{f}(\alpha_i) + \boldsymbol{a}_i]_d, [\boldsymbol{g}(\alpha_i) + \boldsymbol{b}_i]_d$ to P_{king} for all $i \in \{t+2, \ldots, 2t+1\}$. Then for each degree-$d$ packed Shamir sharing $[\boldsymbol{z}]_d \in \{[\boldsymbol{f}(\alpha_i) + \boldsymbol{a}_i]_d, [\boldsymbol{g}(\alpha_i) + \boldsymbol{b}_i]_d\}_{i=t+2}^{2t+1}$,
      i. Upon receiving $2t+1$ shares from all parties, P_{king} checks whether there exists a subset of $(2-\epsilon)t+1$ shares that form a valid degree-d packed Shamir sharing. This is done by running the error-correction algorithm of the Reed-Solomon Code. If not, P_{king} sets $\boldsymbol{z} = \boldsymbol{0}$. Otherwise, P_{king} sets $\boldsymbol{z}$ to be the secrets of the reconstructed degree-d packed Shamir sharing.
      ii. P_{king} computes a degree-$(d-t)$ packed Shamir sharing $[\boldsymbol{z}]_{d-t}$.
      iii. P_{king} uses $\mathcal{F}_{\mathsf{ACSS}}$ to distribute all degree-$(d-t)$ packed Shamir sharings.
   (c) All parties wait to receive $\{[\boldsymbol{f}(\alpha_i) + \boldsymbol{a}_i]_{d-t}, [\boldsymbol{g}(\alpha_i) + \boldsymbol{b}_i]_{d-t}\}_{i=1}^{2t+1}$ from $\mathcal{F}_{\mathsf{ACSS}}$. Then all parties locally compute
$$\begin{aligned}&[\boldsymbol{f}(\alpha_i) * \boldsymbol{g}(\alpha_i)]_{2d-t}\\ =\ &[\boldsymbol{f}(\alpha_i) + \boldsymbol{a}_i]_{d-t} \cdot [\boldsymbol{g}(\beta_i) + \boldsymbol{b}_i]_{d-t} - [\boldsymbol{g}(\beta_i) + \boldsymbol{b}_i]_{d-t}[\boldsymbol{a}_i]_d\\ &- [\boldsymbol{f}(\alpha_i) + \boldsymbol{a}_i]_{d-t}[\boldsymbol{b}_i]_d + [\boldsymbol{c}_i]_d.\end{aligned}$$
4. All parties set a vector of polynomials $\boldsymbol{h}$ of degree $2t$ such that $[\boldsymbol{h}(\alpha_i)]_{2d-t} = [\boldsymbol{c}_i]_d$ for all $i \in \{1, \ldots, t+1\}$ and $[\boldsymbol{h}(\alpha_i)]_{2d-t} = [\boldsymbol{f}(\alpha_i) * \boldsymbol{g}(\alpha_i)]_{2d-t}$ for all $i \in \{t+2, \ldots, 2t+1\}$.
5. All parties output $([\boldsymbol{f}(\alpha_0)]_d, [\boldsymbol{g}(\alpha_0)]_d, [\boldsymbol{h}(\alpha_0)]_{2d-t})$.

4: **Verifying Degree-$(d-t)$ Packed Sharings:**
All parties check that P_{king} indeed distributes degree-$(d-t)$ packed Shamir sharings using $\mathcal{F}_{\mathsf{ACSS}}$. Assume that all degree-$(d-t)$ packed Shamir sharings distributed by P_{king} are $\{[\boldsymbol{z}_\ell]_{d-t}\}_{\ell=0}^{N'}$, where $N' = 2Nt-1$.

1. Each P_i sends a request to $\mathcal{F}_{\mathsf{coin}}$ and waits to receive r from $\mathcal{F}_{\mathsf{coin}}$.
2. All parties locally compute $[\boldsymbol{z}]_{d-t} = \sum_{\ell=0}^{N'} r^\ell [\boldsymbol{z}_\ell]_{d-t}$.
3. All parties send their shares of $[\boldsymbol{z}]_{d-t}$ to every other party.
4. Each party P_i uses the online error correction algorithm to reconstruct a degree-t Shamir secret sharing. Then P_i checks whether all shares lie on a degree-$(d-t)$ polynomial. If not, P_i outputs `fail`.

Lemma 4. *Let $\epsilon = 0.1$. Suppose there are at most $(2+\epsilon)t+1$ parties including at most ϵt corrupted parties. The protocol $\Pi_{\texttt{tripleExtPack}}$ satisfies that, with overwhelming probability, if P_{king} is honest, then all honest parties eventually receive correct packed Beaver triples.*

We refer the readers to Section D.4 for the proof of Lemma 4.

Converting from Packed Beaver Triples to Standard Beaver Triples. To obtain standard Beaver triples, we will rely on an honest P_{king} to transform a degree-$(2d-t)$ packed Shamir sharing $[\boldsymbol{x}]_{2d-t}$ to $d-t+1$ standard packed Shamir sharing $[x_1]_t, \ldots, [x_{d-t+1}]_t$. At a high level, this is done as follows.

– All parties together prepare correlated randomness $([\boldsymbol{r}]_{2d-t}, [r_1]_t, \ldots, [r_{d-t+1}]_t)$.
– All parties locally compute $[\boldsymbol{x}+\boldsymbol{r}]_{2d-t}$ and reconstruct the secrets to P_{king}. Then P_{king} uses $\mathcal{F}_{\mathsf{ACSS}}$ to share $[x_1+r_1]_t, \ldots, [x_{d-t+1}+r_{d-t+1}]_t$.
– Finally, all parties locally compute $[x_i]_t = [x_i+r_i]_t - [r_i]_t$.

We first show how to prepare correlated randomness $([\boldsymbol{r}]_{2d-t}, [r_1]_t, \ldots, [r_{d-t+1}]_t)$. In the following, we use $[x|_i]_t$ to denote a degree-t Shamir sharing with the secret x stored at position β_i. From the technique in [EGPS22], if all parties hold $[x_1|_1]_t, \ldots, [x_{2d-2t+1}|_{2d-2t+1}]_t$, then all parties can locally compute

$$[\boldsymbol{x}]_{2d-t+1} = [\boldsymbol{e}_1]_{2d-2t} \cdot [x_1|_1]_t + \ldots + [\boldsymbol{e}_{2d-2t+1}]_{2d-2t} \cdot [x_{2d-2t+1}|_{2d-2t+1}]_t,$$

where $\boldsymbol{x} = (x_1, \ldots, x_{2d-2t+1})$ and for all $i \in \{1, \ldots, 2d-2t+1\}$, $\boldsymbol{e}_i \in \mathbb{F}^{2d-2t+1}$ is the i-th unit vector (i.e., the i-th entry is 1 while all other entries are 0). This can be easily verified when considering the underlying polynomial of each (packed) Shamir secret sharing. Thus, our idea is to first prepare

$$([r_1|_1]_t, \ldots, [r_{2d-2t+1}|_{2d-2t+1}]_t), ([r_1]_t, \ldots, [r_{d-t+1}]_t),$$

which can be done by using $\mathcal{F}_{\mathsf{ACSS}}$. Then all parties locally compute a degree-$(2d-t)$ packed Shamir sharing of $(r_1, \ldots, r_{2d-2t+1})$ from $([r_1|_1]_t, \ldots, [r_{2d-2t+1}|_{2d-2t+1}]_t)$. Note that if we only focus on the first $d-t+1$ secrets $\boldsymbol{r} = (r_1, \ldots, r_{d-t+1})$, it can be viewed as a random degree-$(2d-t)$ packed Shamir sharing $[\boldsymbol{r}]_{2d-t}$. We describe the protocol $\Pi_{\texttt{ShDepack}}$ below. The communication of $\Pi_{\texttt{ShDepack}}$ is $\mathcal{O}(n^3)$ elements plus one invocation of $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(N \cdot n)$ degree-t Shamir sharings.

Protocol $\Pi_{\texttt{ShDepack}}$

Dealer D

1: Let $(([s_1^{(\ell)}|_1]_t, \ldots, [s_{2d-2t+1}^{(\ell)}|_{2d-2t+1}]_t), ([s_1^{(\ell)}]_t, \ldots, [s_{d-t+1}^{(\ell)}]_t))_{\ell=1}^N$ be the N tuples of sharings that D wants to share.
2: D samples a uniform tuple of correlated random sharings $([s_1^{(0)}|_1]_t, \ldots, [s_{2d-2t+1}^{(0)}|_{2d-2t+1}]_t), ([s_1^{(0)}]_t, \ldots, [s_{d-t+1}^{(0)}]_t)$.
3: D invokes $\mathcal{F}_{\mathsf{ACSS}}$ to distribute these $(N+1) \cdot (3d-3t+2)$ degree-t Shamir sharings.

All Parties

1: Each P_i waits to receive his shares of $(([s_1^{(\ell)}|_1]_t, \ldots, [s_{2d-2t+1}^{(\ell)}|_{2d-2t+1}]_t), ([s_1^{(\ell)}]_t, \ldots, [s_{d-t+1}^{(\ell)}]_t))_{\ell=0}^N$.
2: Each P_i sends a request to $\mathcal{F}_{\mathsf{coin}}$ and waits to receive a challenge r.
3: Upon receiving r from $\mathcal{F}_{\mathsf{coin}}$, all parties locally compute

$$\begin{aligned}&(([s_1|_1]_t, \ldots, [s_{2d-2t+1}|_{2d-2t+1}]_t), ([s_1]_t, \ldots, [s_{d-t+1}]_t)) \\ &= \sum_{\ell=0}^{N} r^\ell (([s_1^{(\ell)}|_1]_t, \ldots, [s_{2d-2t+1}^{(\ell)}|_{2d-2t+1}]_t), ([s_1^{(\ell)}]_t, \ldots, [s_{d-t+1}^{(\ell)}]_t)).\end{aligned}$$

4: All parties invoke $\mathcal{F}_{\mathsf{pubRec}}$ to reconstruct the secrets of $(([s_1|_1]_t, \ldots, [s_{2d-2t+1}|_{2d-2t+1}]_t), ([s_1]_t, \ldots, [s_{d-t+1}]_t))$ and check the correctness of the correlation among the secrets. If the check passes, all parties takes their shares of $(([s_1^{(\ell)}|_1]_t, \ldots, [s_{2d-2t+1}^{(\ell)}|_{2d-2t+1}]_t), ([s_1^{(\ell)}]_t, \ldots, [s_{d-t+1}^{(\ell)}]_t))_{\ell=0}^N$ as output. Otherwise, P_i outputs `fail`.

With $\Pi_{\texttt{ShDepack}}$, we prepare $([r_1|_1]_t, \ldots, [r_{2d-2t+1}|_{2d-2t+1}]_t), ([r_1]_t, \ldots, [r_{d-t+1}]_t)$, the correlated randomness, as follows.

1. Each party acts as a dealer to distribute uniformly random tuples.

2. All parties use $\mathcal{F}_{\mathsf{acs}}$to determine the set of successful dealers.
3. All parties apply randomness extraction to obtain uniformly random tuples that are not known to any party.

We summarize the protocol $\Pi_{\texttt{randDepack}}$ as follows. The communication complexity of $\Pi_{\texttt{randDepack}}$ is $\mathcal{O}(n^4)$ elements plus n invocations of $\mathcal{F}_{\mathsf{ACSS}}$ to share $N \cdot n$ degree-t Shamir sharings in total.

Protocol $\Pi_{\texttt{randDepack}}$

1: Let $N' = N/(t+1)$. Each party P_i acts as a dealer and invokes $\Pi_{\texttt{ShDepack}}$ to distributes N' tuples of correlated randomness $(([s_1^{(i,\ell)}|_1]_t, \ldots, [s_{2d-2t+1}^{(i,\ell)}|_{2d-2t+1}]_t), ([s_1^{(i,\ell)}]_t, \ldots, [s_{d-t+1}^{(i,\ell)}]_t))_{\ell=1}^N$.
2: Each party P_i sets the property Q as P_i terminating $\Pi_{\texttt{ShDepack}}$ when P_j acts as a dealer and his output is not `fail`. Then all parties run $\mathcal{F}_{\mathsf{acs}}$ 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 $(t+1)) \times (2t+1)$.
4: For all $\ell \in \{1, \ldots, N'\}$, all parties locally compute

$$\begin{aligned}&(([r_1^{(i,\ell)}|_1]_t, \ldots, [r_{2d-2t+1}^{(i,\ell)}|_{2d-2t+1}]_t), ([r_1^{(i,\ell)}]_t, \ldots, [r_{d-t+1}^{(i,\ell)}]_t))_{i=1}^{t+1}\\ =&\boldsymbol{M} \cdot (([s_1^{(i,\ell)}|_1]_t, \ldots, [s_{2d-2t+1}^{(i,\ell)}|_{2d-2t+1}]_t), ([s_1^{(i,\ell)}]_t, \ldots, [s_{d-t+1}^{(i,\ell)}]_t))_{i\in\mathcal{D}}.\end{aligned}$$

Finally, all parties output $([r_1^{(i,\ell)}|_1]_t, \ldots, [r_{2d-2t+1}^{(i,\ell)}|_{2d-2t+1}]_t)$, $([r_1^{(i,\ell)}]_t, \ldots, [r_{d-t+1}^{(i,\ell)}]_t)$ for all $i \in \{1, \ldots, t+1\}, \ell \in \{1, \ldots, N'\}$.

Following from similar arguments to those for $\mathcal{F}_{\mathsf{randShare}}$ in Section B, we can show that $\Pi_{\texttt{randDepack}}$ securely realize $\mathcal{F}_{\mathsf{randDepack}}$ defined below.

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

1: For all $\ell \in \{1, \ldots, N\}$, the functionality randomly samples $r_1^{(\ell)}, \ldots, r_{2d-2t+1}^{(\ell)}$.
2: For all $\ell \in \{1, \ldots, N\}$, the functionality waits to receive a set of shares of corrupted parties from $\mathcal{S}$ and samples random degree-t Shamir sharings $([r_1^{(\ell)}|_1]_t, \ldots, [r_{2d-2t+1}^{(\ell)}|_{2d-2t+1}]_t), ([r_1^{(\ell)}]_t, \ldots, [r_{d-t+1}^{(\ell)}]_t)$ based on the shares of corrupted parties and the secrets $r_1^{(\ell)}, \ldots, r_{2d-2t+1}^{(\ell)}$.
3: For all $\ell \in \{1, \ldots, N\}$, the functionality distributes the shares of $([r_1^{(\ell)}|_1]_t, \ldots, [r_{2d-2t+1}^{(\ell)}|_{2d-2t+1}]_t), ([r_1^{(\ell)}]_t, \ldots, [r_{d-t+1}^{(\ell)}]_t)$ to all parties as request-based delayed outputs.

Now we present Π_{depack}, which transforms a batch of degree-$(2d-t)$ packed Shamir sharings to $(d-t+1)$ degree-t Shamir sharings relying on an honest P_{king}. The communication complexity of Π_{depack} is $\mathcal{O}(N \cdot n + n^4)$ elements plus $n+1$ invocations of $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(N \cdot n)$ degree-t Shamir sharings in total.

Protocol Π_{depack}

1: **Input:**
All parties together hold N degree-$(2d-t)$ packed Shamir sharings $([\boldsymbol{x}^{(1)}]_{2d-t}, \ldots, [\boldsymbol{x}^{(N)}]_{2d-t})$ where each $\boldsymbol{x}^{(i)}$ is a vector of dimension $d-t+1$.
2: **Prepare Correlated Randomness:**
All parties invoke $\mathcal{F}_{\mathsf{randDepack}}$ to prepare N tuples of correlated randomness:

$$(([r_1^{(\ell)}|_1]_t, \ldots, [r_{2d-2t+1}^{(\ell)}|_{2d-2t+1}]_t), ([r_1^{(\ell)}]_t, \ldots, [r_{d-t+1}^{(\ell)}]_t))_{\ell=1}^N.$$

Then for all $\ell \in \{1, \ldots, N\}$, all parties locally compute $[\boldsymbol{r}^{(\ell)}]_{2d-t} = [\boldsymbol{e}_1]_{2d-2t} \cdot [r_1^{(\ell)}|_1]_t + \ldots + [\boldsymbol{e}_{2d-2t+1}]_{2d-2t} \cdot [r_{2d-2t+1}^{(\ell)}|_{2d-2t+1}]_t$, where $\boldsymbol{r}^{(\ell)} = (r_1^{(\ell)}, \ldots, r_{d-t+1}^{(\ell)})$.
3: **Applying Transformation:**
For all $\ell \in \{1, \ldots, N\}$, all parties locally compute $[\boldsymbol{x}^{(\ell)} + \boldsymbol{r}^{(\ell)}]_{2d-t} = [\boldsymbol{x}^{(\ell)}]_{2d-t} + [\boldsymbol{r}^{(\ell)}]_{2d-t}$. All parties send their shares of $[\boldsymbol{x}^{(\ell)} + \boldsymbol{r}^{(\ell)}]_{2d-t}$ to P_{king}. Then for each degree-$(2d-t)$ packed Shamir sharing $[\boldsymbol{z}]_{2d-t} \in \{[\boldsymbol{x}^{(\ell)} + \boldsymbol{r}^{(\ell)}]_{2d-t}\}_{\ell=1}^N$,
   1. Upon receiving $2t+1$ shares from all parties, P_{king} checks whether there exists a subset of $(2-\epsilon)t+1$ shares that form a valid degree-$(2d-t)$ packed Shamir sharing. This is done by running the error-

correction algorithm of the Reed-Solomon Code. If not, P_{king} sets $\boldsymbol{z} = \mathbf{0}$. Otherwise, P_{king} sets $\boldsymbol{z}$ to the the secrets of the reconstructed degree-$(2d-t)$ packed Shamir sharing.
2. P_{king} generates $d-t+1$ degree-t Shamir sharings $[z_1]_t, [z_2]_t, \ldots, [z_{d-t+1}]_t$.
3. P_{king} uses $\mathcal{F}_{\mathsf{ACSS}}$ to distribute all degree-t Shamir sharings.

4: **Computing Output:**
All parties wait to receive their shares of $\{[x_i^{(\ell)} + r_i^{(\ell)}]_t\}_{i=1}^{d-t+1}$ for all $\ell \in \{1, \ldots, N\}$. Then all parties locally compute $[x_i^{(\ell)}]_t = [x_i^{(\ell)} + r_i^{(\ell)}]_t - [r_i^{(\ell)}]_t$ for all $i \in \{1, \ldots, d-t+1\}, \ell \in \{1, \ldots, N\}$.
All parties take their shares of $\{[x_i^{(\ell)}]_t\}_{i=1}^{d-t+1}$ for all $\ell \in \{1, \ldots, N\}$ as output.

Summary of Preparing Beaver Triples in Process 2. By combining the above protocols together, we obtain a protocol that prepares random Beaver triples, which is guaranteed to succeed when at most $(2+\epsilon)t+1$ parties participate including at most ϵt corrupted parties. We summarize the protocol in $\Pi_{\mathtt{tripleGen}}$. The communication complexity of $\Pi_{\mathtt{tripleGen}}$ is $\mathcal{O}(N \cdot n + n^7)$ elements plus $n+2$ invocations of $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(N)$ degree-t (or degree-$(d-t)$) Shamir sharings.

Protocol $\Pi_{\mathtt{tripleGen}}$

1: **Preparing Packed Beaver Triples:**
Let $N' = N/(d-t+1)$. All parties invoke $\Pi_{\mathtt{tripleExtPack}}$ to prepare N' packed Beaver triples $\{[\boldsymbol{a}^{(\ell)}]_d, [\boldsymbol{b}^{(\ell)}]_d, [\boldsymbol{c}^{(\ell)}]_{2d-t}\}_{\ell=1}^{N'}$.

2: **Applying Conversion:**
All parties invoke Π_{depack} to transform $\{[\boldsymbol{a}^{(\ell)}]_d, [\boldsymbol{b}^{(\ell)}]_d, [\boldsymbol{c}^{(\ell)}]_{2d-t}\}_{\ell=1}^{N'}$ to $\{([a_i^{(\ell)}]_t, [b_i^{(\ell)}]_t, [c_i^{(\ell)}]_t)\}_{i\in\{1,\ldots,d-t+1\},\ell\in\{1,\ldots,N'\}}$.
All parties take their shares of $([a_i^{(\ell)}]_t, [b_i^{(\ell)}]_t, [c_i^{(\ell)}]_t)$ for all $i \in \{1, \ldots, d-t+1\}, \ell \in \{1, \ldots, N'\}$ as output.

Checking Correctness of Beaver triples. Recall that we need to ensure that if there are at least $(2+\epsilon)t+1$ parties participating or there are at least ϵt corrupted parties, the Beaver triples should be either correct or rejected by all honest parties. For this, parties check the obtained triples at the end of Process 2. The communication of Process 2 is $\mathcal{O}(N \cdot n + n^7)$ elements plus $n+2$ invocations of $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(N)$ degree-t (or degree-$(d-t)$) Shamir sharings.

Process $\Pi_{\mathtt{tripleGen\text{-}GOD}}$

1: **Prepare Random Beaver Triples:**
All parties invoke $\Pi_{\mathtt{tripleGen}}$ to prepare $2N+1$ random Beaver triples, denoted by $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=0}^{2N}$.

2: **Build Polynomials:**
All parties agree on $2N+1$ distinct field elements $\alpha_0, \ldots, \alpha_{2N}$. Then all parties run the following steps.
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 $i \in \{0, \ldots, N\}$.
2. All parties locally compute $[f(\alpha_i)]_t$, $[g(\alpha_i)]_t$ for all $i \in \{N+1, \ldots, 2N\}$.
3. All parties use the Beaver triple $([a_i]_t, [b_i]_t, [c_i]_t)$ to compute $[f(\alpha_i) \cdot g(\alpha_i)]_t$ for all $i \in \{N+1, \ldots, 2N\}$ as follows.
   (a) For all $i \in \{N+1, \ldots, 2N\}$, all parties locally compute $[f(\alpha_i) + a_i]_t, [g(\alpha_i) + b_i]_t$.
   (b) All parties invoke $\mathcal{F}_{\mathsf{pubRec}}$ to reconstruct the secrets $f(\alpha_i) + a_i, g(\beta_i) + b_i$ for all $i \in \{N+1, \ldots, 2N\}$.
   (c) All parties locally compute
$$
\begin{aligned}
[f(\alpha_i) \cdot g(\alpha_i)]_t = &(f(\alpha_i) + a_i) \cdot (g(\beta_i) + b_i) - (g(\beta_i) + b_i)[a_i]_t \\
&- (f(\alpha_i) + a_i)[b_i]_t + [c_i]_t.
\end{aligned}
$$
4. All parties set a polynomial h of degree $2N$ such that $[h(\alpha_i)]_t = [c_i]_t$ for all $i \in \{0, \ldots, N\}$ and $[h(\alpha_i)]_t = [f(\alpha_i) \cdot g(\alpha_i)]_t$ for all $i \in \{N+1, \ldots, 2N\}$.

3: **Verification:**
All parties send requests to $\mathcal{F}_{\mathsf{coin}}$ and wait to receive a random value r.
Upon receiving r, if $r \in \{1, \ldots, N\}$, all parties output `fail` and terminate. Otherwise, all parties locally compute $([f(r)]_t, [g(r)]_t, [h(r)]_t)$. Then all parties invoke $\mathcal{F}_{\mathsf{pubRec}}$ to reconstruct $f(r), g(r), h(r)$ and check whether $f(r) \cdot g(r) = h(r)$. If true, all parties take their shares of $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=1}^{N}$ as output. Otherwise, all parties take `fail` as output and terminate.

4.3 Overall Protocol for Preparing Beaver Triples

We present the protocol for Beaver triples relying on an honest P_{king}.

Protocol $\Pi_{\texttt{tripleKing-GOD}}$

1: **Run Process 1 and Process 2:**
All parties agree on a party P_{king} and invoke $\Pi_{\texttt{tripleExt-GOD}}$ and $\Pi_{\texttt{tripleGen-GOD}}$ in parallel. In particular, every party P_i accepts messages from P_j if and only if P_i terminates $\mathcal{F}_{\mathsf{ACSS}}$ led by P_j in $\Pi_{\texttt{tripleExt-GOD}}$.

2: **Agree on Successful Process:**
Each party P_i sets his input to be $b = 0$ if the first process first succeeds, and sets his input $b = 1$ if the second process first succeeds and his output is not `fail`. (Note that it is possible that both processes succeed.) All parties run a BA protocol. If the final output $b = 0$, then all parties take the output of the first process as the final output. Otherwise, all parties take the output of the second process as the final output.

To remove the assumption that P_{king} is a trusted party, we run $\Pi_{\texttt{tripleKing}}$ n times where each time a different party behaves as P_{king}. Finally, parties use $\mathcal{F}_{\mathsf{acs}}$to agree on the successful kings. In Lemma 5, we will show the following:

- If P_{king} is honest, then all honest parties eventually terminate $\Pi_{\texttt{tripleKing}}$.
- Even if P_{king} is corrupted, if an honest party terminates in the process $\Pi_{\texttt{tripleKing}}$, then all honest parties will eventually terminate and output correct and secure random Beaver triples.

We may set the ACS property Q to be P_i terminating $\Pi_{\texttt{tripleKing}}$ when P_j is the king. After agree on the set of successful kings, all parties output the triples generated in $\Pi_{\texttt{tripleKing}}$ led by successful kings. The communication complexity of $\Pi_{\texttt{triple-GOD}}$ is $\mathcal{O}(N \cdot n + n^8)$ elements plus $\mathcal{O}(n^2)$ invocations of $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(N)$ degree-t Shamir sharings in total. We describe the protocol and functionality and state Lemma 5, whose proof is in Section D.5.

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

1: **Run $\Pi_{\texttt{tripleKing-GOD}}$ with Different Kings:**
For all $i \in \{1, \ldots, n\}$, all parties set P_i as P_{king} and invoke $\Pi_{\texttt{tripleKing-GOD}}$ with $N' = N/(2t + 1)$.

2: **Agree on Successful Kings:**
Each party P_i sets the property Q as P_i terminating $\Pi_{\texttt{tripleKing}}$ when P_j is P_{king}. Then all parties run $\mathcal{F}_{\mathsf{acs}}$ to agree on a set $\mathcal{K}$ of successful kings. All parties output the triples prepared in $\Pi_{\texttt{tripleKing-GOD}}$ led by the first $2t + 1$ successful kings.

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

1: Let N denote the number of random Beaver triples to be prepared. For all $i \in \{1, \ldots, N\}$, the functionality randomly samples a_i, b_i, c_i such that $c_i = a_i \cdot b_i$.

2: For all $i \in \{1, \ldots, N\}$, the functionality waits to receive a set of shares $\{u_{i,j}, v_{i,j}, w_{i,j}\}_{j \in \mathcal{C}orr}$ of corrupted parties from $\mathcal{S}$ and samples 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.

3: For all $i \in \{1, \ldots, N\}$, the functionality distributes the shares of $([a_i]_t, [b_i]_t, [c_i]_t)$ to all parties as request-based delayed outputs.

Lemma 5. *Protocol $\Pi_{\texttt{triple-GOD}}$ securely computes $\mathcal{F}_{\mathsf{triple}}$ in the hybrid model with functionalities $\{\mathcal{F}_{\mathsf{ACSS}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{pubRec}}, \mathcal{F}_{\mathsf{randDepack}}\}$, against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t < n/3$ parties.*

5 Putting it all Together

In Section C, we show a protocol with guaranteed output delivery in the hybrid model with functionalities in $\{\mathcal{F}_{\mathsf{triple}}, \mathcal{F}_{\mathsf{randShare}}, \mathcal{F}_{\mathsf{pubRec}}\}$. The protocol follows standard techniques and achieves linear communication in the online phase. To get a full construction,

– We use our protocol $\Pi_{\texttt{triple-GOD}}$ to instantiate $\mathcal{F}_{\mathsf{triple}}$ in the hybrid model with the functionalities in $\{\mathcal{F}_{\mathsf{ACSS}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{pubRec}}, \mathcal{F}_{\mathsf{randDepack}}\}$.
– Then we instantiate $\mathcal{F}_{\mathsf{randDepack}}$ by the protocol $\Pi_{\texttt{randDepack}}$ in the $\mathcal{F}_{\mathsf{ACSS}}$-hybrid model.
– Finally, $\mathcal{F}_{\mathsf{randShare}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{pubRec}}$ can be instantiated as described in Section B.

Theorem 4. *Let $n = 3t + 1$. For any circuit C of size $|C|$ and depth D, there is a fully malicious asynchronous MPC protocol computing C that is secure against at most t corrupted parties with guaranteed output delivery in the $\mathcal{F}_{\mathsf{ACSS}}$-hybrid model. The achieved communication complexity is $\mathcal{O}(|C| \cdot n + D \cdot n^2 + n^8)$ elements plus $\mathcal{O}(n^2)$ invocations of $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(|C|)$ degree-t Shamir sharings.*

References

[AAPP22] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Asymptotically free broadcast in constant expected time via packed VSS. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 384–414. Springer, Heidelberg, November 2022.

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

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *25th ACM STOC*, pages 52–61. ACM Press, May 1993.

[BFO12] 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, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 663–680. Springer, Heidelberg, August 2012.

[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.

[BH08] Zuzana Beerliova-Trubiniova and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *Theory of Cryptography Conference — TCC 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer-Verlag, 3 2008.

[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.

[Bra84] Gabriel Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, PODC '84, page 154–162, New York, NY, USA, 1984. 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 Beerliová-Trubíniová and Martin Hirt. Efficient multi-party computation with dispute control. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 305–328. Springer, Heidelberg, March 2006.

[BTH08] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 213–230. Springer, Heidelberg, March 2008.

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

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

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

[CDD+99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 311–326. Springer, Heidelberg, May 1999.

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

[CGHZ16] Sandro Coretti, Juan A. Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multiparty 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, Heidelberg, December 2016.

[CHP13] Ashish Choudhury, Martin Hirt, and Arpita Patra. Asynchronous multiparty computation with linear communication complexity. In *International Symposium on Distributed Computing*, pages 388–402. Springer, 2013.

[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, 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.

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 501–520. Springer, Heidelberg, August 2006.

[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.

[EGPS22] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Turbopack: Honest majority mpc with constant online communication. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 951–964, New York, NY, USA, 2022. Association for Computing Machinery.

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[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.

[GLS19] Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional MPC with guaranteed output delivery. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 85–114. Springer, Heidelberg, August 2019.

[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, *Advances in Cryptology – CRYPTO 2022*, pages 3–32, Cham, 2022. Springer Nature Switzerland.

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

[HN06] Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 463–482. Springer, Heidelberg, August 2006.

[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, 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, Heidelberg, July 2008.

[JLS24] Xiaoyu Ji, Junru Li, and Yifan Song. Linear-communication asynchronous complete secret sharing with optimal resilience. Eprint, 2024.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.

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

[Pat11] Arpita Patra. Error-free multi-valued broadcast and byzantine agreement with optimal communication complexity. In Antonio Fernàndez Anta, Giuseppe Lipari, and Matthieu Roy, editors, *Principles of Distributed Systems*, pages 34–49, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

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

[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, Heidelberg, December 2010.

[PCR15] Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Efficient asynchronous verifiable secret sharing and multiparty computation. *Journal of Cryptology*, 28(1):49–109, January 2015.

[PSR02] B. Prabhu, K. Srinathan, and C. Pandu Rangan. Asynchronous unconditionally secure computation: An efficiency improvement. In Alfred Menezes and Palash Sarkar, editors, *INDOCRYPT 2002*, volume 2551 of *LNCS*, pages 93–107. Springer, Heidelberg, December 2002.

[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.

[RS22] Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid mpc for dishonest majority. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 719–749, Cham, 2022. Springer Nature Switzerland.

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

[SR00] K. Srinathan and C. Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In Bimal K. Roy and Eiji Okamoto, editors, *INDOCRYPT 2000*, volume 1977 of *LNCS*, pages 117–129. Springer, Heidelberg, December 2000.

[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 Universal Composability

The Universal Composability (UC) framework was introduced by Canetti [Can01], and is based on the real and ideal world paradigm [Can00]. The model compares a real execution of the protocol among the parties with an ideal execution where a trusted party (the ideal functionality) interacts with the parties. A protocol is then secure if whatever an adversary can do in the real protocol, can be also achieved in the ideal execution.

Real World. In the real world, there is a set of n parties, $P_1, \ldots, P_n$, an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$. The environment provides inputs to the honest parties, receive their outputs and communicates with the adversary $\mathcal{A}$. The adversary has full control over the corrupted parties and the delivery of messages between parties. For simplicity, we consider a static adversary that can corrupt up to t parties at the start of the protocol. The adversary has full control over the corrupted parties.

More concretely, each entity is modelled as an interactive Turing machine (ITM), initialized with the random coins and possible inputs. The protocol proceeds by a sequence of activations, where at each point only a single ITM is active. When a party is activated, it can perform local computation and output or send a messages to other parties. And if the adversary is activated, it can send messages on behalf of the corrupted parties.

Parties have access to a network of point-to-point asynchronous and secure channels. Asynchronous channels guarantee *eventual* delivery [CR98], meaning that messages sent are eventually delivered, and the scheduling of the messages is done by the adversary. The adversary cannot drop, change or inject messages from honest parties, but it can decide which message will be delivered next and when. Such channels have been modeled in UC using the eventual-delivery message transmission functionalities, for example in [CGHZ16, KMTZ13]. The protocol completes once $\mathcal{Z}$ outputs a single bit. We denote by $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z, \overline{r})$ the random variable consisting of the output of $\mathcal{Z}$ with input z, security parameter κ, and interacting with the parties $P_1, \ldots, P_n$ and the adversary $\mathcal{A}$ with random tapes $\overline{r} = (r_1, \ldots, r_n, r_{\mathcal{A}}, r_{\mathcal{Z}})$. Let $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$ denote the random variable $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z, \overline{r})$ for uniform random $\overline{r}$.

Ideal World. The ideal world contains n dummy parties, an ideal-world adversary $\mathcal{S}$ (the simulator), an environment $\mathcal{Z}$ and an ideal functionality $\mathcal{F}$ (the trusted party). The environment gives inputs to the honest parties, receives outputs and also interacts with the ideal adversary. As before, the computation finishes once $\mathcal{Z}$ outputs a single bit.

The ideal functionality models the desired behavior of the computation. In order to model the fact that the adversary can decide when each honest party learns the output, we follow [KMTZ13] and model time via activations. Here, when the functionality $\mathcal{F}$ prepares an output for some party, the party requests $\mathcal{F}$ for the output, and the adversary can instruct $\mathcal{F}$ to delay the output for each party. The party will then eventually receive the output when the environment activates the party sufficiently many times. As in [Coh16, CP23], we say that $\mathcal{F}$ sends a *request-based delayed output* to P_i to describe such behavior.

We denote by $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z, \overline{r})$ the random variable consisting of the output of $\mathcal{Z}$ with input z, security parameter κ, and interacting with the dummy parties $P_1, \ldots, P_n$ and the adversary $\mathcal{S}$ and functionality $\mathcal{F}$ with random tapes $\overline{r} = (r_{\mathcal{S}}, r_{\mathcal{Z}})$. Let $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z)$ denote the random variable $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z, \overline{r})$ for uniform random $\overline{r}$.

Definition 1. *We say that a protocol Π securely computes $\mathcal{F}$ against a fully malicious adversary $\mathcal{A}$ corrupting at most t parties, if for any adversary controlling up to t parties and any environment $\mathcal{Z}$, there exists an ideal adversary $\mathcal{S}$ such that the following ensembles are statistically close:*

$$\{\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*} \approx_{negl(\kappa)} \{\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z)\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*}$$

We also consider a hybrid model where parties have some ideal functionality available which the real protocol can invoke. We denote by $\mathsf{Hybrid}^{\mathcal{F}}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$ the output of $\mathcal{Z}$ from a hybrid execution of Π with ideal calls to $\mathcal{F}$, and $\mathcal{A}$, $\mathcal{Z}$, κ, z as defined above. The composition theorem states that if a protocol Π realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, and protocol ρ realizes $\mathcal{G}$ in the plain model, then the protocol π^ρ realizes $\mathcal{F}$ in the plain model. Here, in the protocol π^ρ the parties invoke the code of the protocol ρ instead of calling $\mathcal{G}$. See [Can01] for details.

In the description of our protocols and functionalities, we try to avoid over-formalism (e.g. we ignore the sid and assume they are implicit).

B Additional Preliminaries

B.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 $\mathcal{F}_{\mathsf{rbc}}$

The functionality runs 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 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 $\mathcal{F}_{\mathsf{ba}}$

The functionality runs with parties $P_1, \ldots, P_n$ and the adversary $\mathcal{S}$. Let $\mathcal{I} = \mathcal{H}$, where $\mathcal{H}$ is the set of honest parties. For each P_i, initialize x_i and y_i to $\perp$.

1: Upon receiving P' from the adversary, with $|P'| \leq t$, if no party has received output, then set $\mathcal{I} = \mathcal{H} \setminus P'$.
2: Upon receiving an input b from party P_i, do as follows.
  – If any party or the adversary has received output, ignore this message; otherwise, set $x_i = b$.
  – If $x_i \neq \perp$ for every $P_i \in \mathcal{I}$, 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.
  – Notify the adversary that P_i has given input.
3: When the output y_i is set to be some value v, the functionality outputs v 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 2. *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 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 a set S_i of parties. We say that Π is a t-resilient ACS protocol for Q if the following holds whenever up to t parties are corrupted:*

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

– *Termination: All honest parties eventually terminate.*

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

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

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

The functionality runs 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 indices k such that there is some honest party that input 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.

B.2 Further Functionalities

Distributing Degree-t Shamir Sharings. The description of $\mathcal{F}_{\mathsf{ACSS}}$ appears below. Note that $\mathcal{F}_{\mathsf{ACSS}}$ only distributes the shares to all parties if it receives the degree-t Shamir sharings from the dealer. Therefore, if the dealer is honest, all parties eventually receive their shares of $\{[s_i]_t\}_{i=1}^N$. If the dealer is corrupted, then the trusted party may wait forever and in this case no honest party receives his shares. Note that, in other words, if an honest party receives his shares from $\mathcal{F}_{\mathsf{ACSS}}$, then all honest parties would eventually receive their shares. Following [CP23], $\mathcal{F}_{\mathsf{ACSS}}$ can be realized with communication complexity $\mathcal{O}(N \cdot n^3 + n^4 \cdot \kappa + n^5)$ elements.

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

The functionality runs with parties $P_1, \ldots, P_n$, where one of the parties is the dealer P_d, and the adversary $\mathcal{S}$.

1: The functionality receives a number N from the dealer, indicating the number of secrets to be shared.
2: The functionality waits to receive N degree-t Shamir sharings $[s_1]_t, \ldots, [s_N]_t$ from the dealer. If received, the functionality distributes the shares to all parties as request-based delayed outputs.

Generating Random Degree-t Shamir Sharings. The description of $\mathcal{F}_{\mathsf{randShare}}$ appears below. $\mathcal{F}_{\mathsf{randShare}}$ can be realized in the $\mathcal{F}_{\mathsf{ACSS}}$-hybrid model relying on known techniques [DN07] in the synchronous setting. For completeness, we give the construction and the security proof below.

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

The functionality runs with parties $P_1, \ldots, P_n$ and the adversary $\mathcal{S}$.

1: For all $\ell \in \{1, \ldots, N\}$, the functionality randomly samples r_ℓ.
2: For all $\ell \in \{1, \ldots, N\}$, the functionality waits to receive 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. (If not received, the functionality sets the shares of corrupted parties to be 0.)
3: For all $\ell \in \{1, \ldots, N\}$, the functionality distributes the shares of $[r_\ell]_t$ to all parties as request-based delayed outputs.

Protocol $\Pi_{\mathtt{randSh}}$

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 $\mathcal{F}_{\mathsf{ACSS}}$ to distribute the shares to all parties.
2: Each party P_i sets the property Q as P_i terminating $\mathcal{F}_{\mathsf{ACSS}}$ when P_j acts as a dealer. Then all parties run Π_{acs}^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 $(t+1) \times (2t+1)$.
4: 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 $\{[r_{\ell,i}]_t\}_{\ell \in \{1,\ldots,N'\}, i \in \{1,\ldots,t+1\}}$.

Lemma 6. *Protocol $\Pi_{\mathtt{randSh}}$ securely computes $\mathcal{F}_{\mathsf{randShare}}$ against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t < n/3$ parties.*

Proof. We first show that all honest parties will eventually terminate the protocol $\Pi_{\mathtt{randSh}}$. By the definition of $\mathcal{F}_{\mathsf{ACSS}}$, the property Q is an ACS property. Thus in Step 2 of $\Pi_{\mathtt{randSh}}$, all honest parties will eventually agree on a set $\mathcal{D}$ of successful dealers. By the definition of $\mathcal{F}_{\mathsf{ACSS}}$ again, for each dealer in $\mathcal{D}$, all honest parties will eventually receive the shares distributed by this dealer. Since Step 3 and Step 4 only involve local computation, all honest parties will eventually terminate the protocol $\Pi_{\mathtt{randSh}}$.

Now we show that the protocol $\Pi_{\mathtt{randSh}}$ securely computes $\mathcal{F}_{\mathsf{randShare}}$. Let $\mathcal{A}$ be a static malicious adversary which controls a set $\mathcal{C}orr$ of $t' \le t$ corrupted parties. Let $\mathcal{Z}$ be an environment. We construct an ideal adversary $\mathcal{S}$ interacting with the environment $\mathcal{Z}$ and the ideal functionality $\mathcal{F}_{\mathsf{fs}}$. $\mathcal{S}$ starts with running $\mathcal{A}$ and passes messages between $\mathcal{Z}$ and $\mathcal{A}$. For corrupted parties, $\mathcal{S}$ faithfully follows the instructions of $\mathcal{A}$. Then $\mathcal{S}$ simulates the behaviors of honest parties as follows. In Step 1, for each honest party P_i, $\mathcal{S}$ generates random values as the shares of corrupted parties. Then $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS}}$ and sends the shares of corrupted parties to them. For each corrupted party P_i, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS}}$ and waits to receive the sharings distributed by P_i from $\mathcal{A}$. If received, $\mathcal{S}$ sends the shares of corrupted parties to them.

In Step 2, $\mathcal{S}$ follows the protocol Π_{acs}^Q and learns the set $\mathcal{D}$ of size $2t+1$. In Step 3 and Step 4, $\mathcal{S}$ follows the protocol and computes the shares of $[r_{\ell,i}]_t$ of corrupted parties for all $\ell \in \{1, \ldots, N'\}, i \in \{1, \ldots, T\}$. Finally, $\mathcal{S}$ sends the shares of $[r_{\ell,i}]_t$ of corrupted parties to $\mathcal{F}_{\mathsf{randShare}}$ and outputs what $\mathcal{A}$ outputs. Whenever an honest party P_i should receive his shares of $\{[r_{\ell,i}]_t\}_{\ell \in \{1,\ldots,N'\}, i \in \{1,\ldots,T\}}$, $\mathcal{S}$ delivers the output from $\mathcal{F}_{\mathsf{randShare}}$ to P_i.

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 $\ell \in \{1, \ldots, N'\}, i \in \{1, \ldots, T\}$. $\mathbf{Hyb}_1$ 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. After randomly sampling the shares of corrupted parties, we delay the generation of the whole sharings until the set $\mathcal{D}$ is determined. Note that in $\mathbf{Hyb}_1$, the shares of honest parties are never sent in the first two steps. Let $\mathcal{C}orr_{\mathsf{succ}}$ denote the set of corrupted parties in $\mathcal{D}$. Let $\mathcal{H}_{\mathsf{succ}}$ denote the set of the first $t+1$ honest parties in $\mathcal{D}$. Then, we generate the whole sharings as $\mathbf{Hyb}_1$ for honest parties not 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. Therefore, for all $\ell \in \{1, \ldots, N'\}$, 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 samples $\{[r_{\ell,i}]_t\}_{i=1}^{t+1}$ 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 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 the last hybrid, we ask $\mathcal{F}_{\mathsf{randShare}}$ to generate $\{[r_{\ell,i}]_t\}_{\ell \in \{1,\ldots,N'\}, i \in \{1,\ldots,t+1\}}$ based on the shares of corrupted parties. Note that the way of generating $\{[r_{\ell,i}]_t\}_{\ell \in \{1,\ldots,N'\}, i \in \{1,\ldots,t+1\}}$ remains unchanged. $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ have the same distribution. Note that $\mathbf{Hyb}_4$ corresponds to the ideal world.

Public Reconstruction. The description of $\mathcal{F}_{\mathsf{pubRec}}$ appears below. Following [CP17], $\mathcal{F}_{\mathsf{pubRec}}$ can be realized with communication complexity $\mathcal{O}(N \cdot n + n^2)$ to reconstruct $\mathcal{O}(N)$ degree-t Shamir sharings.

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

The functionality runs with parties $P_1, \ldots, P_n$ and the adversary $\mathcal{S}$. The functionality is parameterized by the number N of degree-t Shamir sharings to be reconstructed from all parties.

1: The functionality waits for $n-t$ parties that provide shares such that for all $i \in \{1, \ldots, N\}$, the shares lie on a degree-t polynomial sharing $[s_i]_t$. The functionality then sends the whole sharing $[s_i]_t$ to the ideal adversary $\mathcal{S}$. The functionality also computes the secret s_i by using the received shares and sends it to all parties as request-based delayed outputs.

Generating Random Coins. The description of $\mathcal{F}_{\mathsf{coin}}$ appears below. Such a functionality can be realized by first preparing random degree-t Shamir sharings by $\mathcal{F}_{\mathsf{randShare}}$ and then using $\mathcal{F}_{\mathsf{pubRec}}$ to reconstruct the secrets to all parties when needed. When $\mathcal{F}_{\mathsf{ACSS}}$ (which is used in $\mathcal{F}_{\mathsf{randShare}}$) is instantiated by [CP23], $\mathcal{F}_{\mathsf{coin}}$ can be realized with amortized communication complexity $\mathcal{O}(n^3)$ elements per random value.

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

The functionality runs with parties $P_1, \ldots, P_n$ and the adversary $\mathcal{S}$.

1: Upon receiving $2t+1$ parties' requests, the functionality samples a random value r.
2: The functionality sends r to all parties as request-based delayed outputs.

C Main Protocol Blueprint

We now show a standard blueprint of a linear-communication protocol in the $\{\mathcal{F}_{\mathsf{triple}}, \mathcal{F}_{\mathsf{randShare}}, \mathcal{F}_{\mathsf{pubRec}}\}$-hybrid model.

Protocol $\Pi_{\mathtt{main}}$

Offline Phase

1: Let C denote the circuit to be computed. All parties invoke $\mathcal{F}_{\mathsf{triple}}$ to prepare $|C|$ random Beaver triples and assign one random triple to each multiplication gate in the circuit. All parties also invoke $\mathcal{F}_{\mathsf{randShare}}$ to prepare n random degree-t Shamir sharings and assign one random sharing to each party.

Input Phase

1: For every party P_i with input x_i, let $[r_i]_t$ denote the random degree-t Shamir sharing prepared in the offline phase. All parties send their shares of $[r_i]_t$ to P_i.
2: P_i runs the online error correction algorithm to reconstruct $[r_i]_t$ and the secret r_i. Then P_i reliably broadcasts $x_i + r_i$ to all parties.
3: After receiving $x_i + r_i$ from P_i, all parties locally compute $[x_i]_t = (x_i + r_i) - [r_i]_t$.
4: Each party P_i sets the property Q as P_i finishes the broadcast protocol led by P_j. Then all parties run an 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.

Computation Phase

1: For every addition gate with input sharings $[x]_t, [y]_t$, all parties locally compute $[z]_t = [x]_t + [y]_t$.
2: For every multiplication gate, suppose the input degree-t Shamir sharings are denoted by $[x]_t, [y]_t$. Let $([a]_t, [b]_t, [c]_t)$ denote the random Beaver triple assigned to this multiplication gate.
   1. All parties locally compute $[x+a]_t = [x]_t + [a]_t$ and $[y+b]_t = [y]_t + [b]_t$.
   2. All parties invoke $\mathcal{F}_{\mathsf{pubRec}}$ to reconstruct $x+a, y+b$ for all multiplication gates in the current layer.
   3. All parties locally compute

$$[z]_t = (x+a)(y+b) - (x+a)[b]_t - (y+b)[a]_t + [c]_t.$$

Output Phase

1: For every output $[y_i]_t$ that should be reconstructed to P_i. All parties send their shares of $[y_i]_t$ to P_i. Then P_i runs the online error correction algorithm to reconstruct $[y_i]_t$ and the secret y_i. P_i takes y_i as output.

Lemma 7. *Protocol $\Pi_{\mathtt{main}}$ securely computes $\mathcal{F}_{\mathsf{fs}}$ in the $\{\mathcal{F}_{\mathsf{triple}}, \mathcal{F}_{\mathsf{randShare}}, \mathcal{F}_{\mathsf{pubRec}}\}$-hybrid against a fully malicious adversary $\mathcal{A}$ corrupting up to $t < n/3$ parties.*

We refer readers to Section D.6 for the security proof of Lemma 7.

D Security Proofs

D.1 Proof of Lemma 1

Proof. When the dealer D is honest, all honest parties will eventually receive the correct shares in Step 1 from D. Therefore, there will be at least $2t+1$ parties that support D, which includes at least $(2-\epsilon)t+1$ honest parties. For an honest party P_j that reconstructs $f_{\ell,j}(x), g_{\ell,j}(y)$ by using shares from other parties, he will receive $(2-\epsilon)t+1$ shares which includes at least $(2-2\epsilon)t+1$ correct shares from honest parties. By the error-correction of the Reed-Solomon Code, P_j can always reconstruct the correct shares. Then in the verification step, since $\mathcal{B}$ contains at least $(2-\epsilon)t+1$ honest parties and their polynomials lie on a valid degree-d bivariate polynomial, all honest parties will eventually accept the check and terminate with correct shares.

When the dealer D is corrupted, if an honest party terminates, then all honest parties will eventually terminate. This is because an honest party terminates only if he received from $2t+1$ parties supporting D, which includes at least $(2-\epsilon)t+1$ honest parties. The rest of honest parties will eventually receive at least $(2-\epsilon)t+1$ shares and terminate.

Now we argue that if the check fails for an honest party, then it fails for all honest parties. This is because all honest parties check the polynomials broadcast by the same set of parties and the checking process is deterministic. Therefore, if an honest party rejects the check, then all honest parties would eventually reject the check.

Finally we argue that if the check passes, then there exists a set of $(2-3\epsilon)t+1$ honest parties whose shares lie on valid degree-d bivariate polynomials with probability $1 - N \cdot \binom{(2+\epsilon)t+1}{4\epsilon t}/|\mathbb{F}|$.

We first show that if the check passes, then at least $(2-3\epsilon)t+1$ honest parties satisfy that

- they receive their shares before r is sampled by $\mathcal{F}_{\mathsf{coin}}$,
- the polynomials broadcast by these honest parties lie on a valid degree-d bivariate polynomial.

When the first honest party P_i sends an request to $\mathcal{F}_{\mathsf{coin}}$, P_i has received $2t+1$ parties who supports D. Then there are at least $(2-\epsilon)t+1$ honest parties who have received their shares before P_i sends an request to $\mathcal{F}_{\mathsf{coin}}$. On the other hand, passing the check means that there are at least $(2-\epsilon)t+1$ parties who broadcast their polynomials and their polynomials lie on a valid degree-d bivariate polynomial. Since by assumption, there are at most $(2+\epsilon)t+1$ parties, by the inclusion-and-exclusion principle, at least $(2-3\epsilon)t+1$ honest parties receive their shares before r is sampled and the polynomials broadcast by these honest parties lie on a valid degree-d bivariate polynomial.

By the Schwartz-Zippel lemma, if the shares of these honest parties do not lie on valid degree-d bivariate polynomials, then the polynomials broadcast by these honest parties do not lie on a valid degree-d bivariate polynomial with probability at least $N/|\mathbb{F}|$. Now assume that for any set of $(2-3\epsilon)t+1$ honest parties, there exists ℓ such that their shares of $F_\ell(x,y)$ do not lie on a valid degree-d bivariate polynomial, by the union bound, the probability that the check passes is bounded by $N \cdot \binom{(2+\epsilon)t+1}{(2-3\epsilon)t+1}/|\mathbb{F}| = N \cdot \binom{(2+\epsilon)t+1}{4\epsilon t}/|\mathbb{F}|$.

D.2 Proof of Lemma 2

Proof. When the dealer D is honest, $\Pi_{\mathtt{ShBi}}$ ensures that all honest parties will receive correct shares of $\{F_i^{(\ell)}(x,y)\}_{i\in\{1,\ldots,n\},\ell\in\{0,\ldots,N'\}}$. Then in Step 6, each P_i receives at least $(2-\epsilon)t+1$ shares from honest parties, which lie on a degree-d bivariate polynomial. Thus P_i can reconstruct the correct polynomials

$\{F_{j_1}(x,y)\}_{j_1=1}^n$. Then the check in Step 7 also passes since D is honest. And finally, all honest parties will reconstruct the correct shares in Step 9 due to the same reason for Step 6.

When the dealer D is corrupted, if an honest party terminates, all honest parties will eventually terminate $\Pi_{\texttt{ShBi}}$. For the rest of steps, corrupted parties cannot prevent honest parties from termination. Now we argue that either all honest parties take `fail` as output or the shares of all honest parties lie on valid degree-d packed Shamir sharings with overwhelming probability.

In $\Pi_{\texttt{ShBi}}$, either all honest parties take `fail` as output or there exists a set $\mathcal{H}_{\text{valid}}$ of $(2-3\epsilon)t+1$ honest parties whose shares lie on valid degree-d bivariate polynomials with overwhelming probability. In the former case, all honest parties would take `fail` as output.

In the latter case, in Step 6, since there are at most $(2+\epsilon)t+1$ parties, by the inclusion-and-exclusion principle, each honest party P_i receives shares from at least $(2-4\epsilon)t+1$ parties in $\mathcal{H}_{\text{valid}}$. Since $d \leq (2-8\epsilon)t$, for all $j_1 \in \{1,\dots,n\}$, there exists only one degree-d bivariate polynomial such that the shares of $(2-4\epsilon)t+1$ parties that P_i received lie on this degree-d bivariate polynomial. Thus, all honest parties would reconstruct the same bivariate polynomials which are decided by the shares of $\mathcal{H}_{\text{valid}}$. Therefore, the check in Step 6 passes for all honest parties. In Step 7, since all honest parties reconstruct the same bivariate polynomials, they will reach an agreement on whether the check in Step 7 passes or not. If the check fails, then all honest parties would output `fail`. Otherwise, by the Schwartz-Zippel lemma, with overwhelming probability, the secret values decided by the shares of parties in $\mathcal{H}_{\text{valid}}$ form valid degree-d packed Shamir sharings. In the latter case, all honest parties in Step 9 will reconstruct the secrets decided by the shares of parties in $\mathcal{H}_{\text{valid}}$. Therefore, the shares of all honest parties lie on valid degree-d bivariate polynomials.

D.3 Proof of Lemma 3

Proof. When the dealer D is honest, $\Pi_{\texttt{ShPack}}$ ensures that all honest parties will receive correct shares of $\{[\boldsymbol{f}(\alpha_\ell)]_d, [\boldsymbol{g}(\alpha_\ell)]_d\}_{\ell=0}^N$ and $\{[\boldsymbol{h}(\alpha_\ell)]_d\}_{\ell=0}^{2N}$. In Step 3, the probability that $r \in \{1,\dots,N\}$ is negligible. Then in Step 6, each P_i receives at least $(2-\epsilon)t+1$ shares from honest parties, which form a valid degree-d packed Beaver triple. Thus P_i can reconstruct the correct sharings $([\boldsymbol{x}]_d, [\boldsymbol{y}]_d, [\boldsymbol{z}]_d)$ and $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$. Then the checks in Step 6 and Step 7 pass. And finally, all honest parties will take the correct shares as output.

When the dealer D is corrupted, if an honest party terminates, all honest parties will eventually terminate $\Pi_{\texttt{ShPack}}$. For the rest of steps, corrupted parties cannot prevent honest parties from termination. Now we argue that either all honest parties take `fail` as output or all honest parties receive valid degree-d packed Beaver triples with overwhelming probability.

In $\Pi_{\texttt{ShPack}}$, with overwhelming probability, either all honest parties take `fail` as output or all honest parties hold valid degree-d packed Shamir sharings. In the former case, all honest parties would take `fail` as output.

In the latter case, if $r \in \{1,\dots,N\}$ in Step 3, which happens with negligible probability, all honest parties would output `fail`. Otherwise, in Step 6, each honest party P_i receives shares from at least $(2-\epsilon)t+1$ honest parties. Then all honest parties can reconstruct the secrets determined by the shares of honest parties. Therefore, the check in Step 6 passes for all honest parties. In Step 7, since all honest parties reconstruct the same packed Shamir sharings, they will reach an agreement on whether the check in Step 7 passes or not. If the check fails, then all honest parties would output `fail`. Otherwise, by the Schwartz-Zippel lemma, with overwhelming probability, the packed Shamir sharings decided by the shares of honest parties form valid packed Beaver triples. In the latter case, all honest parties in Step 8 will output their shares. Therefore, all honest parties receive valid degree-d packed Beaver triples.

D.4 Proof of Lemma 4

Proof. In the first step, the protocol $\Pi_{\texttt{ShTriple}}$ guarantees that all honest parties will eventually terminate and receive correct packed Beaver triples when the dealer is honest. Furthermore, if an honest party terminates and his output is not `fail`, then all honest parties will eventually terminate and receive correct packed Beaver triples even if the dealer is corrupted. Thus, at least $2t+1$ dealers will successfully distribute random packed Beaver triples in the first step.

When P_{king} is honest, P_{king} will eventually broadcast the set $\mathcal{D}$. Then in Step 3, since for each $P_i \in \mathcal{D}$, at least one honest party terminates $\Pi_{\texttt{ShTriple}}$ led by P_i and his output is not `fail`, all honest parties will eventually receive correct packed Beaver triples distributed by P_i. Thus all honest parties will eventually proceed to Step 3.1.

In Step 3.3.(b), P_{king} will receive $2t+1$ shares from all parties. Since by assumption there are at most ϵt corrupted parties, P_{king} will receive at least $(2-\epsilon)t+1$ shares from honest parties which lie on valid degree-d polynomials. Thus, an honest P_{king} can reconstruct degree-d packed Shamir sharings determined by shares of honest parties. At the end of Step 3.3.(b), all honest parties will eventually receive degree-$(d-t)$ packed Shamir sharings $\{[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_{d-t}, [\boldsymbol{g}(\alpha_i)+\boldsymbol{b}_i]_{d-t}\}_{i=1}^{2t+1}$. Then all honest parties can compute their shares of $([\boldsymbol{f}(\alpha_0)]_d, [\boldsymbol{g}(\alpha_0)]_d, [\boldsymbol{h}(\alpha_0)]_{2d-t})$.

In Step 4, when P_{king} is honest, all honest parties hold valid degree-$(d-t)$ packed Shamir sharing. Since $d-t=\epsilon t-1<t$, the online error correction algorithm ensures that all honest parties can reconstruct $[\boldsymbol{z}]_{d-t}$ determined by the shares of honest parties. Thus, all honest parties will accept the check in Step 4.

D.5 Proof of Lemma 5

Proof. We first show that all honest parties will eventually terminate the protocol $\Pi_{\texttt{triple-GOD}}$. It is sufficient to show that

1. In $\Pi_{\texttt{triple-GOD}}$, the property Q is an ACS property. This is equivalent to show that when P_{king} is honest, then an honest party P_i will eventually terminate $\Pi_{\texttt{tripleKing-GOD}}$.
2. For each party $P_i \in \mathcal{K}$, all honest parties will eventually receive their shares in $\Pi_{\texttt{tripleKing}}$ led by P_i.

For the first point, we argue that when P_{king} is honest, then an honest party will terminate at least one of $\Pi_{\texttt{tripleExt-GOD}}$ and $\Pi_{\texttt{tripleGen-GOD}}$. This guarantees that all honest parties will eventually participate in the BA protocol and terminate. For the sake of contradiction, suppose that for an honest party P_j, neither of $\Pi_{\texttt{tripleExt-GOD}}$ or $\Pi_{\texttt{tripleGen-GOD}}$ terminates.

We first argue that for $\Pi_{\texttt{tripleExt-GOD}}$, if P_{king} broadcasts the set $\mathcal{D}$, then P_j must terminate. This is because for an honest P_{king}, (1) every honest party will eventually receive the set $\mathcal{D}$, and (2) for each party $P_i \in D$, at least one honest party supports P_i. Recall that for $\mathcal{F}_{\mathsf{ACSS}}$, if an honest party terminates, then all honest parties will eventually terminate. Therefore, all honest parties will terminate $\mathcal{F}_{\mathsf{ACSS}}$ when P_i acts as the dealer for all $P_i \in \mathcal{D}$, and therefore terminate $\Pi_{\texttt{tripleExt-GOD}}$. Thus, if P_j does not terminate in $\Pi_{\texttt{tripleExt-GOD}}$, it implies that P_{king} never broadcasts the set $\mathcal{D}$. Again, since P_{king} is honest, this means that at most $L-1 \le (2+\epsilon)t+1$ parties including at most $L-(2t+1)-1 \le \epsilon t$ corrupted parties are supported by at least $t+1$ parties. It means that for each P_i of the rest of parties, no honest party terminates $\mathcal{F}_{\mathsf{ACSS}}$ when P_i is the dealer (which also implies that P_i is a corrupted party).

We then argue that for $\Pi_{\texttt{tripleGen-GOD}}$, if P_{king} is honest and there are at most $(2+\epsilon t)+1$ parties that participate in $\Pi_{\texttt{tripleGen-GOD}}$ including at most ϵt corrupted parties, all honest parties terminate will eventually terminate $\Pi_{\texttt{tripleGen-GOD}}$ and their outputs are not `fail`. By Lemma 4, all honest parties will receive correct packed Beaver triples. Then in Π_{depack}, $\mathcal{F}_{\mathsf{randDepack}}$ is guaranteed to terminate. In Step 3 of Π_{depack}, recall that $2d-t=(1+2\epsilon)t-2$. Since all honest parties hold valid degree-$(2d-t)$ packed Shamir sharings (with overwhelming probability) and an honest P_{king} will receive at least $(2-\epsilon)t+1$ shares from honest parties, by the fact that $2d-t<(2-2\epsilon)t+1$, P_{king} can always reconstruct the secrets determined by the shares of honest parties by the property of the Reed-Solomon Code. Thus, all honest parties can obtain correct degree-t Shamir sharings in Π_{depack}. This implies that all honest parties hold valid Beaver triples in $\Pi_{\texttt{tripleGen}}$. Then the verification in $\Pi_{\texttt{tripleGen-GOD}}$ will succeed with overwhelming probability, indicating that all honest parties will output valid Beaver triples.

In summary, all honest parties will eventually terminate $\Pi_{\texttt{tripleGen-GOD}}$, which contradicts with the assumption that there is an honest party P_j who does not terminate either of $\Pi_{\texttt{tripleExt-GOD}}$ or $\Pi_{\texttt{tripleGen-GOD}}$.

For the second point, for each party $P_i \in \mathcal{K}$, if an honest party P_j terminates $\Pi_{\texttt{tripleKing-GOD}}$ led by P_i, then P_j finishes the BA protocol in $\Pi_{\texttt{tripleKing-GOD}}$. This implies that the input of at least one honest party, say $P_{j'}$, to the BA protocol is equal to the output of the BA protocol b, which further implies that $P_{j'}$ terminates $\Pi_{\texttt{tripleExt-GOD}}$ if $b=0$ or $\Pi_{\texttt{tripleGen-GOD}}$ if $b=1$. We show that when P_{king} is corrupted, if an honest party P_j terminates either of $\Pi_{\texttt{tripleExt-GOD}}$ or $\Pi_{\texttt{tripleGen-GOD}}$, then all honest parties will eventually terminate the same process. Note that this implies that all honest parties will eventually terminate $\Pi_{\texttt{tripleKing-GOD}}$ led by P_i and receive their shares.

Consider the following two cases. In the first case, suppose $P_{j'}$ terminates $\Pi_{\texttt{tripleExt-GOD}}$. Then $P_{j'}$ has received $\mathcal{D}$ from P_{king}. By the properties of the broadcast protocol, all honest parties will eventually receive $\mathcal{D}$. Since $P_{j'}$ terminates $\mathcal{F}_{\mathsf{ACSS}}$ for dealers in $\mathcal{D}$, all honest parties will eventually terminate $\mathcal{F}_{\mathsf{ACSS}}$ for dealers in $\mathcal{D}$. Therefore, all honest parties will eventually receive the shares distributed by dealers in $\mathcal{D}$. Then all honest parties will eventually terminate $\Pi_{\texttt{tripleExt-GOD}}$ and receive their shares.

In the second case, suppose $P_{j'}$ terminates $\Pi_{\texttt{tripleGen-GOD}}$. Note that in $\Pi_{\texttt{ShBi}}, \Pi_{\texttt{ShPack}}, \Pi_{\texttt{ShTriple}}$, all parties check the same set of broadcast values. Thus, for invocations of $\Pi_{\texttt{ShBi}}, \Pi_{\texttt{ShPack}}, \Pi_{\texttt{ShTriple}}$ that $P_{j'}$ terminate, all honest parties will eventually accept the checks and receive their shares in these invocations. Now consider $\Pi_{\texttt{tripleExtPack}}$. Since $P_{j'}$ has received $\mathcal{D}$, by the properties of the broadcast protocol, all honest parties will eventually receive $\mathcal{D}$. Since for every $P_i \in \mathcal{D}$, $P_{j'}$ terminates $\Pi_{\texttt{ShTriple}}$ led by P_i, all honest parties eventually terminate $\Pi_{\texttt{ShTriple}}$ and receive their shares. In Step 3.3, since $P_{j'}$ terminates $\mathcal{F}_{\mathsf{ACSS}}$ led by P_{king}, all honest parties eventually terminate $\mathcal{F}_{\mathsf{ACSS}}$ and receive their shares. In Step 4, by the property of the online error correction algorithm, all honest parties will reconstruct the same degree-t Shamir sharings. Since $P_{j'}$ accepts the check, all honest parties will eventually accept the check.

Similarly, in $\Pi_{\texttt{depack}}$, since $P_{j'}$ terminates $\mathcal{F}_{\mathsf{ACSS}}$ led by P_{king}, all honest parties will eventually terminate $\mathcal{F}_{\mathsf{ACSS}}$ and receive their shares. Finally in $\Pi_{\texttt{tripleGen-GOD}}$, since $P_{j'}$ accepts the check in Step 3, all honest parties will eventually accept the check. Thus, all honest parties will eventually terminate $\Pi_{\texttt{tripleGen-GOD}}$ and receive their shares.

Now we show that $\Pi_{\texttt{triple-GOD}}$ securely computes $\mathcal{F}_{\mathsf{triple}}$. Let $\mathcal{A}$ be a static malicious adversary which controls a set $\mathcal{C}orr$ of $t' \leq t$ corrupted parties. Let $\mathcal{Z}$ be an environment. We construct an ideal adversary $\mathcal{S}$ interacting with the environment $\mathcal{Z}$ and the ideal functionality $\mathcal{F}_{\mathsf{fs}}$. $\mathcal{S}$ starts with running $\mathcal{A}$ and passes messages between $\mathcal{Z}$ and $\mathcal{A}$. For corrupted parties, $\mathcal{S}$ faithfully follows the instructions of $\mathcal{A}$. Then $\mathcal{S}$ simulates the behaviors of honest parties as follows. Let $\mathcal{C}orr'$ be the set of all corrupted parties together with the first $t - t'$ honest parties. Then $|\mathcal{C}orr'| = t$. In the following, we will explicitly generate the shares of all parties in $\mathcal{C}orr'$. In this way, given the shares of parties in $\mathcal{C}orr'$ and the secret, a degree-t Shamir secret sharing is fully determined. In Step 1, for all $i \in \{1, \ldots, n\}$, all parties set P_i as P_{king} and invoke $\Pi_{\texttt{tripleKing-GOD}}$. In $\Pi_{\texttt{tripleKing-GOD}}$, the two processes $\Pi_{\texttt{tripleExt-GOD}}$ and $\Pi_{\texttt{tripleGen-GOD}}$ are invoked in parallel.

Simulation of $\Pi_{\texttt{tripleExt-GOD}}$. In Step 1, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS}}$ as follows: If the dealer is honest, $\mathcal{S}$ samples random values as the shares of parties in $\mathcal{C}orr'$ and then sends those values to parties in $\mathcal{C}orr'$. If the dealer is corrupted, $\mathcal{S}$ waits to receive the whole sharings distributed by the dealer. If received, $\mathcal{S}$ distributes the shares to parties in $\mathcal{C}orr'$.

In Step 2, if P_{king} is an honest party, $\mathcal{S}$ honestly follows the protocol.

In Step 3, after receiving $\mathcal{D}$, $\mathcal{S}$ simulates each honest party to wait for the termination of the executions of $\mathcal{F}_{\mathsf{ACSS}}$ where $P_i \in \mathcal{D}$. Note that for each $P_i \in \mathcal{D}$, if P_i is an honest party, then $\mathcal{S}$ has generated the shares of corrupted parties distributed by P_i. If P_i is a corrupted party, then $\mathcal{S}$ has learnt the whole sharings distributed by P_i. In Step 3.2, $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{coin}}$. If $r \in \{1, \ldots, N'\}$, $\mathcal{S}$ outputs $\perp$ and halts. Otherwise, $\mathcal{S}$ computes the shares of $([f_i(r)]_t, [g_i(r)]_t, [h_i(r)]_t)$ of parties in $\mathcal{C}orr'$. If P_i is a corrupted party, $\mathcal{S}$ also computes the whole sharings $([f_i(r)]_t, [g_i(r)]_t, [h_i(r)]_t)$. Otherwise, $\mathcal{S}$ samples two random values as $f_i(r), g_i(r)$ and sets $h_i(r) = f_i(r) \cdot g_i(r)$. Then $\mathcal{S}$ computes the whole sharings $([f_i(r)]_t, [g_i(r)]_t, [h_i(r)]_t)$ based on the secret values $f_i(r), g_i(r), h_i(r)$ and the shares of corrupted parties. $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{pubRec}}$ and follows Step 3.3. For each corrupted party $P_i \in \mathcal{D}$, if there exists $\ell \in \{1, \ldots, N'\}$ such that $f_i(\ell) \cdot g_i(\ell) \neq h_i(\ell)$ but the check for P_i passes, $\mathcal{S}$ outputs $\perp$ and terminates.

In Step 4, $\mathcal{S}$ follows the protocol and computes the shares of $([f(\alpha_i)]_t, [g(\alpha_i)]_t)$ of parties in $\mathcal{C}orr'$ for all $i \in \{1, \ldots, L\}$. In Step 3.3, if $([a_i]_t, [b_i]_t, [c_i]_t)$ is distributed by a corrupted party, $\mathcal{S}$ samples random values as $f(\alpha_i), g(\alpha_i)$ and computes $f(\alpha_i) + a_i, g(\alpha_i) + b_i$. Otherwise, $\mathcal{S}$ samples random values as $f(\alpha_i) + a_i, g(\alpha_i) + b_i$. Then, $\mathcal{S}$ computes the whole sharings $[f(\alpha_i) + a_i]_t, [g(\alpha_i) + b_i]_t$ using the secrets $f(\alpha_i) + a_i, g(\alpha_i) + b_i$ and the shares of parties in $\mathcal{C}orr'$. After that, $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{pubRec}}$. Finally, $\mathcal{S}$ follows the protocol and computes the shares of $([f(\beta_i)]_t, [g(\beta_i)]_t, [h(\beta_i)]_t)$ of corrupted parties for all $i \in \{1, 2, ..., (L+1)/2 - t\}$.

Simulation of $\Pi_{\texttt{tripleGen-GOD}}$. We first show the simulation of $\Pi_{\texttt{ShBi}}$. We will show that what the adversary can do is to add an arbitrary additive error to each value held by honest parties. Without loss of generality, we assume that corrupted dealers should always distribute all-0 sharings/polynomials. We can assume this because we may think the values that honest parties actually received are the correct values (i.e., 0s) under additive attacks since the adversary knows what honest parties received when the dealer is corrupted. On the other hand, corrupted parties may always change their local values to any values they want. In the following, we show that $\mathcal{S}$ can learn the additive errors chosen by the adversary during the simulation.

When D is corrupted, $\mathcal{S}$ honestly follows the protocol. If the verification fails, $\mathcal{S}$ sets honest parties' output to be `fail`. Otherwise, for each honest party P_i that terminates $\Pi_{\texttt{ShBi}}$, $\mathcal{S}$ sets $\{f_{\ell,i}(x), g_{\ell,i}(y)\}_{\ell=1}^N$ to be all-0 polynomials, and sets $\{\Delta f_{\ell,i}(x), \Delta g_{\ell,i}(y)\}_{\ell=1}^N$ to be the actual outputs of P_i (which are interpreted as the additive errors for $\{f_{\ell,i}(x), g_{\ell,i}(y)\}_{\ell=1}^N$). For each corrupted party P_i, $\mathcal{S}$ sets $\{f_{\ell,i}(x), g_{\ell,i}(y)\}_{\ell=1}^N$ to be all-0 polynomials.

When D is honest, we assume that $\mathcal{S}$ learns $\{f_{\ell,i}(x), g_{\ell,i}(y)\}_{\ell=1}^N$ for all $P_i \in \mathcal{C}orr'$. This will be satisfied when $\mathcal{S}$ simulates $\Pi_{\texttt{ShBi}}$ in $\Pi_{\texttt{tripleGen-GOD}}$. $\mathcal{S}$ samples random degree-d polynomials $\{f_{0,i}(x), g_{0,i}(y)\}$ for all $P_i \in \mathcal{C}orr'$ such that $f_{0,i}(\alpha_j) = g_{0,j}(\alpha_i)$ for all $P_i, P_j \in \mathcal{C}orr'$. Then $\mathcal{S}$ distributes $\{f_{\ell,i}(x), g_{\ell,i}(y)\}_{\ell=0}^N$ to parties in $\mathcal{C}orr'$.

For each honest party P_i, $\mathcal{S}$ simulates P_i and waits to receive shares either from D or other parties. If P_i receives shares from D, $\mathcal{S}$ follows the protocol and broadcasts $(\texttt{support}, P_i, D)$ on behalf of P_i. If P_i receives $(2-\epsilon)t+1$ shares from other parties, for each degree-d polynomial $h(x) \in \{f_{\ell,i}(x), g_{\ell,i}(y)\}_{\ell=0}^N$, $\mathcal{S}$ sets $\Delta\tilde{h}(x)$ as follows:

- If $h(\alpha_j)$ is from an honest party, $\mathcal{S}$ sets $\Delta\tilde{h}(\alpha_j) = 0$.
- If $h(\alpha_j)$ is from a corrupted party, $\mathcal{S}$ sets $\Delta\tilde{h}(\alpha_j)$ to be the difference of the actual value and the value $h(\alpha_j)$ D sends to P_j. Note that $\mathcal{S}$ learns all values that are sent from D to P_j.

$\mathcal{S}$ applies the error-correction algorithm to $\Delta\tilde{h}(x)$. If there exists a degree-d polynomial $\Delta h(x)$ such that $(2-2\epsilon)t+1$ points of $\Delta\tilde{h}(x)$ (that have been assigned above) lie on $\Delta h(x)$, then interpret $\Delta h(x)$ as the additive errors added to the correct polynomial $h(x)$. Otherwise, interpolate $\Delta h(x)$ by using the first $d+1$ points of $\Delta\tilde{h}(x)$ (that have been assigned above) and interpret $\Delta h(x)$ as the additive errors added to the correct polynomial $h(x)$.

In Step 3, $\mathcal{S}$ follows the protocol and emulates $\mathcal{F}_{\mathsf{coin}}$ honestly. Then $\mathcal{S}$ computes $\{f_i(x), g_i(y)\}$ for each party $P_i \in \mathcal{C}orr'$. $\mathcal{S}$ samples a random degree-d bivariate polynomial $F(x,y)$ such that $F(x,i) = f_i(x)$ and $F(i,y) = g_i(y)$ for all $P_i \in \mathcal{C}orr'$. Next, for each honest party P_i, $\mathcal{S}$ computes $\{\Delta f_i(x), \Delta g_i(y)\}$. After that, for each honest party P_i, $\mathcal{S}$ broadcasts $f_i(x) + \Delta f_i(x)$ and $g_i(y) + \Delta g_i(y)$ on behalf of P_i. $\mathcal{S}$ follows the rest of steps in the verification. Finally, in Step 4, if the check passes, $\mathcal{S}$ records $\{f_{\ell,i}(x), g_{\ell,i}(y)\}_{\ell=1}^N$ for each party $P_i \in \mathcal{C}orr'$, and $\mathcal{S}$ records $\{\Delta f_{\ell,i}(x), \Delta g_{\ell,i}(y)\}_{\ell=1}^N$ for each honest party P_i. Otherwise, $\mathcal{S}$ sets the output of honest parties to be `fail`.

Then we show the simulation of $\Pi_{\texttt{ShPack}}$. When D is corrupted, $\mathcal{S}$ honestly follows the protocol. If the checks in Step 6 and Step 7 fail or all honest parties receive `fail` as output in $\Pi_{\texttt{ShBi}}$, $\mathcal{S}$ sets the output of honest parties to be `fail`. Otherwise, $\mathcal{S}$ sets each $[\boldsymbol{s}_\ell]_d$ to be all-0 sharing. $\mathcal{S}$ sets $\Delta[\boldsymbol{s}_\ell]_d$ as follows:

- For each corrupted party P_i, $\mathcal{S}$ sets the i-th entry of $\Delta[\boldsymbol{s}_\ell]_d$ to be 0.
- For each honest party P_i that terminates $\Pi_{\texttt{ShPack}}$, $\mathcal{S}$ sets the i-th entry of $\Delta[\boldsymbol{s}_\ell]_d$ to be the actual share of P_i.

When D is honest, we assume that $\mathcal{S}$ learns the shares of $\{[\boldsymbol{s}_\ell]\}_{\ell=1}^N$ of parties in $\mathcal{C}orr'$. This will be satisfied when $\mathcal{S}$ simulates $\Pi_{\texttt{ShPack}}$ in $\Pi_{\texttt{tripleGen-GOD}}$. For all $F_i^{(\ell)}(x,y)$, $\mathcal{S}$ samples random degree-d polynomials $\{f_{i,j}^{(\ell)}(x), g_{i,j}^{(\ell)}(y)\}$ for all $P_j \in \mathcal{C}orr'$ such that $f_{i,j_1}^{(\ell)}(\alpha_{j_2}) = g_{i,j_2}^{(\ell)}(\alpha_{j_1})$ for all $P_{j_1}, P_{j_2} \in \mathcal{C}orr'$. Then $\mathcal{S}$ simulates $\Pi_{\texttt{ShBi}}$ as described above.

For each honest party P_i, $\mathcal{S}$ simulates P_i and waits to receive the output of $\Pi_{\texttt{ShBi}}$. If the output is `fail`, $\mathcal{S}$ sets the output of P_i to be `fail`. Otherwise, $\mathcal{S}$ learns $\{\Delta F_j^{(\ell)}(x,\alpha_i), \Delta F_j^{(\ell)}(\alpha_i,y)\}$ for all $i \in \{1,\ldots,n\}$ and $\ell \in \{0,\ldots,N'\}$. $\mathcal{S}$ follows the protocol and emulates $\mathcal{F}_{\mathsf{coin}}$ honestly. Then $\mathcal{S}$ computes $\{F_i(x,\alpha_j), F_i(\alpha_j,y)\}_{i=1}^n$ for all parties in $\mathcal{C}orr'$. Next, $\mathcal{S}$ samples B random degree-d packed Shamir sharings $[\boldsymbol{s}_1]_d, \ldots, [\boldsymbol{s}_B]_d$. $\mathcal{S}$ computes $F_1(x,y), \ldots, F_n(x,y)$ as follows:

- $\mathcal{S}$ sets $F_i(\beta_{j_1}, \beta_{j_2})$ to be the i-th share of $[\boldsymbol{s}_{(j_1-1)\epsilon t+j_2}]_d$.
- $\mathcal{S}$ computes $F_i(x,y)$ given $F_i(\beta_{j_1}, \beta_{j_2})$ for all $j_1, j_2 \in \{1,\ldots,\epsilon t\}$ and $F_i(x,\alpha_j), F_i(\alpha_j,y)$ for all $P_j \in \mathcal{C}orr'$. It is not difficult to show that these values fully determine $F_i(x,y)$.

In Step 4, $\mathcal{S}$ broadcasts $F_j(x,\alpha_i) + \Delta F_j(x,\alpha_i)$ and $F_j(\alpha_i,y) + \Delta F_j(\alpha_i,y)$ on behalf of P_i. $\mathcal{S}$ follows the rest of steps until Step 8. If the checks in Step 6 and Step 7 fails, $\mathcal{S}$ sets the outputs of honest parties to be `fail`. Otherwise, for each party $P_j \in \mathcal{C}orr'$, $\mathcal{S}$ computes each $F_j^{(\ell)}(x,y)$ in the same way above. For every honest party P_i and every corrupted party P_j, $\mathcal{S}$ sends $F_j^{(\ell)}(x,\alpha_i) + \Delta F_j^{(\ell)}(x,\alpha_i), F_j^{(\ell)}(\alpha_i,y) + \Delta F_j^{(\ell)}(\alpha_i,y)$ for all $\ell \in \{1,\ldots,N'\}$ to P_j on behalf of P_i. In Step 9, for each honest party P_i, after receiving $2t+1$ shares from party P_j,

– For every $(f_{i,j}^{(\ell)}(x), g_{i,j}^{(\ell)}(y))$ received from an honest party P_j, $\mathcal{S}$ sets $\Delta\tilde{F}_i^{(\ell)}(x,\alpha_j) = \Delta F_i^{(\ell)}(x,\alpha_j)$ and $\Delta\tilde{F}_i^{(\ell)}(\alpha_j, y) = \Delta F_i^{(\ell)}(\alpha_j, y)$.
– For every $(f_{i,j}^{(\ell)}(x), g_{i,j}^{(\ell)}(y))$ received from a corrupted party P_j, $\mathcal{S}$ sets $\Delta\tilde{F}_i^{(\ell)}(x,\alpha_j), \Delta\tilde{F}_i^{(\ell)}(\alpha_j, x)$ to be the difference between the actually received polynomials and the ones D sends to P_j. Note that $\mathcal{S}$ learns all values that are sent from D to P_j.

$\mathcal{S}$ tries to find a degree-d bivariate polynomial such that there exists a subset of $(2-4\epsilon)t+1$ parties satisfying that the assigned polynomials $(\Delta\tilde{F}_i^{(\ell)}(x,\alpha_j), \Delta\tilde{F}_i^{(\ell)}(\alpha_j, x))$ lie on this degree-d bivariate polynomial. If such a degree-d bivariate polynomial exists, § resets $\Delta F_i^{(\ell)}(x,y)$ to be this degree-d bivariate polynomial. Otherwise, § resets $\Delta F_i^{(\ell)}(x,y)$ to be the degree-d bivariate polynomial interpolated from $\Delta\tilde{F}_i^{(\ell)}(x,\alpha_j)$ for the first $d+1$ parties that P_i received shares from. Finally, for all $i \in \{1,\ldots,n\}, j_1, j_2 \in \{1,\ldots,\epsilon t\}$, $\mathcal{S}$ sets $\Delta[\boldsymbol{s}_{(\ell-1)B+(j_1-1)\epsilon t+j_2}]_d$ as follows.

– For each corrupted party P_{j_3}, $\mathcal{S}$ sets the j_3-th entry of $\Delta[\boldsymbol{s}_{(\ell-1)B+(j_1-1)\epsilon t+j_2}]_d$ to be 0.
– For each honest party P_{j_3}, $\mathcal{S}$ sets the j_3-th entry of $\Delta[\boldsymbol{s}_{(\ell-1)B+(j_1-1)\epsilon t+j_2}]_d$ to be $\Delta F_{j_3}^{(\ell)}(\beta_{j_1}, \beta_{j_2})$.

$\mathcal{S}$ records the shares of $\{[\boldsymbol{s}_\ell]_d\}_{\ell=1}^N$ of parties in $\mathcal{C}orr'$, and $\mathcal{S}$ records $\{\Delta[\boldsymbol{s}_\ell]_d\}_{\ell=1}^N$.

Next, we show the simulation of $\Pi_{\texttt{ShTriple}}$. When D is corrupted, $\mathcal{S}$ honestly follows the protocol. If all honest parties receive `fail` as output in $\Pi_{\texttt{ShPack}}$, $\mathcal{S}$ sets the output of honest parties to be `fail`. Otherwise, $\mathcal{S}$ sets $\{[\boldsymbol{a}_\ell]_d, [\boldsymbol{b}_\ell]_d, [\boldsymbol{c}_\ell]_d\}_{\ell=1}^N$ to be all-0 polynomials. For each degree-d packed Shamir sharing $[\boldsymbol{z}]_d \in \{[\boldsymbol{a}_\ell]_d, [\boldsymbol{b}_\ell]_d, [\boldsymbol{c}_\ell]_d\}_{\ell=1}^N$ $\mathcal{S}$ sets $\Delta[\boldsymbol{z}]_d$ as follows:

– For each corrupted party P_i, $\mathcal{S}$ sets the i-th entry of $\Delta[\boldsymbol{z}]_d$ to be 0.
– For each honest party P_i that terminates $\Pi_{\texttt{ShTriple}}$, $\mathcal{S}$ sets the i-th entry of $\Delta[\boldsymbol{z}]_d$ to be the actual share of P_i.

When D is honest, we assume that $\mathcal{S}$ learns the shares of $\{[\boldsymbol{a}_\ell]_d, [\boldsymbol{b}_\ell]_d, [\boldsymbol{c}_\ell]_d\}_{\ell=1}^N$ of parties in $\mathcal{C}orr'$. This will be satisfied when $\mathcal{S}$ simulates $\Pi_{\texttt{ShTriple}}$ in $\Pi_{\texttt{tripleGen-GOD}}$. For all $[\boldsymbol{z}]_d \in \{[\boldsymbol{a}_0]_d, [\boldsymbol{b}_0]_d, [\boldsymbol{c}_0]_d\} \cup \{[\boldsymbol{h}(\alpha_\ell)]_d\}_{\ell=N+1}^{2N}$, $\mathcal{S}$ samples random values as shares of parties in $\mathcal{C}orr'$. Then $\mathcal{S}$ simulates $\Pi_{\texttt{ShPack}}$ as described above.

For each honest party P_i, $\mathcal{S}$ simulates P_i and waits to receive the output of $\Pi_{\texttt{ShPack}}$. If the output is `fail`, $\mathcal{S}$ sets the output of P_i to be `fail`. Otherwise, $\mathcal{S}$ learns $\{\Delta[\boldsymbol{f}(\alpha_\ell)]_d, \Delta[\boldsymbol{g}(\alpha_\ell)]_d\}_{\ell=0}^N$ and $\{\Delta[\boldsymbol{h}(\alpha_\ell)]_d\}_{\ell=0}^{2N}$. $\mathcal{S}$ follows the protocol and emulates $\mathcal{F}_{\mathsf{coin}}$ honestly. If $r \in \{1,\ldots,N\}$, $\mathcal{S}$ outputs $\perp$ and halts. Otherwise, $\mathcal{S}$ computes the shares of $([\boldsymbol{f}(r)]_d, [\boldsymbol{g}(r)]_d, [\boldsymbol{h}(r)]_d)$ of parties in $\mathcal{C}orr'$ and $(\Delta[\boldsymbol{f}(r)]_d, \Delta[\boldsymbol{g}(r)]_d, \Delta[\boldsymbol{h}(r)]_d)$. Then $\mathcal{S}$ randomly samples $\boldsymbol{f}(r), \boldsymbol{g}(r)$ and computes $\boldsymbol{h}(r) = \boldsymbol{f}(r) * \boldsymbol{g}(r)$. Next $\mathcal{S}$ computes the whole sharings $([\boldsymbol{f}(r)]_d, [\boldsymbol{g}(r)]_d, [\boldsymbol{h}(r)]_d)$ by using the secrets and the shares of parties in $\mathcal{C}orr'$.

In Step 4, $\mathcal{S}$ broadcasts the i-th shares of $([\boldsymbol{f}(r)]_d, [\boldsymbol{g}(r)]_d, [\boldsymbol{h}(r)]_d) + (\Delta[\boldsymbol{f}(r)]_d, \Delta[\boldsymbol{g}(r)]_d, \Delta[\boldsymbol{h}(r)]_d)$ on behalf of P_i. $\mathcal{S}$ follows the rest of steps. If the checks in Step 6 and Step 7 fails, $\mathcal{S}$ sets the outputs of honest parties to be `fail`. Otherwise, $\mathcal{S}$ records the shares of $\{[\boldsymbol{a}_\ell]_d, [\boldsymbol{b}_\ell]_d, [\boldsymbol{c}_\ell]_d\}_{\ell=1}^N$ of parties in $\mathcal{C}orr'$, and $\mathcal{S}$ records $\{\Delta[\boldsymbol{a}_\ell]_d, \Delta[\boldsymbol{b}_\ell]_d, \Delta[\boldsymbol{c}_\ell]_d\}_{\ell=1}^N$.

Now, we show the simulation of $\Pi_{\texttt{tripleExtPack}}$. In Step 1, for each honest party P_i, $\mathcal{S}$ samples random values as shares of corrupted parties. Then $\mathcal{S}$ simulates $\Pi_{\texttt{ShTriple}}$ as described above. $\mathcal{S}$ follows the rest of steps in Distribution honestly.

In Step 2, if P_{king} is honest, $\mathcal{S}$ honestly follows the protocol.

In Step 3, for each honest party P_i, $\mathcal{S}$ simulates P_i and waits to receive the output of $\Pi_{\texttt{ShTriple}}$ for each $P_i \in \mathcal{D}$. If the output is `fail` for any $P_i \in \mathcal{D}$, $\mathcal{S}$ sets the output of P_i to be `fail`. Otherwise, $\mathcal{S}$ follows the protocol to extract random triples. From the simulation of $\Pi_{\texttt{ShTriple}}$, $\mathcal{S}$ learns $\{(\Delta[\boldsymbol{a}_i]_d, \Delta[\boldsymbol{b}_i]_d, \Delta[\boldsymbol{c}_i]_d)\}_{i=1}^{2t+1}$. In addition, if $([\boldsymbol{a}_i]_d, [\boldsymbol{b}_i]_d, [\boldsymbol{c}_i]_d)$ is distributed by a corrupted party, then $\mathcal{S}$ learns the whole sharings (which are just all-0 sharings). Otherwise, $\mathcal{S}$ learns the shares of parties in $\mathcal{C}orr'$. $\mathcal{S}$ follows the protocol and computes the shares of $([\boldsymbol{f}(\alpha_i)]_d, [\boldsymbol{g}(\alpha_i)]_d)$ of parties in $\mathcal{C}orr'$ and $(\Delta[\boldsymbol{f}(\alpha_i)]_d, \Delta[\boldsymbol{g}(\alpha_i)]_d)$ for all $i \in \{1,\ldots,2t+1\}$. In Step 3.3, if $([\boldsymbol{a}_i]_d, [\boldsymbol{b}_i]_d, [\boldsymbol{c}_i]_d)$ is distributed by a corrupted party, $\mathcal{S}$ samples random values as $\boldsymbol{f}(\alpha_i), \boldsymbol{g}(\alpha_i)$ and computes $\boldsymbol{f}(\alpha_i) + \boldsymbol{a}_i, \boldsymbol{g}(\alpha_i) + \boldsymbol{b}_i$. Otherwise, $\mathcal{S}$ samples random values as $\boldsymbol{f}(\alpha_i) + \boldsymbol{a}_i, \boldsymbol{g}(\alpha_i) + \boldsymbol{b}_i$. Then, $\mathcal{S}$ computes the whole sharings $[\boldsymbol{f}(\alpha_i) + \boldsymbol{a}_i]_d, [\boldsymbol{g}(\alpha_i) + \boldsymbol{b}_i]_d$ using the secrets $\boldsymbol{f}(\alpha_i) + \boldsymbol{a}_i, \boldsymbol{g}(\alpha_i) + \boldsymbol{b}_i$ and the shares of parties in $\mathcal{C}orr'$.

So far, $\mathcal{S}$ knows $[\boldsymbol{f}(\alpha_i)]_d, [\boldsymbol{g}(\alpha_i)]_d$ for all i where $([\boldsymbol{a}_i]_d, [\boldsymbol{b}_i]_d, [\boldsymbol{c}_i]_d)$ is distributed by a corrupted party in $\mathcal{D}$. If there are $t'' < t$ corrupted parties in $\mathcal{D}$, $\mathcal{S}$ randomly samples $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i$ such that $\boldsymbol{c}_i = \boldsymbol{a}_i * \boldsymbol{b}_i$ for $t-t''$ honest parties in $\mathcal{D}$ and computes $[\boldsymbol{a}_i]_d, [\boldsymbol{b}_i]_d, [\boldsymbol{c}_i]_d$ using the secrets and the shares of parties in $\mathcal{C}orr'$. Then if $i \le t$, $\mathcal{S}$ obtains $[\boldsymbol{f}(\alpha_i)]_d, [\boldsymbol{g}(\alpha_i)]_d$ directly. Otherwise, $\mathcal{S}$ computes $[\boldsymbol{f}(\alpha_i)]_d = [\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_d - [\boldsymbol{a}_i]_d$ and $[\boldsymbol{g}(\alpha_i)]_d$ similarly. In this way, $\mathcal{S}$ knows $[\boldsymbol{f}(\alpha_i)]_d, [\boldsymbol{g}(\alpha_i)]_d$ for t evaluation points. Let $\mathcal{E}$ denote the set of these t evaluation points. Then for all $\alpha_j \notin \mathcal{E}$, $[\boldsymbol{f}(\alpha_j)]_d$ is a linear combination of $[\boldsymbol{f}(\alpha_0)]_d$ and $\{[\boldsymbol{f}(\alpha_i)]_d\}_{\alpha_i \in \mathcal{E}}$ and the same holds for $[\boldsymbol{g}]_d$. In addition, for $\alpha_j \notin \mathcal{E}$ and $j > t$, $[\boldsymbol{a}_i]_d$ is a linear combination of $[\boldsymbol{f}(\alpha_0)]_d$, $\{[\boldsymbol{f}(\alpha_i)]_d\}_{\alpha_i \in \mathcal{E}}$, and $[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_d$. Note that $\mathcal{S}$ knows all these degree-d packed Shamir sharings except $[\boldsymbol{f}(\alpha_0)]_d$. The same holds for $[\boldsymbol{g}]_d$.

After that, for each honest party P_i, $\mathcal{S}$ sends the i-th shares of $[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_d + \Delta[\boldsymbol{f}(\alpha_i)]_d + \Delta[\boldsymbol{a}_i]_d$ and $[\boldsymbol{g}(\alpha_i)+\boldsymbol{b}_i]_d + \Delta[\boldsymbol{g}(\alpha_i)]_d + \Delta[\boldsymbol{b}_i]_d$ to P_{king} on behalf of P_i. If P_{king} is honest, $\mathcal{S}$ honestly follows the protocol. Then $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS}}$ and waits to receive the degree-$(d-t)$ packed Shamir sharings from P_{king}. If received, $\mathcal{S}$ honestly distributes the shares to all parties. Then $\mathcal{S}$ computes $\Delta[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_{d-t}$ and $\Delta[\boldsymbol{g}(\alpha_i)+\boldsymbol{b}_i]_{d-t}$ to be the difference between the packed Shamir sharings $\mathcal{S}$ received when simulating $\mathcal{F}_{\mathsf{ACSS}}$ and those determined by $\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i, \boldsymbol{g}(\alpha_i)+\boldsymbol{b}_i$ sampled by $\mathcal{S}$. In Step 3.3.(c), $\mathcal{S}$ computes

$$\begin{aligned}
&\Delta[\boldsymbol{f}(\alpha_i)*\boldsymbol{g}(\alpha_i)]_{2d-t}\\
=&([\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_{d-t} + \Delta[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_{d-t}) \cdot ([\boldsymbol{g}(\beta_i)+\boldsymbol{b}_i]_{d-t} + \Delta[\boldsymbol{g}(\beta_i)+\boldsymbol{b}_i]_{d-t})\\
&-([\boldsymbol{g}(\beta_i)+\boldsymbol{b}_i]_{d-t} + \Delta[\boldsymbol{g}(\beta_i)+\boldsymbol{b}_i]_{d-t}) \cdot ([\boldsymbol{a}_i]_d + \Delta[\boldsymbol{a}_i]_d)\\
&-([\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_{d-t} + \Delta[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_{d-t}) \cdot ([\boldsymbol{b}_i]_d + \Delta[\boldsymbol{b}_i]_d) + [\boldsymbol{c}_i]_d + \Delta[\boldsymbol{c}_i]_d\\
&-[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_{d-t} \cdot [\boldsymbol{g}(\beta_i)+\boldsymbol{b}_i]_{d-t} + [\boldsymbol{g}(\beta_i)+\boldsymbol{b}_i]_{d-t}[\boldsymbol{a}_i]_d\\
&+[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_{d-t}[\boldsymbol{b}_i]_d - [\boldsymbol{c}_i]_d\\
=&-\Delta[\boldsymbol{g}(\beta_i)+\boldsymbol{b}_i]_{d-t}[\boldsymbol{a}_i]_d - \Delta[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_{d-t}[\boldsymbol{b}_i]_d + \Delta[\boldsymbol{w}_i]_{2d-t},
\end{aligned}$$

where $\Delta[\boldsymbol{w}_i]_{2d-t}$ can be computed by $[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_{d-t}, \Delta[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_{d-t}, [\boldsymbol{g}(\beta_i)+\boldsymbol{b}_i]_{d-t}, \Delta[\boldsymbol{g}(\beta_i)+\boldsymbol{b}_i]_{d-t}$, $\Delta[\boldsymbol{a}_i]_d, \Delta[\boldsymbol{b}_i]_d, \Delta[\boldsymbol{c}_i]_d$ which are all known to $\mathcal{S}$. Since $[\boldsymbol{a}_i]_d$ is a linear combination of $[\boldsymbol{f}(\alpha_0)]_d$, $\{[\boldsymbol{f}(\alpha_i)]_d\}_{\alpha_i \in \mathcal{E}}$, and $[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_d$, and $[\boldsymbol{b}_i]_d$ is a linear combination of $[\boldsymbol{g}(\alpha_0)]_d$, $\{[\boldsymbol{g}(\alpha_i)]_d\}_{\alpha_i \in \mathcal{E}}$, and $[\boldsymbol{g}(\alpha_i)+\boldsymbol{b}_i]_d$, we may further write $\Delta[\boldsymbol{f}(\alpha_i)*\boldsymbol{g}(\alpha_i)]_{2d-t}$ as

$$\begin{aligned}
&\Delta[\boldsymbol{f}(\alpha_i)*\boldsymbol{g}(\alpha_i)]_{2d-t}\\
=&-\Delta[\boldsymbol{g}(\beta_i)+\boldsymbol{b}_i]_{d-t}[\boldsymbol{f}(\alpha_0)]_d - \Delta[\boldsymbol{f}(\alpha_i)+\boldsymbol{a}_i]_{d-t}[\boldsymbol{g}(\alpha_0)]_d + \Delta[\boldsymbol{w}'_i]_{2d-t},
\end{aligned}$$

where $\Delta[\boldsymbol{w}'_i]_{2d-t}$ can be explicitly computed by $\mathcal{S}$.

Following Step 3.5, $\mathcal{S}$ can computes $\Delta[\boldsymbol{u}]_{d-t}, \Delta[\boldsymbol{v}]_{d-t}, \Delta[\boldsymbol{w}]_{2d-t}$ such that

$$\Delta[\boldsymbol{h}(\alpha_0)]_{2d-t} = \Delta[\boldsymbol{u}]_{d-t}[\boldsymbol{f}(\alpha_0)]_d + \Delta[\boldsymbol{v}]_{d-t}[\boldsymbol{g}(\alpha_0)]_d + \Delta[\boldsymbol{w}]_{2d-t}.$$

In Step 4, $\mathcal{S}$ honestly follows the protocol. If any packed Shamir sharing received from P_{king} when simulating $\mathcal{F}_{\mathsf{ACSS}}$ is not of degree $d-t$, but the check in Step 4 passes, $\mathcal{S}$ outputs $\perp$ and terminates. Finally, if the check in Step 4 fails, $\mathcal{S}$ sets the outputs of honest parties to be `fail`. Otherwise, each difference in $\{\Delta[\boldsymbol{f}(\beta_i)+\boldsymbol{a}_i]_{d-t}, \Delta[\boldsymbol{g}(\beta_i)+\boldsymbol{b}_i]_{d-t}\}_{i=t+1}^{2t+1}$ is a valid degree-$(d-t)$ packed Shamir sharing. This implies that $\Delta[\boldsymbol{u}]_{d-t}, \Delta[\boldsymbol{v}]_{d-t}$ are valid degree-$(d-t)$ packed Shamir sharings. For each output packed Beaver triple $([\boldsymbol{f}(\alpha_0)]_d, [\boldsymbol{g}(\alpha_0)]_d, [\boldsymbol{h}(\alpha_0)]_{2d-t})$, $\mathcal{S}$ records the shares of parties in $\mathcal{C}orr'$, and the difference $\Delta[\boldsymbol{f}(\alpha_0)]_d, \Delta[\boldsymbol{g}(\alpha_0)]_d$, and $(\Delta[\boldsymbol{u}]_{d-t}, \Delta[\boldsymbol{v}]_{d-t}, \Delta[\boldsymbol{w}]_{2d-t})$.

In the following, we show the simulation of $\Pi_{\texttt{tripleGen}}$. In Step 1, $\mathcal{S}$ simulates $\Pi_{\texttt{tripleExtPack}}$ as described above. For each packed Beaver triple $([\boldsymbol{a}^{(\ell)}]_d, [\boldsymbol{b}^{(\ell)}]_d, [\boldsymbol{c}^{(\ell)}]_d)$, $\mathcal{S}$ has recorded the shares of parties in $\mathcal{C}orr'$ and $(\Delta[\boldsymbol{a}^{(\ell)}]_d, \Delta[\boldsymbol{b}^{(\ell)}]_d, \Delta[\boldsymbol{u}^{(\ell)}]_{d-t}, \Delta[\boldsymbol{v}^{(\ell)}]_{d-t}, \Delta[\boldsymbol{w}^{(\ell)}]_{2d-t})$. In particular, $\Delta[\boldsymbol{u}^{(\ell)}]_{d-t}, \Delta[\boldsymbol{v}^{(\ell)}]_{d-t}$ are valid degree-$(d-t)$ packed Shamir sharings. In Step 2, $\mathcal{S}$ simulates $\Pi_{\texttt{depack}}$ as follows.

In Step 2 of $\Pi_{\texttt{depack}}$, $\mathcal{S}$ first simulates $\mathcal{F}_{\mathsf{randDepack}}$ and receives the shares of corrupted parties. Then $\mathcal{S}$ randomly samples values as shares of parties in $\mathcal{C}orr' \backslash \mathcal{C}orr$. $\mathcal{S}$ distributes the shares to parties in $\mathcal{C}orr'$. In Step 3 of $\Pi_{\texttt{depack}}$, $\mathcal{S}$ computes the shares of $[\boldsymbol{x}^{(\ell)}+\boldsymbol{r}^{(\ell)}]_{2d-t}$ of parties in $\mathcal{C}orr'$. Then $\mathcal{S}$ samples random values as $\boldsymbol{x}^{(\ell)}+\boldsymbol{r}^{(\ell)}$. $\mathcal{S}$ computes the whole sharing $[\boldsymbol{x}^{(\ell)}+\boldsymbol{r}^{(\ell)}]_{2d-t}$ by using the secrets and the shares of parties in $\mathcal{C}orr'$. We consider two cases:

– If $[\boldsymbol{x}^{(\ell)}]_{2d-t}$ is from some $[\boldsymbol{a}]_d$ (or $[\boldsymbol{b}]_d$) in $\Pi_{\texttt{tripleGen}}$, for each honest party P_i, $\mathcal{S}$ sends the i-th share of $[\boldsymbol{x}^{(\ell)}+\boldsymbol{r}^{(\ell)}]_{2d-t} + \Delta[\boldsymbol{a}]_d$ to P_{king}.

– If $[\boldsymbol{x}^{(\ell)}]_{2d-t}$ is from some $[\boldsymbol{c}]_{2d-t}$ in $\Pi_{\texttt{tripleGen}}$, for each honest party P_i, $\mathcal{S}$ sends the i-th share of $[\boldsymbol{x}^{(\ell)} + \boldsymbol{r}^{(\ell)}]_{2d-t} + \Delta[\boldsymbol{w}]_{2d-t}$ to P_{king}.

If P_{king} is honest, $\mathcal{S}$ honestly follows the protocol. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS}}$ and waits to receive the degree-t Shamir sharings distributed by P_{king}. If received, $\mathcal{S}$ distributes the shares to all parties. Then $\mathcal{S}$ computes $\Delta x_i^{(\ell)}$ as follows:

– If $[\boldsymbol{x}^{(\ell)}]_{2d-t}$ is from some $[\boldsymbol{a}]_d$ (or $[\boldsymbol{b}]_d$) in $\Pi_{\texttt{tripleGen}}$, $\mathcal{S}$ sets $\Delta x_i^{(\ell)}$ to be the difference between the secret of $[x_i^{(\ell)} + r_i^{(\ell)}]_t$ distributed by P_{king} and $x_i^{(\ell)} + r_i^{(\ell)}$ sampled by $\mathcal{S}$.
– If $[\boldsymbol{x}^{(\ell)}]_{2d-t}$ is from some $[\boldsymbol{c}]_d$ in $\Pi_{\texttt{tripleGen}}$, $\mathcal{S}$ sets Δu_i to be the i-th secret of $\Delta[\boldsymbol{u}]_{d-t}$, Δv_i to be the i-th secret of $\Delta[\boldsymbol{v}]_{d-t}$, and Δw_i to be the difference between the secret of $[x_i^{(\ell)} + r_i^{(\ell)}]_t$ distributed by P_{king} and $x_i^{(\ell)} + r_i^{(\ell)}$ sampled by $\mathcal{S}$.

Coming back to $\Pi_{\texttt{tripleGen}}$, for each output Beaver triple $([a]_t, [b]_t, [c]_t)$, $\mathcal{S}$ records the shares of parties in $\mathcal{C}orr'$ and $(\Delta a, \Delta b, \Delta u, \Delta v, \Delta w)$. In hybrid arguments, we will show that all honest parties hold valid degree-t Shamir sharings $[a + \Delta a]_t, [b + \Delta b]_t, [c + \Delta u \cdot a + \Delta v \cdot b + \Delta w]_t$.

Finally, we show the simulation of $\Pi_{\texttt{tripleGen-GOD}}$. In Step 1, $\mathcal{S}$ simulates $\Pi_{\texttt{tripleGen}}$. For each Beaver triple $([a_i]_t, [b_i]_t, [c_i]_t)$, $\mathcal{S}$ has recorded the shares of parties in $\mathcal{C}orr'$ and $(\Delta a_i, \Delta b_i, \Delta u_i, \Delta v_i, \Delta w_i)$. Now for all $i \in \{0, \ldots, 2N\}$, $\mathcal{S}$ checks whether $\Delta a_i = \Delta v_i$, $\Delta b_i = \Delta u_i, \Delta a_i \cdot \Delta b_i = \Delta w_i$. If not, then when a_i, b_i are randomly sampled, with overwhelming probability

$$c_i + \Delta u_i \cdot a_i + \Delta v_i \cdot b_i + \Delta w_i \neq (a_i + \Delta a_i)(b_i + \Delta b_i).$$

In this case, $\mathcal{S}$ randomly samples $\{a_i, b_i, c_i\}_{i=0}^{2N}$ subject to $c_i = a_i \cdot b_i$ and then computes the whole sharings. $\mathcal{S}$ honestly follows the rest of steps. If the verification passes, $\mathcal{S}$ outputs $\perp$ and terminates.

Otherwise, $\mathcal{S}$ sets $a'_i = a_i + \Delta a_i, b'_i = b_i + \Delta b_i, c'_i = a'_i \cdot b'_i$. In this case, all honest parties hold valid Beaver triples. In Step 2, $\mathcal{S}$ follows the protocol to build f', g' based on $\{[a'_i]_t, [b'_i]_t\}_{i=0}^{N}$ and computes the shares of $[f'(\alpha_i)]_t, [g'(\alpha_i)]_t$ of parties in $\mathcal{C}orr'$ for all $i \in \{N+1, \ldots, 2N\}$. In Step 2.3, For all $i \in \{N+1, \ldots, 2N\}$, $\mathcal{S}$ samples two random values as $f'(\alpha_i) + a'_i, g'(\beta_i) + b'_i$. Then $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{pubRec}}$. $\mathcal{S}$ honestly computes the shares of $[f'(\alpha_i) \cdot g'(\alpha_i)]_t$ of parties in $\mathcal{C}orr'$. Next, $\mathcal{S}$ follows the protocol to build h'. In Step 3, $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{coin}}$. If $r \in \{1, \ldots, N\}$, $\mathcal{S}$ outputs $\perp$ and halts. Otherwise, $\mathcal{S}$ randomly samples two values as $f'(r), g'(r)$ and computes $h'(r) = f'(r) \cdot g'(r)$. Then $\mathcal{S}$ computes the whole sharings $[f'(r)]_t, [g'(r)]_t, [h'(r)]_t$ by using the secrets and the shares of parties in $\mathcal{C}orr'$. Finally, $\mathcal{S}$ honestly follows the rest of steps.

Combing back to $\Pi_{\texttt{tripleKing-GOD}}$, after simulating $\Pi_{\texttt{tripleExt-GOD}}$ and $\Pi_{\texttt{tripleGen-GOD}}$ as described above, $\mathcal{S}$ honestly follows the protocol in Step 2. If $b = 0$, $\mathcal{S}$ records the shares of parties in $\mathcal{C}orr'$ obtained when simulating $\Pi_{\texttt{tripleExt-GOD}}$. Otherwise, $\mathcal{S}$ records the shares of parties in $\mathcal{C}orr'$ obtained when simulating $\Pi_{\texttt{tripleGen-GOD}}$. Then in $\Pi_{\texttt{triple-GOD}}$, $\mathcal{S}$ honestly follows the protocol in Step 2. $\mathcal{S}$ records the shares of parties in $\mathcal{C}orr'$ for each successful king in $\mathcal{K}$ obtained when simulating $\Pi_{\texttt{tripleKing-GOD}}$.

Finally, $\mathcal{S}$ provides the shares of corrupted parties to $\mathcal{F}_{\mathsf{triple}}$ and outputs what $\mathcal{A}$ outputs. Whenever an honest party P_i should receive his shares, $\mathcal{S}$ delivers the output from $\mathcal{F}_{\mathsf{triple}}$ to P_i.

We show that the distribution of the output in the ideal world is statistically close 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 small hybrids, we focus on the simulation of $\Pi_{\texttt{tripleExt-GOD}}$.

$\mathbf{Hyb}_{1,1}$: In this hybrid, for each honest party P_i, we change the way of generating each degree-t Shamir sharing. We first generate the shares of parties in $\mathcal{C}orr'$, then compute the whole sharing based on the secret and the shares of parties in $\mathcal{C}orr'$. $\mathbf{Hyb}_{1,1}$ and $\mathbf{Hyb}_0$ have the same distribution.

$\mathbf{Hyb}_{1,2}$: In this hybrid, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ACSS}}$ and learns the shares of corrupted parties if the dealer is honest, and the whole sharings if the dealer is corrupted. Then $\mathcal{S}$ honestly simulates $\mathcal{F}_{\mathsf{coin}}$. If $r \in \{1, \ldots, N'\}$, $\mathcal{S}$ outputs $\perp$ and halts. Note that this happens with negligible probability. $\mathbf{Hyb}_{1,2}$ and $\mathbf{Hyb}_{1,1}$ are statistically close.

$\mathbf{Hyb}_{1,3}$: In this hybrid, for each honest party P_i, we randomly sample $f_i(r), g_i(r), h_i(r)$ such that $h_i(r) = f_i(r) \cdot g_i(r)$ and delay the generation of $(f_i(\alpha_\ell), g_i(\alpha_\ell), h_i(\alpha_\ell))_{\ell=1}^{N'}$ until Step 4. This does not change the distribution of f_i, g_i, h_i and it is sufficient for Step 3. $\mathbf{Hyb}_{1,3}$ and $\mathbf{Hyb}_{1,2}$ have the same distribution.

$\mathbf{Hyb}_{1,4}$: In this hybrid, for each corrupted party P_i, if $h_i(\alpha_\ell) \neq f_i(\alpha_\ell) \cdot g_i(\alpha_\ell)$ for some ℓ but the check in Step 3 passes, $\mathcal{S}$ outputs $\perp$ and terminates. By the Schwartz-Zippel lemma, this happens with negligible probability. $\mathbf{Hyb}_{1,4}$ and $\mathbf{Hyb}_{1,3}$ are statistically close.

$\mathbf{Hyb}_{1,5}$: In this hybrid, for each honest party $P_i \in \mathcal{D}$, we further change the way of determining the first two sharings $([a]_t, [b]_t)$ in each random Beaver triple as follows. At a high level, we first change the way of generating the shared polynomials $[f(\cdot)]_t, [g(\cdot)]_t$ in Step 4 and then decide the degree-t Shamir sharings distributed by honest parties based on $[f(\cdot)]_t, [g(\cdot)]_t$.

To be more concrete, in Step 4, suppose $\mathcal{D} = \{P_{j_1}, \ldots, P_{j_L}\}$. $\mathcal{S}$ first computes the shares of $[f(\cdot)]_t$ of parties in $\mathcal{C}orr'$. In Step 4.1, assume that $([a_i]_t, [b_i]_t, [c_i]_t)$ is distributed by $P_{j_i} \in \mathcal{D}$. For all corrupted party $P_{j_i} \in \mathcal{D}$, if $j_i \leq L' + 1$, set $[f(\alpha_i)]_t = [a_i]_t$. Otherwise, sample a random degree-t Shamir sharing as $[f(\alpha_i)]_t$ given the shares of parties in $\mathcal{C}orr'$. Then for all $i \in \{1, \ldots, (L+1)/2 - t\}$, sample a random degree-t Shamir sharing as $[f(\beta_i)]_t$ given the shares of parties in $\mathcal{C}orr'$. So far, we have fixed at most $t' + (L+1)/2 - t \leq L' + 1$ points. Next, we sample a random degree-L' polynomial $[f(\cdot)]_t$ that satisfies the above assignment and the shares of parties in $\mathcal{C}orr'$. For all honest party $P_{j_i} \in \mathcal{D}$, if $j_i \leq L' + 1$, we set $[a_i]_t = [f(\alpha_i)]_t$. Otherwise, we sample a random degree-t Shamir sharing $[a_i]_t$ given the shares of parties in $\mathcal{C}orr'$. The same process is done for $[b_i]_t$. And finally, $[c_i]_t = [a_i \cdot b_i]_t$ is computed based on the shares of parties in $\mathcal{C}orr'$.

To show that $\mathbf{Hyb}_{1,5}$ and $\mathbf{Hyb}_{1,4}$ are identically distributed, 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 $\mathbf{Hyb}_{1,4}$. To this end, it is sufficient to show that the distribution of the shared polynomial $[f(\cdot)]_t$ in both hybrids are identical. In $\mathbf{Hyb}_{1,4}$, $[f(\cdot)]_t$ is a random shared polynomial given $[f(\alpha_i)]_t = [a_i]_t$ for all $i \in \{1, \ldots, L' + 1\}$ where P_{j_i} is corrupted and given the shares of parties in $\mathcal{C}orr'$. In $\mathbf{Hyb}_{1,5}$, the only difference is that we additionally fix $[f(\alpha_i)]_t$ for all $i \in \{L' + 2, \ldots, L\}$ where P_{j_i} is corrupted and $[f(\beta_i)]_t$ for all $i \in \{1, \ldots, (L+1)/2 - t\}$. However, those degree-$t$ Shamir sharings are randomly sampled. Therefore, the obtained shared polynomial $[f(\cdot)]_t$ has the same distribution as that in $\mathbf{Hyb}_{1,4}$.

$\mathbf{Hyb}_{1,6}$: In this hybrid, for all $i \in \{L' + 2, \ldots, L\}$ where P_{j_i} is honest, instead of randomly sample degree-t Shamir sharings $[a_i]_t, [b_i]_t$, we first randomly sample $[f(\alpha_i) + a_i]_t, [g(\beta_i) + b_i]_t$ and then recompute $[a_i]_t, [b_i]_t$. The distributions of $\mathbf{Hyb}_{1,6}$ and $\mathbf{Hyb}_{1,5}$ are identical.

$\mathbf{Hyb}_{1,7}$: In this hybrid, we no longer generate the whole random Beaver triples for each honest party P_i. Instead, for each output Beaver triple $([f(\beta_i)]_t, [g(\beta_i)]_t, [h(\beta_i)]_t)$, we compute $h(\beta_i) = f(\beta_i) \cdot g(\beta_i)$ and then compute the whole sharing based on the secret and the shares of parties in $\mathcal{C}orr'$. Note that starting from $\mathbf{Hyb}_{1,4}$, for each Beaver triple $([a_i]_t, [b_i]_t, [c_i]_t)$, the shares of honest parties form a valid Beaver triple. Thus, each output Beaver triple is also correct. The distributions of $\mathbf{Hyb}_{1,7}$ and $\mathbf{Hyb}_{1,6}$ are identical.

$\mathbf{Hyb}_2$: In the following small hybrids, we focus on the simulation of $\mathit{\Pi}_{\mathtt{tripleGen\text{-}GOD}}$.

$\mathbf{Hyb}_{2,1}$: We first focus on the simulation of $\mathit{\Pi}_{\mathtt{ShBi}}$.

$\mathbf{Hyb}_{2,1,1}$: In this hybrid, $\mathit{\Pi}_{\mathtt{ShBi}}$ is simulated by $\mathcal{S}$ as described above when D is corrupted. Note that $\mathcal{S}$ just follows the protocol and records the polynomials that should be distributed by D (which are assumed to be all-0 polynomials) and the additive errors to shares of honest parties. The distributions of $\mathbf{Hyb}_{2,1,1}$ and $\mathbf{Hyb}_{1,7}$ are identical.

$\mathbf{Hyb}_{2,1,2}$: In this hybrid, when D is honest, we compute $\{\Delta f_{\ell,i}(x), \Delta g_{\ell,i}(y)\}_{\ell=0}^N$ for each honest party P_i as described above. We claim that for each $h(x) \in \{f_{\ell,i}(x), g_{\ell,i}(y)\}_{\ell=0}^N$, $\Delta h(x)$ is the difference between the polynomial that P_i actually received and the one he should receive. We consider two cases:

- If there exists $(2-2\epsilon)t+1$ different shares lie on a degree-d polynomial, since $d+1 \leq (2-4\epsilon)t+1$, such a degree-$d$ polynomial is unique. Since $h(x)$ that P_i should receive is a valid degree-d polynomial, the additive errors corresponds to these $(2-2\epsilon)t+1$ different shares also lie on a degree-d polynomial and such a degree-d polynomial is unique. Since $\Delta h(x)$ is equal to the difference between the polynomial that P_i actually received and the one he should receive for $(2-\epsilon)t+1 \geq d+1$ different evaluation points, $\Delta h(x)$ is the additive error that is added to the polynomial $h(x)$ that P_i should receive.
- Otherwise, any $(2-2\epsilon)t+1$ different shares do not lie on a degree-d polynomial. In this case, the additive errors to any $(2-2\epsilon)t+1$ different shares do not lie on a degree-d polynomial either. Then $\Delta h(x)$ is computed by using the additive errors to the first $d+1$ received shares. $\Delta h(x)$ is the additive error that is added to the polynomial $h(x)$ that P_i should receive.

$\mathbf{Hyb}_{2,1,3}$: In this hybrid, when D is honest, $\mathcal{S}$ only samples random degree-d polynomials $\{f_{0,i}(x), g_{0,i}(y)\}$ for all $P_i \in \mathcal{C}orr'$ such that $f_{0,i}(\alpha_j) = g_{0,j}(\alpha_i)$ for all $P_i, P_j \in \mathcal{C}orr'$ for D. Then in the verification step, $\mathcal{S}$ computes $\{f_i(x), g_i(y)\}$ for each party $P_i \in \mathcal{C}orr'$ and samples a random degree-d bivariate polynomial $F(x,y)$ such that $F(x,i) = f_i(x)$ and $F(i,y) = g_i(y)$ for all $P_i \in \mathcal{C}orr'$. Finally $\mathcal{S}$ computes $F_0(x,y) = F(x,y) - \sum_{\ell=1}^{N} r^i \cdot F_\ell(x,y)$. The distributions of $\mathbf{Hyb}_{2,1,3}$ and $\mathbf{Hyb}_{2,1,2}$ are identical.

$\mathbf{Hyb}_{2,1,4}$: In this hybrid, when D is honest, $\mathcal{S}$ no longer computes $F_0(x,y)$ and $\mathcal{S}$ simulates the verification step as described above. The only difference is that when an honest party broadcast his polynomials, we use the polynomial he should receive adding with the additive errors. The distributions of $\mathbf{Hyb}_{2,1,4}$ and $\mathbf{Hyb}_{2,1,3}$ are identical.

$\mathbf{Hyb}_{2,2}$: We then focus on the simulation of $\Pi_{\texttt{ShPack}}$.

$\mathbf{Hyb}_{2,2,1}$: In this hybrid, $\Pi_{\texttt{ShPack}}$ is simulated by $\mathcal{S}$ as described above when D is corrupted. Note that $\mathcal{S}$ just follows the protocol and records the shares that should be distributed by D (which are assumed to be all-0 shares) and the additive errors to shares of honest parties. The distributions of $\mathbf{Hyb}_{2,2,1}$ and $\mathbf{Hyb}_{2,1,4}$ are identical.

$\mathbf{Hyb}_{2,2,2}$: In this hybrid, when D is honest, for each degree-d bivariate polynomial, we first sample the shares of parties in $\mathcal{C}orr'$ and then compute the rest of shares based on the secrets and the shares of parties in $\mathcal{C}orr'$. The distributions of $\mathbf{Hyb}_{2,2,2}$ and $\mathbf{Hyb}_{2,2,1}$ are identical.

$\mathbf{Hyb}_{2,2,3}$: In this hybrid, when D is honest, $\mathcal{S}$ changes the way of preparing $\{F_j^{(0)}(x,y)\}_{j=1}^n$. In the verification step, $\mathcal{S}$ computes $\{F_i(x,\alpha_j), F_i(\alpha_j,y)\}_{i=1}^n$ for all parties in $\mathcal{C}orr'$. Then, $\mathcal{S}$ samples B random degree-d packed Shamir sharings $[\boldsymbol{s}_1]_d, \ldots, [\boldsymbol{s}_B]_d$. $\mathcal{S}$ computes $F_1(x,y), \ldots, F_n(x,y)$ as follows:

- $\mathcal{S}$ sets $F_i(\beta_{j_1}, \beta_{j_2})$ to be the i-th share of $[\boldsymbol{s}_{(j_1-1)\epsilon t + j_2}]_d$.
- $\mathcal{S}$ computes $F_i(x,y)$ given $F_i(\beta_{j_1}, \beta_{j_2})$ for all $j_1, j_2 \in \{1, \ldots, \epsilon t\}$ and $F_i(x,\alpha_j), F_i(\alpha_j, y)$ for all $P_j \in \mathcal{C}orr'$.

Finally, $\mathcal{S}$ computes $F_j^{(0)}(x,y) = F_j(x,y) - \sum_{\ell=1}^{N'} r^i F_j^{(\ell)}(x,y)$. The distributions of $\mathbf{Hyb}_{2,2,3}$ and $\mathbf{Hyb}_{2,2,2}$ are identical.

$\mathbf{Hyb}_{2,2,4}$: In this hybrid, when D is honest, the first 8 steps are simulated by $\mathcal{S}$ described above. The only difference is that we replace the actual shares of each honest party by the shares he should receive adding with the additive errors obtained when simulating $\Pi_{\texttt{ShBi}}$. The distributions of $\mathbf{Hyb}_{2,2,4}$ and $\mathbf{Hyb}_{2,2,3}$ are identical.

$\mathbf{Hyb}_{2,2,5}$: In this hybrid, when D is honest, $\mathcal{S}$ computes the additive errors $\{\Delta[\boldsymbol{s}_\ell]_d\}_{\ell=1}^N$ as described above. Following the same argument as that in $\mathbf{Hyb}_{2,1,2}$, the computed additive errors are identical to the difference between the shares of honest parties that they actually received and the shares they should receive. (Note that the additive errors for shares of corrupted parties are always 0.)

$\mathbf{Hyb}_{2,2,6}$: In this hybrid, when D is honest, $\mathcal{S}$ no longer generates the whole degree-d bivariate polynomials but only keeps the shares of parties in $\mathcal{C}orr'$. The distributions of $\mathbf{Hyb}_{2,2,6}$ and $\mathbf{Hyb}_{2,2,5}$ are identical.

$\mathbf{Hyb}_{2,3}$: Next, we focus on the simulation of $\Pi_{\texttt{ShTriple}}$.

$\mathbf{Hyb}_{2,3,1}$: In this hybrid, $\Pi_{\texttt{ShTriple}}$ is simulated by $\mathcal{S}$ as described above when D is corrupted. Note that $\mathcal{S}$ just follows the protocol and records the shares that should be distributed by D (which are assumed to be all-0 shares) and the additive errors to shares of honest parties. The only difference is that if $r \in \{1, \ldots, N\}$, $\mathcal{S}$ outputs $\perp$ and halts. This happens with negligible probability. The distributions of $\mathbf{Hyb}_{2,3,1}$ and $\mathbf{Hyb}_{2,2,6}$ are statistically close.

$\mathbf{Hyb}_{2,3,2}$: In this hybrid, when D is honest, for all $[\boldsymbol{z}]_d \in \{[\boldsymbol{a}_0]_d, [\boldsymbol{b}_0]_d, [\boldsymbol{c}_0]_d\} \cup \{[\boldsymbol{h}(\alpha_\ell)]_d\}_{\ell=N+1}^{2N}$, $\mathcal{S}$ only samples random values as shares of parties in $\mathcal{C}orr'$. Then in Step 3 after r is sampled, if $r \in \{1, \ldots, N\}$, $\mathcal{S}$ outputs $\perp$ and halts. Otherwise, $\mathcal{S}$ computes the shares of $([\boldsymbol{f}(r)]_d, [\boldsymbol{g}(r)]_d, [\boldsymbol{h}(r)]_d)$ of parties in $\mathcal{C}orr'$ and the additive errors $(\Delta[\boldsymbol{f}(r)]_d, \Delta[\boldsymbol{g}(r)]_d, \Delta[\boldsymbol{h}(r)]_d)$. Next $\mathcal{S}$ randomly samples $\boldsymbol{f}(r), \boldsymbol{g}(r), \boldsymbol{h}(r)$ such that $\boldsymbol{h}(r) = \boldsymbol{f}(r) * \boldsymbol{g}(r)$ and computes the whole sharings $([\boldsymbol{f}(r)]_d, [\boldsymbol{g}(r)]_d, [\boldsymbol{h}(r)]_d)$. In Step 4, $\mathcal{S}$ computes the shares of honest parties by adding the additive errors obtained when simulating $\Pi_{\texttt{ShPack}}$. The distributions of $\mathbf{Hyb}_{2,3,2}$ and $\mathbf{Hyb}_{2,3,1}$ are statistically close.

$\mathbf{Hyb}_{2,4}$: We focus on the simulation of $\Pi_{\texttt{tripleExtPack}}$.

$\mathbf{Hyb}_{2,4,1}$: In this hybrid, for each honest party $P_i \in \mathcal{D}$, we change the way of determining the first two sharings $([\boldsymbol{a}]_d, [\boldsymbol{b}]_d)$ in each random packed Beaver triple as follows. At a high level, we first change the way of generating the shared polynomials $[\boldsymbol{f}(\cdot)]_d, [\boldsymbol{g}(\cdot)]_d$ in Step 3 and then decide the degree-t Shamir sharings distributed by honest parties based on $[\boldsymbol{f}(\cdot)]_d, [\boldsymbol{g}(\cdot)]_d$.

To be more concrete, in Step 3, suppose $\mathcal{D} = \{P_{j_1}, \ldots, P_{j_{2t+1}}\}$. $\mathcal{S}$ first computes the shares of $[\boldsymbol{f}(\cdot)]_d$ of parties in $\mathcal{C}orr'$. In Step 4.1, assume that $([\boldsymbol{a}_i]_d, [\boldsymbol{b}_i]_d, [\boldsymbol{c}_i]_d)$ is distributed by $P_{j_i} \in \mathcal{D}$. For all corrupted party $P_{j_i} \in \mathcal{D}$, if $j_i \leq t+1$, set $[\boldsymbol{f}(\alpha_i)]_d = [\boldsymbol{a}_i]_d$. Otherwise, sample a random degree-d packed Shamir sharing as $[\boldsymbol{f}(\alpha_i)]_d$ given the shares of parties in $\mathcal{C}orr'$. Let t'' denote the number of corrupted parties in $\mathcal{D}$. If $t'' < t$, for each P_{j_i} of the first $t - t''$ honest party in $\mathcal{D}$, sample a random degree-d packed Shamir sharing as $[\boldsymbol{f}(\alpha_i)]_d$ given the shares of parties in $\mathcal{C}orr'$. So far, we have fixed t evaluation points. Next, we sample a random degree-d packed Shamir sharing $[\boldsymbol{f}(\alpha_0)]_d$ based on the shares of parties in $\mathcal{C}orr'$. Now we interpolate $[\boldsymbol{f}(\cdot)]_d$ using the above $t+1$ evaluation points. For all honest party $P_{j_i} \in \mathcal{D}$, if $j_i \leq t+1$, we set $[\boldsymbol{a}_i]_d = [\boldsymbol{f}(\alpha_i)]_d$. Otherwise, we sample a random degree-d packed Shamir sharing $[\boldsymbol{a}_i]_d$ given the shares of parties in $\mathcal{C}orr'$. The same process is done for $[\boldsymbol{b}_i]_d$. And finally, $[\boldsymbol{c}_i]_t = [\boldsymbol{a}_i * \boldsymbol{b}_i]_d$ is computed based on the shares of parties in $\mathcal{C}orr'$.

To show that $\mathbf{Hyb}_{2,4,1}$ and $\mathbf{Hyb}_{2,3,2}$ are identically distributed, it is sufficient to show that the degree-d packed Shamir sharings of honest parties generated in the above approach are identically distributed to those in $\mathbf{Hyb}_{2,3,2}$. To this end, it is sufficient to show that the distribution of the shared polynomials $[\boldsymbol{f}(\cdot)]_d$ in both hybrids are identical. In $\mathbf{Hyb}_{2,3,2}$, $[\boldsymbol{f}(\cdot)]_d$ is a random vector of shared polynomials given $[\boldsymbol{f}(\alpha_i)]_d = [\boldsymbol{a}_i]_d$ for all $i \in \{1, \ldots, t+1\}$ where P_{j_i} is corrupted and given the shares of parties in $\mathcal{C}orr'$. In $\mathbf{Hyb}_{2,4,1}$, the only difference is that we randomly sample $[\boldsymbol{f}(\alpha_0)]_d$, $[\boldsymbol{f}(\alpha_i)]_d$ for all $i \in \{t+2, \ldots, 2t+1\}$ where P_{j_i} is corrupted, and $[\boldsymbol{f}(\alpha_i)]_d$ for the first $t - t''$ honest parties in $\mathcal{D}$. The obtained shared polynomials $[\boldsymbol{f}(\cdot)]_d$ has the same distribution as that in $\mathbf{Hyb}_{2,3,2}$.

$\mathbf{Hyb}_{2,4,2}$: In this hybrid, for all $i \in \{t+2, \ldots, 2t+1\}$ where P_{j_i} is honest, instead of randomly sample degree-d packed Shamir sharings $[\boldsymbol{a}_i]_d, [\boldsymbol{b}_i]_d$, we first randomly sample $[\boldsymbol{f}(\alpha_i) + \boldsymbol{a}_i]_d, [\boldsymbol{g}(\beta_i) + \boldsymbol{b}_i]_d$ and then recompute $[\boldsymbol{a}_i]_d, [\boldsymbol{b}_i]_d$. The distributions of $\mathbf{Hyb}_{2,4,2}$ and $\mathbf{Hyb}_{2,4,1}$ are identical.

$\mathbf{Hyb}_{2,4,3}$: In this hybrid, $\mathcal{S}$ simulates $\Pi_{\texttt{tripleExtPack}}$ until Step 3.(c). The only difference is that when sending shares to P_{king}, each honest parties' shares are prepared by using the shares they should hold adding with the additive errors obtained when simulating $\Pi_{\texttt{ShTriple}}$. The distributions of $\mathbf{Hyb}_{2,4,3}$ and $\mathbf{Hyb}_{2,4,2}$ are identical.

$\mathbf{Hyb}_{2,4,4}$: In this hybrid, for each $[\boldsymbol{f}(\alpha_i) * \boldsymbol{g}(\alpha_i)]_{2d-t}$, $\mathcal{S}$ computes the difference between the shares honest parties actually hold and the shares they should hold. In particular, we have

$$\begin{aligned}
&\Delta[\boldsymbol{f}(\alpha_i) * \boldsymbol{g}(\alpha_i)]_{2d-t} \\
=& -\Delta[\boldsymbol{g}(\beta_i) + \boldsymbol{b}_i]_{d-t}[\boldsymbol{f}(\alpha_0)]_d - \Delta[\boldsymbol{f}(\alpha_i) + \boldsymbol{a}_i]_{d-t}[\boldsymbol{g}(\alpha_0)]_d + \Delta[\boldsymbol{w}'_i]_{2d-t}.
\end{aligned}$$

$\mathbf{Hyb}_{2,4,5}$: In this hybrid, for $[\boldsymbol{h}(\alpha_0)]_{2d-t}$, $\mathcal{S}$ computes the difference between the shares honest parties actually hold and the shares they should hold. In particular, we have

$$\Delta[\boldsymbol{h}(\alpha_0)]_{2d-t} = \Delta[\boldsymbol{u}]_{d-t}[\boldsymbol{f}(\alpha_0)]_d + \Delta[\boldsymbol{v}]_{d-t}[\boldsymbol{g}(\alpha_0)]_d + \Delta[\boldsymbol{w}]_{2d-t}.$$

$\mathbf{Hyb}_{2,4,6}$: In this hybrid, Step 4 is simulated by $\mathcal{S}$. The only difference is that if any packed Shamir sharing received from P_{king} when simulating $\mathcal{F}_{\mathsf{ACSS}}$ is not of degree $d-t$, but the check in Step 4 passes, $\mathcal{S}$ outputs $\perp$ and terminates. By the Schwartz-Zippel lemma, this happens with negligible probability. Thus, the distributions of $\mathbf{Hyb}_{2,4,6}$ and $\mathbf{Hyb}_{2,4,5}$ are statistically close. Note that if $\mathcal{S}$ does not terminate, $(\Delta[\boldsymbol{u}]_{d-t}, \Delta[\boldsymbol{v}]_{d-t})$ are valid degree-$(d-t)$ packed Shamir sharings.

$\mathbf{Hyb}_{2,4,7}$: In this hybrid, we delay the sampling of $[\boldsymbol{f}(\alpha_0)]_d, [\boldsymbol{g}(\alpha_0)]_d, [\boldsymbol{h}(\alpha_0)]_{2d-t}$ to the end of $\Pi_{\texttt{tripleExtPack}}$. Note that these sharings are not needed in the simulation of $\Pi_{\texttt{tripleExtPack}}$.

$\mathbf{Hyb}_{2,5}$: In the following, we focus on the simulation of $\Pi_{\texttt{tripleGen}}$.

$\mathbf{Hyb}_{2,5,1}$: In this hybrid, $\mathcal{F}_{\mathsf{randDepack}}$ in Π_{depack} is simulated by $\mathcal{S}$. Then the shares of honest parties are generated given the shares of parties in $\mathcal{C}orr'$. The distribution of $\mathbf{Hyb}_{2,5,1}$ is identical to that of $\mathbf{Hyb}_{2,4,7}$.

$\mathbf{Hyb}_{2,5,2}$: In this hybrid, we change the way of preparing correlated randomness in $\mathcal{F}_{\mathsf{randDepack}}$. In Step 3 of Π_{depack}, $\mathcal{S}$ computes the shares of $[\boldsymbol{x}^{(\ell)} + \boldsymbol{r}^{(\ell)}]_{2d-t}$ of parties in $\mathcal{C}orr'$. Then $\mathcal{S}$ samples random values as $\boldsymbol{x}^{(\ell)} + \boldsymbol{r}^{(\ell)}$. $\mathcal{S}$ computes the whole sharing $[\boldsymbol{x}^{(\ell)} + \boldsymbol{r}^{(\ell)}]_{2d-t}$ by using the secrets and the shares of parties in $\mathcal{C}orr'$.

– If $[\boldsymbol{x}^{(\ell)}]_{2d-t}$ is from some $[\boldsymbol{a}]_d$ (or $[\boldsymbol{b}]_d$) in $\Pi_{\texttt{tripleGen}}$, we set $[\boldsymbol{r}^{(\ell)}]_{2d-t} = [\boldsymbol{x}^{(\ell)} + \boldsymbol{r}^{(\ell)}]_{2d-t} - [\boldsymbol{a}]_d$. Effectively, here we set $[\boldsymbol{x}^{(\ell)}]_{2d-t} = [\boldsymbol{a}]_d$. Then $\Delta[\boldsymbol{x}^{(\ell)}]_{2d-t} = \Delta[\boldsymbol{a}]_d$.

– If $[\boldsymbol{x}^{(\ell)}]_{2d-t}$ is from some $[\boldsymbol{c}]_{2d-t}$ in $\Pi_{\texttt{tripleGen}}$, then $\mathcal{S}$ has computed $\Delta[\boldsymbol{u}]_{d-t}, \Delta[\boldsymbol{v}]_{d-t}, \Delta[\boldsymbol{w}]_{2d-t}$ such that honest parties actual shares of $[\boldsymbol{c}]_{2d-t}$ are

$$[\boldsymbol{c}]_{2d-t} + \Delta[\boldsymbol{u}]_{d-t} \cdot [\boldsymbol{a}]_d + \Delta[\boldsymbol{v}]_{d-t} \cdot [\boldsymbol{b}]_d + \Delta[\boldsymbol{w}]_{2d-t}.$$

In particular, $\Delta[\boldsymbol{u}]_{d-t}$ and $\Delta[\boldsymbol{v}]_{d-t}$ are valid degree-$(d-t)$ packed Shamir sharings. Therefore, $[\boldsymbol{c}]_{2d-t} + \Delta[\boldsymbol{u}]_{d-t} \cdot [\boldsymbol{a}]_d + \Delta[\boldsymbol{v}]_{d-t} \cdot [\boldsymbol{b}]_d$ is a valid degree-$(2d-t)$ packed Shamir sharing. We set $[\boldsymbol{r}^{(\ell)}]_{2d-t} = [\boldsymbol{x}^{(\ell)} + \boldsymbol{r}^{(\ell)}]_{2d-t} - [\boldsymbol{c}]_{2d-t} - \Delta[\boldsymbol{u}]_{d-t} \cdot [\boldsymbol{a}]_d - \Delta[\boldsymbol{v}]_{d-t} \cdot [\boldsymbol{b}]_d$. for each honest party P_i, $\mathcal{S}$ sends the i-th share of $[\boldsymbol{x}^{(\ell)} + \boldsymbol{r}^{(\ell)}]_{2d-t} + \Delta[\boldsymbol{w}]_{2d-t}$ to P_{king}. Effectively, here we set $[\boldsymbol{x}^{(\ell)}]_{2d-t} = [\boldsymbol{c}]_{2d-t} + \Delta[\boldsymbol{u}]_{d-t} \cdot [\boldsymbol{a}]_d + \Delta[\boldsymbol{v}]_{d-t} \cdot [\boldsymbol{b}]_d$. Then $\Delta[\boldsymbol{x}^{(\ell)}]_{2d-t} = \Delta[\boldsymbol{w}]_{2d-t}$.

Note that $[\boldsymbol{r}^{(\ell)}]_{2d-t}$ is still a random degree-$(2d-t)$ packed Shamir sharing given the shares of parties in $\mathcal{C}orr'$. The distributions of $\mathbf{Hyb}_{2,5,2}$ and $\mathbf{Hyb}_{2,5,1}$ are identical.

$\mathbf{Hyb}_{2,5,3}$: In this hybrid, $\Pi_{\texttt{depack}}$ is simulated by $\mathcal{S}$ described above. The only difference is that when sending shares to P_{king}, each honest parties' shares are prepared by using the shares they should hold adding with the additive errors obtained when simulating $\Pi_{\texttt{tripleExtPack}}$. The distributions of $\mathbf{Hyb}_{2,5,3}$ and $\mathbf{Hyb}_{2,5,2}$ are identical.

$\mathbf{Hyb}_{2,5,4}$: In this hybrid, $\mathcal{S}$ computes $\Delta x_i^{(\ell)}$ as described above. We have the following two facts.

– If $[\boldsymbol{x}^{(\ell)}]_{2d-t}$ is from some $[\boldsymbol{a}]_d$ (or $[\boldsymbol{b}]_d$) in $\Pi_{\texttt{tripleGen}}$, $\mathcal{S}$ sets $\Delta x_i^{(\ell)}$ to be the difference between the secret of $[x_i^{(\ell)} + r_i^{(\ell)}]_t$ distributed by P_{king} and $x_i^{(\ell)} + r_i^{(\ell)}$ sampled by $\mathcal{S}$. Recall that in this case, $\boldsymbol{x}^{(\ell)} = \boldsymbol{a}$. So the additive error to $x_i^{(\ell)}$ is equal to the additive to a_i.
– If $[\boldsymbol{x}^{(\ell)}]_{2d-t}$ is from some $[\boldsymbol{c}]_d$ in $\Pi_{\texttt{tripleGen}}$, $\mathcal{S}$ sets Δu_i to be the i-th secret of $\Delta[\boldsymbol{u}]_{d-t}$, Δv_i to be the i-th secret of $\Delta[\boldsymbol{v}]_{d-t}$, and Δw_i to be the difference between the secret of $[x_i^{(\ell)} + r_i^{(\ell)}]_t$ distributed by P_{king} and $x_i^{(\ell)} + r_i^{(\ell)}$ sampled by $\mathcal{S}$. Recall that in this case $\boldsymbol{x}^{(\ell)} = \boldsymbol{c} + \boldsymbol{u} * \boldsymbol{a} + \boldsymbol{v} * \boldsymbol{b}$. Therefore, considering the additive error to $x_i^{(\ell)}$, the error to c_i is $u_i \cdot a_i + v_i \cdot b_i + w_i$.

$\mathbf{Hyb}_{2,5,5}$: In this hybrid, we do not generate the whole sharings $\{[\boldsymbol{a}^{(\ell)}]_d, [\boldsymbol{b}^{(\ell)}]_d, [\boldsymbol{c}^{(\ell)}]_{2d-t}\}_{\ell=1}^{N'}$. Instead, we only generate $([a_i^{(\ell)}]_t, [b_i^{(\ell)}]_t, [c_i^{(\ell)}]_t)$ for all $i \in \{1, \ldots, d-t+1\}, \ell \in \{1, \ldots, N'\}$ at the end of $\Pi_{\texttt{tripleGen}}$. Note that the simulation does not need to use the whole sharings of $\{[\boldsymbol{a}^{(\ell)}]_d, [\boldsymbol{b}^{(\ell)}]_d, [\boldsymbol{c}^{(\ell)}]_{2d-t}\}_{\ell=1}^{N'}$ or $([a_i^{(\ell)}]_t, [b_i^{(\ell)}]_t, [c_i^{(\ell)}]_t)$ for all $i \in \{1, \ldots, d-t+1\}, \ell \in \{1, \ldots, N'\}$.

$\mathbf{Hyb}_{2,6}$: Now we focus on the simulation of $\Pi_{\texttt{tripleGen-GOD}}$.

$\mathbf{Hyb}_{2,6,1}$: In this hybrid, if there exists $i \in \{0, \ldots, 2N\}$ such that at least one of $\Delta a_i = \Delta v_i$, $\Delta b_i = \Delta u_i, \Delta a_i \cdot \Delta b_i = \Delta w_i$ does not hold, $\Pi_{\texttt{tripleGen-GOD}}$ is simulated by $\mathcal{S}$ as described above. The only difference is that if the verification passes, $\mathcal{S}$ outputs $\perp$ and terminates.

We show that this happens with negligible probability. First note that when at least one of $\Delta a_i = \Delta v_i$, $\Delta b_i = \Delta u_i, \Delta a_i \cdot \Delta b_i = \Delta w_i$ does not hold, with overwhelming probability

$$c_i + \Delta u_i \cdot a_i + \Delta v_i \cdot b_i + \Delta w_i \neq (a_i + \Delta a_i)(b_i + \Delta b_i).$$

In other words, with overwhelming probability, at least one of the Beaver triples all parties hold are incorrect. Now we argue that in this case, $f \cdot g \neq h$. If one of the first $N+1$ Beaver triples are incorrect, say the i-th one, then we immediately have $f(\alpha_i) \cdot g(\alpha_i) \neq h(\alpha_i)$. Otherwise, if the first $N+1$ Beaver triples are correct, then for some $i \geq N+2$, the i-th triple is incorrect. In this case, we must have $f(\alpha_i) \cdot g(\alpha_i) \neq h(\alpha_i)$.

In the verification, if $f \cdot g \neq h$, with overwhelming probability, $f(r) \cdot g(r) \neq h(r)$. In this case, the verification fails. Thus, the probability that the verification passes is negligible. The distributions of $\mathbf{Hyb}_{2,6,1}$ and $\mathbf{Hyb}_{2,5,5}$ are statistically close.

$\mathbf{Hyb}_{2,6,2}$: In this hybrid, if for all $i \in \{0, \ldots, 2N\}$, $\Delta a_i = \Delta v_i$, $\Delta b_i = \Delta u_i, \Delta a_i \cdot \Delta b_i = \Delta w_i$, $\Pi_{\texttt{tripleGen-GOD}}$ is simulated by $\mathcal{S}$ described above. The differences are that (1) $\mathcal{S}$ samples random values as $f'(\alpha_i) + a'_i, g'(\beta_i) + b'_i$, and (2) if $r \in \{1, \ldots, N\}$, then $\mathcal{S}$ outputs $\perp$ and halts. Since $a'_i = a_i + \Delta a_i$ and a_i is a random value, $f'(\alpha_i) + a'_i$ is a random value in $\mathbf{Hyb}_{2,6,1}$. Since r is a random value, the probability that $r \in \{1, \ldots, N\}$ is negligible. Thus, the distributions of $\mathbf{Hyb}_{2,6,2}$ and $\mathbf{Hyb}_{2,6,1}$ are statistically close.

$\mathbf{Hyb}_{2,6,3}$: In this hybrid, if for all $i \in \{0, \ldots, 2N\}$, $\Delta a_i = \Delta v_i$, $\Delta b_i = \Delta u_i, \Delta a_i \cdot \Delta b_i = \Delta w_i$, $\mathcal{S}$ does not generate the whole sharings $\{[a_i]_t, [b_i]_t, [c_i]_t\}_{i=0}^{2N}$ but only generate the whole sharings $\{[a'_i]_t, [b'_i]_t, [c'_i]_t\}_{i=1}^{N}$ at the end of $\Pi_{\texttt{tripleGen-GOD}}$. Note that those sharings are not used in the simulation.

$\mathbf{Hyb}_{2,6,4}$: In this hybrid, let $\mathcal{D}$ be the set of parties such that for each $P_i \in \mathcal{D}$, at least one honest party terminates $\mathcal{F}_{\mathsf{ACSS}}$ led by P_i in $\Pi_{\mathtt{tripleExt\text{-}GOD}}$ at the end of $\Pi_{\mathtt{tripleGen\text{-}GOD}}$. If all parties take `fail` as output while P_{king} is honest, $|\mathcal{D}| \leq (2+\epsilon)t+1$, and $\mathcal{D}$ contains at most ϵt corrupted parties, $\mathcal{S}$ outputs $\perp$ and halts. As we argued above (where we prove that $\Pi_{\mathtt{triple\text{-}GOD}}$ eventually terminates), this happens with negligible probability. Thus, the distributions of $\mathbf{Hyb}_{2,6,4}$ and $\mathbf{Hyb}_{2,6,3}$ are statistically close.

$\mathbf{Hyb}_3$: In this hybrid, $\mathcal{S}$ honestly follows Step 2 of $\Pi_{\mathtt{tripleKing\text{-}GOD}}$ and Step 2 of $\Pi_{\mathtt{triple\text{-}GOD}}$, for each of the first $n-t$ successful kings in $\mathcal{K}$, $\mathcal{S}$ provides the shares of the output triples of corrupted parties to $\mathcal{F}_{\mathsf{triple}}$ and does not generate the shares of honest parties by itself. Instead, the shares of honest parties are generated by $\mathcal{F}_{\mathsf{triple}}$. Note that those triples are generated in the same way. $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_{2,6,4}$ are identically distributed.

Since $\mathbf{Hyb}_3$ corresponds to the ideal world, $\Pi_{\mathtt{triple\text{-}GOD}}$ securely computes $\mathcal{F}_{\mathsf{triple}}$.

D.6 Proof of Lemma 7

Proof. We first show that all honest parties will eventually terminate the protocol $\Pi_{\mathtt{main}}$.

- In the offline phase, all parties are guaranteed to finish $\mathcal{F}_{\mathsf{triple}}$ and $\mathcal{F}_{\mathsf{randShare}}$.
- In the input phase, by the online error correction algorithm, every party P_i will eventually reconstruct $[r_i]_t$ and the secret r_i. Therefore, every honest party P_i will eventually start the broadcast protocol. According to the property of the broadcast protocol, every honest party P_i will finishes the protocol led by another honest party P_j. Thus, Q is an ACS property. Therefore all parties will eventually terminate Π_{acs}^Q and agree on a set $\mathcal{D}$ of at least $n-t$ parties that successfully share their inputs. In particular, by the property of the broadcast channel, all honest parties will obtain their shares of inputs of parties in $\mathcal{D}$.
- In the computation phase, all parties are guaranteed to finish $\mathcal{F}_{\mathsf{pubRec}}$, which is the only interactive step.
- In the output phase, by the online error correction algorithm, every party P_i will eventually reconstruct $[y_i]_t$ and the secret y_i.

Now we show that the protocol $\Pi_{\mathtt{main}}$ securely computes $\mathcal{F}_{\mathsf{fs}}$. Let $\mathcal{A}$ be a static malicious adversary which controls a set $\mathcal{C}orr$ of $t' \leq t$ corrupted parties. Let $\mathcal{Z}$ be an environment. We construct an ideal adversary $\mathcal{S}$ interacting with the environment $\mathcal{Z}$ and the ideal functionality $\mathcal{F}_{\mathsf{fs}}$. $\mathcal{S}$ starts with running $\mathcal{A}$ and passes messages between $\mathcal{Z}$ and $\mathcal{A}$. For corrupted parties, $\mathcal{S}$ faithfully follows the instructions of $\mathcal{A}$. Then $\mathcal{S}$ simulates the behaviors of honest parties as follows. Let $\mathcal{C}orr'$ be the set of all corrupted parties together with the first $t-t'$ honest parties. Then $|\mathcal{C}orr'| = t$. In the following, we will explicitly generate the shares of all parties in $\mathcal{C}orr'$. In this way, given the shares of parties in $\mathcal{C}orr'$ and the secret, a degree-t Shamir secret sharing is fully determined.

In the offline phase, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{triple}}$ and $\mathcal{F}_{\mathsf{randShare}}$ and receives the shares of corrupted parties from $\mathcal{A}$. Then $\mathcal{S}$ samples random values as shares of parties in $\mathcal{C}orr' \backslash \mathcal{C}orr$.

In the input phase, for each honest party P_i, $\mathcal{S}$ waits to receive shares from all parties. After receiving $2t+1$ correct shares (note that the share from an honest party is always correct, and $\mathcal{S}$ knows the shares of corrupted parties and can check the correctness of the share received from a corrupted party), $\mathcal{S}$ samples a random value as $x_i + r_i$ and honestly broadcasts $x_i + r_i$. Then $\mathcal{S}$ computes the shares of $[x_i]_t$ of parties in $\mathcal{C}orr'$. For each corrupted party P_i, $\mathcal{S}$ samples a random degree-t Shamir sharing $[r_i]_t$ based on the shares of parties in $\mathcal{C}orr'$. Then $\mathcal{S}$ sends the shares of honest parties to P_i on behalf of honest parties. $\mathcal{S}$ honestly follows the ACS protocol. For each corrupted party $P_i \in \mathcal{D}$, $\mathcal{S}$ receives $x_i + r_i$ from P_i and computes $x_i = (x_i + r_i) - r_i$ and the shares of $[x_i]_t$ of parties in $\mathcal{C}orr'$. For each corrupted party $P_i \notin \mathcal{D}$, $\mathcal{S}$ sets $x_i = 0$.

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 $\mathcal{S}$ samples two random degree-t Shamir sharings as $[x+a]_t, [y+b]_t$ based on the shares of parties in $\mathcal{C}orr'$. $\mathcal{S}$ honestly follows $\mathcal{F}_{\mathsf{pubRec}}$. Finally, $\mathcal{S}$ follows the protocol and computes the shares of $[z]_t$ of parties in $\mathcal{C}orr'$.

In the output phase, $\mathcal{S}$ provides the inputs of corrupted parties and the set $\mathcal{D}$ to $\mathcal{F}_{\mathsf{fs}}$ and receives the outputs of corrupted parties. For each corrupted party P_i, $\mathcal{S}$ computes the shares of $[y_i]_t$ of honest parties given the shares of parties in $\mathcal{C}orr'$ and the output y_i. $\mathcal{S}$ sends the shares of $[y_i]_t$ of honest parties to P_i. For each honest party P_i, $\mathcal{S}$ waits to receive shares from all parties. After receiving $2t+1$ correct shares, $\mathcal{S}$ delivers the output from $\mathcal{F}_{\mathsf{fs}}$ to P_i.

Finally, $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

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, in $\mathcal{F}_{\mathsf{triple}}$ and $\mathcal{F}_{\mathsf{randShare}}$, for each degree-$t$ Shamir sharing, we first sample random values as the shares of parties in $\mathcal{C}orr'\backslash\mathcal{C}orr$ and then generate the rest of shares accordingly. This does not change the distribution of the output of $\mathcal{F}_{\mathsf{triple}}$ and $\mathcal{F}_{\mathsf{randShare}}$. The distribution of $\mathbf{Hyb}_1$ is identical to that of $\mathbf{Hyb}_0$.

$\mathbf{Hyb}_2$: In this hybrid, in the input phase, for each honest party P_i, $\mathcal{S}$ samples a random value $x_i + r_i$ and then computes the whole sharing of $[r_i]_t$ given the shares of parties in $\mathcal{C}orr'$ and the secret r_i. The only difference is that in $\mathbf{Hyb}_1$ we first sample a random value r_i and then compute $x_i + r_i$ while in $\mathbf{Hyb}_2$ we switch the order. The distribution of $\mathbf{Hyb}_2$ is identical to that of $\mathbf{Hyb}_1$.

$\mathbf{Hyb}_3$: In this hybrid, in the input phase, for each honest party $P_i \in \mathcal{D}$, $\mathcal{S}$ computes the shares of $[x_i]_t$ of parties in $\mathcal{C}orr'$ and then generate the whole sharing based on the secret x_i. Since a degree-t Shamir sharing is fully determined by the shares of parties in $\mathcal{C}orr'$ and the secret, this does not change the distribution of the shares of honest parties. Note that $[r_i]_t$ is no longer used in the input phase (except the shares of parties in $\mathcal{C}orr'$). We do not generate the full sharing of $[r_i]_t$.

$\mathbf{Hyb}_4$: In this hybrid, in the computation phase, for every multiplication gate, $\mathcal{S}$ computes the shares of $[x+a]_t, [y+b]_t$ of parties in $\mathcal{C}orr'$ and then samples two random degree-t Shamir sharings as $[x+a]_t, [y+b]_t$ given the shares of parties in $\mathcal{C}orr'$. The only difference is that in $\mathbf{Hyb}_3$, we first randomly sample $[a]_t, [b]_t$ and then compute $[x+a]_t, [y+b]_t$ while in $\mathbf{Hyb}_4$, we switch the order. The distributions of $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ are identical. Note that $[a]_t, [b]_t$ are no longer used in the computation phase (except the shares of parties in $\mathcal{C}orr'$). We do not generate the full sharings of $[a]_t, [b]_t$.

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

$\mathbf{Hyb}_6$: In this hybrid, in the output phase, $\mathcal{S}$ computes the function output based on the extracted inputs of corrupted parties and the set $\mathcal{D}$. By the correctness of the protocol, the function output y_i of P_i is identical to the secret of $[y_i]_t$ computed following the protocol. Then $\mathcal{S}$ determines the shares of $[y_i]_t$ of honest parties by the shares of parties in $\mathcal{C}orr'$ and the secret y_i. The distribution of $\mathbf{Hyb}_6$ is identical to that of $\mathbf{Hyb}_5$.

$\mathbf{Hyb}_7$: 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 $\mathbf{Hyb}_6$.

$\mathbf{Hyb}_8$: In this hybrid, $\mathcal{S}$ provides the inputs of corrupted parties and $\mathcal{D}$ to $\mathcal{F}_{\mathsf{fs}}$ and uses the output received from $\mathcal{F}_{\mathsf{fs}}$. Since $\mathcal{F}_{\mathsf{fs}}$ computes the function in the same way as $\mathcal{S}$ does in $\mathbf{Hyb}_7$. The distributions of $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ are identical.

Since $\mathbf{Hyb}_8$ corresponds to the ideal world, $\Pi_{\mathtt{main}}$ securely computes $\mathcal{F}_{\mathsf{fs}}$.