Computationally Efficient Asynchronous MPC with Linear Communication and Low Additive Overhead

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

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

Abstract. We explore the setting of asynchronous multi-party computation (AMPC) with optimal resilience $n = 3t + 1$, and develop an efficient protocol that optimizes both communication and computation.

The recent work by Goyal, Liu-Zhang, and Song [Crypto' 24] was the first to achieve AMPC with amortized linear communication cost without using computationally heavy public-key cryptography. However, its $\mathcal{O}(n^{14})$ additive communication overhead renders it impractical for most real-world applications.

It is possible to reduce the communication overhead significantly by leveraging cryptographic tools such as homomorphic commitments, public-key cryptography, or zero-knowledge proofs; however, the corresponding AMPC protocols introduce computation overhead of $\Omega(nC)$ public-key cryptographic operations that become bottleneck as n grows. Overall, achieving AMPC with linear communication complexity, low additive communication overhead, and low computation overhead remains an open challenge.

In this work, we resolve this efficiency challenge by utilizing the random oracle model. By relying solely on computationally efficient and *lightweight* cryptographic primitives such as random oracle hash and symmetric-key cryptography, our protocol is not only efficient in terms of computation and communication overhead but also post-quantum secure. For a circuit with $|C|$ multiplication gates, our protocol achieves $\mathcal{O}(|C| \cdot n + D \cdot n^2 + n^4)$ communication complexity. In terms of computation, our protocol requires $\mathcal{O}(n|C|)$ lightweight cryptographic operations.

1 Introduction

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

The cryptographic literature has studied MPC for more than forty years and the last decade has seen tremendous progress towards making it practical. However, most existing MPC systems still rely on strong networking assumptions such as (bounded) synchrony that make their practicability questionable, especially for low-latency application scenarios.

In the *synchronous* model, messages are assumed to be delivered within a fixed time frame. In high-throughput low-latency application scenarios in the real world, we cannot set synchronous time bounds generously and instead unpredictable delays must be tolerated. This makes most existing synchronous MPC systems inadequate. While protocols designed in the *asynchronous* model are resilient to such delays, the current designs may not scale as n grows: Indeed, current asynchronous MPC (AMPC) protocols struggle to scale to hundreds of parties either due to their 1) huge communication complexity from the information-theoretic (IT) approach [BKR94, PCR10, PCR08, CP23, GLZS24] which does not rely on any cryptographic assumptions or 2) computational complexity from the usage of threshold cryptography for common coins, additive homomorphic encryption/commitments and/or non-interactive zero-knowledge proofs [CP15, Coh16, BKLZL20, HNP05, HNP08, CHLZ21]. Concretely, the best-known IT-secure AMPC with $t < n/3$ resilience protocols require $\mathcal{O}(nC + \kappa n^{14})$ [GLZS24] for a circuit with C gates and n parties, here $\mathcal{O}(\kappa n^{14})$ is an additive overhead of communication cost which is independent

of the circuit. Current computational AMPC protocols [LYK+19, SLL+24] rely on significant "heavyweight" public-key cryptography and/or non-interactive zero-knowledge proofs requiring $\Omega(nC)$ public-key cryptographic operations, which becomes the primary scalability bottleneck with increasing n.

Lightweight Cryptography. We aim to reduce the high computational cost of heavyweight cryptography-based MPC by developing an MPC protocol that relies exclusively on *lightweight cryptography*. The term lightweight cryptography, first introduced by Shoup and Smart [SS24], refers to computationally efficient cryptographic primitives such as hash functions and symmetric key encryption. These operations are at least 100-1000× faster than simple public-key cryptographic operations such as discrete-log exponentiations. Consequently, replacing heavyweight cryptography with lightweight alternatives can significantly improve the computation time of the AMPC protocol. At the same time, as the offered security is computational and not information-theoretic, communication complexity can be significantly smaller than IT-secure AMPC protocols.

Moreover, similar to IT-secure protocols, lightweight cryptographic tools such as `SHA3` (a cryptographic hash function) and `AES` (a symmetric key encryption algorithm) also provide post-quantum security. As quantum computing inches closer to practical realization, major companies like Apple have begun transitioning from traditional cryptographic schemes to post-quantum alternatives. Notably, lightweight cryptography inherently offers post-quantum security at no additional cost.

However, a key limitation of lightweight cryptographic tools is that they do not naturally support homomorphic transcript aggregation, making it challenging to match the communication efficiency of heavyweight cryptography-based MPC. Despite this, specific prior works [BBB+24, DDL+24] have shown that the performance benefits of computational efficiency often outweigh the increase in communication complexity—provided the increase remains justifiable. For instance, HashRand [BBB+24], an asynchronous random beacon protocol based on hash functions, achieved a 5× increase in throughput compared to Spurt, a discrete-log-based beacon protocol, at $n = 136$ parties despite having an $\mathcal{O}(n)$ factor higher communication and hash computation complexity, mainly because of better computational efficiency.

However, recent attempts at building an MPC from lightweight cryptography, such as Momose [Mom24] give up a lot of communication in exchange for computational complexity. For instance, their protocol requires at least $\mathcal{O}(n^2C)$ communication complexity for computing a circuit of size C while achieving Guaranteed Output Delivery (GOD). They also have a high concrete overhead of $\mathcal{O}(\kappa n^6)$ bits of communication, which is at least $\mathcal{O}(n^3)$ factor higher than primitives based on heavyweight cryptography and makes their protocol too expensive in communication for practical use [CP15, SLL+24][6]. We aim to address these limitations and strike a balanced middle ground between the communication complexity of information-theoretic MPC and the computational overhead of heavyweight cryptography-based MPC.

1.1 Our Contributions

In this work, we consider the setting of AMPC with linear communication and optimal resilience $t < n/3$ [BKR94, ADS20] active corruptions. Given the above considerations, and in particular, the fact that the only linear protocol has an $\mathcal{O}(\kappa n^{14})$ additive overhead, we ask whether lightweight cryptography can be used to achieve AMPC with linear communication complexity and low additive overhead:

Can we achieve optimally resilient AMPC with linear communication and a low additive overhead from lightweight cryptography?

We answer in the affirmative by presenting a scalable AMPC protocol that combines the computational efficiency of IT-secure protocols with practical communication overhead, without compromising post-quantum security. More precisely, we achieve communication complexity of $\mathcal{O}(n)$ elements per multiplication, and $\mathcal{O}(n^4)$ elements of additive overhead.

Theorem 1. *Let $n = 3t + 1$ and κ denote the security parameter. Further let $\mathbb{F}$ be a finite field of size $2^{\Omega(\kappa)}$ and C be a circuit of size $|C|$ and depth D. Assuming random oracles, there is a fully malicious*

[6] We note that this is different from the claimed efficiency in [Mom24], which claims linear cost in the worst-case and $\mathcal{O}(\kappa n^5)$ additive overhead. But after private communication with the author, it was acknowledged that the cost is quadratic in the worst case and linear in the optimistic case. In Section 2.1, we analyze this cost in detail and mainly demonstrate the difficulties we need to address in this setting to achieve linear communication with $\mathcal{O}(\kappa n^4)$ overhead.

asynchronous MPC protocol computing the circuit that is secure against at most t corrupted parties with guaranteed output delivery. The achieved communication complexity is $\mathcal{O}(|C| \cdot n + D \cdot n^2 + n^4)$ *field elements and round complexity is* $\mathcal{O}(D + n)$.

Computational Efficiency. On top of communication efficiency, our protocols are computationally efficient. Our AMPC with GOD protocol requires $\mathcal{O}(n|C|)$ lightweight cryptographic operations overall. In comparison, protocols based on homomorphic public-key cryptography like [CP15, AJM+23] require $\Omega(n|C|)$ public-key cryptographic operations per party, where each such operation is 100× to 1000× more expensive than a lightweight cryptographic operation.

1.2 Related Work

The communication complexity in AMPC has been the subject of a very significant line of work.
Information-Theoretic MPC. In the IT setting, the first protocol with optimal resilience $t < n/3$ was provided by Ben-Or, Kelmer, and Rabin [BKR94]. The works [PCR10, PCR08] achieved $\mathcal{O}(n^5)$ field elements per multiplication, which was further improved in [CP23] to $\mathcal{O}(n^4)$. The works [GLZS24, JLS24] recently improved the scope to $\mathcal{O}(n)$ elements, but the additive overhead is $\Omega(n^{14})$ elements, making it impractical. In the case of $t < n/4$ and perfect security, the recent work [AAPP24] achieves linear communication $\mathcal{O}(n)$ elements per multiplication with an additive overhead of $\mathcal{O}(n^5)$ elements.
Cryptographic MPC. Several communication-efficient protocols with optimal resilience $t < n/3$ under different assumptions have been proposed. However, most of these works make use of heavy cryptography, typically in the form of threshold (somewhat homomorphic) encryption and/or non-interactive zero-knowledge proofs, and/or homomorphic commitments, which considerably increases the computational overhead. Our protocols only make use of hash functions, which are orders of magnitude faster.

The works [HNP05, HNP08, CHLZ21] make use of an additive homomorphic encryption, with [HNP08, CHLZ21] communicating $\mathcal{O}(n^2)$ elements per multiplication. The recent work Dumbo-MPC [SLL+24] improves upon this work and presents a protocol based on homomorphic commitments that communicates $\mathcal{O}(n^2)$ elements per multiplication in the worst-case, and $\mathcal{O}(n)$ elements in the optimistic case when the network is synchronous and all parties are honest. The work [CP15] achieves $\mathcal{O}(n)$ elements per multiplication at the cost of using somewhat-homomorphic encryption. The work [CHLZ21] also achieves linear cost using additive-homomorphic encryption for $t < (1 - \epsilon)n/3$, but considers the atomic-send model. The works [Coh16, BKLZL20] achieve a communication independent of the circuit size using fully-homomorphic encryption.

The works [LYK+19, DGKN09] introduce AMPC protocols where the preprocessing phase may not terminate, i.e. they are not live. This undesirable condition is more critical than the standard security with abort, as in the latter case, parties realize that the protocol failed (they obtain $\perp$ as output).

There also exist works that use homomorphic commitments to achieve ACSS with linear communication [AJM+23] per secret. However, this protocol requires $\mathcal{O}(n)$ discrete-log operations per secret, which is a scalability bottleneck.
Lightweight Protocols. Some protocols based on lightweight cryptography have appeared for concrete functionalities, including asynchronous distributed random beacons [BBB+24], asynchronous common subset [DDL+24], and asynchronous verifiable secret sharing [BKP11, SS24]. The work [BKP11] introduces an ACSS protocol with $\mathcal{O}(n^3 L)$ communication for sharing L secrets. [SS24] improves this complexity to $\mathcal{O}(nL + \kappa n^2 \log(n))$ bits when the dealer is honest using Hash-based Zero-Knowledge proofs and Pseudorandom functions (PRFs). However, a malicious dealer can increase communication to $\mathcal{O}(n^2 L + \kappa n^3)$ bits. This ACSS is used in a recent work [Mom24], which introduces an AMPC with $\mathcal{O}(n^2)$ elements per multiplication in the worst-case, and $\mathcal{O}(n)$ elements in the optimistic case.

2 Technical Overview

In the following, we use $[s]_d$ to denote a degree-d Shamir secret sharing of s and $\alpha_0, \ldots, \alpha_n$ to denote distinct field elements. Here, we denote the security parameter by κ and require the field size to be $2^{\Omega(\kappa)}$.

Following [CP23, GLZS24], an asynchronous MPC (AMPC) can be obtained in three steps. The first step is to realize an *asynchronous complete secret sharing* (ACSS) [PCR09] protocol which ensures that all honest parties can obtain their shares of a degree-t Shamir sharing $[s]_t$ distributed by a dealer. Then,

Table 1: Comparison of relevant Asynchronous MPC protocols.

Protocol	Computation Complexity	Communication		Cryptographic Assumption
		Cost Per Multiplication	Additive Overhead	
[SLL$^+$24]*	$\mathcal{O}(n^2\|C\|)$ DLE	$\mathcal{O}(n^2)$	$\mathcal{O}(\kappa n^3)$	DLog + q-SDH
[Mom24]	$\mathcal{O}(n^2\|C\|)$ LWC	$\mathcal{O}(n^2)$	$\mathcal{O}(\kappa n^6)$	RO
[GLZS24]	$\mathcal{O}(n\|C\|)$ FA	$\mathcal{O}(n)$	$\mathcal{O}(\kappa^2 n^{14})$	None
Ours	$\mathcal{O}(n\|C\|)$ LWC	$\mathcal{O}(n)$	$\mathcal{O}(\kappa n^4)$	RO

Computation complexity: The computation cost of our protocols is composed of $\mathcal{O}(nC)$ lightweight cryptographic (LWC) operations like cryptographic Hash functions and Pseudorandom number generation. In comparison, each discrete-log exponentiation (DLE) used by [SLL$^+$24] is 100× more expensive than an LWC operation. [GLZS24] uses information-theoretic cryptography whose computation costs involve $\mathcal{O}(\kappa)$-sized finite field arithmetic (FA) operations, which are 10× cheaper than LWC operations. **Communication**: We compare the number of field elements in the amortized per-gate communication cost and the overall additive overhead. *The work [SLL$^+$24] is only an offline phase protocol. We describe its costs here by pairing its offline phase with HoneyBadgerMPC's [LYK$^+$19] online phase. **Cryptographic Assumption**: Our work and [Mom24] are based in the Random Oracle (RO) model, and [GLZS24] is IT-secure. In contrast, [SLL$^+$24] utilizes Discrete-Log (DLog) and q-Strong Diffie Hellman (q-SDH) hardness assumptions.

the second step is to prepare Beaver triples [Bea92] with the help of ACSS in the offline phase. After preparing a sufficient number of Beaver triples, all parties only need to do public reconstruction in the online phase, which can be achieved with linear communication complexity and high concrete efficiency.

In the information-theoretic setting, the recent two works [JLS24] and [GLZS24] have addressed the first two steps respectively with linear communication complexity, thus yielding a full asynchronous MPC protocol with linear communication complexity. However, both works incur a large (circuit-size independent) communication overhead, making the final asynchronous MPC protocol not practical at all. Concretely, the ACSS protocol in [JLS24] incurs an additive overhead of $\mathcal{O}(n^{12})$ and the triple generation protocol in [GLZS24] incurs another additive overhead of $\mathcal{O}(n^7)$ (regardless of the costs of ACSS).

In this work, we aim to improve the concrete efficiency of asynchronous MPC by using computationally efficient cryptographic tools such as pseudorandom number generators (PRGs), symmetric-key encryptions, and random oracles. A recent prior work by Shoup and Smart [SS24] uses these tools to construct an ACSS protocol with linear communication in the *optimistic* case (where all parties follow the protocol) where the additive overhead is as small as $\mathcal{O}(n^2 \log n)$. Building upon [SS24], Momose [Mom24] constructs an asynchronous MPC assuming random oracles with linear communication in the optimistic case where the additive overhead is $\mathcal{O}(n^5)$. However both works would fall back to quadratic communication in the pessimistic case as we will discuss later.

We note that the quadratic fallback of [Mom24] is due to the fallback procedure of the ACSS protocol in [SS24]. Thus, to achieve linear communication, it is sufficient to use an ACSS protocol with linear communication in [Mom24]. However, to the best of our knowledge, known solutions for ACSS with linear communication either incurs a large additive overhead [JLS24] or requires computationally expensive cryptographic primitives [AJM$^+$23] such as the homomorphic KZG polynomial commitment scheme.

Our contribution lies on two aspects. First, building upon [Mom24], we show how to achieve overall linear communication while maintaining the $\mathcal{O}(n^5)$ overhead. Then, we show how to further reduce the additive communication overhead from $\mathcal{O}(n^5)$ to $\mathcal{O}(n^4)$. In the following, we start by recapping the techniques in [Mom24] and then introduce our new solutions.

2.1 Overviews of Previous Works

The construction in [Mom24] follows the general paradigm we mentioned above: Parties first prepare random Beaver triples in the offline phase, and then use them to compute the circuit in the online phase.

Sketch of Triple Generation in [Mom24]. For the preparation of triples, [Mom24] extends the DN technique [DN07] and party elimination framework [HMP00] to the asynchronous communication setting. To generate a random Beaver triple, all parties run the following steps.

1. All parties prepare random Shamir sharings $([a]_t, [b]_t)$ and random double sharings $([r]_t, [r]_{2t})$.
2. All parties compute their shares of $[z]_{2t} = [a]_t \cdot [b]_t + [r]_{2t}$ and together reconstruct the secret z.
3. Each party locally computes his share of $[c]_t = z - [r]_t$ and terminates with his share of $([a]_t, [b]_t, [c]_t)$.

After preparing a sufficient number of (possibly incorrect) random Beaver triples, all parties verify the correctness of the triples. If the verification passes, all parties move on to the online phase. Otherwise, all parties will together identify a corrupted party who deviates from the protocol.

To achieve GOD, the party elimination framework [HMP00] is used: For a parameter L, the triple generation process is divided into $|C|/L$ segments, where $|C|$ is the total number of Beaver triples required in the online phase. In each segment, all parties run the following steps to prepare L random Beaver triples. If a corrupted party is identified, all parties eliminate this party and rerun this segment. Since there are at most t corrupted parties, the number of rerun is bounded by t. By carefully choosing the parameter L, it eventually leads to a linear communication per triple.

Technical Difficulties in [Mom24]. To make the above idea work, there are two main challenges.

- Prepare degree-t and degree-$2t$ Shamir sharings with linear cost per sharing.
- Detect a corrupted party when the verification fails.

We start by introducing their techniques in handling the second challenge as it would eventually lead to a solution for the first challenge as well.

In the synchronous setting, the second challenge can be resolved by letting every party send his view of the generation process to each other. Then each party, after receiving the views from all parties, can find out the corrupted party that deviates from the protocol. However, in the asynchronous setting, an honest party can only expect to receive $n - t = 2t + 1$ parties' views since corrupted parties may never respond. With only $2t + 1$ parties' views, it may be insufficient to identify the corrupted party. To address this issue, the authors in [SS24] and subsequently the author in [Mom24] make use of a functionality $\mathcal{F}_{\mathsf{PrivSend}}$ which allows the sender to verifiably send his message to the receiver. To be more concrete, $\mathcal{F}_{\mathsf{PrivSend}}$ ensures that all parties acknowledge that the sender has dispersed the message among all parties (named `DispersePhase`), and when needed, all parties can recover this message to any receiver they agreed (named `RevealPhase`). We refer the readers to Appendix C for the realization of this functionality. With the help of this functionality, when a sender D wants to send a message to a receiver R, all parties can jointly invoke an instance of $\mathcal{F}_{\mathsf{PrivSend}}$. Once they terminate the `RevealPhase` of $\mathcal{F}_{\mathsf{PrivSend}}$, they can guarantee the delivery of this message from D to R in the `RevealPhase`.

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

$\mathcal{F}_{\mathsf{PrivSend}}$ proceeds as follows, running with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, a dealer D, and an adversary $\mathcal{S}$.

Disperse Phase

1: Upon receiving a message m from D, for the first time of such a request, the trusted party records m.

2: Upon receiving the request $(\mathsf{Init}, \mathsf{PrivSend}, D)$ from an honest party, the functionality forwards it together with the identity of this party to $\mathcal{S}$. When the functionality receives this request from $2t + 1$ distinct parties and the message m from D, the functionality sends a request-based delayed output $(\mathsf{Dispersed}, \mathsf{PrivSend})$ to all parties.

Reveal Phase

3: Upon receiving the request $(\mathsf{Deliver}, R)$ from an honest party, the functionality forwards this request together with the identity of this honest party to $\mathcal{S}$.
- If R is corrupted, the functionality reveal m to R.

When it receives this request from $t+1$ distinct parties, if it has sent $(\mathsf{Dispersed}, \mathsf{PrivSend})$ to all parties, the trusted party sends a request-based delayed output m to R.

Now in the triple generation phase, every point-to-point message is sent via $\mathcal{F}_{\mathsf{PrivSend}}$. Then when the verification fails, all exchanged messages will be reconstructed to all parties so that they will obtain the full view of the triple generation process. This addresses the second challenge.

Coming back to the first challenge, the solution is to let each party act as the dealer and distribute random degree-t and degree-$2t$ Shamir sharings. Then they agree on a set of $n - t = 2t + 1$ successful dealers and extract random sharings following the techniques in [DN07]. For degree-t Shamir sharings, [Mom24] uses the ACSS protocol in [SS24]. For degree-$2t$ Shamir sharings, the dealer uses $\mathcal{F}_{\mathsf{PrivSend}}$ to send the shares to all parties. As shown in [Mom24], there is no need to ensure that the dealer distributes a valid degree-$2t$ Shamir sharings. Instead, any deviation of the protocol would be caught in the triple verification with overwhelming probability.

Communication Overhead of [Mom24]. During the triple generation phase, to prepare L random Beaver triples, all parties need to communicate $\mathcal{O}(Ln+n^4)$ field elements if the triple verification passes. Here the term $\mathcal{O}(n^4)$ comes from $\mathcal{O}(n^2)$ instances of $\mathcal{F}_{\mathsf{PrivSend}}$, one instance for each pair of parties, and each instance of $\mathcal{F}_{\mathsf{PrivSend}}$ needs $\mathcal{O}(|m| + n^2)$ to send an m-bit message. If the triple verification fails, the communication becomes $\mathcal{O}(Ln^2+n^4)$. This is because all parties need to reconstruct the entire view of the generation phase to all parties via $\mathcal{F}_{\mathsf{PrivSend}}$, and $\mathcal{F}_{\mathsf{PrivSend}}$ needs $\mathcal{O}(|m|n+n^2)$ field elements communication to reconstruct the message m to all parties. Since at most t segments fail, to prepare $|C|$ random Beaver triples, the communication is $|C|/L \cdot \mathcal{O}(Ln + n^4) + t \cdot \mathcal{O}(Ln^2 + n^4) = \mathcal{O}(|C|n + |C|n^4/L + Ln^3 + n^5)$. To ensure the second term is bounded by $|C|n$, we have $L \geq n^3$ and thus the communication complexity of [Mom24] is $\mathcal{O}(|C|n + n^6)$.

However, the above analysis assumes the underlying ACSS protocol from [SS24] achieves linear communication per sharing and does not consider the fallback procedure in the pessimistic case. We note that the ACSS protocol in [SS24] only achieves linear communication in the optimistic case, but requires quadratic communication in the pessimistic case. To be more concrete, the ACSS protocol in [SS24] guarantees that each party P_i either obtains his correct share or a proof against a corrupted dealer D. In the latter case, to recover P_i's share, P_i first sends the proof to all other parties. Then upon accepting the proof, all parties invoke the fallback procedure which reconstructs the whole degree-t Shamir sharing to P_i to let him recover his share. In the worst case, $\mathcal{O}(n)$ parties may trigger the fallback procedure, resulting in $\mathcal{O}(n^2)$ communication per sharing.

Unfortunately, the quadratic fallback is also inherited by the asynchronous MPC in [Mom24]. At a first glance, one may try to resolve this issue by using the party elimination framework: when P_i sends the proof to all parties, instead of reconstructing the whole sharings to P_i, all parties can just eliminate the dealer D and discard the sharings distributed by D. However, this idea only works in the synchronous network where a party without receiving his share will be noticed immediately. In the asynchronous network setting, due to the unknown network latency, parties may not receive the proof from P_i on time. Consider the scenario where the fallback process is triggered only after all $|C|$ random Beaver triples have been generated. Discarding the sharings generated by D would make all $|C|$ random Beaver triples invalid.

In summary, the asynchronous MPC in [Mom24] achieves $\mathcal{O}(|C|n + Dn^2 + n^6)$ (the Dn^2 term comes from the online evaluation phase) in the optimistic case where the fallback procedures of the ACSS protocol are not triggered, and $\mathcal{O}(|C|n^2 + n^6)$ in the pessimistic case.

In this work, we still follow the paradigm of the construction in [Mom24], where all parties prepare Beaver triples in the offline phase and evaluate the circuit in the online phase. To avoid the quadratic multiplicative overhead in the worst case, parties will not trigger the fallback procedure of the ACSS protocol even if some honest parties do not obtain their shares. However, the missing shares from honest parties may result in failures when applying the online error correction algorithm in the online phase. To make this idea work, in section 2.2, we introduce new techniques to guarantee the success of circuit evaluation with linear communication cost in the online phase. Then we show how to further reduce the additive overhead of the whole construction to $\mathcal{O}(n^4)$ in section 2.3.

2.2 Achieve Linear Cost in the Pessimistic Case

High-Level Idea. In the online phase, with random Beaver triples prepared, all parties can evaluate addition gates locally and evaluate multiplication gates by only reconstructing degree-t Shamir sharings. For a multiplication gate with input sharings $[x]_t, [y]_t$ with a random Beaver triple $([a]_t, [b]_t, [c]_t)$, all

parties locally compute $[x+a]_t, [y+b]_t$ and publicly reconstruct $x+a, y+b$. Then the multiplication result can be locally computed by $[z]_t := (x+a)(y+b) - (x+a)[b]_t - (y+b)[a]_t + [c]_t$.

It is known that if all honest parties have their shares of a degree-t Shamir sharing, the public reconstruction is guaranteed to succeed. However, the optimistic path of [SS24] with linear communication does not guarantee that every honest party obtains his shares from the dealer. The pessimistic path, while ensuring shares of honest parties, requires quadratic communication per sharing.

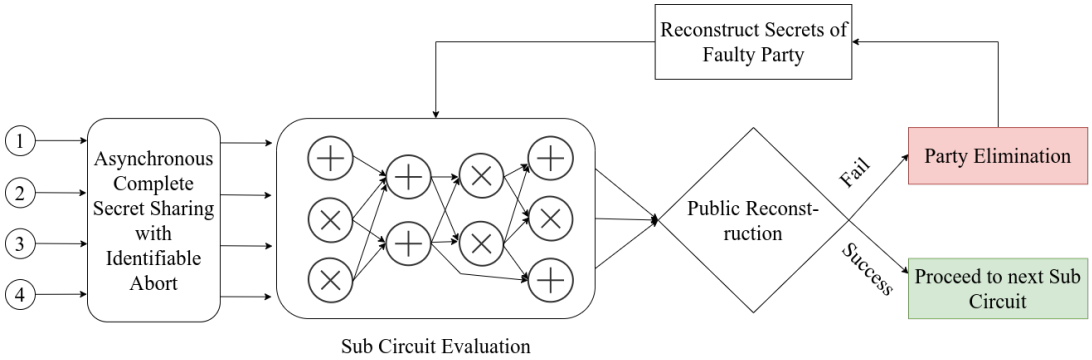


Fig. 1: **Online Phase**: We achieve linear communication in our online phase by splitting our circuit into $\mathcal{O}(n)$ smaller sub circuits. Parties first share their inputs using ACSS-Id protocol. Then, they evaluate the sub-circuit using Beaver triples generated during the preprocessing phase. At the end of sub-circuit evaluation, parties publicly reconstruct the output gates. In case the reconstruction fails and parties output $\perp$ because of a faulty party P_i, all parties participate in party elimination. Here, they reconstruct P_i's secrets with linear communication and re-evaluate the sub circuit by replacing P_i's shares with the secrets.

To achieve linear communication, our idea is to avoid the fallback procedure in [SS24]. After all parties accept the proof from P_i, instead of recovering P_i's shares, all parties publicly reconstruct the secrets of the corrupted dealer $D = P_j$. As we will show later, such a step can be done efficiently with linear communication per secret/sharing. However this still does not help us since P_i cannot obtain his shares from the secrets. To make progress, we borrow an important observation in [BSFO12, GSZ20]: for each sharing $[s]_t \in \{[x+a]_t, [y+b]_t\}$ to be reconstructed, $[s]_t$ can be represented by $[s]_t = \sum_{i=1}^{n} [s_i]_t$ where $[s_i]_t$ is a linear combination of the degree-t Shamir sharings distributed by P_i through ACSS. Now with the publicly reconstructed secrets distributed by the corrupted dealer $D = P_j$, all parties can compute s_j in clear. Then they will replace their shares of $[s_j]_t$ by s_j and recompute their shares of $[s]_t$. In this way, P_i can compute his share of $[s]_t$.

To make this idea work, we construct a weaker variant of the ACSS protocol, which we refer to as ACSS-Id, with the following guarantee: (1) Each party P_i either receives his correct share from the dealer D or a proof against the corrupted dealer D; (2) Upon receiving $t+1$ parties' requests, all parties can together reconstruct the secrets of all degree-t Shamir sharings distributed by D with linear communication. We point out that the optimistic path in [SS24] has already achieved the first property but not the second property, which is the key idea towards achieving linear communication in our construction. Jumping ahead, we show how to achieve linear communication in the online phase assuming ACSS-Id.

Party-Elimination Based Public Reconstruction. In the online phase, all parties perform the public reconstruction of $[s]_t = \sum_{i=1}^{n} [s_i]_t$ as follows, where each $[s_i]_t$ is a linear combination of degree-t Shamir sharings distributed by P_i through ACSS-Id.

Step 1: Check Degree-t Shares. For $i \in [n]$, each party checks whether he holds his share of $[s_i]_t$. If true, he computes his share of $[s]_t$ and broadcasts it. Otherwise, he broadcasts the proof against the corrupted P_i.

Step 2: Public Reconstruction. Each party waits to receive messages from others:

- When receiving enough shares of $[s]_t$ and reconstructing the secret s by online error correction, he sets the reconstruction result as s.

- When receiving a valid proof against some party P_i, he sets the reconstruction result as $\perp$.

Step 3: Agreement on Public Reconstruction Result. All parties run an agreement protocol for the reconstruction result. If the agreement result is not $\perp$, all parties output the result and terminate. Otherwise, all parties continue to agree on a corrupted dealer P_i.

Step 4: Public Reconstruction of Corrupted Dealer's Secrets. In case a corrupted dealer P_i is identified, all parties reconstruct the secrets shared by P_i, compute s_i by the linear combination of P_i's secrets, and replace their shares of $[s_i]_t$ by s_i.

For completeness, note that whenever a corrupted dealer P_i is identified, all parties will replace $[s_i]_t$ by the constant value s_i. This ensures that the above procedure will not fail due to P_i again. Thus, all parties can eventually reconstruct the secret s by repeating the above four steps and removing corrupted dealers. To achieve linear communication, we replace the public reconstruction in Step 1 and Step 2 by the efficient public reconstruction protocol in [DN07], and apply the party-elimination framework where the online phase is divided into $\mathcal{O}(n)$ segments of equal size. We refer the readers to Section 5 for more details.

Realization of ACSS-Id. As we mentioned above, the optimistic path in [SS24] has already achieved the first property required by ACSS-Id. We show how to upgrade the construction in [SS24] to also achieve the second property.

We first note that the optimistic path in [SS24] is also an asynchronous verifiable secret sharing protocol, which allows all parties to reconstruct the whole sharings to a receiver with linear communication. Indeed, the fallback procedure just uses this property to let a party P_i recover his share. To achieve efficient public reconstruction, we combine the ACSS protocol in [SS24] with the efficient public reconstruction technique [DN07].

Suppose the dealer D wants to share $t+1$ degree-t Shamir sharings, denoted by $[s_0]_t, \ldots, [s_t]_t$. The dealer first constructs a degree-t polynomial $[f(x)]_t = [s_0]_t + [s_1]_t \cdot x + \cdots + [s_t]_t \cdot x^t$. Then the dealer uses an instance of the ACSS protocol in [SS24] to share $[f(\alpha_i)]_t$ for all $i \in [n]$. On one hand, parties with their shares of $\{[f(\alpha_i)]_t\}_{i=1}^n$ can locally compute their shares of $\{[s_i]_t\}_{i=0}^t$. On the other hand, to publicly reconstruct the secrets shared by D, we first reconstruct $f(\alpha_i)$ to P_i. By the property of the ACSS protocol in [SS24], this step can be achieved with linear communication per sharing. Then each party P_i sends $f(\alpha_i)$ to all other parties. Relying on online error correction, all parties can recover $s_0, s_1, \ldots, s_n$. Note that the above construction only uses the optimistic path in [SS24].

In Section 4, we give the concrete construction and prove its security. Apart from the above description, our construction does not use the ACSS protocol in [SS24] directly. Instead, we use a similar construction with $\mathcal{O}(n^3)$ overhead, which is worse than $\mathcal{O}(n^2 \log n)$ achieved in [SS24], but the construction itself is simpler, and this result is sufficient for our purpose.

Efficiently Agree on the Corrupted Party. One issue we omitted in the above sketch is the cost of agreeing on a single corrupted party when the public reconstruction fails. To prove a dealer D is corrupted, a party P_i needs to provide all his shares from D as the proof against D. In the worst case where $\mathcal{O}(n)$ parties accuse the same dealer D, we need to pay quadratic communication just for sending the proofs.

To address this issue, the idea is to let all parties first agree on a single party P_i who wants to accuse some dealer D. Then all parties only open P_i's shares (Both [SS24] and our construction allow parties to verifiably reconstruct the shares P_i received from D). To be more concrete, all parties do the following in each segment.

1. At the beginning of this segment, each party reliably broadcasts the identity of the corrupted dealer he wants to accuse, and all parties will accept his accusation if they terminate the sharing phase of ACSS-Id invoked by this dealer.
2. If the public reconstruction succeeds, all parties move to the next segment.
3. Otherwise, all parties agree on a single party's accusation, say P_i accusing D, and only open the shares D sends to P_i. If P_i's shares are incorrect, all parties reconstruct the secrets shared by D and re-evaluate the current segment. Otherwise, all parties mark P_i as a corrupted party, and re-evaluate the current segment.

In summary, with the above new techniques, we manage to achieve $\mathcal{O}(|C|n + Dn^2 + n^4)$ communication in the online phase. We refer the readers to Appendix F for the detailed cost analysis.

2.3 Optimizing Triple Generation

In this subsection, we show how to improve the triple generation phase and achieve $\mathcal{O}(|C|n + n^4)$ communication for generating $|C|$ random Beaver triples.

High-Level Idea. Recall that in [Mom24], to prepare L random Beaver triples, the communication complexity is $\mathcal{O}(Ln + n^4)$ if the triple verification passes, and $\mathcal{O}(Ln^2 + n^4)$ otherwise. To achieve our goal, we have to bring down the cost in both cases to $\mathcal{O}(Ln + n^3)$. This means that

- In each segment, we cannot afford to use $\mathcal{F}_{\mathsf{PrivSend}}$ to send point-to-point messages between every pair of parties since it would cost $\mathcal{O}(n^4)$ immediately.
- When the verification fails, we cannot afford to let every party receive the entire view of the triple generation process since it would cost at least $\mathcal{O}(Ln^2)$.

Our idea is to divide each segment further into n groups. For the i-th group, we ask P_i to act as P_{king} and take the lead. When using the DN technique to compute multiplications, P_{king} is responsible to reconstruct the degree-$2t$ sharing $[z]_{2t}$ by collecting shares from all parties and broadcasting the reconstruction results. The triple verification is also done for each group separately. Note that we no longer use $\mathcal{F}_{\mathsf{PrivSend}}$ for point-to-point messages.

Now when the triple verification fails for the i-th group, we ask P_i who acts as P_{king} for this group to find the corrupted party who deviates from the protocol description. For each multiplication gate, P_i has already received $[z]_{2t}$ from all parties. Thus, it is sufficient to reconstruct the random sharings $[a]_t, [b]_t, [r]_t, [r]_{2t}$ to P_i.

- For degree-t Shamir sharings, each party directly sends his shares and P_i may use online error correction to reconstruct the whole sharings.
- For the degree-$2t$ Shamir sharing $[r]_{2t}$, it is a linear combination of degree-$2t$ Shamir sharings dealt by each dealer. Thus, we will let all parties verifiably reconstruct the degree-$2t$ Shamir sharings used in this group dealt by each dealer to P_i, which is realized by a protocol $\Pi_{\texttt{Sh2t-Id}}$ introduced below. With all information in hand, P_i is able to find the corrupted party.

However, this is not the end of the story. Although P_i can localize a corrupted party P_j, it is not clear how P_i can convince others that P_j is indeed corrupted. To overcome this issue, our final construction makes use of the dispute control [BTH06]: A dispute pair of parties (P_i, P_j) satisfies that at least one of these two parties is corrupted. Now P_i simply broadcasts that he has conflict with P_j, and all parties take the dispute pair (P_i, P_j) as output.

Note that when P_{king} is honest, the above execution will eventually terminate. Thus in each segment, we can expect that $n - t = 2t + 1$ groups terminate.

- If the triple verification passes for at least $t + 1$ groups, then we obtain $\mathcal{O}(L)$ random Beaver triples from these $t + 1$ groups. All parties move on to the next segment.
- Otherwise, we obtain at least $t+1$ dispute pairs, and each pair is broadcast by a distinct P_{king}. These contain at least $(t + 1)/2 = \mathcal{O}(n)$ distinct dispute pairs since each dispute pair can only be counted twice. In this case, parties need to rerun the current segment. Since there are at most $\mathcal{O}(n^2)$ dispute pairs, the number of rerun is bounded by $\mathcal{O}(n)$.

Ideally, this would give us a solution with $\mathcal{O}(Ln + n^3)$ communication in each segment. To make the above idea work, it is crucial that the same dispute pair will not be identified multiple times by the same P_{king}.

In the following, we first discuss how to realize $\Pi_{\texttt{Sh2t-Id}}$ and then show how to avoid an honest P_{king} from finding the same dispute pair multiple times.

Preparation of Degree-$2t$ Sharings with Partial Reconstruction. We follow the idea in [Mom24] to prepare degree-$2t$ Shamir sharings. Recall that in [Mom24], the dealer uses n instances of $\mathcal{F}_{\mathsf{PrivSend}}$ to send the shares to all parties. As we have mentioned, to achieve $\mathcal{O}(|C|n + n^4)$ communication, we cannot afford n^2 instances of $\mathcal{F}_{\mathsf{PrivSend}}$ in each segment. Thus, we will let the dealer distribute a sufficient number of random degree-$2t$ Shamir sharings in one shot. Then these random degree-$2t$ Shamir sharings are divided into $\mathcal{O}(n^2)$ groups, where the group with label (j, i) is used in the i-th group of the j-th segment.

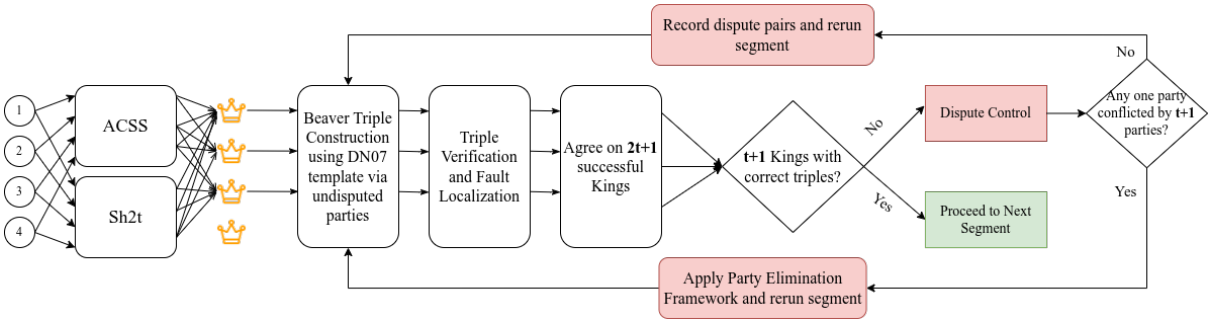


Fig. 2: **Beaver Triple Generation Overview**: The flowchart describes a segment of our Beaver triple generation protocol. For simplicity, we assume that all paries will get their shares from ACSS and Sh2t, later we will replace it by ACSS-Id and Sh2t-Id in detailed construction. First, each party acts as dealer and invokes ACSS and Sh2t to distribute random degree-t and $2t$ sharings. Then, each party acts as a king and compiles a list of $2t+1$ dealers whose protocols it terminated. Each king then uses the technique in [DN07] to generate triples, and parties verify them using a triple verification process. If the verification fails, the parties cooperate with the king to locate the faulty party. A king succeeds if he either successfully generates correct triples or identifies a faulty party. Parties then agree on a set of $2t+1$ successful kings. The overall success of the segment depends on whether at least $t+1$ kings successfully generated triples. If not, parties enter the dispute control phase for this segment, where each king who did not generate triples successfully reports a dispute with a party. Each king records this dispute and reruns the segment while ignoring all messages from its disputed party. Furthermore, if a party conflicts with $t+1$ parties, then it is marked malicious and parties trigger the party elimination framework on it.

In the above sketch of preparing random Beaver triples, we need all parties to be able to reconstruct the degree-$2t$ Shamir sharings used in a group with label (j,i) to the party P_i. To this end, we let the dealer also broadcast a commitment for the shares sent to each party P_k in each group. In this way, P_k can simply send his shares to P_i and P_i can verify P_k's shares by checking the commitment broadcast by the dealer. Note that doing it naively requires each dealer to broadcast $\mathcal{O}(n^3)$ commitments, leading to $\mathcal{O}(n^4)$ communication per dealer, and $\mathcal{O}(n^5)$ communication in total. To reduce a factor of n, we let the dealer compute the hash on every n commitment, send the corresponding commitments to P_i for verification, and only broadcast these $\mathcal{O}(n^2)$ hash results. We refer the readers to Section 6.1 for more details.

Dispute Control. To apply the dispute control framework, all parties together maintain a set of identified disputed pairs. Note that a party who has conflict with $t+1$ parties is identified as a corrupted party, and an identified corrupted party automatically has conflict with every other party. Now for an honest P_{king}, we need to ensure that P_{king} would not identify the same pair of disputed parties multiple times (or otherwise, there is no guarantee on the number of rerun). Note that this is unlike the party elimination framework where the identified party is eliminated, and thus the same party would not be identified twice.

The idea is to avoid using the sharings or messages from parties that have conflict with P_{king}. To be more concrete, in the beginning of each segment, P_{king} broadcasts a set W of $2t+1$ parties who successfully distribute degree-t and degree-$2t$ random sharings and do not conflict with P_{king}. Then all parties will use the sharings generated by parties in W to extract random sharings $([a]_t, [b]_t, [r]_t, [r]_{2t})$ for this group. During the reconstruction of $[z]_{2t}$, P_{king} will only accept messages from parties that do not conflict with P_{king} (but may not in the set W chosen by P_{king} in the beginning of this segment). In this way, an honest P_{king} will always find a new dispute pair when the verification fails. We refer the readers to Section 6.2 for more details.

Final Solution for Triple Generation. We summarize the outline of our triple generation protocol. First, we divide the generation of triples into $\mathcal{O}(n)$ segments. Let $L = \mathcal{O}(C/n)$. Then in each segment, the goal is to prepare L random Beaver triples.

In the beginning, each party uses $\Pi_{\texttt{ACSS-Id}}$ and $\Pi_{\texttt{Sh2t-Id}}$ to distribute sufficient number of random degree-t and degree-$2t$ Shamir sharings. Then all parties do the following for each segment.

1. Each party acts as P_{king} and leads one group to prepare $\mathcal{O}(L/n)$ random Beaver triples. P_{king} first broadcasts a set W of $2t+1$ parties that have no conflict with P_{king}. Then all parties use the degree-t and degree-$2t$ random sharings dealt by parties in W to locally prepare $([a]_t, [b]_t, [r]_t, [r]_{2t})$ for each triple. Next, P_{king} helps reconstruct the degree-$2t$ Shamir sharings required by the DN protocol and the triple verification process. In particular, P_{king} only accepts messages from parties that have no conflict with him.
   - If the triple verification succeeds, all parties take their shares of the prepared Beaver triples as output and terminate.
   - Otherwise, all parties reconstruct $[a]_t, [b]_t, [r]_t$ and the degree-$2t$ Shamir sharings used in this group and dealt by parties in W to P_{king}. Then P_{king} identifies the corrupted party and announces a new dispute pair. Finally, all parties take the new dispute pair as output.
2. All parties agree on a set of successful kings (of size $n-t$).
3. After agreeing on such a set, if at least $t+1$ groups provide correct triples, all parties proceed to the next segment. Otherwise, all parties find at least $\mathcal{O}(n)$ new dispute pairs. They update the set of identified disputed pairs and rerun the current segment.

After at most $\mathcal{O}(n)$ rerun, all parties will terminate with $|C|$ random Beaver triples. The achieved communication complexity is $\mathcal{O}(|C|n + n^4)$.

For simplicity, the above sketch omits the problem that a party may not obtain his correct shares in $\Pi_{\texttt{ACSS-Id}}$ or $\Pi_{\texttt{Sh2t-Id}}$, but only a proof against the corrupted dealer. We address this problem using a similar approach to that in the online phase. We refer the readers to Section 6.2 for more details.

3 Preliminaries

3.1 Model

We consider protocols among a set $\mathcal{P}$ of n parties $P_1, \ldots, P_n$. For the security of our protocols, we use the UC framework introduced by Canetti [Can01], based on the *real and ideal world paradigm* [Can00]. Parties have access to a network of point-to-point asynchronous and secure channels (for details of the asynchronous network model, we refer the reader to [CR98]). Asynchronous channels guarantee *eventual* delivery, meaning that messages sent are eventually delivered, and the adversary does the scheduling of the messages. In particular, the adversary can arbitrarily (but finitely) delay all messages sent and deliver them out of order. We also consider the fully malicious adversary, that can completely control the behavior of corrupted parties.

Functionality of Asynchronous MPC. We use the functionality of AMPC in [CP23] as follows.

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

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

1: Upon receiving an input v from $P_i \in \mathcal{P}$, if $\mathsf{CoreSet}$ has not been recorded yet or if $P_i \in \mathsf{CoreSet}$, set $x^{(i)} = v$.
2: Upon receiving an input $\mathsf{CoreSet}$ from $\mathcal{S}$, verify that $\mathsf{CoreSet}$ is a subset of $\mathcal{P}$ of size at least $n-t$, else ignore the message. If $\mathsf{CoreSet}$ has not been recorded yet, then record $\mathsf{CoreSet}$ and for every $P_i \notin \mathsf{CoreSet}$, set $x^{(i)} = 0$.
3: If the $\mathsf{CoreSet}$ has been recorded and the value $x^{(i)}$ has been set to a value different from $\perp$ for every $P_i \in \mathsf{CoreSet}$, then compute $y = f(x^{(1)}, \ldots, x^{(n)})$ and generate a request-based delayed output $y^{(i)} = y$ for every $P_i \in \mathcal{P}$.

3.2 Distributed Zero-Knowledge Proof

The distributed zero-knowledge proof (DZK) allows a prover to prove the correctness of degree-d Shamir sharings that have been distributed to all parties, and each party can verify whether his shares are correct. In the random oracle model, the authors in [ABCP23, CCG25] give a construction of the DZK

protocol in the synchronous setting. In this work, we introduce how to apply their techniques to the asynchronous setting to realize the following functionality $\mathcal{F}_{\mathsf{dZK}}$. We refer the reader to Appendix B for the detailed construction.

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

Public Input: Let $(\alpha_1, \ldots, \alpha_n)$ denote n distinct field elements, N denote the number of input polynomials, $d \leq n-1$ denote the degree of input polynomials.
$\mathcal{F}_{\mathsf{dZK}}$ runs with parties $\mathcal{P} = \{V_1, \ldots, V_n\}$, a prover $P \in \mathcal{P}$, the set $\mathcal{C}orr$ of corrupted parties, and an adversary $\mathcal{S}$. The functionality initializes the set $\mathcal{P}_{\texttt{Vrfy}} = \mathcal{P}$. The set $\mathcal{P}_{\texttt{Vrfy}}$ includes the identities of parties whose shares can be publicly verified. Let $\mathcal{H} := \mathcal{P} \setminus \mathcal{C}orr$ denote the set of honest parties.

Distributing Proof Phase

1: Upon receiving degree-d polynomials $f_1(x), \ldots, f_N(x)$ and a set $\mathcal{P}'_{\texttt{Vrfy}} \subseteq [n]$ from prover P, the functionality records them.
   - If $P \in \mathcal{C}orr$, the functionality sets $\mathcal{P}_{\texttt{Vrfy}} \leftarrow \mathcal{P}'_{\texttt{Vrfy}}$.
   - Otherwise, the functionality sends $f_1(\alpha_i), \ldots, f_N(\alpha_i)$ for all corrupted $V_i \in \mathcal{C}orr$ to $\mathcal{S}$.

2: Upon receiving a request $(\mathsf{Init}, \mathsf{DZK}, P)$ from an honest party, the functionality forwards this request together with the identity of this honest party to $\mathcal{S}$. When it receives this request from $2t+1$ distinct parties, if the functionality has recorded the prover's degree-d polynomials, it sends a request-based delayed output $(\mathsf{Delivered}, \mathsf{DZK})$ to all parties.

Verification Phase

3: Upon receiving the request $(\texttt{PrivVerifyDZK}, j, R)$ from an honest party, the functionality forwards this request together with the identity of this honest party to $\mathcal{S}$.
   - If R is corrupted, the functionality also sends $(f_1(\alpha_j), \ldots, f_N(\alpha_j))$ to $\mathcal{S}$.

   Upon receiving the request $(\texttt{PrivVerifyDZK}, s_1, \ldots, s_N, j)$ from the verifier R, if R is honest, the functionality sends $(\texttt{PrivVerifyDZK}, j, R)$ and the identity of R to $\mathcal{S}$.

4: When the functionality receives $(\texttt{PrivVerifyDZK}, s_1, \ldots, s_N, j)$ from the verifier R and $(\texttt{PrivVerifyDZK}, j, R)$ from $t+1$ parties, it sends a request-based delayed output `true` to R if the following conditions hold and `false` otherwise.
   - (i) It has sent $(\mathsf{Delivered}, \mathsf{DZK})$ to all parties; (ii) $V_j \in \mathcal{P}_{\texttt{Vrfy}}$; (iii) $(s_1, \ldots, s_N) = (f_1(\alpha_j), \ldots, f_N(\alpha_j))$.

Transfer Proof Phase

5: Upon receiving the request $(\texttt{TransferDZK}, s_1, \ldots, s_N, V_j \to R)$ from V_j, it sends a request-based delayed output `true` to V_j if the following conditions hold and `false` otherwise.
   - (i) It has sent $(\mathsf{Delivered}, \mathsf{DZK})$ to all parties; (ii) $V_j \in \mathcal{P}_{\texttt{Vrfy}} \cap \mathcal{H}$ or $V_j \in \mathcal{C}orr$; (iii) $(s_1, \ldots, s_N) = (f_1(\alpha_j), \ldots, f_N(\alpha_j))$.

   If the functionality decides to send `true` to V_j, it also sends a request-based delayed output $f_1(\alpha_j), \ldots, f_N(\alpha_j)$ to R.

3.3 Building Blocks

Our construction makes use of the following building blocks, and we give the definitions of them in Appendix A.1. Assuming the random oracle model, all of them can be efficiently realized in the asynchronous setting against a malicious adversary and 1/3 corruptions.

- **Reliable Broadcast $\mathcal{F}_{\mathsf{rbc}}$.** It allows parties to agree on the value of a sender and can be realized with $\mathcal{O}(L \cdot n + n^2)$ elements communication complexity for broadcasting a message of size L elements [DXR21].
- **Byzantine Agreement $\mathcal{F}_{\mathsf{ba}}$.** It allows parties to agree on a common output and can be realized with $\mathcal{O}(L \cdot n + \kappa \cdot n^2 \log(n))$ elements communication complexity for the agreement of a message of size L elements [NRS+20].
- **Reliable Agreement $\mathcal{F}_{\mathsf{ra}}$.** It is the agreement version of the reliable broadcast where all parties have input. When all parties terminate it, at least $t+1$ of them provide matching input. It can be realized with $\mathcal{O}(L \cdot n^2)$ elements communication complexity for the agreement of a message of size L elements [DDL+24].
- **Agreement on a Common Set $\mathcal{F}_{\mathsf{acs}}$.** If allows parties to agree on a set of at least $n-t$ parties that satisfy a certain property. It can be realized with $\mathcal{O}(n^3)$ elements communication complexity [DDL+24].

– **Verifiable Private Send $\mathcal{F}_{\mathsf{PrivSend}}$.** It allows a sender to verifiably send a message to a receiver. For a message of size L elements, we give a construction with communication complexity $\mathcal{O}(L+n^2)$ elements in the `DispersePhase` and $\mathcal{O}(L+n^2)$ elements in each time of `RevealPhase`. Refer to Appendix C for more details.

4 Asynchronous Completing Secret sharing with Identified Abort

4.1 Functionality of ACSS-Id

We describe the ACSS with Identifiable Abort functionality in Section 4.1. Notice that the adversary $\mathcal{S}$ can change the output of any honest party to $(\texttt{Corrupt}, D)$ when the dealer D is corrupt. Furthermore, each party can request the functionality to open the view of a party P_i. On receiving $t+1$ such requests, the functionality changes outputs of parties to $(\texttt{Corrupt}, D)$ if the proof is valid, indicating that D is corrupt. If the proof is invalid, parties instead mark P_i as corrupt because he accused a correct dealer of being corrupt. Parties can also reconstruct the secrets distributed by D by sending a `Public-Recon` request.

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

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

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

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

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

3: Upon receiving request $(\texttt{Open-Proof}, P_i)$ from $t+1$ parties, do the following.
   – If $P_i \in \mathcal{P}_{\text{proof}}$, send a request-based delayed output $(\texttt{Corrupt}, D)$ to all parties.
   – If $P_i \in \mathcal{P}_{\mathcal{C}orr}$ and $D \notin \mathcal{P}_{\mathcal{C}orr}$, send a request-based delayed output $(\texttt{Corrupt}, P_i)$ to all parties.
   – Otherwise, send all views to $\mathcal{S}$ and give up security.

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

4.2 Construction

Protocol $\Pi_{\texttt{ACSS-Id}}$

Let $\alpha_0, \ldots, \alpha_n$ be distinct field elements, L be the number of secrets.

Distribution Phase

1: D possesses L degree-t polynomials $f_1(x), \ldots, f_L(x)$ as inputs, then he divides them into $t+1$ groups, each of size $L' = L/(t+1)$. Denote the degree-t polynomials in the k-th group as $f_1^{(k)}(x), \ldots, f_{L'}^{(k)}(x)$, where $k \in [t+1]$.

2: For each $\ell \in [L']$, we define a degree-(t,t) bivariate polynomial $g_\ell(x,y)$ as follows:

$$g_\ell(x,y) := f_\ell^{(1)}(x) + f_\ell^{(2)}(x) \cdot y + \cdots + f_\ell^{(t+1)}(x) \cdot y^t$$

For each $i \in [n]$, D sends $g_1(x,\alpha_i), \ldots, g_{L'}(x,\alpha_i)$ to an instance of $\mathcal{F}_{\mathsf{dZK}}$, denoted by $\mathcal{F}_{\mathsf{dZK}}^{(i)}$. All parties send request $(\mathsf{Init}, \mathsf{DZK}, D)$ to each $\mathcal{F}_{\mathsf{dZK}}^{(i)}$.

3: For $i \in [n]$, let $m_i = \{g_\ell(\alpha_i, y)\}_{\ell \in [L']}$. All parties jointly invoke an instance of $\mathcal{F}_{\mathsf{PrivSend}}$, denoted by $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$. D sends m_i to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$, all parties send request $(\mathsf{Init}, \mathsf{PrivSend}, D)$ to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$.

Verification Phase

1: Upon receiving $(\mathsf{Dispersed}, \mathsf{PrivSend})$ from $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$ and $(\mathsf{Delivered}, \mathsf{DZK})$ from $\mathcal{F}_{\mathsf{dZK}}^{(i)}$ for all $i \in [n]$, all parties proceed.

2: For each party P_i, all parties send request $(\mathsf{Deliver}, P_i)$ to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$. When P_i receives m_i from $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$, he parses it to $\{g_\ell(\alpha_i, y)\}_{\ell\in[L']}$ and check whether each $g_\ell(\alpha_i, y)$ is a degree-t polynomial. If true, he computes $f_\ell^{(1)}(\alpha_i), \ldots, f_\ell^{(t+1)}(\alpha_i)$ from the coefficients of $g_\ell(\alpha_i, y)$. Then he sends $(\texttt{PrivVerifyDZK}, \{g_\ell(\alpha_i, \alpha_j)\}_{\ell\in[L']}, i)$ to $\mathcal{F}_{\mathsf{dZK}}^{(j)}$ for each $j \in [n]$ and records the output of each $\mathcal{F}_{\mathsf{dZK}}^{(j)}$. All parties send request $(\texttt{PrivVerifyDZK}, i, P_i)$ to $\mathcal{F}_{\mathsf{dZK}}^{(j)}$.

3: If P_i gets $\{f_\ell^{(1)}(\alpha_i), \ldots, f_\ell^{(t+1)}(\alpha_i)\}_{\ell\in[L']}$ and terminates all $\mathcal{F}_{\mathsf{dZK}}^{(j)}$ with `true`, he accepts his shares and sets them as his output. Otherwise, he sets his output as $(\texttt{Corrupt}, D)$.

Termination Phase

1: All parties jointly invoke an instance of $\mathcal{F}_{\mathsf{ra}}$, if a party accepts his shares, he sets his input for $\mathcal{F}_{\mathsf{ra}}$ as 1. When all parties terminate $\mathcal{F}_{\mathsf{ra}}$ with 1, they terminate with their output.

In our ACSS-Id construction, the dealer D encodes $t+1$ degree-t sharings into a degree-(t, t) bivariate polynomial. D uses $\mathcal{F}_{\mathsf{PrivSend}}$ to send the ith row polynomial to each party. D and all parties then use $\mathcal{F}_{\mathsf{dZK}}$ to check if their received shares are valid. On receiving and validating their shares, each party P_i participates in an instance of Reliable Agreement $\mathcal{F}_{\mathsf{ra}}$ with input 1. Then, on terminating $\mathcal{F}_{\mathsf{ra}}$, parties terminate the sharing phase.

Protocol $\Pi_{\texttt{ACSS-Id}}$

Upon receiving a request $(\texttt{Open-Proof}, P_i)$ from the environment, if all parties have terminated the sharing phase, they do the following.

Accusation Phase

1: They send request $(\mathsf{Deliver}, P_k)$ to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$ for all $k \in [n]$.

2: Upon receiving m_i from $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$, all parties honestly follow the verification phase to do the check. If the verification result is wrong, all parties terminate with $(\texttt{Corrupt}, D)$ and $(\texttt{Corrupt}, P_i)$ otherwise.

Upon receiving a request `Public-Recon` from the environment, if all parties have terminated the sharing phase, they do the following.

Public Reconstruction Phase

1: Each party P_i who has set his input as 1 for $\mathcal{F}_{\mathsf{ra}}$ during the termination phase will send $(\texttt{TransferDZK}, \{g_\ell(\alpha_i, \alpha_j)\}_{\ell\in[L']}, P_i \to P_j)$ to $\mathcal{F}_{\mathsf{dZK}}^{(j)}$.

2: When P_i receives $\{g_\ell(\alpha_j, \alpha_i)\}_{\ell\in[L']}$ from $\mathcal{F}_{\mathsf{dZK}}^{(i)}$, he accepts these values. Upon accepts $\{g_\ell(\alpha_j, \alpha_i)\}_{\ell\in[L']}$ from $t+1$ distinct j, he reconstructs $\{g_\ell(\alpha_0, \alpha_i)\}_{\ell\in[L']}$ and sends them to all parties.

3: When P_i receives $\{g_\ell(\alpha_0, \alpha_j)\}_{\ell\in[L']}$ from P_j, he records it. When P_i succeeds in reconstructing $\{g_\ell(\alpha_0, y)\}_{\ell\in[L']}$ by using the online error correction, he outputs the coefficients of $\{g_\ell(\alpha_0, y)\}_{\ell\in[L']}$.

If a corrupt dealer D sends incorrect shares to a party P_i, then this party outputs $(\texttt{Corrupt}, D)$, and accuses D of being corrupt. Parties later react to the accusation by participating in the Accusation phase. Essentially, parties reconstruct their views to everyone using $\mathcal{F}_{\mathsf{PrivSend}}$. If any view is inconsistent with $\mathcal{F}_{\mathsf{dZK}}$ or is not a valid degree-t polynomial, parties mark D as corrupt. Otherwise, they mark the accuser as corrupt for wrongly accusing a correct dealer D. In either case, all parties mark one party as corrupt.

In case parties mark a dealer as corrupt, they aim to publicly reconstruct its secrets after receiving a `Public-Recon` message from the environment. Parties try to reconstruct the point $g_\ell(\alpha_0, \alpha_i)$ to party P_i. Each party P_j sends points on its row $g_\ell(\alpha_j, \alpha_i)$ for $\ell \in [1, \ldots, L']$ to P_i. Party P_i uses $\mathcal{F}_{\mathsf{dZK}}$ to verify each set of points and waits to accept $t+1$ points on the column $g_\ell(x, \alpha_i)$. Note that at least $t+1$ honest parties accepted their shares, so P_i eventually receives $t+1$ valid points and reconstructs its column and point $g_\ell(\alpha_i, \alpha_0)$. Then, parties use Online Error Correction to reconstruct polynomial $g_\ell(x, \alpha_0)$ and output its coefficients.

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

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

5 Party Elimination Based Public Reconstruction

In this section, we give the construction of *party elimination based public reconstruction* (refer to section 2.2) based on ACSS-Id. We start with the construction of two sub-protocols $\Pi_{\texttt{BatchPubRec}}$ and $\Pi_{\texttt{Agreement}}$.

For $\Pi_{\texttt{BatchPubRec}}$, we follow the technique in [DN07] to realize a public reconstruction with linear cost per secret. Compared to standard public reconstruction, $\Pi_{\texttt{BatchPubRec}}$ allows some parties not to terminate with a failure symbol $\perp$, but they will learn an accusation between a dealer and a party. The detailed construction is as follows.

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

For each party P_i:

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

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

(2). To reconstruct $f(\alpha_i)$, P_i waits to receive messages:

- Upon receiving new shares of $[f(\alpha_i)]_t$, P_i uses online error correction on all received shares of $[f(\alpha_i)]_t$ to reconstruct $f(\alpha_i)$. If it succeeds, P_i sends $f(\alpha_i)$ to all parties and moves to Step 1.(3). Otherwise, P_i keeps waiting for more messages.
- Upon receiving $(\texttt{Accusation}, P_k, \texttt{dealer}, \texttt{ACSS})$ from P_k's broadcast and when he has terminated $\mathcal{F}_{\texttt{ACSS-Id}}$ invoked by $\texttt{dealer}$, P_i records the identity of P_k and sets $\boldsymbol{m}_i = \perp$ and moves to Step 2.

(3). To reconstruct $f(X)$, P_i waits to receive messages from all parties:

- Upon receiving $f(\alpha_k)$ from P_k, P_i uses online error correction on all received $f(\alpha_k)$ to reconstruct $f(X)$. If it succeeds, P_i sets the k-th sub-vector $\boldsymbol{m}_{i,k}$ as $(s^{(0)}, \ldots, s^{(t+1)})$ (the coefficients of $f(x)$). Otherwise, P_i keeps waiting for more messages.
- Upon receiving $(\texttt{Accusation}, P_k, \texttt{dealer}, \texttt{ACSS})$ from P_k's broadcast and he has terminated $\mathcal{F}_{\texttt{ACSS-Id}}$ invoked by $\texttt{dealer}$, P_i records the identity of P_k and sets $\boldsymbol{m}_i = \perp$ and moves to Step 2.

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

When all parties terminate $\Pi_{\texttt{BatchPubRec}}$, they do the following to agree on the same output of $\Pi_{\texttt{BatchPubRec}}$, which is either the correct public reconstruction result or a failure symbol $\perp$. This is achieved by two instances of BA. For the first one, they use the output of $\Pi_{\texttt{BatchPubRec}}$ as inputs. Upon terminating, they check whether the agreement results are equal to their inputs. If true, they set input 1 for the second BA and 0 otherwise. This can ensure that if all parties terminate the second BA with 1, then the output of the first BA must be some honest party's input. That can prevent all parties from agreeing on an incorrect result chosen by the adversary. The detailed construction is as follows.

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

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

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

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

(1). For each party, if he has recorded the identity of a party P_k during $\Pi_{\texttt{BatchPubRec}}$, he reliably broadcasts the first one he has recorded.

(2). All parties set the property Q as follows, a party P_i will like P_j if P_i (1) receive an identity P_k from P_j, (2) receive $(\texttt{Accusation}, P_k, \texttt{dealer}, \texttt{ACSS})$ from P_k's broadcast, (3) terminate $\mathcal{F}_{\texttt{ACSS-Id}}$ invoked by $\texttt{dealer}$. Then, all parties jointly invoke $\mathcal{F}_{\texttt{acs}}$ with property Q to agree on a set $\mathcal{D}$ of size $n - t$.

(3). For the identity P_k broadcast by the first party in this set, all parties send $(\texttt{Open-Proof}, P_k)$ to $\mathcal{F}_{\texttt{ACSS-Id}}$ invoked by $\texttt{dealer}$ and terminate with the output of $\mathcal{F}_{\texttt{ACSS-Id}}$, which is $(\texttt{Corrupt}, \texttt{dealer})$ or $(\texttt{Corrupt}, P_k)$.

Sub-Circuit Evaluation. With the above two sub-protocols, we give the so-called *sub-circuit evaluation* protocol $\Pi_{\texttt{SubCktEval}}$, which is used in the online phase later. Recall that with the help of Beaver triples, all parties in the online phase only need to do public reconstruction. Then based on our idea of *party elimination based public reconstruction*, all parties can execute $\Pi_{\texttt{SubCktEval}}$ to evaluate a circuit with the help of Beaver triples, and they will either succeed in evaluating this circuit or agree on a corrupted party. For a circuit C' of depth D' with $|C'|$ multiplication gates and C'_O output gates, the communication complexity is $\mathcal{O}((C' + C'_O)n + D' \cdot n^2 + n^3)$ field elements.

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

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

If true, he moves to Step 2. Otherwise, he received $(\texttt{Corrupt}, \texttt{dealer})$ from $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ invoked by one $\texttt{dealer}$. Then he sets the output $\boldsymbol{m}_i = \perp$, reliably broadcasts $(\texttt{Accusation}, P_i, \texttt{dealer}, \texttt{ACSS})$, and moves to Step 4.

2: **Circuit Evaluation.**
From $k = 1$ to D', for the k-th layer in the circuit C':
- For every addition gate with input sharings $[x]_t, [y]_t$, locally compute $[z]_t = [x]_t + [y]_t$.
- Let L be the number of multiplication gates in the k-th layer. Suppose the input degree-t Shamir sharings are denoted by $([x_i]_t, [y_i]_t)_{i=1}^L$. Let $([a_i]_t, [b_i]_t, [c_i]_t)_{i=1}^L$ denote the random Beaver triples assigned to these L gates. Each party P_j executes $\Pi_{\texttt{BatchPubRec}}$ with his shares of $([x_i + a_i]_t, [y_i + b_i]_t)_{i=1}^L$, and gets output $\boldsymbol{m}_j^{(k)}$ after terminating $\Pi_{\texttt{BatchPubRec}}$.
  - If $\boldsymbol{m}_j^{(k)} = \perp$, P_j moves to Step 4.
  - Otherwise, for all $i \in [L]$, P_j parses $\boldsymbol{m}_j^{(k)}$ to get $\{x_i + a_i, y_i + b_i\}_{i=1}^L$ and locally compute

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

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

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

6 Triple Generation

In this section, we give the construction of a triple generation protocol.

6.1 Preparing Random degree-2t Shamir Sharing

We start with the functionality of $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ as follows. It divides the sharings received from the dealer into n^2 groups and allows all parties to open a partial of them to a fixed receiver. Similarly to $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$, some honest parties may terminate with $(\texttt{Corrupt}, D)$ when the dealer is corrupted, and they can jointly open an accusation and learn the identity of a corrupted party.

Functionality $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$

Public Input: $(\alpha_0, \ldots, \alpha_n), N$
$\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ runs with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, a dealer $D \in \mathcal{P}$, and an adversary $\mathcal{S}$.

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

2: Upon receiving N polynomials $q_1(\cdot), \ldots, q_N(\cdot)$ from D (when the dealer is honest, the degree should be $2t$), divide it into n^2 groups, each of size $N' = N/n^2$. Denote the polynomials in the k-th group as $q_1^{(k)}(\cdot), \ldots, q_{N'}^{(k)}(\cdot)$, send a requested-based delayed output $\{q_1^{(k)}(\alpha_i), \ldots, q_{N'}^{(k)}(\alpha_i)\}_{j\in[n],k\in[n]}$ to each party P_i.
   - Upon receiving a request $(\texttt{proof}, P_i)$ from $\mathcal{S}$, if $D \in \mathcal{P}_{\mathcal{C}orr}$ and the output of P_i has not been delivered, change the output of P_i to $(\texttt{Corrupt}, D)$ and add P_i in $\mathcal{P}_{\text{proof}}$. Otherwise, ignore this request.

3: Upon receiving request $(\texttt{Open-Proof}, P_i)$ from $t+1$ parties, do the following.
   - If $P_i \in \mathcal{P}_{\text{proof}}$, send a requested-based delayed output $(\texttt{Corrupt}, D)$ to all parties.
   - If $P_i \in \mathcal{P}_{\mathcal{C}orr}$ and $D \notin \mathcal{P}_{\mathcal{C}orr}$, send a requested-based delayed output $(\texttt{Corrupt}, P_i)$ to all parties.
   - Otherwise, send all views to $\mathcal{S}$ and give up security.

4: Upon receiving request $(\texttt{Accusation}, D, P_i)$ from $t+1$ parties, if both D and P_i are honest, send all views to $\mathcal{S}$ and give up security. Otherwise, send a requested-based delayed output $(\texttt{Accusation}, D, P_i)$ to all parties.

5: Upon receiving $(\texttt{Private-Recon-Vrfy}, k, P_j)$ from $t+1$ parties and $k \in [(j-1)\cdot n+1, j\cdot n]$, let $\mathcal{M}$ denote the set of first $2t+1$ parties, for each $\ell \in [N']$, compute a degree-$2t$ polynomial $f_\ell(x)$ based on $\{q_\ell^{(k)}(\alpha_i)\}_{i\in\mathcal{M}}$. Then if $f_\ell(x) = q_\ell^{(k)}(x)$ for all $\ell \in [N']$, send a request-based delayed output $\texttt{true}$ and $\{f_\ell(x)\}_{\ell\in[N']}$ to P_j. Otherwise, send a request-based delayed output $\texttt{false}$ to P_j.
   - Upon receiving a set $\mathcal{M}'$ from $\mathcal{S}$, if $|\mathcal{M}'| = 2t+1$, $\mathcal{M}' \cap \mathcal{P}_{\text{proof}} = \varnothing$, and the output of P_j has not been delivered, set $\mathcal{M} = \mathcal{M}'$ and do the above thing again.
   - Upon receiving a request $(\texttt{Accusation}, D, P_i)$ from $\mathcal{S}$, if it has sent $(\texttt{Accusation}, D, P_i)$ to all parties before and the output of R has not been delivered, change the output of P_j by the identity of P_i.

In the sharing phase, the dealer sends each party's shares through $\mathcal{F}_{\mathsf{PrivSend}}$. To guarantee verifiable reconstruction, the dealer needs to compute commitments on each party's shares. After each party receives his shares from $\mathcal{F}_{\mathsf{PrivSend}}$, he will check whether the commitments broadcast by the dealer are correct. If not, he can let all parties open his shares delivered by $\mathcal{F}_{\mathsf{PrivSend}}$ to accuse this dealer later.

Protocol $\Pi_{\texttt{Sh2t-Id}}$

Let $\alpha_0, \ldots, \alpha_n$ be distinct field elements, L be the number of sharings to be prepared. The termination and accusation phases are the same as the construction in $\Pi_{\texttt{ACSS-Id}}$.

Distribution Phase

1: D possesses degree-$2t$ polynomial $f_1(x), \ldots, f_L(x)$ as inputs. He randomly samples degree-$2t$ polynomial $\tilde{f}_1(x), \ldots, \tilde{f}_L(x)$.

2: For $f_1(x), \ldots, f_L(x)$, D divides them into n^2 groups, each of size $L' = L/n^2$. Denote the degree-$2t$ polynomials in k-th group as $f_1^{(k)}(x), \ldots, f_{L'}^{(k)}(x)$. For $i \in [n], k \in [n^2]$, we define:

$$m_*^{(k)}[i] := (f_1^{(k)}(\alpha_i), \ldots, f_{L'}^{(k)}(\alpha_i)), \tilde{m}_*^{(k)}[i] := (\tilde{f}_1^{(k)}(\alpha_i), \ldots, \tilde{f}_{L'}^{(k)}(\alpha_i))$$

Then D computes $h_*^{(k)}[i] = H(m_*^{(k)}[i], \tilde{m}_*^{(k)}[i])$, and a matrix Com of size $n \times n$ such that $\mathsf{Com}[i][j] = H(h_*^{((j-1)\cdot n+1)}[i], \ldots, h_*^{(j\cdot n)}[i])$.

3: D reliably broadcasts Com. All parties jointly invoke an instance of $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$ for each $i \in [n]$, D sends $\{f_\ell(\alpha_i), \tilde{f}_\ell(\alpha_i)\}_{\ell\in[L]}$, $\{h_*^{((i-1)\cdot n+1)}[j], \ldots, h_*^{(i\cdot n)}[j]\}_{j\in[n]}$ to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$, all parties send request $(\mathsf{Init}, \mathsf{PrivSend}, D)$ to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$.

Verification Phase

1: Upon receiving $(\mathsf{Dispersed}, \mathsf{PrivSend})$ from $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$ for all $i \in [n]$ and Com from D's broadcast, all parties proceed. For each $P_i \in \mathcal{P}$, all parties send request $(\mathsf{Deliver}, P_i)$ to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$.

2: Each party P_i waits to receive $\{f_\ell(\alpha_i), \tilde{f}_\ell(\alpha_i)\}_{\ell\in[L]}$ and $\{h_*^{((i-1)\cdot n+1)}[j], \ldots, h_*^{(i\cdot n)}[j]\}_{j\in[n]}$ from $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$. Then he checks whether for all $j \in [n]$, $\mathsf{Com}[i][j] = H(h_*^{((j-1)\cdot n+1)}[i], \ldots, h_*^{(j\cdot n)}[i])$ and $\mathsf{Com}[j][i] = H(h_*^{((i-1)\cdot n+1)}[j], \ldots, h_*^{(i\cdot n)}[j])$. If true, he accepts his shares. Otherwise, he sets his output as $(\texttt{Corrupt}, D)$.

The following construction realizes steps 4 and 5 in $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$, it allows all parties to reconstruct a group of sharings distributed by the dealer to one party P_j. In our construction, to reduce the additive

overhead to $\mathcal{O}(n^4)$, each party P_j can only verify $1/n$ fraction of sharings, where the group index is between $(j-1)\cdot n+1$ and $j\cdot n$.

Protocol $\Pi_{\texttt{Sh2t-Id}}$

Upon receiving a request $(\texttt{Accusation}, D, P_i)$ from the environment, all parties do the following.

Agreement Accusation Phase

1: All parties jointly invoke an instance of $\mathcal{F}_{\mathsf{ra}}$ with input message $(\texttt{Accusation}, D, P_i)$. When they terminate $\mathcal{F}_{\mathsf{ra}}$, they set their output as $(\texttt{Accusation}, D, P_i)$ and terminate.

Upon receiving request $(\texttt{Private-Recon-Vrfy}, k, P_j)$ and $k \in [(j-1)\cdot n+1, j\cdot n]$ from the environment, if all parties have terminated the sharing phase, they do the following. During the following protocol, P_j will listen to the output of the Agreement Accusation Phase.

Private Reconstruction and Verification Phase

1: For each party P_i, if he has accepted his shares, he sends $m_*^{(k)}[i], \tilde{m}_*^{(k)}[i]$ to P_j.

2: If P_i's output in the sharing phase is $(\texttt{Corrupt}, D)$, he waits until he gets $(\texttt{Accusation}, D, P_i)$ (i can be equal to j) and then terminates with it. Otherwise, P_j has received $h_*^{((j-1)\cdot n+1)}[i], \ldots, h_*^{(j\cdot n)}[i]$ for all $i \in [n]$ in the sharing phase and waits to receive messages from each party P_i.
   - If he receives $m_*^{(k)}[i], \tilde{m}_*^{(k)}[i]$, he accepts them if $h_{\ell,*}^{(k)}[i] = H(m_*^{(k)}[i], \tilde{m}_*^{(k)}[i])$ and $\mathsf{Com}[i][j] = H(h_*^{((j-1)\cdot n+1)}[i], \ldots, h_*^{(j\cdot n)}[i])$.
   - Otherwise, if he gets $(\texttt{Accusation}, D, P_i)$, he records the identity of P_i.

   If P_j first accepts $2t+1$ distinct $m_*^{(k)}[i], \tilde{m}_*^{(k)}[i]$, he reconstructs degree-$2t$ polynomials $f_1^{(k)}(x), \ldots f_{L'}^{(k)}(x)$, $\tilde{f}_1^{(k)}(x), \ldots \tilde{f}_{L'}^{(k)}(x)$ and proceed. Otherwise, P_j terminates with the identity of the first P_i he has recorded.

3: For each $i \in [n]$, P_j checks whether $h_*^{(k)}[i] = H(m_*^{(k)}[i], \tilde{m}_*^{(k)}[i])$. If true, he outputs $\texttt{true}$ and $f_1^{(k)}(x), \ldots f_{L'}^{(k)}(x)$. Otherwise, he outputs $\texttt{false}$.

Lemma 2. *Protocol* $\Pi_{\texttt{Sh2t-Id}}$ *securely computes* $\mathcal{F}_{\texttt{Sh2t-Id}}$ *against a fully malicious adversary* $\mathcal{A}$ *who corrupts at most* $t < n/3$ *parties.*

We prove Lemma 2 and analyze the costs in Appendix E.

6.2 Generating Triples by P_{king}

In this subsection, we construct $\Pi_{\texttt{TripleKingDN}}$ which allows a king to generate Beaver triples and will be used as a sub-protocol later. All parties takes their degree-t and $2t$ shares distributed by a set of dealers as inputs, and then extract random sharings. Following from the observation in [GLO+21], for *double sharings*, we may instead prepare $([r]_t, [o]_{2t})$ where $[o]_{2t}$ is a random degree-$2t$ Shamir secret sharing of 0. This allows us to decouple the relation of these two sharings. With these random sharings, all parties follow the DN technique to generate triples.

If king notifies all parties to wait for an accusation broadcast by a party, they will terminate $\Pi_{\texttt{TripleKingDN}}$ with this accusation until they receive it from this party, and they can open it later to agree on a corrupted party. Otherwise, they need to check whether the triples generated by king are correct or not.

Protocol $\Pi_{\texttt{TripleKingDN}}$

Let N be the number of Beaver triples prepared by P_{king} and $N' = (2N+1)/(t+1)$, $\mathcal{W}$ be the set of successful dealers broadcast by P_{king} and $|\mathcal{W}| = 2t+1$. Denote the sharings distributed by each $P_j \in \mathcal{W}$ as $\{[s_{\ell'}^{(j)}]_t\}_{\ell'=0}^{3N'}$ and $\{[o_{\ell'}^{(j)}]_{2t}\}_{\ell' \in [N']}$. P_{king} takes set $\texttt{Dispute}_{\mathsf{king}}$ as input and ignores messages received from parties in $\texttt{Dispute}_{\mathsf{king}}$.

Generating Random Shamir Sharings.

1: All parties agree on a Vandermonde matrix $\boldsymbol{V}$ of size $(t+1)\times(2t+1)$ and locally compute:

$$\forall \ell' \in [3N'], \quad ([s_{\ell',1}]_t, \ldots, [s_{\ell',t+1}]_t) = \boldsymbol{V}\cdot([s_{\ell'}^{(j)}]_t)_{j\in\mathcal{W}}.$$

$$\forall \ell' \in [N'], \quad ([o_{\ell',1}]_{2t}, \ldots, [o_{\ell',t+1}]_{2t}) = \boldsymbol{V}\cdot([o_{\ell'}^{(j)}]_{2t})_{j\in\mathcal{W}}.$$

2: All parties transform the above Shamir sharings into $\{[a_\ell]_t, [b_\ell]_t, [r_\ell]_t, [o_\ell]_{2t}\}_{\ell\in[2N+1]}$ and compute their share of $[r]_t := \sum [s_0^{(j)}]_t, j \in \mathcal{W}$.

Generating Triples by P_{king}

1: Let $[z_\ell]_{2t} := [a_\ell]_t \cdot [b_\ell]_t + [r_\ell]_t + [o_\ell]_{2t}$, each party P_i locally computes his share of $\{[z_\ell]_{2t}\}_{\ell\in[2N+1]}$ and send them to P_{king}. Each party P_i who cannot compute his shares has reliably broadcast $(\texttt{Accusation}, P_i, D, \texttt{ACSS})$ or $(\texttt{Accusation}, P_i, D, \texttt{Sh2t})$ for on $D \in \mathcal{W}$ outside the protocol.

2: P_{king} waits to receive messages from each party $P_i \notin \texttt{Dispute}_{\mathsf{king}}$:
- If P_{king} receives P_i's shares of $\{[z_\ell]_{2t}\}_{\ell\in[2N+1]}$, P_{king} records them.
- Otherwise, when P_{king} receives $(\texttt{Accusation}, P_i, D, \texttt{ACSS})$ or $(\texttt{Accusation}, P_i, D, \texttt{Sh2t})$ from P_i's broadcast for one $D \in \mathcal{W}$, he records it.

If P_{king} first accepts $2t+1$ parties' shares of $\{[z_\ell]_{2t}\}_{\ell\in[2N+1]}$, he locally reconstructs $\{z_\ell\}_{\ell\in[2N+1]}$ and reliably broadcasts them. Otherwise, P_{king} reliably broadcasts the first message broadcast by a party he has recorded.

3: Each party waits to receive messages from P_{king}:
- If he receives $\{z_\ell\}_{\ell\in[2N+1]}$ and he has shares of $\{[r_\ell]_t\}_{\ell\in[2N+1]}$, he locally computes his share of $[c_\ell]_t = z_\ell - [r_\ell]_t$ for all $\ell \in [2N+1]$ and proceeds. If he does not have shares of $[r_\ell]_t$, he set his shares of $[c_\ell]_t$ as $\perp$. Then all parties move to verify the triples generated by P_{king}.
- If he receives $(\texttt{Accusation}, P_i, D, \texttt{ACSS})$ or $(\texttt{Accusation}, P_i, D, \texttt{Sh2t})$ for one $D \in \mathcal{W}$, upon receiving it from P_i's broadcast, he terminates with this message.

The verification process follows the approach in [NV18, BSFO12]. At a high-level idea, all parties first prepare $2N+1$ Beaver triples $\{[a_\ell]_t, [b_\ell]_t, [c_\ell]_t\}_{\ell=0}^{2N}$ and a random degree-t sharing $[r]_t$. Then they set two polynomials f, g of degree N such that $[f(\alpha_\ell)]_t = [a_\ell]_t$ and $[g(\alpha_\ell)]_t = [b_\ell]_t$ for all $\ell \in [0, N]$. Then for all $\ell \in [N+1, 2N]$, all parties use the ℓ-th Beaver triple $\{[a_\ell]_t, [b_\ell]_t, [c_\ell]_t\}$ to compute $[f(\alpha_\ell) \cdot g(\alpha_\ell)]_t$. Now all parties set a degree-$2N$ polynomial h such that $[h(\alpha_\ell)]_t = [c_\ell]_t$ for all $\ell \in [0, N]$ and $[h(\alpha_\ell)]_t = [f(\alpha_\ell) \cdot g(\alpha_\ell)]_t$.

The main observation is that, if all random Beaver triples are correct, then we have $h = f \cdot g$ and vice verse. Therefore, to check whether all Beaver triples are correct, it is sufficient to check whether $h = f \cdot g$. By Schwartz-Zippel lemma, it is sufficient to test a random evaluation point. All parties will reconstruct $f(r), g(r), h(r)$ and check whether $h(r) = f(r) \cdot g(r)$.

Similarly, during the verification process, king may notify all parties to wait for an accusation and all parties can terminate as above. If all parties get the verification result, they will accept their triples if the result is true. Otherwise, they will help king to detect a corrupted party.

Protocol $\Pi_{\texttt{TripleKingDN}}$

All parties take their shares of $\{[a_\ell]_t, [b_\ell]_t, [c_\ell]_t\}_{\ell=0}^{2N}$ and $[r]_t$ as inputs and do the following to verify their triples.

Build Polynomials

1: All parties set two polynomials f, g of degree N such that $[f(\alpha_\ell)]_t = [a_\ell]_t$ and $[g(\alpha_\ell)]_t = [b_\ell]_t$ for all $\ell \in [0, N]$.

2: For all $\ell \in [N+1, 2N]$, all parties locally compute $[f(\alpha_\ell)]_t, [g(\alpha_\ell)]_t$. Then they send $[x_\ell]_t := [f(\alpha_\ell) + a_\ell]_t, [y_\ell]_t := [g(\alpha_\ell) + b_\ell]_t$ to P_{king}.

3: If P_{king} first succeeds in using OEC to reconstruct $\{x_\ell, y_\ell\}_{\ell=N+1}^{2N}$, he reliably broadcasts them. Otherwise, P_{king} first receives $(\texttt{Accusation}, P_i, D, \texttt{ACSS})$ and will reliably broadcast it.

4: All parties wait to receive messages from P_{king}. If they receive $\{x_\ell, y_\ell\}_{\ell=N+1}^{2N}$ from P_{king}, they proceed. Otherwise, they terminate with $(\texttt{Accusation}, P_i, D, \texttt{ACSS})$ broadcast by P_{king}.

5: For all $\ell \in [N+1, 2N]$, all parties locally compute:

$$[f(\alpha_\ell) \cdot g(\alpha_\ell)]_t = x_\ell \cdot y_\ell - x_\ell [b_\ell]_t - y_\ell [a_\ell]_t + [c_\ell]_t$$

Then they set a polynomial h of degree $2N$ such that $[h(\alpha_\ell)]_t = [c_\ell]_t$ for all $\ell \in [N]$ and $[h(\alpha_\ell)]_t = [f(\alpha_\ell) \cdot g(\alpha_\ell)]_t$ for all $\ell \in [N+1, 2N]$.

Verification of Triples

1: Al parties send their share of $[r]_t$ to P_{king}. If P_{king} first succeeds in using OEC to reconstruct the whole sharing $[r]_t$, he reliably broadcasts it. Otherwise, he reliably broadcasts the first $(\texttt{Accusation}, P_i, D, \texttt{ACSS})$ he received.

2: If all parties receive $(\texttt{Accusation}, P_i, D, \texttt{ACSS})$, they terminate with it. Otherwise, all parties invoke an instance of $\mathcal{F}_{\mathsf{ra}}$ with input 1 if his share of $[r]_t$ broadcast by P_{king} is correct. When all parties terminate $\mathcal{F}_{\mathsf{ra}}$ with 1, if $r \notin \{\alpha_1, \ldots, \alpha_N\}$, they proceed. Otherwise, all parties aborts.

3: All parties locally compute their shares of $\{[f(r)]_t, [g(r)]_t, [h(r)]_t\}$ and send them to P_{king}. P_{king} does the same thing as above to do reconstruction. He will broadcast the whole sharing $\{[f(r)]_t, [g(r)]_t, [h(r)]_t\}$ or $(\texttt{Accusation}, P_i, D, \texttt{ACSS})$.
4: If all parties receive $(\texttt{Accusation}, P_i, D, \texttt{ACSS})$, they terminate with it. Otherwise, they proceed.
5: All parties invoke an instance of $\mathcal{F}_{\mathsf{ra}}$ with input 1 if their shares of $\{[f(r)]_t, [g(r)]_t, [h(r)]_t\}$ broadcast by P_{king} are correct. When they terminate $\mathcal{F}_{\mathsf{ra}}$ with 1, they check whether $h(r) = f(r) \cdot g(r)$. If true, they terminate with shares of $\{[a_\ell]_t, [b_\ell]_t, [c_\ell]_t\}_{\ell=1}^{N}$. Otherwise, they help P_{king} to do fault localization.

To help king detect a corrupted party, all parties reconstruct the whole degree-t and $2t$ sharings to king. For degree-t sharings, all parties can send the extracted random sharings to king since king can use OEC to do reconstruction. For degree-$2t$ sharings, all parties verifiably reconstruct the sharings distributed by each dealer. Finally, king may receive an accusation from a party or detect a corrupted party. For the former case, he will let all parties terminate with this party's accusation. For the latter case, he will broadcast a dispute pair to claim that he conflicts with this party and ignore the messages received from this party from then on.

Protocol $\Pi_{\texttt{TripleKingDN}}$

Fault Localization

1: All parties do the following, let k be the index of the group of sharings used for P_{king} in this segment.
(1). Send shares of $\{([a_\ell]_t, [b_\ell]_t, [r_\ell]_t)\}_{\ell=0}^{2N}$ to P_{king}.
(2). Send $(\texttt{Private-Recon-Vrfy}, k, P_{\mathsf{king}})$ to $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ invoked by dealers in $\mathcal{W}$.
2: P_{king} first checks the degree-$2t$ sharings of zero distributed by each $P_i \in \mathcal{W}$. For the outputs of all $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$:
- If one of them is $(\texttt{Accusation}, D, P_i)$, P_{king} reliably broadcasts $(\texttt{Accusation}, P_i, D, \texttt{Sh2t})$ when he receives it from P_i's broadcast.
- If one of them is `false`, P_{king} reliably broadcasts $(\texttt{Dispute}, P_i, P_{\mathsf{king}})$ and terminates.
- Otherwise, P_{king} gets sharings distributed by P_i, denoted each one by $[o^{(i)}]_{2t}$. If one of secret $o^{(i)} \neq 0$, P_{king} reliably broadcasts $(\texttt{Dispute}, P_i, P_{\mathsf{king}})$ and terminates. Otherwise, P_{king} proceeds.

3: Then P_{king} checks which party provides incorrect shares of $[z_\ell]_{2t}$.
(1). If P_{king} can uses OEC to reconstruct $\{([a_\ell]_t, [b_\ell]_t, [r_\ell]_t)\}_{\ell=0}^{2N}$, he proceeds. Otherwise, he reliably broadcasts the first $(\texttt{Accusation}, P_i, D, \texttt{ACSS})$ he received from P_i's broadcast and terminates.
(2). P_{king} computes each $[o_\ell]_{2t}$ from $\{o^{(i)}\}_{i \in \mathcal{W}}$. Then he follows the protocol to compute $[z_\ell]_{2t} = [a_\ell]_t \cdot [b_\ell]_t + [r_\ell]_t + [o_\ell]_{2t}$. For the first party P_i who provides incorrect shares of $[z_\ell]_{2t}$, P_{king} reliably broadcasts $(\texttt{Dispute}, P_i, P_{\mathsf{king}})$ and terminates.

4: All parties wait for the message from P_{king}'s broadcast.
- If it is $(\texttt{Accusation}, P_i, D, \texttt{ACSS})$ or $(\texttt{Accusation}, P_i, D, \texttt{Sh2t})$ and $D \in \mathcal{W}$, all parties terminate with it when they receive it from P_i's broadcast.
- If it is $(\texttt{Dispute}, P_i, P_{\mathsf{king}})$, all parties terminate with it.

Summary. During the protocol $\Pi_{\texttt{TripleKingDN}}$, all parties follow the DN technique to generate Beaver triples. All parties may terminate with (1) Valid triples, (2) An accusation broadcast by a party, or (3) A dispute pair broadcast by king. For both cases (2) and (3), all parties can help king to find a corrupted party later. Then this king can ignore the messages received from this party from then on and after at most t failure times, an honest king will always generate valid triples. The total communication complexity of $\Pi_{\texttt{TripleKingDN}}$ is $\mathcal{O}(Nn + n^3)$ field elements for preparing N Beaver triples.

6.3 Triple Generation Procedure

In this subsection, we give the construction of triple generation protocol $\Pi_{\texttt{Triple}}$. Each party will act as the dealer and invoke $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ to distribute degree-t random sharings and $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ to distribute degree-$2t$ random sharings of zero. For the shares received from each dealer, all parties divide them into $\mathcal{O}(n^2)$ groups, and each one will be used for each king to generate triples in each group.

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

Let N be the number of Beaver triples to be prepared and $N' = 6N/(t+1) + n$.

Preparation of Random Shamir Sharings.

1: Each party acts as the dealer and invokes $\mathcal{F}_{\text{ACSS-Id}}$ and $\mathcal{F}_{\text{Sh2t-Id}}$ to distribute $9N'$ random degree-t Shamir sharings and $3N'$ random degree-$2t$ Shamir sharings of 0 respectively.
2: Each party P_i initializes a set $\mathcal{W}_i = \varnothing$, once P_i terminates $\mathcal{F}_{\text{ACSS-Id}}$ and $\mathcal{F}_{\text{Sh2t-Id}}$ invoked by a dealer P_j, he adds P_j to $\mathcal{W}_i$:
   – If P_i terminates $\mathcal{F}_{\text{ACSS-Id}}$ or $\mathcal{F}_{\text{Sh2t-Id}}$ with $(\texttt{Corrupt}, P_j)$, he reliably broadcasts $(\texttt{Accusation}, P_i, P_j, \texttt{ACSS})$ or $(\texttt{Accusation}, P_i, P_j, \texttt{Sh2t})$.
   – Otherwise, P_i divides his degree-t and $2t$ shares into n^2 groups, each group contains $9N'/n^2$ degree-t sharings and $3N'/n^2$ degree-$2t$ sharings. They assign the group with indices in $[(j-1)\cdot n, j\cdot n]$ to each party P_j.

In the generation phase, each party will act as a king and all parties help him to generate $\mathcal{O}(N/n^2)$ triples. Then all parties agree on a set of successful kings (of size $n-t$), and decide whether they can move to the next segment. Recall that a king is considered to be successful if all parties terminate with (1) Valid Beaver triples, (2) An accusation between a party and a dealer, and (3) A dispute pair between a king and a party. If there are not a sufficient number of valid triples generated by these $n-t$ successful kings, all parties will either open an accusation to publicly eliminate a corrupted party or record the dispute pairs. Since each honest king is guaranteed to generate valid triples after removing all corrupted parties, all parties will eventually generate a sufficient number of triples in each segment. The communication complexity of $\Pi_{\texttt{Triple}}$ is $\mathcal{O}(Nn + n^4)$ field elements for generating N Beaver triples.

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

Generation Phase.

Divide the generation of N random Beaver triples into t segments, each party P_i initializes a set $\texttt{Dispute}_i = \varnothing$, in the following, P_i will ignore messages received from parties in $\texttt{Dispute}_i$.

1: **Generation of Random Triples:**
Each party acts as P_{king} and leads an instance of $\Pi_{\texttt{TripleKingDN}}$, when $|\mathcal{W}_{\text{king}}| \geq 2t+1$, P_{king} reliably broadcasts a set $\mathcal{W} \subseteq \mathcal{W}_{\text{king}}$ of size $2t+1$.
For each party P_i, upon receiving $\mathcal{W} \subseteq \mathcal{W}_i$ from P_{king}, if P_{king} has not broadcast $(\texttt{Dispute}, P_j, P_{\text{king}})$ for all $P_j \in \mathcal{W}$, he participates in $\Pi_{\texttt{TripleKingDN}}$ with the first unused group of shares distributed by dealers in $\mathcal{W}$.

2: **Determine the Set of Successful Kings:**
Each party sets the property Q as he terminates the $\Pi_{\texttt{TripleKingDN}}$ led by one P_{king}, and all parties invoke $\mathcal{F}_{\text{acs}}$ with property Q to agree on a set $\mathcal{D}$ of successful kings with size $|\mathcal{D}| = 2t+1$.

3: **Determine the Outputs:**
For the outputs of all $P_{\text{king}} \in \mathcal{D}$, all parties check the following things in order:
   – If at least $t+1$ of them are valid Beaver triples, all parties terminate the current segment with the first $t+1$ of them and move to the next segment.
   – If at least one of them is $(\texttt{Accusation}, P_i, D, \texttt{ACSS})$ or $(\texttt{Accusation}, P_i, D, \texttt{Sh2t})$, for P_i with the smallest index, all parties send request $(\texttt{Open-Proof}, P_i)$ to $\mathcal{F}_{\text{ACSS-Id}}$ or $\mathcal{F}_{\text{Sh2t-Id}}$ invoked by D. Upon receiving the output $(\texttt{Corrupt}, D)$ or $(\texttt{Corrupt}, P_i)$, all parties move to Step 4 with the identity of the corrupted party.
   – Otherwise, all parties records all dispute pairs broadcast by all P_{king}, and each P_{king} whose output is $(\texttt{Dispute}, P_i, P_{\text{king}})$ locally removes P_i from $\mathcal{W}_{\text{king}}$ (if exists) and adds P_i to $\texttt{Dispute}_{\text{king}}$. When all parties find that one party conflicts with at least $t+1$ parties, they agree on the identity of this party and move to Step 4.

4: **Reconstruct Corrupted Party's Secret**
When all parties agree on a corrupted party P_k, each party P_i adds P_k to $\texttt{Dispute}_i$ and removes P_k from $\mathcal{W}_i$ (if exists). Then all parties do the following.
   – If P_k has distributed degree-t Shamir sharings, all parties send $\texttt{Public-Recon}$ to the $\mathcal{F}_{\text{ACSS-Id}}$ invoked by him and replace their shares by the secrets received from $\mathcal{F}_{\text{ACSS-Id}}$. All parties locally update their shares of triples prepared in the previous segment.
Finally, all parties execute the current one again.

7 Main Protocol

In this section, we construct $\Pi_{\texttt{Main}}$ to realize $\mathcal{F}_{\mathsf{AMPC}}$. All parties first execute $\Pi_{\texttt{Triple}}$ to prepare triples in the offline phase and then use them to evaluate a circuit in the online phase. To achieve linear cost, we use the *party elimination framework* and divide the circuit into t disjoint sub-circuits $C_1, \ldots, C_t$ (sorted by the topology), each containing $|C|/t$ multiplication gates and the depth is bounded by $\mathcal{O}(D/n)$. For each circuit segment, all parties only need to do public reconstruction, which can be realized by $\Pi_{\texttt{SubCktEval}}$. As a result, we obtain the following theorem. We refer the reader to Appendix F for detailed construction, security proof, and cost analysis.

Theorem 1. *Let $n = 3t + 1$ and κ denote the security parameter. Further let $\mathbb{F}$ be a finite field of size $2^{\Omega(\kappa)}$ and C be a circuit of size $|C|$ and depth D. Assuming random oracles, there is a fully malicious asynchronous MPC protocol computing the circuit that is secure against at most t corrupted parties with guaranteed output delivery. The achieved communication complexity is $\mathcal{O}(|C| \cdot n + D \cdot n^2 + n^4)$ field elements and round complexity is $\mathcal{O}(D + n)$.*

8 Conclusion

In this work, we presented an asynchronous Multi-Party Computation protocol that relies exclusively on computationally efficient lightweight cryptographic tools. To address the absence of transcript homomorphism, we introduced several techniques to optimize the communication overhead of our protocols. The first of these is the asynchronous party elimination framework with efficient public reconstruction, which is mainly used in the online phase and allows us to achieve GOD with only linear cost per gate. Additionally, our second technique—an asynchronous dispute control framework, which is mainly used for triple generation in the offline phase and enables us to attain an additive communication overhead of $\mathcal{O}(n^4)$, marking an $\mathcal{O}(n^2)$ improvement over previous approaches in this domain. Together, these contributions advance distributed cryptography by leveraging lightweight cryptography to verifiably detect malicious behavior and effectively manage corrupted parties in an asynchronous network.

Acknowledgements. X. Ji and Y. Song were supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61361136003. Y. Song was also supported in part by the Shanghai Qi Zhi Institute Innovation Program SQZ202313. C. Liu-Zhang and Y. Song were supported in part by the ETH Zurich Leading House Research Partnership Grant RPG-072023-19. A. Bandarupalli and A. Kate were supported in part by Sui Academic Research Award and NIFA/USDA award number 2021-67021-34252.

References

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

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

[ADD+22] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Brief announcement: Asynchronous verifiable information dispersal with near-optimal communication. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 418–420, 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.

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

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

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

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

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

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

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

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

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

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

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

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

[CCG25] Ignacio Cascudo, Daniele Cozzo, and Emanuele Giunta. Verifiable secret sharing from symmetric key cryptography with improved optimistic complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 100–128. Springer, 2025.

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

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

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

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

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

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

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

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

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

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

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

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

[GLZS24] Vipul Goyal, Chen-Da Liu-Zhang, and Yifan Song. Towards achieving asynchronous MPC with linear communication and optimal resilience. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 170–206. Springer, Cham, August 2024.

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

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

[HMP00] Martin Hirt, Ueli Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In *International conference on the theory and application of cryptology and information security*, pages 143–161. Springer, 2000.

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

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

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

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

[Mom24] Atsuki Momose. Practical asynchronous mpc from lightweight cryptography. *Cryptology ePrint Archive*, 2024.

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

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

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

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

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

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

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

[SLL+24] Yuan Su, Yuan Lu, Jiliang Li, Yuyi Wang, Chengyi Dong, and Qiang Tang. Dumbo-mpc: Efficient fully asynchronous mpc with optimal resilience. *Cryptology ePrint Archive*, 2024.

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

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

A Additional Preliminaries

A.1 Definitions of Agreement Primitives

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

We state the traditional property-based formalization of ACS.

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

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

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

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

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

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

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

A.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 *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 $i \in [n]$. Each party P_i holds a share x_i and the whole sharing is denoted by $[x]_d$. We recall the properties of the Shamir secret sharing scheme:

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

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

a polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most d such that $f(\beta_i) = x_i$ for all $i \in [k]$, and $f(\alpha_i) = x_i$ for all $i \in [n]$.

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

B Construction and Security Proof of DZK

In this section, we introduce the construction of the distributed zero-knowledge (DZK) protocol in the asynchronous setting based on the ideas in [CCG25], and provide the corresponding security proof and cost analysis.

To build the DZK protocol, we rely on the Merkle tree commitment and the PrivSend functionality $\mathcal{F}_{\mathsf{PrivSend}}$. We first review the definition of the Merkle tree commitment as follows.

Merkle Tree Commitment. A Merkle tree commitment is a commitment scheme for a vector of messages $\mathbf{m} = (m_1, \ldots, m_N)$ based on a collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^\kappa$, where κ denotes the security parameter. For each message m_i, the prover samples an independent random string $r_i \xleftarrow{\$} \{0,1\}^\kappa$ and defines the leaf node as $L_i = H(m_i, r_i), \quad i \in [N]$. Internal nodes are computed recursively by hashing the concatenation of their children. Namely, if u and v are the left and right children of a node, then the parent node is $P = H(u, v)$. The root of the resulting binary hash tree, denoted by root, serves as the commitment to the entire vector $\mathbf{m}$.

To open the commitment for a leaf m_i, the prover provides (m_i, r_i) together with an authentication path consisting of the sibling hashes along the path from L_i to root. Given this proof, any verifier can recompute the hashes along the path and check consistency with the committed root. The size of the opening proof and the verification complexity are both $O(\log N)$.

The randomness r_i ensures that the leaf commitment L_i computationally hides the underlying message m_i, so that the Merkle root does not reveal information about the committed inputs beyond what is explicitly opened.

To be more concrete, a Merkle tree commitment scheme consists of the following algorithms $\mathtt{MT.Com}$, $\mathtt{MT.Open}$, $\mathtt{MT.Vrfy}$.

- $(\mathsf{root}, \mathsf{aux}) \leftarrow \mathtt{MT.Com}(\mathbf{m})$: Given a vector of messages $\mathbf{m} = (m_1, \ldots, m_N)$, the prover samples independent random strings $r_1, \ldots, r_N \xleftarrow{\$} \{0,1\}^\kappa$, and computes the leaf nodes $L_i = H(m_i, r_i), \quad i \in [N]$. The internal nodes are computed recursively by hashing the concatenation of their two children. The root of the resulting binary hash tree, denoted by root, is the commitment. The algorithm outputs $(\mathsf{root}, \mathsf{aux})$, where aux denotes the entire Merkle tree.
- $\pi_i \leftarrow \mathtt{MT.Open}(\mathsf{aux}, i)$: To open the i-th message, the prover outputs $\pi_i = (r_i, \mathsf{path}_i)$, where path_i is the authentication path consisting of the sibling hashes along the path from leaf L_i to the root.
- $\{0,1\} \leftarrow \mathtt{MT.Vrfy}(\mathsf{root}, m_i, i, \pi_i)$: Given a commitment root, an index i, the message m_i, and an opening proof $\pi_i = (r_i, \mathsf{path}_i)$, the verifier first recomputes the leaf $L_i = H(m_i, r_i)$, and then iteratively recomputes the hashes along the authentication path. The verifier outputs 1 if the resulting root equals root, and outputs 0 otherwise.

B.1 Technical Overview

The idea of Hash-based DZK proof was first described in [SS24], who used this construction to design an ACSS protocol only using lightweight cryptography. Subsequently, [ABCP23] described a synchronous construction of DZK in the lightweight setting as well. [ABCP23] also first provided a functionality-based definition for DZK, but only in the synchronous network setting. A subsequent work [CCG25] improved the efficiency of DZK as compared to [ABCP23], also in the synchronous network setting.

Recap of DZK Technique in [SS24]. We briefly recap the construction of the DZK protocol in [SS24]. In this protocol, the n parties with L shares $f_i(\alpha_\ell)$ for $i \in [L]$ act as verifiers P_ℓ, and the dealer D who dealt these shares acts as a prover trying to prove the shares belong to valid degree-d polynomials.

First, the prover samples a random degree-d *challenge* polynomial $Y(x)$, generates commitments $\boldsymbol{C}[\ell]$ to points $Y(\alpha_\ell)$ for $\ell \in [n]$ using the random oracle H, and broadcasts $\boldsymbol{C}$. The verifiers then sample a random challenge point p and ask the prover to reliably broadcast $r(x) := Y(x) - \sum_{i=1}^{L} p^i f_i(x)$. Then, each verifier P_ℓ verifies whether the challenge is successful by checking $H(r(\alpha_\ell) + \sum_{i=1}^{L} p^i f_i(\alpha_\ell)) \stackrel{?}{=} \boldsymbol{C}[\ell]$. The prover's probability of passing this check with an invalid shares is negligible because of the Schwartz-Zippel lemma, which prevents a non-zero polynomial evaluating to zero on a randomly sampled point with probability no more than $\frac{L}{|\mathbb{F}|}$. This proof technique can be made non-interactive by using the Fiat-Shamir heuristic, where the prover creates the challenge point p by applying H on generated commitments. At the end of the verification procedure, if all parties agree that the prover provides a valid proof, then we can guarantee that all honest parties have a valid DZK proof and get correct shares.

For the DZK proof, we use π to denote the broadcast proof and π_i to denote the private proof for verifier P_i (excluding P_i's shares $f_1(\alpha_i), \ldots, f_L(\alpha_i)$, we also do not count the cost for distributing these shares inside the DZK protocol), and the party P_i can verify its shares once it receives both the π and π_i. The broadcast proof $\pi = r(x), \boldsymbol{C}$ has size $\mathcal{O}(n)$ field elements, whereas each private proof $\pi_i = Y(\alpha_i)$ has size $\mathcal{O}(1)$ field elements. Each field element is represented using κ bits, where κ is the security parameter.

Subtle Issues in the Asynchronous Setting. [SS24] noted subtle issues with the above construction in the asynchronous network setting. The first issue is that a corrupted prover may not distribute the DZK proof. Note that in the synchronous setting, with the help of the time bound, all parties are guaranteed to receive the DZK proof from the prover. However, there is no such guarantee in the asynchronous setting, and a corrupted prover may not send the DZK proof to an honest party via the P2P channel. Since we can only expect to receive $n - t$ response, then only $n - 2t$ honest parties can receive their DZK proof in the worst case. To guarantee all honest parties can eventually receive their DZK proof, the prover uses the $\mathcal{F}_{\mathsf{PrivSend}}$ primitive to deliver each party's DZK proof. Recall that for the message sent by $\mathcal{F}_{\mathsf{PrivSend}}$, all parties can guarantee the recovery of this message to any receiver in the `RevealPhase` once they terminate the `DispersePhase`.

The second issue is that the above verification method achieves only a soundness error of $2^{n-\kappa}$ in the asynchronous setting, rather than the desired $2^{-\kappa}$. The reason is that a computationally unbounded corrupted prover can first distribute shares corresponding to a polynomial of degree greater than d to the honest parties. It can then check whether there exists a subset of honest parties $\mathcal{W}$ whose random linear combinations of shares happen to interpolate to a unique degree-d polynomial $r(x)$, even though their original shares are not consistent with $r(x)$, and the probability that the adversary can succeed is $2^{n-\kappa}$. Although the random linear combinations held by the remaining honest parties will not lie on $r(x)$ with overwhelming probability, the adversary can simply block their messages, and the honest parties in $\mathcal{W}$ cannot be aware of this. Consequently, to guarantee termination in an asynchronous network, the protocol may proceed using only the responses from at most $n-t$ parties, which may only include honest parties in $\mathcal{W}$ and corrupted parties. This attack is specific to the asynchronous setting. In contrast, this attack does not work in the synchronous setting, because every party can expect to receive responses from all parties before completing the verification.

To amplify the soundness to $2^{-\kappa}$, we can use a *challenge* polynomial $Y(x)$ in a larger field $\mathbb{G}$ of size $\Theta(2^{n+\kappa})$. This modification would increase the communication complexity to at least $\mathcal{O}((n+\kappa)n^2)$ bits since the dealer needs to broadcast the polynomial $r(x)$, which is of size $\mathcal{O}((n+\kappa)n)$ bits. Recall that in this work, we need to invoke $\mathcal{O}(n)$ instances of DZK in each instance of ACSS, and there are $\mathcal{O}(n)$ instances of ACSS in total. Then the total overhead for DZK is at least $\mathcal{O}((n+\kappa)n^4)$ bits, the term $\mathcal{O}(n^5)$ bits exceeds our target to achieving $\mathcal{O}(n^4)$ overall field elements (each field element is of size κ bit) when $n > \kappa$. To reduce it, we consider a more efficient construction of DZK introduced in [CCG25].

What is achieved in [CCG25]. We do not recap the technique in [CCG25] and refer the reader to their original paper for more details. Compared to the result in [ABCP23], the main contribution of [CCG25] is to trade off the size of the broadcast proof and each party's private proof. In [SS24], the broadcast proof size is $\mathcal{O}((n+\kappa)n)$ bits as we discussed above, and each party's private proof size is $\mathcal{O}(n+\kappa)$ bits. By applying the technique in [CCG25] to the asynchronous setting, we can reduce the broadcast proof size to $\mathcal{O}(n + \kappa \log n)$ bits, but improve each party's private proof size to $\mathcal{O}((n + \kappa \log n) \log n)$ bits. As we will show next, for $\mathcal{O}(n)$ parallel instances invoked by the same dealer, this trade-off allows us to bound the overall cost of DZK within $\mathcal{O}(n^3)$ field elements. Then, for $\mathcal{O}(n)$ dealers, the overall cost of DZK is $\mathcal{O}(n^4)$ field elements.

B.2 Construction

We give the construction of DZK as follows. Our protocol $\Pi_{\mathtt{DZK}}$ generalizes the techniques in [ABCP23, CCG25] by introducing a parameter called the compression factor u. By setting $u = 2$, $\Pi_{\mathtt{DZK}}$ is the case of applying the technique in [CCG25] to the asynchronous setting. By setting $u = n$, we note that $\tau = 0$ in this case, and the prover will not enter the loop for compression, but just broadcast a linear combination of its input polynomials. This is the case of applying the technique in [ABCP23] to the asynchronous setting.

Protocol $\Pi_{\mathtt{DZK}}$

Parameter: Let $(\alpha_1, \ldots, \alpha_n)$ denote n distinct field elements, N denote the number of input polynomials, $d \in \mathbb{N}$ with $d \leq n-1$ be the degree of input polynomials, compress factor $u \in \mathbb{N}$, and $\tau \in \mathbb{N}$ with $\tau = \lceil \log_u d \rceil$. Let $d_k = \lceil d_{k-1}/u \rceil$ for all $k \in [\tau]$. Let κ be the security parameter, $\mathbb{F}$ be a finite field of size $\Theta(2^\kappa)$, and $\mathbb{G}$ be an extension field of $\mathbb{F}$ of size $\Theta(2^{n+\kappa})$.

Distributing Proof Phase

When the prover P receives degree-d polynomials $f_1(x), \ldots, f_N(x) \in \mathbb{F}[x]$ from the environment, and each party receives $(\mathsf{Init}, \mathsf{DZK}, P)$ from the environment, they proceed as follows.

1: P randomly samples a degree-d polynomial $f_0(x) \in \mathbb{G}[x]$. For each $i \in [n]$, P sets $m_{0,i} = \{f_0(\alpha_i), f_1(\alpha_i), \ldots, f_N(\alpha_i)\}$, and computes $(c_0, \mathtt{aux}_0) = \mathtt{MT.Com}(\{m_{0,1}, \ldots, m_{0,n}\})$ and $\mathtt{op}_{0,i} = \mathtt{MT.Open}(i, \mathtt{aux}_0)$ for all $i \in [n]$.

2: P computes $\mu_0 = H(c_0) \in \mathbb{G}$, and sets $f^{(0)} = \sum_{\ell=0}^{N} \mu_0^\ell \cdot f_\ell$. For all $k \in [\tau]$, P repeats the following steps.

(1). P computes $g_0^{(k)}, g_1^{(k)}, \ldots, g_{u-1}^{(k)}$ of degree $\leq d_k$ such that:

$$f^{(k-1)} = g_0^{(k)} + \sum_{\ell \in [u-1]} x^{d_{k-1} - \ell \cdot d_k} \cdot g_{u-\ell}^{(k)}.$$

(2). For each $i \in [n]$, P computes $m_{k,i} = \{g_0^{(k)}(\alpha_i), g_1^{(k)}(\alpha_i), \ldots, g_{u-1}^{(k)}(\alpha_i)\}$, and $(c_k, \mathtt{aux}_k) = \mathtt{MT.Com}(\{m_{k,1}, \ldots, m_{k,n}\})$ and $\mathtt{op}_{k,i} = \mathtt{MT.Open}(i, \mathtt{aux}_k)$ for all $i \in [n]$.

(3). P computes $\mu_k = H(c_0, c_1, \ldots, c_k) \in \mathbb{G}$, and sets:

$$f^{(k)} = g_0^{(k)} + \mu_k \cdot g_1^{(k)} + \cdots + \mu_k^{u-1} \cdot g_{u-1}^{(k)}.$$

3: Upon getting $f^{(\tau)}$, P sets:

$$\pi \leftarrow (\{c_k\}_{k=0}^{\tau}, f^{(\tau)}), \pi_i \leftarrow (\{m_{k,i}\}_{k=1}^{\tau}, \{\mathtt{op}_{k,i}\}_{k=0}^{\tau}, f_0(\alpha_i)).$$

Then P broadcasts π. For each party P_i, all parties jointly invoke an instance of $\mathcal{F}_{\mathsf{PrivSend}}$, denoted by $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$. P sends π_i to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$, and all parties send $(\mathsf{Init}, \mathsf{PrivSend}, P)$ to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$.

4: Upon receiving $(\mathsf{Dispersed}, \mathsf{PrivSend})$ from $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$ for all $i \in [n]$, each party sends the request request $(\mathsf{Deliver}, V_i)$ to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$ for all $i \in [n]$.

5: Upon receiving π from P's reliable broadcast and π_i from $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$, each party V_i terminates with $(\mathsf{Delivered}, \mathsf{DZK})$.

Verification Phase

When the verifier R receive $(\mathtt{PrivVerifyDZK}, s_1, \ldots, s_N, j)$ from the environment, it proceeds the following steps. When parties receive $(\mathtt{PrivVerifyDZK}, j, R)$ from the environment, they execute the following steps.

1: For each party that has terminated the distributing proof phase, it sends the request $(\mathsf{Deliver}, R)$ to $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$. The receiver R waits to receive π_j from $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$.

2: Upon getting π, π_j, R does the following steps to do verification. It first sets $m_{0,j} = \{f_0(\alpha_j), s_1, \ldots, s_N\}$. Then R computes:

$$\forall k \in [0, \tau-1], f^{(k)}(\alpha_j) \leftarrow g_0^{(k)}(\alpha_j) + \sum_{\ell \in [u-1]} \alpha_j^{d_{k-1} - \ell \cdot d_k} \cdot g_{u-\ell}^{(k)}(\alpha_j).$$

Then R computes $\mu_k = H(c_0, \ldots, c_k) \in \mathbb{G}$ for all $k \in [0, \tau]$. Finally, R checks whether the following conditions hold.

- $1 = \mathtt{MT.Vrfy}(c_k, m_{k,j}, j, \mathtt{op}_{k,j})$ for all $k \in [0, \tau]$.
- $\deg(f^{(\tau)}) \leq d_\tau$.
- $f^{(0)}(\alpha_j) = f_0(\alpha_j) + \sum_{\ell=1}^{N} \mu_0^\ell \cdot s_\ell$.
- $f^{(k)}(\alpha_j) = g_0^{(k)}(\alpha_j) + \mu_k \cdot g_1^{(k)}(\alpha_j) + \cdots + \mu_k^{u-1} \cdot g_{u-1}^{(k)}(\alpha_j)$ for all $k \in [\tau]$.

If true, R terminates with `true`. Otherwise, R terminates with `false`.

Transfer Proof Phase

When V_j receives the request $(\texttt{TransferDZK}, s_1, \ldots, s_N, V_j \to R)$ from the environment, it executes the following steps.

1: V_j does the same thing as in the Verification Phase to do checks whether $s_1, \ldots, s_N$ are correct shares under its DZK proof π, π_j. If it succeeds, then it sends $s_1, \ldots, s_N$ and π_j to R, and terminates with `true`. Otherwise, it terminates with `false`.

2: Upon receiving $s_1, \ldots, s_N$ and π_j from V_j, R performs the same verification again. If it succeeds in verification, it outputs $s_1, \ldots, s_N$. Otherwise, it ignores the messages received from V_j.

Theorem 2. *Let $n = 3t + 1$ and κ denote the security parameter. Assuming random oracles, protocol Π_{DZK} securely computes $\mathcal{F}_{\mathsf{dZK}}$ against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t < n/3$ parties.*

Communication Complexity. For the size of π broadcast by P, since each commitment $c_k \in \mathbb{F}$, $d = \mathcal{O}(n)$, and the degree-d_τ polynomial $f^{(\tau)} \in \mathbb{G}[x]$, then the total broadcast size is $\mathcal{O}(u{\cdot}n{+}\kappa(\log_u n{+}u))$ bits. For the size of each V_i's DZK proof, since each element in vector $m_{k,i}$ is in the field $\mathbb{G}$ (for all $k \in [\tau]$), each Merkle proof $\mathtt{op}_{k,i}$ is of size $\mathcal{O}(\kappa \log n)$ bits (for all $k \in [0, \tau]$), and $f_0(\alpha_i) \in \mathbb{G}$. Then the total size of each V_i's DZK proof is $\mathcal{O}((u \cdot n + u \cdot \kappa + \kappa \cdot \log n) \log_u n)$ bits.

Then the prover P broadcasts π and invoke $\mathcal{F}^{(i)}_{\mathsf{PrivSend}}$ to distribute π_i. By using the broadcast protocol in [ADD+22] and realization of $\mathcal{F}_{\mathsf{PrivSend}}$ in Appendix C, the total communication complexity is $\mathcal{O}((un + u\kappa + \kappa \log n)n \log_u n + \kappa n^3) = \mathcal{O}(\kappa n^3)$ bits.

When all parties invoke the verification phase for a receiver R, it costs $\mathcal{O}((u{\cdot}n{+}u{\cdot}\kappa{+}\kappa{\cdot}\log n) \log_u n{+}\kappa n^2) = \mathcal{O}(\kappa n^2)$ bits.

When a party V_j invokes the transfer proof phase for a receive R, it costs $\mathcal{O}(N\kappa + (u \cdot n + u \cdot \kappa + \kappa \cdot \log n) \log_u n)$ bits.

Parallel Optimization. For $\mathcal{O}(n)$ instances of Π_{DZK} invoked by the same dealer, we can let D only invoke a single instance of $\mathcal{F}^{(i)}_{\mathsf{PrivSend}}$ to distribute all DZK proof in these $\mathcal{O}(n)$ instances for a single party P_i. Then the total broadcast size is $\mathcal{O}(u \cdot n^2 + \kappa n(\log_u n + u))$ bits, and the cost for broadcasting these message is $\mathcal{O}(u \cdot n^3 + \kappa n^2(\log_u n + u))$ bits. The total input size for each $\mathcal{F}^{(i)}_{\mathsf{PrivSend}}$ is $\mathcal{O}((u{\cdot}n{+}u{\cdot}\kappa{+}\kappa{\cdot}\log n)n \log_u n)$, then the communication complexity for the instantiation of $\mathcal{F}^{(i)}_{\mathsf{PrivSend}}$ is $\mathcal{O}((u{\cdot}n{+}u{\cdot}\kappa)n \log_u n{+}\kappa n^2)$ bits. Therefore, the total communication complexity for the distributing proof phase (for $\mathcal{O}(n)$ instances of Π_{DZK}) is $\mathcal{O}((u{\cdot}n{+}u{\cdot}\kappa)n^2 \log_u n{+}\kappa n^3)$ bits. When all parties invoke the verification phase for a receiver R for all these $\mathcal{O}(n)$ instances of Π_{DZK}, it costs $\mathcal{O}((u{\cdot}n{+}u{\cdot}\kappa)n \log_u n{+}\kappa n^2)$ bits. When a party V_j invokes the transfer proof phase for a receive R for one instance of Π_{DZK}, it costs $\mathcal{O}(N\kappa{+}(u{\cdot}n{+}u{\cdot}\kappa{+}\kappa{\cdot}\log n) \log_u n)$ bits.

B.3 Security Proof

We prove Theorem 2 as follows. We first construct the simulator for an honest prover:

Simulator $\mathcal{S}$

For honest Prover P

Distributing Proof Phase

1: In Step 1, $\mathcal{S}$ receives $\{f_1(\alpha_i), \ldots, f_N(\alpha_i)\}_{i \in \mathcal{C}orr}$ from $\mathcal{F}_{\mathsf{dZK}}$ for all corrupted $V_i \in \mathcal{C}orr$, for each V_i's Merkle proof opening $\mathtt{op}_{0,i}$, we parse it into the following form:

$$\mathtt{op}_{0,i} = \{r_{0,i}, h^{(1)}_{0,i}, \ldots, h^{(\log n - 1)}_{0,i}\}.$$

where $r_{0,i}$ denotes the randomness used to hash the leaf, and $\{h^{(1)}_{0,i}, \ldots, h^{(\log n-1)}_{0,i}\}$ denote the hash nodes in the Merkle proof opening. $\mathcal{S}$ first randomly samples values as $h^{(0)}_{0,i}$ for all $i \in [n]$, then it computes the whole Merkle tree based on $\{h^{(0)}_{0,i}\}_{i \in [n]}$ and gets the Merkle root c_0.

For each corrupted V_i, $\mathcal{S}$ randomly samples its $f_0(\alpha_i)$, and sets $m_{0,i} = (f_0(\alpha_i), \ldots, f_N(\alpha_i))$. Then $\mathcal{S}$ randomly samples $r_{0,i}$ and programs the Random Oracle $H(\cdot)$ such that $h^{(0)}_{0,i} = H(m_{0,i}, r_{0,i})$. Then

for all corrupted P_i, $\texttt{MT.Vrfy}(c_0, m_{0,i}, i, \mathtt{op}_{0,i}) = 1$ is satisfied. During this procedure, if $H(m_{0,i}, r_{0,i})$ have been queried by the adversary, $\mathcal{S}$ aborts the simulation.

During the simulation of the random oracle, for each honest V_i's hash value of leaf $h_{0,i}^{(0)}$, if $\mathcal{S}$ finds it has been mapped to some input queried by the adversary different from the one generated by $\mathcal{S}$, $\mathcal{S}$ aborts the simulation.

2: In Step 2, $\mathcal{S}$ computes $f^{(0)}(\alpha_i)$ for all corrupted V_i. Based on all corrupted parties' $f^{(0)}(\alpha_i)$, $\mathcal{S}$ randomly samples the whole degree-d polynomial $f^{(0)}(x)$ and honestly does the rest of the iterations in Step 2.

3: In Step 3, upon getting $f^{(\tau)}$, then $\mathcal{S}$ broadcasts π on behalf of P. When $\mathcal{S}$ receives $\mathcal{F}_{\mathsf{dZK}}$ that an honest party has sent the request $(\mathsf{Init}, \mathsf{DZK}, P)$, it simulates the behavior of sending request $(\mathsf{Init}, \mathsf{PrivSend}, P)$ from this honest party to each $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$. Then $\mathcal{S}$ emulates each $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$ as follows:

(1). When $\mathcal{S}$ learns that P's input is delivered to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$, if V_i is corrupted, it sends π_i to V_i on behalf of $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$.

(2). When an honest party's request $(\mathsf{Init}, \mathsf{PrivSend}, P)$ has been delivered to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$, $\mathcal{S}$ forwards it and the identity of this honest party to the adversary.

(3). When $\mathcal{S}$ learns that $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$ received request $(\mathsf{Init}, \mathsf{PrivSend}, P)$ from $2t+1$ parties and P's input, it sends $(\mathsf{Dispersed}, \mathsf{PrivSend})$ to all corrupted parties on behalf of $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$. For each honest party, $\mathcal{S}$ simulates the behavior of sending $(\mathsf{Dispersed}, \mathsf{PrivSend})$ from $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$ to this honest party.

4: In Step 4, $\mathcal{S}$ sends request $(\mathsf{Init}, \mathsf{DZK}, P)$ to $\mathcal{F}_{\mathsf{dZK}}$ on behalf of all corrupted parties. For each honest party, when $\mathcal{S}$ learns that it has received π from P's broadcast and $(\mathsf{Dispersed}, \mathsf{PrivSend})$ from $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$ for all $i \in [n]$, it delivers the output $(\mathsf{Delivered}, \mathsf{DZK})$ to this honest party.

Verification Phase

1: If R is honest, $\mathcal{S}$ does the following steps.

(1). During the simulation of $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$, when $\mathcal{S}$ receives the request $(\mathsf{Deliver}, R)$ from a corrupted party on behalf of $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$, it sends request $(\texttt{PrivVerifyDZK}, j, R)$ to $\mathcal{F}_{\mathsf{dZK}}$ on behalf of this corrupted party.

(2). When $\mathcal{S}$ receives $\mathcal{F}_{\mathsf{dZK}}$ that an honest party has sent the request $(\texttt{PrivVerifyDZK}, j, R)$, $\mathcal{S}$ simulates the behavior of sending $(\mathsf{Deliver}, R)$ from this honest party to $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$. When this honest party's request $(\mathsf{Deliver}, R)$ has been delivered to $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$, $\mathcal{S}$ forwards it and the identity of this honest party to the adversary.

(3). When $\mathcal{S}$ receives $2t+1$ request $(\mathsf{Deliver}, R)$ on behalf of $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$, $\mathcal{S}$ simulates the behavior of sending messages from $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$ to R. If $\mathcal{S}$ has delivered the output $(\mathsf{Delivered}, \mathsf{DZK})$ to R before, it delivers the verification result from $\mathcal{F}_{\mathsf{dZK}}$ to R.

2: If R is corrupted, $\mathcal{S}$ does the following steps.

(1). When $\mathcal{S}$ receives from $\mathcal{F}_{\mathsf{dZK}}$ that an honest party has sent the request $(\texttt{PrivVerifyDZK}, j, R)$, $\mathcal{S}$ simulates the behavior of sending the request $(\mathsf{Deliver}, R)$ from this honest party to $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$. When this honest party's request $(\mathsf{Deliver}, R)$ has been delivered to $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$, $\mathcal{S}$ forwards it and the identity of this honest party to the adversary.

(2). After $\mathcal{S}$ learns that one honest party has sent the request $(\texttt{PrivVerifyDZK}, j, R)$, $\mathcal{S}$ receives $(f_1(\alpha_j), \ldots, f_N(\alpha_j))$ from $\mathcal{F}_{\mathsf{dZK}}$. Then if V_j is corrupted, $\mathcal{S}$ sends π_j to the adversary on behalf of $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$. Otherwise, $\mathcal{S}$ does the following steps to generate honest V_j's π_j: The rest of the missing values in π_j are $f_0(\alpha_j)$ and $r_{0,j} \in \mathtt{op}_{0,j}$:

- For $f_0(\alpha_j)$, $\mathcal{S}$ computes it from $f^{(0)}(\alpha_j) - \sum_{\ell=1}^{N} \mu_0^{\ell} \cdot f_\ell(\alpha_j)$.
- For $r_{0,j}$, $\mathcal{S}$ first sets $m_{0,j} = \{f_0(\alpha_j), f_1(\alpha_j), \ldots, f_N(\alpha_j)\}$, then randomly samples a value as $r_{0,j}$ and program the Random Oracle $H(\cdot)$ such that $h_{0,j}^{(0)} = H(m_{0,j}, r_{0,j})$. If $H(m_{0,j}, r_{0,j})$ has been queried by the adversary, $\mathcal{S}$ aborts the simulation.

Then $\mathcal{S}$ generates the whole π_j and sends it to the adversary on behalf of $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$. If $\mathcal{S}$ has generated V_j's dzk proof π_j in the transfer proof phase, then $\mathcal{S}$ will reuse this proof.

(3). When $\mathcal{S}$ receives $2t+1$ request $(\mathsf{Deliver}, R)$ on behalf of $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$, $\mathcal{S}$ sends π_j to R on behalf of $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$.

Transfer Proof Phase

1: If both V_j and R are honest, $\mathcal{S}$ simulates the behavior of sending messages between V_j and R. When R receives the messages from V_j, $\mathcal{S}$ delivers the output from $\mathcal{F}_{\mathsf{dZK}}$ to R.

2: If both V_j and R are corrupted, $\mathcal{S}$ does nothing.

3: If V_j is honest and R is corrupted, $\mathcal{S}$ first waits to receive messages from $\mathcal{F}_{\mathsf{dZK}}$. If it receives `Fail` from $\mathcal{F}_{\mathsf{dZK}}$, then $\mathcal{S}$ does nothing on behalf of V_j. Otherwise, $\mathcal{S}$ receives `true` and $(f_1(\alpha_j), \ldots, f_N(\alpha_j))$ from $\mathcal{F}_{\mathsf{dZK}}$. Then $\mathcal{S}$ does the same thing as the verification phase to generate V_j's DZK proof π_j and sends it

together with $(f_1(\alpha_j), \ldots, f_N(\alpha_j))$ to R on behalf of V_j. If $\mathcal{S}$ has generated V_j's DZK proof π_j in the verification phase, then $\mathcal{S}$ will reuse this proof.

4: If V_j is corrupted and R is honest, $\mathcal{S}$ honestly executes R to do verification. When it succeeds in the verification, $\mathcal{S}$ checks whether the shares received from the corrupted V_j are the shares received from $\mathcal{F}_{\mathsf{dZK}}$ before. If true, $\mathcal{S}$ sends request $(\texttt{TransferDZK}, f_1(\alpha_j), \ldots, f_N(\alpha_j), V_j \to R)$ to $\mathcal{F}_{\mathsf{dZK}}$. Otherwise, $\mathcal{S}$ aborts the simulation.

The hybrid argument for an honest prover P is as follows.

$\mathbf{Hyb}_0$: This hybrid corresponds to the real-world execution.

$\mathbf{Hyb}_1$: In this hybrid, we change the generation of the degree-d polynomial $f_0(x)$. $\mathcal{S}$ first randomly samples corrupted parties' shares and the secret $f_0(\alpha_0)$, then samples the whole polynomial based on the secret and shares of corrupted parties. According to the property of Shamir secret sharing, this adjustment does not affect the output distribution. The output distribution of $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ is identical.

$\mathbf{Hyb}_2$: In this hybrid, $\mathcal{S}$ first randomly samples the hash node leaf $h_{0,i}^{(0)}$ for all $i \in [n]$, then programs the random oracle such that $h_{0,i}^{(0)} = H(m_{0,i}, r_{0,i})$. The rest of the steps remain the same. If $H(m_{0,i}, r_{0,i})$ has been queried by the adversary, $\mathcal{S}$ aborts the simulation. Since the output space of the random oracle is in $\mathbb{F}$ and the adversary can only query the random oracle for $\mathrm{poly}(\kappa)$ times, when $|\mathbb{F}| = 2^{\Omega(\kappa)}$, the probability that this happens is negligible in the security parameter. The output distribution of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ is statistically close.

$\mathbf{Hyb}_3$: In this hybrid, we change the generation of honest party V_i's $f_0(\alpha_i)$. $\mathcal{S}$ first randomly samples $f^{(0)}(\alpha_0)$, then computing $f_0(\alpha_0) = f^{(0)}(\alpha_0) - \sum_{\ell=1}^{N} \mu_0^\ell \cdot f_\ell(\alpha_0)$. Based on the secret $f_0(\alpha_0)$ and corrupted parties' shares, $\mathcal{S}$ randomly samples degree-d polynomial $f_0(x)$ and computes honest V_i's $f_0(\alpha_i)$. Since both $f_0(\alpha_0)$ and $f^{(0)}(\alpha_0)$ are uniformly distributed, the distribution of honest party V_i's $f_0(\alpha_i)$ remains the same, and the output distribution of $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ is identical.

$\mathbf{Hyb}_3$: In this hybrid, we defer the generation of honest party V_i's $m_{0,i}, r_{0,i}$ to the end of the distributing proof phase, and we change the generation of the Merkle root c_0. $\mathcal{S}$ first randomly samples values as the hash of leaves in the Merkle tree, then computes the root c_0 accordingly. Then $\mathcal{S}$ programs the Random oracle $H(\cdot)$ such that $H(m_{0,i}, r_{0,i})$ equals the hash of V_i's leaf in the Merkle tree, where $r_{0,i}$ is the random mask sampled by $\mathcal{S}$ in the operation $\texttt{MT.Com}(\cdot)$. During the simulation of the random oracle, if the following conditions hold, $\mathcal{S}$ aborts the simulation.

- $(m_{0,i}, r_{0,i})$ has been queried by the adversary before it is revealed to the adversary.
- An honest V_i's hash node leaf $h_{0,i}^{(0)}$ has been mapped to another input.

Since the adversary can only query the Random Oracle for $\mathrm{poly}(\kappa)$ times, $r_{0,i}$ is uniformly distributed in $\mathbb{F}$, the probability that $\mathcal{S}$ finds that $H(m_{0,i}, r_{0,i})$ has been queried by the adversary before it is revealed to the adversary is at most $\mathrm{poly}(\kappa)/|\mathbb{F}|$. Since the output space of the random oracle is in $\mathbb{F}$, then the probability that $\mathcal{S}$ finds that an honest V_i's hash node leaf $h_{0,i}^{(0)}$ has been mapped to another input is also at most $\mathrm{poly}(\kappa)/|\mathbb{F}|$. Therefore, when $|\mathbb{F}| = 2^{\Omega(\kappa)}$, the probability that $\mathcal{S}$ will abort is negligible, and the output distribution of $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ is statistically close.

$\mathbf{Hyb}_4$: In this hybrid, for the verification procedure, when an honest R needs to verify an honest V_j's shares, $\mathcal{S}$ will not generate V_j's DZK proof π_j, and only simulate the behavior of sending messages from $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$ to R. When R receives the messages from $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$, $\mathcal{S}$ delivers the verification output from $\mathcal{F}_{\mathsf{dZK}}$ to the honest R.

The difference between $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ is that $(s_1, \ldots, s_N)$ can match its DZK proof π_j, but $(s_1, \ldots, s_N) \neq (f_1(\alpha_j), \ldots, f_N(\alpha_j))$. As a result, the honest V_j will accept incorrect shares received from the environment. Recall that the prover is honest, so an honest V_j's DZK proof is valid for V_j's original shares $f_1(\alpha_j), \ldots, f_N(\alpha_j)$. Let $m'_{0,j} = (f_0(\alpha_j), s_1, \ldots, s_N)$ and $r_{0,j}$ be the randomness used in the commitment of V_j's Merkle tree leaf, then the probability that $s_1, \ldots, s_N$ can also pass the DZK proof is the case that the hash value of $H(m'_{0,j}, r_{0,j}) = H(m_{0,j}, r_{0,j})$, which is negligible in the security parameter if $m'_{0,j} \neq m_{0,j}$. Therefore, the output distribution of $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ is statistically close.

$\mathbf{Hyb}_5$: In this hybrid, we defer the generation of honest V_i's $m_{0,i}, r_{0,i}$ until they are needed to be revealed to a corrupted party. $\mathcal{S}$ continues to check the following two conditions, and will abort the simulation if at least one of them happens:

- $(m_{0,i}, r_{0,i})$ has been queried by the adversary before it is revealed to the adversary.

– An honest V_i's hash node leaf $h_{0,i}^{(0)}$ has been mapped to another input.

The differences between $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ are in two cases:

– If $\mathcal{S}$ needs to reveal honest V_i's $m_{0,i}, r_{0,i}$ to the corrupted party, the difference between $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_4$ is that $\mathcal{S}$ will abort later than that in $\mathbf{Hyb}_4$ if $(m_{0,i}, r_{0,i})$ has been queried by the adversary before it is revealed to the adversary. Following a similar argument in $\mathbf{Hyb}_3$, the output distribution of $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ is statistically close in this case.
– If $\mathcal{S}$ never reveals honest V_i's $m_{0,i}, r_{0,i}$ to the corrupted party, then we never generate V_i's $m_{0,i}, r_{0,i}$ since they are never used. $\mathcal{S}$ will only abort in this case for the second abort condition, where it finds that an honest V_i's hash node leaf $h_{0,i}^{(0)}$ has been mapped to another input. Thus the difference between $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ is that in $\mathbf{Hyb}_4$ $\mathcal{S}$ will also abort if $(m_{0,i}, r_{0,i})$ has been queried by the adversary. Following a similar argument in $\mathbf{Hyb}_3$, the output distribution of $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ is statistically close in this case.

Therefore, the output distribution of $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ is statistically close.

$\mathbf{Hyb}_6$: In this hybrid, in the transfer proof phase, if an honest R accepts a corrupted V_j's DZK proof, but the shares are not the same as the shares received from $\mathcal{F}_{\mathsf{dZK}}$, $\mathcal{S}$ aborts the simulation. This can only happen in the case that the adversary finds a collision for the hash node $h_{0,j}^{(0)}$ with different inputs. Since the adversary can only make $\mathrm{poly}(\kappa)$ times queries to the random oracle, this will happen with negligible probability when $|\mathbb{F}| = 2^{\Omega(\kappa)}$. Therefore, the output distribution of $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ is statistically close.

$\mathbf{Hyb}_7$: In this hybrid, $\mathcal{S}$ no longer generates an honest V_i's $m_{0,i}$. When an honest V_i's DZK proof needed to be revealed to corrupted parties, $\mathcal{S}$ receives $(f_1(\alpha_i), \ldots, f_N(\alpha_i))$ from $\mathcal{F}_{\mathsf{dZK}}$ and then computes $m_{0,i}$ accordingly. The output distribution of $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ is identical.

Note that $\mathbf{Hyb}_7$ corresponds to the idea world, $\mathbf{Hyb}_0$ corresponds to the real world, and $\mathbf{Hyb}_7 \approx_s \mathbf{Hyb}_0$. Therefore, when the adversary can only make $\mathrm{poly}(\kappa)$ queries to the random oracle, Π_{DZK} computes $\mathcal{F}_{\mathsf{dZK}}$ with statistical security when the prover is honest.

Then we consider the case when the prover is corrupted.

Simulator $\mathcal{S}$

For Corrupted Prover P

Distributing Proof Phase

1: $\mathcal{S}$ initializes a set $\mathcal{Q} = \varnothing$, during the simulation of the Random Oracle $H(\cdot)$, for each queries received from the adversary, $\mathcal{S}$ adds the pre-image and hash result pair to $\mathcal{Q}$.
2: In Step 3, when $\mathcal{S}$ receives $\mathcal{F}_{\mathsf{dZK}}$ that an honest party has sent the request $(\mathsf{Init}, \mathsf{DZK}, P)$, it simulates the behavior of sending request $(\mathsf{Init}, \mathsf{PrivSend}, P)$ from this honest party to each $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$. During the simulation of $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$:
   (1). $\mathcal{S}$ first receives $\{\pi_1, \ldots, \pi_n\}$ from P, and sends π_i to V_i for all corrupted V_i.
   (2). When an honest party's request $(\mathsf{Init}, \mathsf{PrivSend}, P)$ has been delivered to $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$, $\mathcal{S}$ forwards it and the identity of this honest party to the adversary.
   (3). When $\mathcal{S}$ learns that $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$ received request $(\mathsf{Init}, \mathsf{PrivSend}, P)$ from $2t+1$ parties, it sends $(\mathsf{Dispersed}, \mathsf{PrivSend})$ to all parties on behalf of $\mathcal{F}_{\mathsf{PrivSend}}^{(i)}$.
3: $\mathcal{S}$ initializes a set $\mathcal{W}_{\mathtt{NoProof}} = \varnothing$. Based on the Merkle commitment $c_k \in \pi$ for all $k \in [0, \tau]$, $\mathcal{S}$ first tries to recursively recover the corresponding Merkle tree based on the set Π. During the reconstruction of the whole Merkle tree, if $\mathcal{S}$ finds that some internal hash node value has not been queried by the adversary, for all parties whose DZK proof relies on the value of this hash node, $\mathcal{S}$ adds the identity to $\mathcal{W}_{\mathtt{NoProof}}$. As a result, upon terminating the reconstruction of the Merkle tree procedure, for each honest $V_i \notin \mathcal{W}_{\mathtt{NoProof}}$, $\mathcal{S}$ learns its shares $\tilde{m}_{0,i}$ and DZK proof $\tilde{\pi}_i$.
4: $\mathcal{S}$ initializes a set $\mathcal{P}'_{\mathtt{Vrfy}} = \varnothing$. For each $V_i \notin \mathcal{W}_{\mathtt{NoProof}}$ (including corrupted V_i), $\mathcal{S}$ will add V_i to $\mathcal{P}'_{\mathtt{Vrfy}}$ if $\pi_i = \tilde{\pi}_i$.
5: Upon terminating the verification for all $V_i \in \mathcal{P}$, according to the size of $\mathcal{P}'_{\mathtt{Vrfy}}$:
   – If $|\mathcal{P} \setminus \mathcal{W}_{\mathtt{NoProof}}| < d+1$, $\mathcal{S}$ interpolates a degree degree-d polynomials $f_0(x), f_1(x), \ldots, f_N(x)$ such that $(f_0(\alpha_i), \ldots, f_N(\alpha_i)) = \tilde{m}_{0,i}$ for all $V_i \in \mathcal{P}'_{\mathtt{Vrfy}}$ and the first $d+1-|\mathcal{P} \setminus \mathcal{W}_{\mathtt{NoProof}}|$ honest parties not in $\mathcal{W}_{\mathtt{NoProof}}$.
   – Otherwise, $\mathcal{S}$ first interpolates degree-d polynomial $f_0(x), f_1(x), \ldots, f_N(x)$ based on the first $d+1$ distinct $V_i \notin \mathcal{W}_{\mathtt{NoProof}}$'s $\tilde{m}_{0,i}$. Then for the rest of honest $V_i \in \mathcal{P}'_{\mathtt{Vrfy}}$, $\mathcal{S}$ checks whether their $\tilde{m}_{0,i}$ lie on $f_0(x), f_1(x), \ldots, f_N(x)$. If not, $\mathcal{S}$ aborts the simulation.

6: $\mathcal{S}$ sends $f_1(x), \ldots, f_N(x)$ and $\mathcal{P}'_{\texttt{Vrfy}}$ to $\mathcal{F}_{\mathsf{dZK}}$. When each honest V_i receives π and π_i, $\mathcal{S}$ delivers the output $(\mathsf{Delivered}, \mathsf{DZK})$ from $\mathcal{F}_{\mathsf{dZK}}$ to V_i.

Verification Phase

1: During the simulation of $\mathcal{F}^{(j)}_{\mathsf{PrivSend}}$, when $\mathcal{S}$ receives the request $(\mathsf{Deliver}, R)$ from a corrupted party on behalf of $\mathcal{F}^{(j)}_{\mathsf{PrivSend}}$, it sends request $(\texttt{PrivVerifyDZK}, j, R)$ to $\mathcal{F}_{\mathsf{dZK}}$ on behalf of this corrupted party.
2: When $\mathcal{S}$ receives $\mathcal{F}_{\mathsf{dZK}}$ that an honest party has sent the request $(\texttt{PrivVerifyDZK}, j, R)$, $\mathcal{S}$ simulates the behavior of sending request $(\mathsf{Deliver}, R)$ from this honest party to $\mathcal{F}^{(j)}_{\mathsf{PrivSend}}$. When this honest party's request $(\mathsf{Deliver}, R)$ has been delivered to $\mathcal{F}^{(j)}_{\mathsf{PrivSend}}$, $\mathcal{S}$ forwards it and the identity of this honest party to the adversary.
3: When $\mathcal{S}$ receives $2t+1$ request $(\mathsf{Deliver}, R)$ on behalf of $\mathcal{F}^{(j)}_{\mathsf{PrivSend}}$:
- If R is corrupted, $\mathcal{S}$ sends π_j to R on behalf of $\mathcal{F}^{(j)}_{\mathsf{PrivSend}}$.
- Otherwise, if $\mathcal{S}$ has delivered the output $(\mathsf{Delivered}, \mathsf{DZK})$ to R before, $\mathcal{S}$ sends π_j from $\mathcal{F}^{(j)}_{\mathsf{PrivSend}}$ to R. Then $\mathcal{S}$ checks whether $V_j \notin \mathcal{P}'_{\texttt{Vrfy}}$ but π_j is a valid DZK proof for the shares $(f_0(\alpha_j), s_1, \ldots, s_N)$. If true, $\mathcal{S}$ aborts the simulation. Otherwise, $\mathcal{S}$ delivers the verification result from $\mathcal{F}_{\mathsf{dZK}}$ to R.

Transfer Proof Phase

1: If both V_j and R are honest, $\mathcal{S}$ simulates the behavior of sending messages between V_j and R. When R receives the messages from V_j, $\mathcal{S}$ delivers the output from $\mathcal{F}_{\mathsf{dZK}}$ to R.
2: If both V_j and R are corrupted, $\mathcal{S}$ does nothing.
3: If V_j is honest and R is corrupted, $\mathcal{S}$ honestly execute V_j to do simulation.
4: If V_j is corrupted and R is honest, $\mathcal{S}$ honestly executes R to do verification. When it succeeds in the verification, $\mathcal{S}$ checks whether the shares received from the corrupted V_j are the shares received from $\mathcal{F}_{\mathsf{dZK}}$ before. If true, $\mathcal{S}$ sends request $(\texttt{TransferDZK}, f_1(\alpha_j), \ldots, f_N(\alpha_j), V_j \to R)$ to $\mathcal{F}_{\mathsf{dZK}}$. Otherwise, $\mathcal{S}$ aborts the simulation.

The hybrid argument for a corrupted prover P is as follows.

$\mathbf{Hyb}_0$: This hybrid corresponds to the real-world execution.

$\mathbf{Hyb}_1$: In this hybrid, $\mathcal{S}$ does the following steps to extract P's input degree-d polynomials $f_1(x), \ldots, f_N(x)$. $\mathcal{S}$ first tries to recursively recover the whole Merkle tree based on the Merkle roots $\{c_k\}_{k\in[0,\tau]}$ broadcast by P, and $\mathcal{S}$ does the thing as above to collect the set $\mathcal{W}_{\texttt{NoProof}}$, where each $V_i \notin \mathcal{W}_{\texttt{NoProof}}$ has a valid DZK proof π_i on its shares. Then $\mathcal{S}$ initializes the set $\mathcal{P}'_{\texttt{Vrfy}} = \varnothing$, and adds each $V_i \notin \mathcal{W}_{\texttt{NoProof}}$ to $\mathcal{P}'_{\texttt{Vrfy}}$ if $\pi_i = \tilde{\pi}_i$.

Upon terminating the verification for all V_i, if $|\mathcal{P} \setminus \mathcal{W}_{\texttt{NoProof}}| < d+1$, $\mathcal{S}$ interpolates degree-d polynomials $f_0(x), \ldots, f_N(x)$ based on $\tilde{m}_{0,i}$ for all $V_i \notin \mathcal{W}_{\texttt{NoProof}}$ and the first $d+1-|\mathcal{P} \setminus \mathcal{W}_{\texttt{NoProof}}|$ honest parties in $\mathcal{W}_{\texttt{NoProof}}$.

The difference between $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ is that when $|\mathcal{P} \setminus \mathcal{W}_{\texttt{NoProof}}| < d+1$, for several honest $V_i \notin \mathcal{P}'_{\texttt{Vrfy}}$, the values $(f_0(\alpha_i), \ldots, f_N(\alpha_i))$ may be different. Since these shares do not have the corresponding DZK proof, and cannot be verifiably opened to honest parties, so these shares are never used in the whole execution. Therefore, this adjustment does not affect the output distribution, and the output distribution of $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ is identical.

$\mathbf{Hyb}_2$: In this hybrid, if $|\mathcal{P} \setminus \mathcal{W}_{\texttt{NoProof}}| \geq d+1$, we let $\mathcal{S}$ interpolate P's degree-d polynomials $f_0(x), f_1(x), \ldots, f_N(x)$ based on the first $d+1$ honest $V_i \in \mathcal{P}'_{\texttt{Vrfy}}$'s $\tilde{m}_{0,i}$. In the following hybrid argument, we will argue that for the rest of honest $V_i \in \mathcal{P}'_{\texttt{Vrfy}}$'s $\tilde{m}_{0,i}$ also lie on $f_0(x), f_1(x), \ldots, f_N(x)$ with overwhelming probability. If not, $\mathcal{S}$ will abort the simulation, and this probability is negligible.

$\mathbf{Hyb}_{2.1}$: In this hybrid, for all honest parties that accept their DZK proof (the honest parties in the set $\mathcal{P}'_{\texttt{Vrfy}}$), $\mathcal{S}$ checks whether (i) their $f^{(\tau)}(\alpha_i)$ lie on a unique degree-d_τ polynomial $f^{(\tau)}(x)$ and (ii) their $g_0^{(\tau)}(\alpha_i), \ldots, g_{u-1}^{(\tau)}(\alpha_i)$ lie on a unique degree-d_τ polynomials. If not, $\mathcal{S}$ aborts the simulation.

For (i), since in the verification, we let P broadcast $f^{(\tau)}(x)$ and each honest P_i will check whether their points lie on this polynomial, this is always true for all honest verifiers in $\mathcal{P}'_{\texttt{Vrfy}}$. For (ii), we first use the first $d_\tau+1$ honest parties' (in $\mathcal{P}'_{\texttt{Vrfy}}$) shares to interpolate the $g_0^{(\tau)}(x), \ldots, g_{u-1}^{(\tau)}(x)$ and check whether the rest of the honest parties' shares lie on these two polynomials. If one honest V_i's shares do not lie on these two polynomials, but it still accepts its DZK proof, this means V_i's shares $\tilde{g}_0^{(\tau)}(\alpha_i), \ldots, \tilde{g}_{u-1}^{(\tau)}(\alpha_i)$ satisfies $\tilde{g}_0^{(\tau)}(\alpha_i) + \mu_\tau \cdot \tilde{g}_1^{(\tau)}(\alpha_i) + \cdots + \mu_\tau^{u-1} \cdot \tilde{g}_{u-1}^{(\tau)}(\alpha_i) = f^{(\tau)}(\alpha_i) = g_0^{(\tau)}(\alpha_i) + \mu_\tau \cdot g_1^{(\tau)}(\alpha_i) + \cdots + \mu_\tau^{u-1} \cdot g_{u-1}^{(\tau)}(\alpha_i)$.

Recall that μ_τ is generated by P by querying $H(c_0, \ldots, c_\tau)$, where c_τ is the Merkle root for the Merkle tree in the τth iteration. Since this V_i accepts its DZK proof, then it must hold a valid Merkle proof $\mathtt{op}_{k,i}$ for its shares $\tilde{g}_0^{(\tau)}(\alpha_i), \ldots, \tilde{g}_{u-1}^{(\tau)}(\alpha_i)$. Then this means the corrupted P can only first fix all

V_i's $\tilde{g}_0^{(\tau)}(\alpha_i), \tilde{g}_1^{(\tau)}(\alpha_i)$ then learn the challenge μ_τ, except with negligible probability that it can find hash collision within poly(κ) queries to the random oracle. Therefore, since $\mu_k \in \mathbb{G}$ the probability that for a random μ_τ such that the relation $\tilde{g}_0^{(\tau)}(\alpha_i) + \cdots + \mu_\tau^{u-1} \cdot \tilde{g}_{u-1}^{(\tau)}(\alpha_i) = g_0^{(\tau)}(\alpha_i) + \cdots + \mu_\tau^{u-1} \cdot g_{u-1}^{(\tau)}(\alpha_i)$ holds is at most $u/|\mathbb{G}| \approx 2^{-(n+\kappa)}$, which is also negligible probability.

Finally, we take the union bound for all subsets of honest parties to interpolate $g_0^{(\tau)}(x), \ldots, g_{u-1}^{(\tau)}(x)$ and for all V_i's verification. The total probability that $\mathcal{S}$ will abort is at most $n/2^\kappa$, which is negligible in the security parameter κ. Therefore, the output distribution of $\mathbf{Hyb}_{2.1}$ and $\mathbf{Hyb}_2$ is statistically close.

From $k = 1$ to $\tau - 1$, we consider the following hybrid arguments.

$\mathbf{Hyb}_{2.k}$: In this hybrid, for all honest parties that accept their DZK proof (the honest parties in the set $\mathcal{P}'_{\mathtt{Vrfy}}$), $\mathcal{S}$ checks whether (i) their $f^{(\tau-k)}(\alpha_i)$ lie on a unique degree-$d_{\tau-k}$ polynomial $f^{(\tau-k)}(x)$ and (ii) their $g_0^{(\tau-k)}(\alpha_i), \ldots, g_{u-1}^{(\tau-k)}(\alpha_i)$ lie on a unique degree-$d_{\tau-k}$ polynomials. If not, $\mathcal{S}$ aborts the simulation.

For (i), since $f^{(\tau-k)} = g_0^{(\tau-k+1)} + \sum_{\ell \in [u-1]} x^{d_{\tau-k} - \ell \cdot d_{\tau-k+1}} \cdot g_{u-\ell}^{(\tau-k+1)}$, and in $\mathbf{Hyb}_{2.k-1}$ we have shown that all these honest V_i's $g_0^{(\tau-k+1)}(\alpha_i), \ldots, g_{u-1}^{(\tau-k+1)}(\alpha_i)$ lie on unique degree-$d_{\tau-k+1}$ polynomials $g_0^{(\tau-k+1)}(x), \ldots, g_{u-1}^{(\tau-k+1)}(x)$, then this means all their shares lie on the unique degree-$d_{\tau-k}$ polynomial $f^{(\tau-k)}(x)$.

For (ii), following a similar argument as in $\mathbf{Hyb}_{2.1}$, we know the probability that $\mathcal{S}$ will abort is also negligible. Therefore, the output distribution of $\mathbf{Hyb}_{2.k}$ and $\mathbf{Hyb}_{2.k-1}$ is statistically close.

$\mathbf{Hyb}_{2.\tau}$: In this hybrid, for all honest parties that accept their DZK proof (the honest parties in the set $\mathcal{P}'_{\mathtt{Vrfy}}$), $\mathcal{S}$ checks whether their $f^{(0)}(\alpha_i)$ lie on a unique degree-d ($d = d_0$) polynomial $f^{(0)}(x)$. Since $f^{(0)}(x) = g_0^{(1)} + \sum_{\ell \in [u-1]} x^{d_0 - \ell \cdot d_1} \cdot g_{u-\ell}^{(1)}$, and in $\mathbf{Hyb}_{2.\tau-1}$ we have shown that all these honest V_i's $g_0^{(1)}(\alpha_i), \ldots, g_{u-1}^{(1)}(\alpha_i)$ lie on unique degree-d_1 polynomials $g_0^{(1)}(x), \ldots, g_{u-1}^{(1)}(x)$, then this means all their shares lie on the unique degree-d polynomial $f^{(0)}(x)$. Therefore, the output distribution of $\mathbf{Hyb}_{2.\tau}$ and $\mathbf{Hyb}_{2.\tau-1}$ is identical.

$\mathbf{Hyb}_3$: In this hybrid, for all honest parties that accept their DZK proof (the honest parties in the set $\mathcal{P}'_{\mathtt{Vrfy}}$), $\mathcal{S}$ checks whether their $\tilde{m}_{0,i}$ lie on a unique degree-d polynomials $f_0(x), f_1(x), \ldots, f_N(x)$. If not, $\mathcal{S}$ aborts the simulation.

We first interpolate the degree-d polynomial $f_0(x), f_1(x), \ldots, f_N(x)$ based on a subset of $d+1$ honest parties' shares, where these honest parties are in the set $\mathcal{P}'_{\mathtt{Vrfy}}$. Then we consider the probability that an honest V_i's $f^{(0)}(\alpha_i)$ lies on the unique degree-d polynomial $f^{(0)}(x)$, but its $\tilde{m}_{0,i} \neq (f_0(\alpha_i), \ldots, f_N(\alpha_i))$. We parse $\tilde{m}_{0,i}$ into $(f'_0(\alpha_i), \ldots, f'_N(\alpha_i))$, then this means the following equation holds for a random challenge μ_0:

$$\sum_{\ell=0}^{N} \mu_0^\ell \cdot f_\ell(\alpha_i) = \sum_{\ell=0}^{N} \mu_0^\ell \cdot f'_\ell(\alpha_i).$$

According to the Schwartz-Zipple lemma, the probability that the above equation holds is at most $N/|\mathbb{G}| = \frac{N}{2^{n+\kappa}}$. Then we take the union bound for all subsets of $d+1$ honest parties that can interpolate the degree-d polynomials $f_0(x), f_1(x), \ldots, f_N(x)$, and all V_i's $\tilde{m}_{0,i}$, the total probability is at most $\frac{Nn}{2^\kappa}$, which is negligible in the security parameter.

Therefore, $\mathcal{S}$ will abort the simulation with negligible probability, and the output distribution of $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_{2.\tau}$ is statistically close.

$\mathbf{Hyb}_4$: In this hybrid, for the verification procedure, when an honest R needs to verify V_j's shares, if $V_j \in \mathcal{P}'_{\mathtt{Vrfy}}$, when R receives the messages from $\mathcal{F}_{\mathsf{PrivSend}}^{(j)}$, $\mathcal{S}$ delivers the verification output from $\mathcal{F}_{\mathsf{dZK}}$ to the honest R.

The difference between $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ is that for an honest $V_j \in \mathcal{P}'_{\mathtt{Vrfy}}$, $(s_1, \ldots, s_N)$ can match its DZK proof π_j, but $(s_1, \ldots, s_N) \neq (f_1(\alpha_j), \ldots, f_N(\alpha_j))$. As a result, the honest V_j will accept incorrect shares received from the environment. Recall that if $V_j \in \mathcal{P}'_{\mathtt{Vrfy}}$, then its DZK proof is valid for V_j's original shares $f_1(\alpha_j), \ldots, f_N(\alpha_j)$. Let $m'_{0,j} = (f_0(\alpha_j), s_1, \ldots, s_N)$ and $r_{0,j}$ be the randomness used in the commitment of V_j's Merkle tree leaf, then the probability that $s_1, \ldots, s_N$ can also pass the DZK proof is the case that the hash value of $H(m'_{0,j}, r_{0,j}) = H(m_{0,j}, r_{0,j})$, which is negligible in the security parameter if $m'_{0,j} \neq m_{0,j}$. Therefore, the output distribution of $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ is statistically close.

$\mathbf{Hyb}_5$: In this hybrid, for verification procedure, when an honest R needs to verify V_j's shares, if $V_j \notin \mathcal{P}'_{\texttt{Vrfy}}$, $\mathcal{S}$ checks whether their π_j is a valid DZK proof for the shares $(f_0(\alpha_j), s_1, \ldots, s_N)$. If true, $\mathcal{S}$ aborts the simulation.

The difference between $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ is in two cases:

- For $V_j \in \mathcal{W}_{\texttt{NoProof}}$, some hash nodes in their π_j are never queried by the adversary during the simulation of the random oracle. Then, when $\mathcal{S}$ does the verification, the probability that these values chosen by the adversary happen to be a valid mapping is negligible since the output space of the random oracle is at least of size $2^{\Omega(\kappa)}$.
- For $V_j \notin \mathcal{W}_{\texttt{NoProof}} \cup \mathcal{P}'_{\texttt{Vrfy}}$, the adversary distributed incorrect DZK proof $\pi_j \neq \tilde{\pi}_j$ for them through $\mathcal{F}^{(j)}_{\mathsf{PrivSend}}$. Then if $\mathcal{S}$ can succeed in the verification, then all hash values computed from $\pi_j, \tilde{\pi}_j$ and $(s_1, \ldots, s_N)$ are the same. This can only happen when the adversary finds a collision within $\text{poly}(\kappa)$ queries to the random oracle, which is negligible in the security parameter when $|\mathbb{F}| = 2^{\Omega(\kappa)}$.

Therefore, the output distribution of $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ is statistically close.

$\mathbf{Hyb}_6$: In this hybrid, in the transfer proof phase, if an honest R accepts a corrupted V_j's DZK proof, but the shares are not the same as the shares received from $\mathcal{F}_{\mathsf{dZK}}$, $\mathcal{S}$ aborts the simulation. This can only happen in the case that the adversary finds a collision for the hash node $h^{(0)}_{0,j}$ with different inputs. Since the adversary can only make $\text{poly}(\kappa)$ times queries to the random oracle, this will happen with negligible probability when $|\mathbb{F}| = 2^{\Omega(\kappa)}$. Therefore, the output distribution of $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ is statistically close.

Note that $\mathbf{Hyb}_6$ corresponds to the idea world, $\mathbf{Hyb}_0$ corresponds to the real world, and $\mathbf{Hyb}_6 \approx_s \mathbf{Hyb}_0$. Therefore, when the adversary can only make $\text{poly}(\kappa)$ queries to the random oracle, Π_{DZK} computes $\mathcal{F}_{\mathsf{dZK}}$ with statistical security when the prover is corrupted.

C Verifiable Private Send

Recall the functionality of $\mathcal{F}_{\mathsf{PrivSend}}$ in section 2.1, the author in [Mom24] assumes the random oracle and gives a construction of it based on the primitive *Asynchronous Verifiable Information Dispersal* (AVID) defined below. AVID also contains two phases, a disperse phase and a retrieve phase. For L bits message, the author in [ADD+22] realizes $\mathcal{F}_{\mathsf{AVID}}$ with communication complexity $\mathcal{O}(L + n^2)$ in the disperse phase and $\mathcal{O}(L + n)$ in the retrieve phase for each receiver.

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

$\mathcal{F}_{\mathsf{AVID}}$ proceeds as follows, running with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, a dealer D and an adversary $\mathcal{S}$.

Disperse Phase

1: Upon receiving a message M from D, for the first time of such a request, the trusted party records m and reveals it to $\mathcal{S}$.

2: Upon receiving the request $(\mathsf{Init}, \mathsf{AVID}, D)$ from an honest party, the functionality forwards it together with the identity of this party to $\mathcal{S}$. When the functionality receives this request from $2t+1$ distinct parties and the message m from D, the functionality sends a request-based delayed output $(\mathsf{Dispersed}, \mathsf{AVID})$ to all parties.

Retrieve Phase

3: Upon receiving the request $(\mathsf{Deliver}, R)$ from an honest party, the functionality forwards this request together with the identity of this honest party to $\mathcal{S}$. When it receives this request from $t+1$ distinct parties, if it has sent $(\mathsf{Dispersed}, \mathsf{AVID})$ to all parties, the trusted party sends a request-based delayed output m to R.

To realize his construction, he also assumes that between the sender and receiver, they know a symmetric key key. To prepare these symmetric keys, the authors in [SS24] introduce a key distribution protocol. In the following, we introduce a more simplified construction $\Pi_{\texttt{PrepKey}}$, which is sufficient for our construction. The amortized cost of the Disperse Phase for $\mathcal{O}(n)$ dealers is $\mathcal{O}(n^2)$ elements, and the Reveal Phase for each receiver is $\mathcal{O}(n)$ elements.

Protocol $\Pi_{\texttt{PrepKey}}$

Public Input: Field elements $(\alpha_0, \ldots, \alpha_n)$, each of them is in the field $\mathbb{F}$, and $|\mathbb{F}| = 2^{\Theta(\kappa)}$, where κ is the security parameter.
The dealer D takes $\mathsf{key} \in \mathbb{F}$ as input.

Disperse Phase

1: The dealer D randomly samples a value $r \in \mathbb{F}$ and two random degree-t sharings $[\mathsf{key}]_t, [r]_t$. Then D distributes the sharings $[\mathsf{key}]_t, [r]_t$ to all parties.
2: For each $i \in [n]$, let key_i, r_i denote P_i's share of $[\mathsf{key}]_t, [r]_t$, then D randomly computes $c_i = H(\mathsf{key}_i, r_i)$. D reliable broadcasts $c_1, \ldots, c_n$ and sends r_i to each party P_i.
3: For each party P_i, it randomly samples a pair of value (m_i, e_i). Then P_i reliable broadcasts $d_i = H(m_i, e_i)$. This step can be executed for a distinct dealer in parallel.
4: Upon receiving key_i, r_i and $c_1, \ldots, c_n$ from D, P_i accepts its key_i, r_i if $c_i = H(\mathsf{key}_i, r_i)$. If P_i accepts its share, it sends m_i, e_i to D.
5: D initializes a set $\mathcal{M} = \varnothing$. Upon receiving m_i, e_i from P_i and $d_i = H(m_i, e_i)$, D adds (P_i, m_i, e_i) to $\mathcal{M}$. Upon getting $\mathcal{M} = 2t + 1$, D reliable broadcasts $\mathcal{M}$.
6: Upon receiving $\mathcal{M}$ from D, all parties accept it if $d_i = H(m_i, e_i)$ for all $P_i \in \mathcal{M}$. Once they accept the set $\mathcal{M}$, they terminate.

All parties agree on a receiver R and execute the following step.

Reveal Phase

1: Each $P_i \in \mathcal{M}$ sends key_i, r_i to the receiver R.
2: Upon receiving key_i, r_i from P_i, R accepts it if $c_i = H(\mathsf{key}_i, r_i)$. Upon accepting $t + 1$ distinct key_i, r_i, R interpolates the whole sharing $[\mathsf{key}]_t, [r]_t$. Then R outputs key if $c_i = H(\mathsf{key}_i, r_i)$ for all $i \in [n]$ and $\perp$ otherwise.

Based on the prepared symmetric key, we give the construction of $\Pi_{\texttt{PrivSend}}$ as follows.

Protocol $\Pi_{\texttt{PrivSend}}$

Public Input: Field elements $(\alpha_0, \ldots, \alpha_n)$, each of them is in the field $\mathbb{F}$, and $|\mathbb{F}| = 2^{\Theta(\kappa)}$, where κ is the security parameter.

Disperse Phase

1: The sender takes a message m as input, then he randomly samples a value $\mathsf{key} \in \mathbb{F}$ and computes $c = m \oplus H(\mathsf{key})$. Then he sends c to $\mathcal{F}_{\mathsf{AVID}}$.
2: D leads an instance of $\Pi_{\texttt{PrepKey}}$ and take key as input.
3: When all parties receive $(\mathsf{Dispersed}, \mathsf{AVID})$ from $\mathcal{F}_{\mathsf{AVID}}$ and terminate the Disperse Phase of $\Pi_{\texttt{PrepKey}}$, they terminate with $(\mathsf{Dispersed}, \mathsf{PrivSend})$.

Upon receiving the request $(\mathsf{Deliver}, R)$ from the environment, all parties proceed with the following steps.

Reveal Phase

1: All parties send $(\texttt{Retrieve}, R)$ to $\mathcal{F}_{\mathsf{AVID}}$ and invoke the Reveal Phase of $\Pi_{\texttt{PrepKey}}$ with the identity of R.
2: When the party R receives c from $\mathcal{F}_{\mathsf{AVID}}$ and key from $\mathcal{F}_{\mathsf{AVSS}}$, he outputs $m = c \oplus H(\mathsf{key})$.

Lemma 3. *Assuming random oracles, protocol $\Pi_{\texttt{PrivSend}}$ securely computes $\mathcal{F}_{\mathsf{PrivSend}}$ against a fully malicious adversary $\mathcal{A}$ who corrupts at most $t < n/3$ parties.*

We first analyze the cost of $\Pi_{\texttt{PrivSend}}$ and then prove Lemma 3. Here we exclude the cost for the preparation and reconstruction of key, which will be shown in section **??**.

Communication Complexity. Based on the construction of AVID in [ADD+22], in each disperse phase, it requires $\mathcal{O}(|m| + n^2)$ field elements plus the Disperse phase of $\Pi_{\texttt{PrepKey}}$ (which is amortized $\mathcal{O}(n^2)$ elements for $\mathcal{O}(n)$ dealers). In the reveal phase, it requires $\mathcal{O}(|m| + n)$ field elements plus the Disperse phase of $\Pi_{\texttt{PrepKey}}$ (which is $\mathcal{O}(n)$ elements).

Round Complexity. In the disperse phase and reveal phase, based on the construction of AVID in [ADD+22], it is $R_{\mathsf{bc}} + 1 = 4 + 1 = 5$ and 1, respectively.

D Secure and Cost Analysis of ACSS-Id

D.1 Cost Analysis

We analyze the communication, round, and computation complexity as follows.

Communication Complexity. In the sharing phase, the dealer invoke $t+1$ instances of $\mathcal{F}_{\mathsf{dZK}}$ and the *Disperse Phase* of n instances of $\mathcal{F}_{\mathsf{PrivSend}}$, and the *Reveal Phase* of these n instances of $\mathcal{F}_{\mathsf{PrivSend}}$ for each party. For $\mathcal{F}_{\mathsf{dZK}}$, we set the compress factor $u = 2$, and the costs for this step are bounded by $\mathcal{O}(n^3)$ field elements. For $\mathcal{F}_{\mathsf{PrivSend}}$, it costs $\mathcal{O}(Ln + n^3)$ communication complexity in total. All parties invoke an instance of $\mathcal{F}_{\mathsf{ra}}$, which requires $\mathcal{O}(n^2)$ field elements. Therefore, the total costs in the sharing phase are $\mathcal{O}(Ln + n^3)$ field elements.

In the accusation phase, all parties execute the *Reveal Phase* of $\mathcal{F}^{(i)}_{\mathsf{PrivSend}}$, which requires $\mathcal{O}(Ln + n^3)$ field elements in total.

In the public reconstruction phase, all parties execute the transfer proof phase for each pair of parties. The total communication complexity is bounded by $\mathcal{O}(Ln + n^3)$ field elements.

Round Complexity. Denote the round complexity of reliably broadcast as $R_{\mathtt{bc}}$, which can be realized by 4 by the construction in [DXR21]. We recall the round complexity of each functionality.

- For $\mathcal{F}_{\mathsf{dZK}}$, it is a non-interactive protocol and the round complexity is $R_{\mathtt{bc}}$.
- For $\mathcal{F}_{\mathsf{PrivSend}}$, excluding the preparation of the symmetric key, it is 5 in the *Disperse Phase* and 1 in the *Reveal Phase*.

Therefore, in the sharing phase, the round complexity is $\mathtt{max}\{R_{\mathtt{bc}} + R_{\mathtt{ra}}, 5\} = 6$. In the accusation phase, the round complexity is 1. In the public reconstruction phase, the round complexity is 2.

Computation Complexity For disseminating shares with $\mathcal{F}_{\mathsf{PrivSend}}$, the dealer uses a symmetric-key Pseudorandom Function (PRF) to encrypt shares. This step requires $\mathcal{O}(nL)$ bits of PRF generation and requires $\mathcal{O}(nL)$ lightweight cryptographic operations. The dealer also generates $\mathcal{O}(n^2)$ commitments for $\mathcal{O}(n)$ $\mathcal{F}_{\mathsf{PrivSend}}$ instances. In addition, the dealer also generates $\mathcal{O}(n^2)$ commitments for $\mathcal{F}_{\mathsf{dZK}}$ and each commitment is generated on a message of size $\mathcal{O}(\frac{L}{n})$ field elements. Even though we count only the number of Random Oracle invocations in our cost, in practice, cryptographic Hash functions like `SHA256` have a computational complexity proportional to the input size. Overall, the commitment generation also has a computational cost of $\mathcal{O}(nL)$ lightweight cryptographic operations.

D.2 Security Analysis

We prove lemma 1 and start with constructing the ideal adversary $\mathcal{S}$ as follows. We first consider the case of the honest dealer.

Simulator $\mathcal{S}$

For honest dealer

Distribution Phase

1: $\mathcal{S}$ receives shares of corrupted parties from $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$. Then for each corrupted party P_j, $\mathcal{S}$ computes $g_1(\alpha_j, y), \ldots, g_{L'}(\alpha_j, y)$.

2: $\mathcal{S}$ simulates each $\mathcal{F}^{(i)}_{\mathsf{dZK}}$ as follows.

1. $\mathcal{S}$ honestly simulates the Distributing Proof Phase of $\mathcal{F}^{(i)}_{\mathsf{dZK}}$.
2. In the Verification phase, when $\mathcal{S}$ receives request $(\mathtt{PrivVerifyDZK}, s_1, \ldots, s_N, k)$ from a corrupted party, if k is an index of a corrupted party and $s_1, \ldots, s_N$ equal $g_1(\alpha_k, \alpha_i), \ldots, g_{L'}(\alpha_k, \alpha_i)$, $\mathcal{S}$ sends `true` to this corrupted party as the output of $\mathcal{F}^{(i)}_{\mathsf{dZK}}$. Otherwise, $\mathcal{S}$ sends `false` to this corrupted party as the output of $\mathcal{F}^{(i)}_{\mathsf{dZK}}$.

3: $\mathcal{S}$ honestly emulates each $\mathcal{F}^{(i)}_{\mathsf{PrivSend}}$.

Verification Phase

4: $\mathcal{S}$ does nothing. When each honest party gets his shares, $\mathcal{S}$ considers this honest party to accept his shares.

Termination Phase

5: When an honest party accepts his shares, $\mathcal{S}$ sets this honest party's input for $\mathcal{F}_{\mathsf{ra}}$ as 1. Then $\mathcal{S}$ honestly emulate $\mathcal{F}_{\mathsf{ra}}$.

Accusation Phase

6: For request $(\texttt{Open-Proof}, P_i)$ received from the environment, if i is the index of a honest party, $\mathcal{S}$ aborts the simulation. Otherwise, $\mathcal{S}$ proceeds.

7: $\mathcal{S}$ sends $\texttt{Reveal}$ to $\mathcal{F}^{(i)}_{\mathsf{PrivSend}}$ on behalf of each honest party. Then $\mathcal{S}$ computes and sends m_i to each party on behalf of $\mathcal{F}^{(i)}_{\mathsf{PrivSend}}$.

Public Reconstruction Phase

8: $\mathcal{S}$ sends request $\texttt{Public-Recon}$ to $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ on behalf of each honest party and receives $f_1(\alpha_0), \ldots, f_L(\alpha_0)$. Then $\mathcal{S}$ computes $g_1(\alpha_0, y), \ldots, g_{L'}(\alpha_0, y)$.

9: For each $\ell \in [L']$, $\mathcal{S}$ randomly samples a degree-(t,t) bivariate polynomial $g_\ell(x,y)$ based on $g_\ell(\alpha_0, y)$ and $\{g_\ell(\alpha_j, y)\}_{j \in Corr}$. Then $\mathcal{S}$ honestly execute each honest party P_i with his $g_1(\alpha_i, y), \ldots, g_{L'}(\alpha_i, y)$. $\mathcal{S}$ also honestly emulates each $\mathcal{F}^{(i)}_{\mathsf{dZK}}$ with inputs $g_1(x, \alpha_i), \ldots, g_{L'}(x, \alpha_i)$.

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

The hybrid arguments are as follows.

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

$\mathbf{Hyb}_1$: In this hybrid, $\mathcal{S}$ first generate corrupted parties' shares. Then he samples honest parties' shares based on the secret $f_1(\alpha_0), \ldots, f_L(\alpha_0)$ and shares of corrupted parties. Based on the property of the Shamir sharing scheme, this makes no difference. The distributions of $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are identical.

$\mathbf{Hyb}_2$: In this hybrid, $\mathcal{S}$ change the simulation of each $\mathcal{F}^{(i)}_{\mathsf{dZK}}$ as above. The difference between $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ is $\mathcal{S}$ will reply $\texttt{false}$ when the adversary sends request $(\texttt{PrivVerifyDZK}, \{g_\ell(\alpha_j, \alpha_i)\}_{\ell \in [L']}, j)$ to $\mathcal{F}^{(i)}_{\mathsf{dZK}}$ and j is the index of an honest party. That means the adversary can correctly guess the shares of honest parties, which is negligible in the security parameter. Therefore, the distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are statistically close.

$\mathbf{Hyb}_3$: In this hybrid, during the accusation phase, if the identity of P_i is an honest party, $\mathcal{S}$ aborts the simulation. Since we do not allow this case to happen in the real world, the distributions of $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ are still identical.

$\mathbf{Hyb}_4$: In this hybrid, $\mathcal{S}$ no longer generates honest parties' shares since they are never used. Then $\mathcal{S}$ also does not require the dealer's secrets $f_1(\alpha_0), \ldots, f_L(\alpha_0)$ in the sharing phase. He will learn them from $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ during the public reconstruction phase. The distributions of $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ are identical.

Note that $\mathbf{Hyb}_4$ corresponds to the ideal world, then $\Pi_{\texttt{ACSS-Id}}$ securely computes $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ when the dealer is honest.

Then we consider the case of the corrupted dealer.

Simulator $\mathcal{S}$

For Corrupted dealer

Distribution Phase

1: $\mathcal{S}$ honestly emulates each $\mathcal{F}^{(i)}_{\mathsf{PrivSend}}$ and $\mathcal{F}^{(i)}_{\mathsf{dZK}}$.

Verification Phase

2: For each honest party P_i, $\mathcal{S}$ gets $g_1(\alpha_i, y), \ldots, g_{L'}(\alpha_i, y)$. Then $\mathcal{S}$ does the following check on behalf of P_i.
- For $\ell \in [L']$, each $g_\ell(\alpha_i, y)$ is a degree-t polynomial.
- For each $j \in [n]$, send $(\texttt{VerifyDZK}, g_1(\alpha_i, \alpha_j), \ldots, g_{L'}(\alpha_i, \alpha_j), \alpha_i)$ to $\mathcal{F}^{(j)}_{\mathsf{dZK}}$ and the output is $\texttt{true}$.

If so, $\mathcal{S}$ considers P_i accepts his $g_1(\alpha_i, y), \ldots, g_{L'}(\alpha_i, y)$. Otherwise, $\mathcal{S}$ sends $(\texttt{proof}, P_i)$ to $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$.

Termination Phase

3: If all parties terminate $\mathcal{F}_{\mathsf{ra}}$ with 1, let $\mathcal{H}$ denote the set of the first $t+1$ honest parties who have accepted their $g_1(\alpha_i, y), \ldots, g_{L'}(\alpha_i, y)$. $\mathcal{S}$ uses $\{g_1(\alpha_i, y), \ldots, g_{L'}(\alpha_i, y)\}_{i \in \mathcal{H}}$ to reconstruct $g_1(x, y), \ldots, g_{L'}(x, y)$ and gets degree-t polynomials $f_1(x), \ldots, f_L(x)$.

4: $\mathcal{S}$ sends $f_1(x), \ldots, f_L(x)$ to $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ and deliveries the output from $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ to each honest party.

Accusation Phase

5: $\mathcal{S}$ follows the protocol and executes each honest party honestly.

Public Reconstruction Phase

6: $\mathcal{S}$ follows the protocol and executes each honest party honestly.

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

The hybrid arguments are as follows.

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

Hyb$_1$: In this hybrid, $\mathcal{S}$ uses $\{g_1(\alpha_i, y), \ldots, g_{L'}(\alpha_i, y)\}_{i\in\mathcal{H}}$ to reconstruct $g_1(x, y), \ldots, g_{L'}(x, y)$. For each honest $P_i \notin \mathcal{H}$, if he accepts his $g'_1(\alpha_i, y), \ldots, g'_{L'}(\alpha_i, y)$ but there exists $\ell \in [L']$ such that $g'_\ell(\alpha_i, y) \neq g_\ell(\alpha_i, y)$, $\mathcal{S}$ aborts the simulation.

Since each honest party $P_i \in \mathcal{H}$ accepts their $g_1(\alpha_i, y), \ldots, g_{L'}(\alpha_i, y)$, then their outputs of each $\mathcal{F}^{(j)}_{\mathsf{dZK}}$ for $j \in [n]$ are `true`. Then for each $j \in [n]$, these $t+1$ honest parties' $g_1(\alpha_i, \alpha_j), \ldots, g_{L'}(\alpha_i, \alpha_j)$ determines the dealer's degree-t input polynomial $g_1(x, \alpha_j), \ldots, g_{L'}(x, \alpha_j)$. For a honest party $P_i \notin \mathcal{H}$, if he also accepts his $g'_1(\alpha_i, y), \ldots, g'_{L'}(\alpha_i, y)$, then for each $j \in [n]$ and $\ell \in [L']$, $g'_\ell(\alpha_i, \alpha_j) = g_\ell(\alpha_i, \alpha_j)$, which implies $g'_\ell(\alpha_i, y) = g_\ell(\alpha_i, y)$. Therefore, $\mathcal{S}$ will never abort the simulation. The distributions between **Hyb**$_1$ and **Hyb**$_0$ are identical.

Hyb$_2$: In this hybrid, $\mathcal{S}$ sends $f_1(x), \ldots, f_L(x)$ to $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ and let it deliver the output to each party. The distributions between **Hyb**$_2$ and **Hyb**$_1$ are identical.

Note that **Hyb**$_2$ corresponds to the ideal world, then $\Pi_{\texttt{ACSS-Id}}$ securely computes $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ when the dealer is corrupted.

E Secure and Cost Analysis of Sh2t-Id

E.1 Cost Analysis

We analyze the communication and round complexity as follows.

Communication Complexity. In the sharing phase, it contains n instances of $\mathcal{F}_{\mathsf{PrivSend}}$ (each message size is $\mathcal{O}(L + n^2)$) and an instance of reliably broadcast (for n^2 field elements), results in $\mathcal{O}(Ln + n^4)$ communication cost in total. The accusation phase is the same as $\Pi_{\texttt{ACSS-Id}}$, which is $\mathcal{O}(Ln + n^3)$ field elements.

In the agreement accusation phase, it is $\mathcal{O}(n^2)$ field elements. In the private reconstruction and verification phase, for one $j \in [n]$, when all parties reconstruct $f^{(k)}_1(x), \ldots, f^{(k)}_{L'}(x)$ to party P_j, it requires $\mathcal{O}(L/n + n)$ field elements.

Round Complexity. In the sharing and accusation phase, it is the same as $\Pi_{\texttt{ACSS-Id}}$, which are 6 and 2, respectively. In the agreement accusation phase, it is $R_{\texttt{ra}}$=2. In the private reconstruction and verification phase, it is 1.

Computation Complexity For disseminating shares with $\mathcal{F}_{\mathsf{PrivSend}}$, the complexity is the same as $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ with $\mathcal{O}(nL)$ lightweight cryptographic operations. In addition, dealer also generates $\mathcal{O}(n^3)$ commitments and each commitment is generated on a message of size $\mathcal{O}(\frac{L}{n^2})$ field elements. Overall, the commitment generation also has a computational cost of $\mathcal{O}(nL)$ of lightweight cryptographic operations.

E.2 Security Analysis

We prove lemma 2 and start with constructing the ideal adversary $\mathcal{S}$ as follows. We first consider the case of the honest dealer.

Simulator $\mathcal{S}$

For honest dealer

Distribution Phase

1: $\mathcal{S}$ receives shares of corrupted parties from $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$. For each corrupted party P_i, he randomly samples $\tilde{f}_1(\alpha_i), \ldots, \tilde{f}_L(\alpha_i)$ and computes $m^{(k)}_*[i], \tilde{m}^{(k)}_*[i]$ for all $k \in [n^2]$.

2: For each P_i, if P_i is honest, $\mathcal{S}$ randomly samples values as $\{h^{(k)}_*[i]\}_{k\in[n^2]}$ and then computes $\mathsf{Com}[i][j]$ for all $j \in [n]$. If P_i is corrupted, $\mathcal{S}$ follows the protocol to compute each $\mathsf{Com}[i][j]$.

3: $\mathcal{S}$ honestly emulates each $\mathcal{F}^{(i)}_{\mathsf{PrivSend}}$ and reliably broadcasts Com on behalf of D.

Verification, Termination and Accusation Phase

4: $\mathcal{S}$ does the same simulation as the construction in $\Pi_{\texttt{ACSS-Id}}$.

Agreement Accusation Phase

5: $\mathcal{S}$ honest execute each honest party. When all parties terminate with $(\texttt{Accusation}, D, P_i)$, if P_i is the identity of an honest party, $\mathcal{S}$ aborts the simulation. Otherwise, $\mathcal{S}$ records the identity of P_i.

Private Reconstruction and Verification Phase

6: If $R = P_j$ is honest, $\mathcal{S}$ does the following simulation.

- If P_i is an honest party, $\mathcal{S}$ considers R accepts P_i's shares when his messages are delivered.
- If P_i is an corrupted party, when $\mathcal{S}$ receives $m'^{(k)}_*[i], \tilde{m'}^{(k)}_*[i]$ from P_i, he checks whether $h^{(k)}_*[i] = H(m'^{(k)}_*[i], \tilde{m'}^{(k)}_*[i])$. If true but $m'^{(k)}_*[i], \tilde{m'}^{(k)}_*[i]$ are not equal to $m^{(k)}_*[i], \tilde{m}^{(k)}_*[i]$, $\mathcal{S}$ aborts the simulation.

If $\mathcal{S}$ records the identity of a corrupted party P_i during the Agreement Accusation Phase before R accepts shares from $2t+1$ parties, $\mathcal{S}$ sends $(\texttt{Accusation}, D, P_i)$ to $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$.

7: If $R = P_j$ is corrupted, $\mathcal{S}$ receives $f^{(k)}_1(x), \ldots, f^{(k)}_{L'}(x)$ from $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$. For each honest party P_i, $\mathcal{S}$ does the following.
- Randomly sample $\tilde{f}^{(k)}_1(\alpha_i), \ldots, \tilde{f}^{(k)}_{L'}(\alpha_i)$ based on shares of corrupted parties, then compute $m^{(k)}_*[i], \tilde{m}^{(k)}_*[i]$ and map $H(m^{(k)}_*[i], \tilde{m}^{(k)}_*[i])$ to $h^{(k)}_*[i]$. If $h^{(k)}_*[i]$ has been mapped to other inputs queried by the adversary or $H(m^{(k)}_*[i], \tilde{m}^{(k)}_*[i])$ has been mapped to other output, $\mathcal{S}$ aborts the simulation. Otherwise, $\mathcal{S}$ sends $m^{(k)}_*[i], \tilde{m}^{(k)}_*[i]$ to R on behalf of P_i.

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

The hybrid arguments are as follows.

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

$\mathbf{Hyb}_1$: In this hybrid, $\mathcal{S}$ first generates shares of corrupted parties, then he samples shares of honest parties based on the dealer's secrets and the shares of corrupted parties. Based on the Shamir secret sharing scheme, this makes no difference. Therefore, the distributions of $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are identical.

$\mathbf{Hyb}_2$: In this hybrid, we delay the generation of honest parties' shares. For honest party P_i's $\mathsf{Com}[i][j]$, $\mathcal{S}$ randomly samples values as $\{h^{(k)}_*[i]\}_{k\in[n^2]}$ and computes it. Since honest parties' shares are never used so far, this makes no difference. Therefore, the distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are identical.

$\mathbf{Hyb}_3$: In this hybrid, if the identity of P_i is an honest party, $\mathcal{S}$ aborts the simulation. Since we do not allow this case to happen in the real work, the distributions of $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ are identical.

$\mathbf{Hyb}_4$: In this hybrid, in the private reconstruction and verification phase, for a corrupted receiver R, $\mathcal{S}$ does the simulation as above. The probability $\mathcal{S}$ aborts is equal to the adversary finding a collision, which is negligible in the security parameter for a computationally bounded adversary. Therefore, the distributions of $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ are computationally indistinguishable.

$\mathbf{Hyb}_5$: In this hybrid, in the private reconstruction and verification phase, for an honest receiver R, $\mathcal{S}$ does the simulation as above. The difference between $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ is that a corrupted party P_i may send incorrect shares to R and still match the commitment. While the probability equals the adversary finds collision, which is negligible in the security parameter. Therefore, the distributions of $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ are computationally indistinguishable.

$\mathbf{Hyb}_6$: In this hybrid, $\mathcal{S}$ no longer requires honest dealer's inputs. $\mathcal{S}$ lets $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ do the same thing the deliver the output to each party, which makes no difference. Therefore, the distributions of $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ are identical.

Note that $\mathbf{Hyb}_6$ corresponds to the ideal world, then $\Pi_{\texttt{Sh2t-Id}}$ securely computes $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ when the dealer is honest.

Then we consider the case of the corrupted dealer.

Simulator $\mathcal{S}$

For Corrupted dealer

Distribution Phase

1: For each party P_i, $\mathcal{S}$ receives his shares during the simulation of $\mathcal{F}^{(i)}_{\mathsf{PrivSend}}$. Then $\mathcal{S}$ interpolates polynomials $f_1(x), \ldots, f_L(x)$ based on all parties' shares.

Verification and Termination Phase

2: $\mathcal{S}$ follows the protocol to do verification on behalf of each honest party. For each honest party P_i whose output is $(\texttt{Corrupt}, D)$, he sends $(\texttt{proof}, P_i)$ to $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$.

3: When all parties terminate the sharing phase, $\mathcal{S}$ sends $f_1(x), \ldots, f_L(x)$ to $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$.

Accusation and Agreement Accusation Phase

4: $\mathcal{S}$ honest execute each honest party. When all parties terminate with $(\texttt{Accusation}, D, P_i)$, $\mathcal{S}$ records the identity of P_i.

Private Reconstruction and Verification Phase

5: If $R = P_j$ is corrupted, $\mathcal{S}$ honestly follows the protocol to execute each honest party.

6: If $R = P_j$ is honest and his output is $(\texttt{Corrupt}, D)$, he follows the protocol to execute R. Otherwise, $\mathcal{S}$ initializes a set $\mathcal{M}'$ and does the following simulation on behalf of R.
   (1). When $\mathcal{S}$ receives shares from honest party P_i, he consider R accepts his shares and $\mathcal{S}$ adds P_i to $\mathcal{M}'$. When $\mathcal{S}$ receives shares from a corrupted party P_i, he follows the protocol to do the check. If true, he adds P_i to $\mathcal{M}'$.
   (2). If $\mathcal{S}$ first gets $\mathcal{M}'$ such that $|\mathcal{M}'| = 2t + 1$, he proceeds. Otherwise, he considers R first receives $(\texttt{Accusation}, D, P_i)$ and will forward it to $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$.
   (3). For each corrupted party $P_i \in \mathcal{M}'$, $\mathcal{S}$ checks whether the shares received from then lie on the polynomial $f_1^{(k)}(x), \ldots, f_{L'}^{(k)}(x)$. If true, $\mathcal{S}$ sends $\mathcal{M}'$ to $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ and proceeds. Otherwise, $\mathcal{S}$ aborts the simulation.
   (4). $\mathcal{S}$ reconstructs degree-$2t$ polynomials based on shares of parties in $\mathcal{M}'$ and checks whether these polynomials are equal to $f_1^{(k)}(x), \ldots, f_{L'}^{(k)}(x)$. If true, deliver the output $\texttt{true}$ and $f_1^{(k)}(x), \ldots, f_{L'}^{(k)}(x)$ from $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ to R. Otherwise, deliver the output $\texttt{false}$ from $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ to R (after $\mathcal{S}$ considers R can terminate).
7: $\mathcal{S}$ outputs the views of $\mathcal{A}$.

The hybrid arguments are as follows.

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

$\mathbf{Hyb}_1$: In this hybrid, in the private reconstruction and verification Phase, $\mathcal{S}$ will abort the simulation as above. The probability equals the adversary finding a collision, which is negligible in the security parameter for a computationally bounded adversary. Therefore, the distributions between $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are computationally indistinguishable.

$\mathbf{Hyb}_2$: In this hybrid, $\mathcal{S}$ let $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ deliver the output to each party, which makes no difference. Therefore, the distributions between $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are identical.

Note that $\mathbf{Hyb}_2$ corresponds to the ideal world, then $\Pi_{\texttt{Sh2t-Id}}$ securely computes $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ when the dealer is corrupted.

F Secure and Cost Analysis of AMPC

F.1 Construction of Main Protocol

We give the protocol $\Pi_{\texttt{Main}}$ which realizes $\mathcal{F}_{\mathsf{AMPC}}$ as follows.

Protocol $\Pi_{\texttt{Main}}$

Offline Phase

1: **Initialization of Verifiable Private Channel.** For each pair of parties (P_i, P_j), all parties initialize an instance of $\mathcal{F}_{\mathsf{PrivSend}}$.
2: **Preparation of Beaver Triples.** Let C denote the circuit to be computed. All parties invoke $\Pi_{\texttt{Triple}}$ to prepare $|C|$ random Beaver triples and assign one random triple with each multiplication gate in the circuit.

Online Phase

1: **Distributing Inputs.** Each party P_i invokes $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ to shares his secret x_i.
2: **Agreement the Set of the Inputs.** Each party P_i sets the property Q as P_i terminates $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ where P_j acts as the dealer. Then all parties invoke $\mathcal{F}_{\mathsf{acs}}$ with property Q to agree on a set $\mathcal{D}$ of parties that successfully share their inputs. For every $P_i \notin \mathcal{D}$, all parties set their shares of P_i's input as 0.
3: **Preprocessing for the Circuit.** All parties divide the circuit C into t disjoint sub-circuits $C_1, \ldots, C_t$ (sorted by topology) such that each sub-circuit contains $|C|/t$ multiplication gates.
4: **Circuit Evaluation.** From $k = 1$ to t, for each sub-circuit C_k, let $([a_i]_t, [b_i]_t, [c_i]_t)_{i=1}^{|C|/t}$ denote the random Beaver triples assigned to multiplication gates in C_k, all parties execute $\Pi_{\texttt{SubCktEval}}$ with input sharings for C_k and these $|C|/t$ random Beaver triples to evaluate C_k.
   – If the output of $\Pi_{\texttt{SubCktEval}}$ is the identity of a corrupted party, all parties send $\texttt{Public-Recon}$ to the $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ invoked by this corrupted party and wait to receive his secrets from $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$. Then they replace their shares with the secrets and invoke $\Pi_{\texttt{SubCktEval}}$ for the current C_k again.
   – Otherwise, all parties set $k = k + 1$.
5: **Output Reconstruction.** Upon all parties terminating the last sub-circuit C_k and learning their shares of output, they divide C_O output wires into t segments and invoke $\Pi_{\texttt{SubCktEval}}$ as above for each segment to do public reconstruction in order. As a result, all parties get all C_O outputs.

F.2 Cost Analysis

We analyze the communication and round complexity as follows.

Communication Complexity of $\Pi_{\texttt{Triple}}$. First, all parties need to distribute $\mathcal{O}(N/n+n)$ random degree-t and $2t$ sharings, which requires $\mathcal{O}(N \cdot n + n^4)$ field elements. All parties also need to broadcast their accusation, which requires $\mathcal{O}(n^4)$ field elements in total.

Then, all parties invoke n instances of $\Pi_{\texttt{TripleKingDN}}$, each king needs to prepare $\mathcal{O}(N/n^2)$ triples in each segment, for each of them.

- P_{king} broadcasts a set, which requires $\mathcal{O}(n^2)$ field elements.
- All parties send their shares to P_{king}, which requires $\mathcal{O}(N/n+n)$ field elements.
- P_{king} broadcasts the reconstruction results or an accusation, which requires $\mathcal{O}(N/n+n^2)$ field elements.
- During the process of building polynomials, they let king help to do reconstruction, which requires $\mathcal{O}(N/n+n^2)$ field elements.
- During the verification of triples, king needs to broadcast $\mathcal{O}(n)$ field elements, and all parties need to invoke two instances of $\mathcal{F}_{\mathsf{ra}}$. The total cost is $\mathcal{O}(n^2)$ field elements.
- During the fault localization process, all parties send their degree-t shares to P_{king}, which requires $\mathcal{O}(N/n+n)$ field elements. They reconstruct each dealer's degree-$2t$ sharings (of size $\mathcal{O}(N/n^3)$) to king, which requires $\mathcal{O}(N/n+n^2)$ field elements. Then king broadcasts his output, which requires $\mathcal{O}(n^2)$ field elements.

Therefore, the total cost for each king in each segment is $\mathcal{O}(N/n+n^2)$ field elements, resulting in the total cost for all kings in this segment is $\mathcal{O}(N+n^3)$ field elements.

Then, in each segment, all parties invoke an instance of $\mathcal{F}_{\mathsf{acs}}$, which requires $\mathcal{O}(n^3)$ field elements. All parties agree on the generation result, if they need to agree on a corrupted party, they require $\mathcal{O}(N+n^3)$ field elements to open an accusation and reconstruct this corrupted party's secrets.

Therefore, the total cost in each segment is $\mathcal{O}(N+n^3)$ field elements, and the total cost for the whole triple generation protocol is $\mathcal{O}(Nn+n^4)$ field elements.

Computation Complexity of $\Pi_{\texttt{Triple}}$ Each party distributes $\mathcal{O}(\frac{N}{n})$ sharings through $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ and $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$. The computational cost of this phase is $\mathcal{O}(N)$ lightweight cryptographic operations. Then, in the fault-localization process, each party sends its degree-$2t$ shares to P_{king}, which needs to compute $\mathcal{O}(n^2)$ commitments per segment, each on messages of size $\mathcal{O}(\frac{N}{n^3})$. Over n segments, this cost amounts to $\mathcal{O}(n^3)$ commitments per P_{king}, corresponding to $\mathcal{O}(N)$ lightweight cryptographic operations per P_{king} over all n segments. In addition, each party also verifies at most $\mathcal{O}(n)$ accusations through $\mathcal{F}_{\mathsf{PrivSend}}$ (one for each party), which also contributes to $\mathcal{O}(N)$ lightweight cryptographic operations to the computational cost. Overall, the total computation complexity is $\mathcal{O}(nN)$ lightweight cryptographic operations.

Communication Complexity of $\Pi_{\texttt{Main}}$. In the offline phase, all parties first initialize $\mathcal{F}_{\mathsf{PrivSend}}$ for each pair of parties, each one contains an instance of $\mathcal{F}_{\mathsf{AVSS}}$, and requires $\mathcal{O}(n^4)$ field elements in total. Then they invoke $\Pi_{\texttt{Triple}}$ to prepare triples, which requires $\mathcal{O}(Cn+n^4)$ field elements.

In the online phase, all parties first invoke $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ to distribute their inputs and $\mathcal{F}_{\mathsf{acs}}$ to agree on a set of successful dealers, which requires $\mathcal{O}(C_I \cdot n + n^4)$ field elements. Then they invoke $\Pi_{\texttt{SubCktEval}}$ to do reconstruction in each circuit segment, for each instance of $\Pi_{\texttt{SubCktEval}}$, denote the number of multiplication gates and output gates as C', C'_O and the circuit depth as D', then

- All parties broadcast their accusation, which requires $\mathcal{O}(n^3)$ field elements.
- During the circuit Evaluation process, it requires $\mathcal{O}(C'n+D'n^2)$ field elements.
- During the Output Reconstruction, it requires $\mathcal{O}(C'_O n+n^2)$ field elements.
- During the agreement, all parties first prepare $\mathcal{O}(1)$ random coin for $\mathcal{F}_{\mathsf{ba}}$, which can be realized with $\mathcal{O}(n^3)$ field elements. Then they invoke two instance of $\mathcal{F}_{\mathsf{ba}}$, which requires $\mathcal{O}((C'+C_O)n+n^2\log n)$ field elements.

Therefore, the total cost of $\Pi_{\texttt{SubCktEval}}$ is $\mathcal{O}((C'+C'_O)n+D'n^2+n^3)$ field elements. For the preparation of random coins, it can be built based on our construction of $\Pi_{\texttt{AVSS}}$, and we refer the reader to see the idea introduced in [Mom24]. In the online phase, all parties will fail the $\Pi_{\texttt{SubCktEval}}$ for at most t times. When they fail, they reconstruct the corrupted party's secrets, which requires $\mathcal{O}(C+n^2)$ field

elements. Let $D_{\mathtt{max}} = \max\{D_1, \ldots, D_t\}$, if $D_{\mathtt{max}}$ is bounded by $\mathcal{O}(C/n^2 + D/n)$, the total cost will be $\mathcal{O}((C + C_I + C_O)n + D \cdot n^2 + n^4) = \mathcal{O}(Cn + D \cdot n^2 + n^4)$ field elements.

Computational Complexity. Parties publicly reconstruct the secrets proposed by at most $\mathcal{O}(t)$ faulty parties. For each public reconstruction, each party verifies $\mathcal{O}(n)$ commitments on messages of size $\mathcal{O}(\frac{C}{n^2})$, contributing $\mathcal{O}(\frac{C}{n})$ per public reconstruction per party. Therefore, the overall computational cost is $\mathcal{O}(C)$ lightweight cryptographic operations per party and $\mathcal{O}(nC)$ operations for the system.

Round Complexity. We denote the round complexity of $\mathcal{F}_{\mathsf{acs}}$ and $\mathcal{F}_{\mathsf{ba}}$ as $R_{\mathbf{acs}}, R_{\mathbf{ba}}$. In the offline phase, the initialization of $\mathcal{F}_{\mathsf{PrivSend}}$ requires 7 rounds. For the preparation of triples, in the preparation process, all parties distribute random sharings and broadcast their accusation, which requires 10 rounds in total. In each segment,

- Each king broadcasts a set of successful dealers, which requires 4 rounds.
- During the generation of triples, all parties send shares to king, and king broadcasts the result, which requires $1 + 4 = 5$ rounds.
- During the verification of triples, it requires 19 rounds.
- During the fault localization, all parties help king to reconstruct shares, and king broadcasts his output, which requires 5 rounds.
- All parties agree on a set of successful kings, which requires $R_{\mathbf{acs}}$.
- If all parties need to open an accusation and reconstruct a corrupted party's secret, it requires 3 rounds.

Then, the total rounds for each segment are $33 + R_{\mathbf{acs}}$ (if they do not open an accusation). Since all parties may execute the segments for at most n times, then the total rounds in the offline phase are:

$$10 + (33 + R_{\mathbf{acs}}) \cdot n + 3 \cdot t = 10 + (34 + R_{\mathbf{acs}})n.$$

In the online phase, for each $\Pi_{\mathtt{SubCktEval}}$:

- In the check shares phase, it requires 4 rounds for broadcast.
- In the circuit evaluation, it requires $2D'$ rounds, where $D' \in \{D_1, \ldots, D_t\}$.
- In the output reconstruction phase, it requires 2 rounds.
- In the agreement phase, it requires $2R_{\mathbf{ba}}$ rounds. If all parties need to open an accusation and reconstruct the secrets, it requires $7 + R_{\mathbf{acs}}$ rounds.

Therefore, the total round complexity in the online phase is:

$$(6 + 2R_{\mathbf{ba}}) \cdot 2t + 2(D + D_{\mathtt{max}}) + (7 + R_{\mathbf{acs}}) \cdot t = 2D + t \cdot (D_{\mathtt{max}} + 19 + 4R_{\mathbf{ba}} + R_{\mathbf{acs}}).$$

Since $\mathcal{F}_{\mathsf{acs}}$ and $\mathcal{F}_{\mathsf{ba}}$ can be realized with constant rounds, if $D_{\mathtt{max}}$ is bounded by $\mathcal{O}(D/n)$, the total round complexity is $\mathcal{O}(D + n)$.

F.3 Security Analysis

We prove theorem 1 and start with constructing the ideal adversary $\mathcal{S}$ as follows. Let $\mathcal{C}orr$ denote the set of corrupted parties, then $|\mathcal{C}orr| = t' \leq t$. Let $\mathcal{C}orr'$ be the set of all corrupted parties with the first $t - t'$ honest parties, then $|\mathcal{C}orr'| = t$.

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

Simulation of $\Pi_{\mathtt{TripleKingDN}}$.

Generating Random Shamir Sharings.

1: $\mathcal{S}$ follows the protocol to compute corrupted parties' shares of $\{[a_\ell]_t, [b_\ell]_t, [r_\ell]_t, [o_\ell]_{2t}\}_{\ell \in [0,2N+1]}$ and $[r]_t$.
2: Denote $\mathcal{H}$ as the set of honest parties in $\mathcal{W}$, we rewrite each $[o_\ell]_{2t} = [o^{\mathcal{H}}]_{2t} + [o^{\mathcal{C}orr}]_{2t}$, where $[o_\ell^{\mathcal{H}}]_{2t}$ is honest dealers' contribution to $[o_\ell]_{2t}$ and $[o_\ell^{\mathcal{C}orr}]_{2t}$ is the corrupted parties' contribution.

Generating Triples.

3: Let $[z'_\ell]_{2t} = [a_\ell]_t \cdot [b_\ell]_t + [r_\ell]_t + [o_\ell^{\mathcal{H}}]_{2t}$, $\mathcal{S}$ first randomly samples the whole sharing $[z'_\ell]_{2t}$ based on shares of corrupted parties. Then for each honest party, P_{king} sets his share of $[z_\ell]_{2t}$ as $[z'_\ell]_{2t} + [o_\ell^{\mathcal{C}orr}]_{2t}$.
4: If P_{king} is honest, $\mathcal{S}$ honestly execute P_{king} and records the shares of $[z_\ell]_{2t}$ received from corrupted parties.

– If P_{king} first receives an accusation, $\mathcal{S}$ reliably broadcast it on behalf of P_{king}.
– Otherwise, $\mathcal{S}$ uses the first $2t+1$ shares of $[z_\ell]_{2t}$ he received to reconstruct z_ℓ and broadcasts them.

If P_{king} is corrupted, $\mathcal{S}$ sends each honest party's shares of $[z_\ell]_{2t}$ to P_{king}. Upon receiving secret z_ℓ from P_{king}, $\mathcal{S}$ records the additive error $d_\ell = z_\ell - z'_\ell$.

Build Polynomials.

5: $\mathcal{S}$ randomly samples the whole $\{[x_\ell]_t, [y_\ell]_t\}$ based on shares of corrupted parties. If P_{king} is honest, $\mathcal{S}$ does the following things on behalf of him:
– If P_{king} first receives an accusation, $\mathcal{S}$ reliably broadcasts this accusation.
– Otherwise, $\mathcal{S}$ reliably broadcasts $\{x_\ell, y_\ell\}$.

If P_{king} is corrupted, $\mathcal{S}$ send each honest party's shares of $\{[x_\ell]_t, [y_\ell]_t\}$ to P_{king}. Upon receiving secret x'_ℓ, y'_ℓ from P_{king}, $\mathcal{S}$ computes $e_\ell^{(x)} = x'_\ell - x_\ell, e_\ell^{(y)} = y'_\ell - y_\ell$.

6: For all $\ell \in [0, 2N]$, $\mathcal{S}$ follows the protocol to compute corrupted parties' shares of $[h(\alpha_\ell)]_t$. Then $\mathcal{S}$ computes a degree-$2N$ polynomial $d(\cdot)$ such that $d(\alpha_\ell) = d_\ell$.

Verification of Triples.

7: Upon getting $\{x_\ell, y_\ell\}$, $\mathcal{S}$ randomly samples the whole sharing $[r]_t$ based on shares of corrupted parties. If P_{king} is corrupted, $\mathcal{S}$ send each honest party's shares of $[r]_t$ to P_{king}. Otherwise, $\mathcal{S}$ does the following things on behalf of honest P_{king}:
– If P_{king} first receives an accusation, $\mathcal{S}$ reliably broadcasts this accusation.
– Otherwise, $\mathcal{S}$ reliably broadcasts the whole sharing $[r]_t$.

When all parties terminate $\mathcal{F}_{\mathsf{ra}}$ with 1, $\mathcal{S}$ continues.

8: For all $\ell \in [N+1, 2N]$, $\mathcal{S}$ checks that if any of $e_\ell^{(x)}, e_\ell^{(y)}$ is none zero:
– If true, $\mathcal{S}$ randomly samples $\{[a_\ell]_t, [b_\ell]_t, [r_\ell]_t\}_{\ell=0}^{2N}$ based on shares of corrupted parties. Then, he follows the protocol to execute each honest party. If $\mathcal{S}$ gets $h(r) = f(r) \cdot g(r)$, he aborts the simulation.
– Otherwise, $\mathcal{S}$ randomly samples values as $f(r), g(r)$ and computes $h(r) = f(r) \cdot g(r) + d(r)$. Then he randomly samples the whole sharings $\{[f(r)]_t, [g(r)]_t, [h(r)]_t\}$ based on shares of corrupted parties and secrets $f(r), g(r), h(r)$.

9: If P_{king} is honest, $\mathcal{S}$ reliably broadcasts the whole sharings $\{[f(r)]_t, [g(r)]_t, [h(r)]_t\}$ on behalf of P_{king}. Otherwise, $\mathcal{S}$ sends each honest parties' shares of $\{[f(r)]_t, [g(r)]_t, [h(r)]_t\}$ to P_{king} and executes the rest of steps honestly.

Fault Localization.

10: If P_{king} is corrupted and $\mathcal{S}$ has not samples $\{[a_\ell]_t, [b_\ell]_t, [r_\ell]_t\}_{\ell=0}^{2N}$, $\mathcal{S}$ randomly samples $\{[a_\ell]_t, [b_\ell]_t, [r_\ell]_t\}_{\ell=0}^{2N}$. $\mathcal{S}$ also random samples the degree-$2t$ sharings of zero distributed by each honest dealer $P_i \in \mathcal{W}$. Then $\mathcal{S}$ honestly executes the rest of the steps.

11: If P_{king} is honest, $\mathcal{S}$ honestly follows the protocol to execute P_{king}.

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

Simulation of $\Pi_{\mathtt{Triple}}$

Preparation of Random Shamir Sharings.

1: $\mathcal{S}$ simulates each $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ and $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ invoked by a party P_i as follows.
– If P_i is honest, $\mathcal{S}$ randomly samples shares of corrupted parties.
– If P_i is corrupted, $\mathcal{S}$ honestly execute $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}, \mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ and records each honest party's outputs.

2: For each honest party P_i, if he terminates $\mathcal{F}_{\mathsf{ACSS\text{-}Id}}$ or $\mathcal{F}_{\mathsf{Sh2t\text{-}Id}}$ invoked by a corrupted dealer D with $(\mathtt{Corrupt}, D)$, $\mathcal{S}$ reliably broadcasts $(\mathtt{Accusation}, P_i, P_j, \mathtt{ACSS})$ or $(\mathtt{Accusation}, P_i, P_j, \mathtt{Sh2t})$ on behalf of P_i.

Generation Phase.

3: In step 1, for each honest P_{king}, $\mathcal{S}$ follows the protocol to reliably broadcast $\mathcal{W}$ on behalf of this P_{king}. Then $\mathcal{S}$ invokes $\mathcal{S}_{\mathtt{TripleKingDN}}$ for each king. If all parties get valid shares of triples, $\mathcal{S}$ records shares of corrupted parties.

4: For the rest of the steps, $\mathcal{S}$ follows the protocol to honestly execute each honest party.

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

For circuit evaluation, $\mathcal{S}$ takes shares of degree-t Shamir sharings $([x^{(i)}]_t, [y^{(i)}]_t)$ and $([a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t$ for parties in $\mathcal{C}orr'$ as inputs. For public reconstruction, $\mathcal{S}$ takes the whole degree-t output Shamir sharings as inputs.

Simulation of $\Pi_{\mathtt{SubCktEval}}$

1: In Step 1, if an honest party P_i has received $(\texttt{Corrupt}, \texttt{dealer})$ before, $\mathcal{S}$ reliably broadcasts $(\texttt{Accusation}, P_i, \texttt{dealer}, \texttt{ACSS})$ on behalf of P_i.
2: In Step 2, in each layer, for every addition gate, $\mathcal{S}$ computes shares of $[z^{(i)}]_t$ for parties in $\mathcal{C}orr'$. For a group of L multiplication gates, $\mathcal{S}$ first computes shares of $[x^{(i)} + a^{(i)}]_t, [y^{(i)} + b^{(i)}]_t$ for parties in $\mathcal{C}orr'$, then randomly samples the whole $[x^{(i)} + a^{(i)}]_t, [y^{(i)} + b^{(i)}]_t$ based on shares of parties in $\mathcal{C}orr'$ and simulates $\Pi_{\texttt{BatchPubRec}}$ with honest parties' shares. Then $\mathcal{S}$ computes shares of $[z^{(i)}]_t$ for parties in $\mathcal{C}orr'$.
3: In Step 3, $\mathcal{S}$ honestly simulates $\Pi_{\texttt{BatchPubRec}}$ with honest parties' shares.
4: In Step 4, $\mathcal{S}$ follows the protocol to compute each honest party P_j's input $\boldsymbol{m}_j$. Then $\mathcal{S}$ honestly simulates $\Pi_{\texttt{Agreement}}$ and learns each honest party's output.

Simulator $\mathcal{S}$

Simulation of $\Pi_{\texttt{Main}}$

Offline Phase

1: $\mathcal{S}$ honestly emulates $\mathcal{F}_{\texttt{PrivSend}}$ and invokes $\mathcal{S}_{\texttt{TripleKingDN}}$ to learns corrupted parties' shares of Beaver triples. For honest parties in $\mathcal{C}orr' \setminus \mathcal{C}orr$, $\mathcal{S}$ randomly samples values as their shares of Beaver triples. $\mathcal{S}$ also learns which honest party does not get his shares but an ACSS proof during $\mathcal{S}_{\texttt{Triple}}$.

Online Phase

2: In Step 1, $\mathcal{S}$ simulates $\mathcal{F}_{\texttt{ACSS-Id}}$ as follows:
- For each honest dealer, $\mathcal{S}$ randomly samples shares of parties in $\mathcal{C}orr'$.
- For each corrupted dealer, $\mathcal{S}$ waits to receive degree-t Shamir sharings and learns the identity of the honest party whose output is $(\texttt{Corrupt}, D)$. $\mathcal{S}$ uses these degree-t Shamir sharing to compute the shares of honest parties in $\mathcal{C}orr'$.

3: In Step 2, $\mathcal{S}$ honestly simulates $\mathcal{F}_{\texttt{acs}}$ and learns a set $\mathcal{D}$ of size $2t + 1$.
4: In Steps 3 and 4, for each sub-circuit C_k and $k \in [t]$, $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{SubCktEval}}$ with shares of parties in $\mathcal{C}orr'$ for C_k:
- If $\mathcal{S}$ learns that all parties agree on a corrupted dealer during $\mathcal{S}_{\texttt{SubCktEval}}$, $\mathcal{S}$ sends $\texttt{Public-Recon}$ to $\mathcal{F}_{\texttt{ACSS-Id}}$ invoked by this dealer on behalf of honest parties. When $\mathcal{S}$ receives $t + 1$ requests, $\mathcal{S}$ sends this corrupted dealer's secrets to all parties on behalf of $\mathcal{F}_{\texttt{ACSS-Id}}$. Then $\mathcal{S}$ updates each honest party's shares distributed by this corrupted dealer with secrets. Finally, $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{SubCktEval}}$ again.
- Otherwise, $\mathcal{S}$ proceeds.

5: In Step 5, for each output wire $[y]_t$, upon getting shares of $[y]_t$ for parties in $\mathcal{C}orr'$, $\mathcal{S}$ sends the inputs of corrupted parties and the set $\mathcal{D}$ to $\mathcal{F}_{\texttt{AMPC}}$ and receives the output y. Then $\mathcal{S}$ computes the whole $[y]_t$ based on the secret y and shares of parties in $\mathcal{C}orr'$ and honestly follows the protocol to invoke $\mathcal{S}_{\texttt{SubCktEval}}$.
6: $\mathcal{S}$ outputs the views of $\mathcal{A}$.

The hybrid arguments are as follows.

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

$\mathbf{Hyb}_1$: In this hybrid, when $\mathcal{S}$ simulates $\mathcal{F}_{\texttt{ACSS-Id}}$ and $\mathcal{F}_{\texttt{Sh2t-Id}}$ on behalf of honest dealers, $\mathcal{S}$ first randomly samples corrupted parties' shares and then generates the whole sharings based on shares of corrupted parties and the secrets. According to the property of Shamir sharing, the distribution of the whole sharing is the same. Therefore, the distributions of $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are identical.

$\mathbf{Hyb}_2$: In the following, we focus on the simulation of $\Pi_{\texttt{Triple}}$ in the offline phase.

$\mathbf{Hyb}_{2.1}$: In this hybrid, we delay the generation of honest parties' shares until the set $\mathcal{W}$ is broadcast by each king. Since their shares have not been used before, this makes no difference. The distributions of $\mathbf{Hyb}_{2.1}$ and $\mathbf{Hyb}_1$ are identical.

$\mathbf{Hyb}_{2.2}$: In this hybrid, we change the generation of honest parties' shares. Let $\mathcal{H}$ denote the set of honest parties in $\mathcal{W}$ and $\mathcal{H}' \subseteq \mathcal{H}, |\mathcal{H}'| = t + 1$. For honest dealers in $\mathcal{H} \setminus \mathcal{H}'$, $\mathcal{S}$ still generates the whole sharings. Then for degree-t sharings, $\mathcal{S}$ first randomly samples the sharings $([s_{\ell',1}]_t, \ldots, [s_{\ell',t+1}]_t)$ based on shares of corrupted parties, then he computes the whole sharings $[s_{\ell'}^{(j)}]_t$ distributed by dealers in $\mathcal{H}'$. Since $\boldsymbol{V}$ is a Vandermonde matrix, any $(t + 1) \times (t + 1)$ sub-matrix of $\boldsymbol{V}$ is invertible, then given $([s_{\ell',1}]_t, \ldots, [s_{\ell',t+1}]_t)$, there is a one to one map between these sharings and $\{[s_{\ell'}^{(j)}]_t\}_{j \in \mathcal{H}'}$. Therefore, The distributions of $\mathbf{Hyb}_{2.2}$ and $\mathbf{Hyb}_{2.1}$ are identical.

$\mathbf{Hyb}_{2.3}$: In this hybrid, we change the generation of each secret r_ℓ. $\mathcal{S}$ first randomly samples a value as $z'_\ell = a_\ell \cdot b_\ell + r_\ell$ and then computes $r_\ell = z'_\ell - a_\ell \cdot b_\ell$. This does not change the distribution of $\{[a_\ell]_t, [b_\ell]_t, [r_\ell]_t\}$, then the distributions of $\mathbf{Hyb}_{2.3}$ and $\mathbf{Hyb}_{2.2}$ are identical.

$\mathbf{Hyb}_{2.4}$: In this hybrid, we rewrite each $[o_\ell]_{2t} = [o_\ell^{\mathcal{H}}]_{2t} + [o_\ell^{\mathcal{C}orr}]_{2t}$ and change the generation of $[o_\ell^{\mathcal{H}}]_{2t}$. $\mathcal{S}$ first randomly samples the whole sharings $[z'_\ell]_{2t} = [a_\ell]_t \cdot [b_\ell]_t + [r_\ell]_t + [o_\ell^{\mathcal{H}}]_{2t}$ based on shares of corrupted parties and secret z'_ℓ, then computes the whole sharing $[o_\ell^{\mathcal{H}}]_{2t} = [z'_\ell]_{2t} - [a_\ell]_t \cdot [b_\ell]_t - [r_\ell]_t$. This does not change the distribution of $[o_\ell^{\mathcal{H}}]_{2t}$, so the distributions of $\mathbf{Hyb}_{2.4}$ and $\mathbf{Hyb}_{2.3}$ are identical.

$\mathbf{Hyb}_{2.5}$: In this hybrid, we delay the generation of honest party's shares of $\{[a_\ell]_t, [b_\ell]_t, [r_\ell]_t\}$. When a party needs to send his shares of $[z_\ell]_{2t}$ to P_{king}, $\mathcal{S}$ sets this party's shares as $[z'_\ell]_{2t} + [o_\ell^{\mathcal{C}orr}]_{2t}$ and sends it to P_{king} on behalf of this party. The distributions of $\mathbf{Hyb}_{2.5}$ and $\mathbf{Hyb}_{2.4}$ are identical.

$\mathbf{Hyb}_{2.6}$: In this hybrid, we change the way of sampling $[r_\ell]_t$. If P_{king} first receives $2t+1$ shares of $[z_\ell]_{2t}$ and succeeds in broadcasting the secret z_ℓ, $\mathcal{S}$ computes honest parties' shares of $[c_\ell]_t$, then randomly sampling the whole $[c_\ell]_t$ based on shares of corrupted parties and secret $c_\ell = a_\ell \cdot b_\ell + z_\ell - z'_\ell$. Finally, $\mathcal{S}$ computes $[r_\ell]_t = z_\ell - [c_\ell]_t$. This does not change the distribution of $[r_\ell]]_t$, the distributions of $\mathbf{Hyb}_{2.6}$ and $\mathbf{Hyb}_{2.5}$ are identical.

$\mathbf{Hyb}_{2.7}$: In this hybrid, let $d = h - f \cdot g$. If $r \notin \{\alpha_1, \ldots, \alpha_N\}, d \not\equiv 0$ and $d(r) = 0$, $\mathcal{S}$ aborts the simulation. By the Schwartz-Zipple lemma, the probability is at most $\frac{2Nn}{2^\kappa - N}$, which is negligible in the security parameter κ. Thus, the distributions of $\mathbf{Hyb}_{2.7}$ and $\mathbf{Hyb}_{2.6}$ are statistically close.

$\mathbf{Hyb}_{2.8}$: In this hybrid, if any of $e_\ell^{(x)}, e_\ell^{(y)}$ is not zero but $h(r) = f(r) \cdot g(r)$, $\mathcal{S}$ aborts the simulation. Due to the same reason in $\mathbf{Hyb}_{2.7}$, the distributions of $\mathbf{Hyb}_{2.8}$ and $\mathbf{Hyb}_{2.7}$ are statistically close.

$\mathbf{Hyb}_{2.9}$: In this hybrid, we delay the generation of $([a_\ell]_t, [b_\ell]_t)$ for all $\ell \in [N+1, 2N]$:

For $[a_\ell]_t, [b_\ell]_t$, $\mathcal{S}$ first follows the protocol to compute corrupted parties' shares of $[f(\alpha_\ell) + a_\ell]_t, [g(\alpha_\ell) + b_\ell]_t$ for all $k \in [2t+1], i \in [N+1, 2N]$. Then $\mathcal{S}$ randomly samples values as $f(\alpha_\ell) + a_\ell, g(\alpha_\ell) + b_\ell$ and recomputes a_ℓ, b_ℓ. Finally $\mathcal{S}$ randomly samples the whole $[a_\ell]_t, [b_\ell]_t$ based on the secrets a_ℓ, b_ℓ and shares of corrupted parties.

To compute $[c_\ell]_t$, $\mathcal{S}$ first computes $c_\ell = a_\ell \cdot b_\ell + d_\ell$ and then samples the whole $[c_\ell]_t$ based on the secret c_ℓ and shares of corrupted parties. The distributions of $([a_\ell]_t, [b_\ell]_t, [c_\ell]_t)$ remain unchanged, so the distributions of $\mathbf{Hyb}_{2.9}$ and $\mathbf{Hyb}_{2.8}$ are identical.

$\mathbf{Hyb}_{2.10}$: In this hybrid, we delay the generation of $([c_\ell]_t)$ for all $i \in [N+1, 2N]$. $\mathcal{S}$ first follows the protocol to compute corrupted parties' shares of $[h(\alpha_\ell)]_t$, then randomly samples the whole $[h(\alpha_\ell)]_t$ based on the secret $h(\alpha_\ell) = f(\alpha_\ell) \cdot g(\alpha_\ell) + d(\alpha_\ell)$ and shares of corrupted parties. Finally $\mathcal{S}$ computes $[c_\ell]_t = [h(\alpha_\ell)]_t - (f(\alpha_\ell) + a_\ell) \cdot (g(\alpha_\ell) + b_\ell) + (f(\alpha_\ell) + a_\ell)[b]_t + (g(\alpha_\ell) + b_\ell)[a]_t$. The distributions of $\mathbf{Hyb}_{2.10}$ and $\mathbf{Hyb}_{2.9}$ are identical.

$\mathbf{Hyb}_{2.11}$: In this hybrid, we change the generation of $([a_0]_t, [b_0]_t, [c_0]_t)$. If $r \notin \{\alpha_1, \ldots, \alpha_N\}$, $f(r)$ is a linear combination of $\{f(\alpha_\ell)\}_{\ell=0}^N$ and the coefficient of $f(\alpha_0) = a_0$ is non-zero. Then $f(\alpha_0)$ also can be computed by the linear combination of $\{f(\alpha_\ell)\}_{\ell=1}^N$ and $f(r)$. We let $\mathcal{S}$ first compute corrupted parties' shares of $[f(r)]_t$ by the linear combination of $\{[f(\alpha_\ell)]_t\}_{\ell=0}^N$, then randomly samples the whole $[f(r)]_t$ based on shares of corrupted parties. $\mathcal{S}$ does the same thing to generate $[g(r)]_t$. Finally, $\mathcal{S}$ computes $[a_0]_t, [b_0]_t$ by the linear combination of $\{[f(\alpha_\ell)]_t, [g(\alpha_\ell)]_t\}_{\ell=1}^N$ and $\{[f(r)]_t, [g(r)]_t\}$. $\mathcal{S}$ also computes $c_0 = a_0 \cdot b_0 + d_0$ and randomly samples the whole $[c_0]_t$ based on the secret c_0 and shares of corrupted parties. The distributions of $\mathbf{Hyb}_{2.11}$ and $\mathbf{Hyb}_{2.10}$ are identical.

$\mathbf{Hyb}_{2.12}$: In this hybrid, we change the generation of $[h(r)]_t$. When $r \notin \{\alpha_0, \ldots, \alpha_{2N}\}$, we let $\mathcal{S}$ randomly sample $[h(r)]_t$ based on the secret $h(r) = f(r) \cdot g(r) + d(r)$ and shares of corrupted parties. Since $[h(r)]_t$ is a linear combination of $\{[h(\alpha_\ell)]_t\}_{\ell=0}^{2N}$ and when $r \notin \{\alpha_0, \ldots, \alpha_{2N}\}$, the coefficient of $[h(\alpha_0)]_t$ is non-zero. $\mathcal{S}$ computes $[h(\alpha_0)]_t$ by a proper linear combination of $\{[h(\alpha_\ell)]_t\}_{\ell=1}^{2N}$ and $[h(r)]_t$. Note that in $\mathbf{Hyb}_{2.11}$, $[h(r)]_t$ is a random degree-t Shamir sharings given shares of corrupted parties and the secret $h(r)$. The distributions of $\mathbf{Hyb}_{2.12}$ and $\mathbf{Hyb}_{2.11}$ are identical.

$\mathbf{Hyb}_{2.13}$: In this hybrid, we no longer generate honest parties' shares of $\{[a_0]_t, [b_0]_t, [c_0]_t\} \cup \{[a_\ell]_t, [b_\ell]_t, [c_\ell]_t\}_{i=N+1}^{2N}$ since they are never used. During the fault localization, for corrupted P_{king}, $\mathcal{S}$ will randomly degree-t and $2t$ sharings as the sharings distributed by honest dealers and honestly execute the protocol. The distributions of $\mathbf{Hyb}_{2.13}$ and $\mathbf{Hyb}_{2.12}$ are identical.

$\mathbf{Hyb}_3$: In this hybrid, we focus on the simulation of $\Pi_{\mathtt{Main}}$ in the online phase.

$\mathbf{Hyb}_{3.1}$: In this hybrid, in the input phase, for each honest dealer, after randomly sampling shares of parties in $\mathcal{C}orr'$, $\mathcal{S}$ delays the generation of the rest of honest parties' shares until the set $\mathcal{D}$ is determined.

Since these honest parties' shares are not used in the input phase, the distributions of $\mathbf{Hyb}_{3.1}$ and $\mathbf{Hyb}_{2.13}$ are identical.

$\mathbf{Hyb}_{3.2}$: In this hybrid, in the input phase, for each honest dealer not in $\mathcal{D}$, $\mathcal{S}$ does not generate shares of honest parties. Since these honest dealers' sharings are never used, the distributions of $\mathbf{Hyb}_{3.2}$ and $\mathbf{Hyb}_{3.1}$ are identical.

$\mathbf{Hyb}_{3.3}$: In this hybrid, in the computation phase, $\mathcal{S}$ invokes $\mathcal{S}_{\texttt{SubCktEval}}$ to simulate each $\Pi_{\texttt{SubCktEval}}$. The difference is as follows, in $\mathbf{Hyb}_{3.2}$, $\mathcal{S}$ first randomly samples $[a]_t, [b]_t$ then computes $[x+a]_t, [y+b]_t$, while in $\mathbf{Hyb}_{3.3}$ we let $\mathcal{S}$ first randomly samples $[x+a]_t, [y+b]_t$ and then computes $[a]_t, [b]_t$, which makes no difference. The distributions of $\mathbf{Hyb}_{3.3}$ and $\mathbf{Hyb}_{3.2}$ are identical.

$\mathbf{Hyb}_{3.4}$: In this hybrid, $\mathcal{S}$ first computes $y = f(x_1, \ldots, x_n)$, then computes the whole $[y]_t$ based on secret y and shares of parties in $\mathcal{C}orr'$. For honest parties not in $\mathcal{C}orr' \setminus \mathcal{C}orr$, we also no longer generate their shares during each $\Pi_{\texttt{SubCktEval}}$ since they are never used. The distributions of $\mathbf{Hyb}_{3.4}$ and $\mathbf{Hyb}_{3.3}$ are identical.

$\mathbf{Hyb}_4$: In this hybrid, for honest parties not in $\mathcal{C}orr' \setminus \mathcal{C}orr$, we no longer generate their shares of Beaver triples since they are never used. The distributions of $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_{3.4}$ are identical.

$\mathbf{Hyb}_5$: In this hybrid, $\mathcal{S}$ sends the inputs of corrupted parties and the set $\mathcal{D}$ to $\mathcal{F}_{\mathsf{AMPC}}$ and receives the output y. Since $\mathcal{F}_{\mathsf{AMPC}}$ computes y in the same way as $\mathcal{S}$, $\mathcal{S}$ no longer needs honest parties' inputs. The distributions of $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ are identical.

Note that $\mathbf{Hyb}_5$ corresponds to the ideal world, then $\Pi_{\texttt{Main}}$ securely computes $\mathcal{F}_{\mathsf{AMPC}}$ when the dealer is corrupted.