Linear-Communication Asynchronous Complete Secret Sharing with Optimal Resilience

Xiaoyu Ji
Tsinghua University
jixy23@mails.tsinghua.edu.cn

Junru Li
ShanghaiTech University
lijr2@shanghaitech.edu.cn

Yifan Song
Tsinghua University
and Shanghai Qi Zhi Institute
yfsong@mail.tsinghua.edu.cn

Abstract

Secure multiparty computation (MPC) allows a set of n parties to jointly compute a function on their private inputs. In this work, we focus on the information-theoretic MPC in the *asynchronous network* setting with optimal resilience ($t < n/3$). The best-known result in this setting is achieved by Choudhury and Patra [J. Cryptol '23], which requires $O(n^4\kappa)$ bits per multiplication gate, where κ is the size of a field element.

An asynchronous complete secret sharing (ACSS) protocol allows a dealer to share a batch of Shamir sharings such that all parties eventually receive their shares. ACSS is an important building block in AMPC. The best-known result of ACSS is due to Choudhury and Patra [J. Cryptol '23], which requires $O(n^3\kappa)$ bits per sharing. On the other hand, in the synchronous setting, it is known that distributing Shamir sharings can be achieved with $O(n\kappa)$ bits per sharing. There is a gap of n^2 in the communication between the synchronous setting and the asynchronous setting.

Our work closes this gap by presenting the first ACSS protocol that achieves $O(n\kappa)$ bits per sharing. When combined with the compiler from ACSS to AMPC by Choudhury and Patra [IEEE Trans. Inf. Theory '17], we obtain an AMPC with $O(n^2\kappa)$ bits per multiplication gate, improving the previously best-known result by a factor of n^2. Moreover, with a concurrent work that improves the compiler by Choudhury and Patra by a factor of n, we obtain the first AMPC with $O(n\kappa)$ bits per multiplication gate.

1 Introduction

Secure multiparty computation (MPC) enables a group of n parties to compute a public function on their private inputs while protecting the secrecy of each party's input. There are two main network settings considered in the literature of MPC: the *synchronous network* setting and the *asynchronous network* setting.

Most of the existing works on MPC study in the synchronous network setting [BGW88, CCD88, GMW87, RB89, Yao82], where a synchronized global clock exists and there is an upper bound on the network communication delay. In the synchronous network setting, every party can expect to receive messages from other parties within a bounded amount of time, which makes it easier to construct MPC protocols. However, this does not capture the network condition in the real world where there is no fixed time bound for the network communication delay and all parties are asynchronous. This gave rise to the asynchronous network setting where the messages may be arbitrarily delayed and delivered out of order with the only guarantee that all messages will be eventually delivered. In particular, an MPC protocol in the asynchronous network setting (or AMPC for short) does not rely on any assumption of timing.

One of the main challenges in constructing an AMPC protocol is that one cannot distinguish between a corrupted party not sending a message and an honest party that sent a message that is delayed by the adversary. As a result, to achieve liveness, a party cannot expect or wait for messages from all other parties and should proceed once he receives a certain number of messages. Typically, a party can only afford to wait for messages from $n-t$ parties, where t is the number of corrupted parties, since corrupted parties may never send messages. On the other hand, in the worst case, the t missing messages may come from honest parties due to the network delay, and t messages received from these $n-t$ parties may be from corrupted parties. Due to this challenge, protocols in the synchronous network completely fail since the security of these protocols typically requires each party to receive messages from all other (honest) parties.

In this work, we focus on the communication complexity of AMPC in the information-theoretic setting. Compared with MPC protocols in the synchronous setting, AMPC usually requires a smaller corruption threshold. For example, it is known that in the synchronous setting, perfect security can be achieved with $t < n/3$ corruption [BGW88, BH08] and guaranteed output delivery (with negligible error) can be achieved with the honest majority ($t < n/2$) [RB89] assuming broadcast channels. On the other hand, in the asynchronous setting, perfect security is only possible when $t < n/4$ [BOCG93] while guaranteed output delivery (with negligible error) requires $t < n/3$ [BOKR94, ADS20]. Also, AMPC appears less efficient than synchronous MPC in terms of asymptotic communication complexity: It has been known for a decade that guaranteed output delivery with optimal resilience $t < n/2$ in the synchronous setting can be achieved with linear communication complexity in the number of parties per multiplication gate. However, in the asynchronous setting, the best-known result with optimal resilience $t < n/3$ [CP23] still requires $\mathcal{O}(n^4\kappa)$ bits of communication per multiplication gate, where κ is the size of a field element. A long-standing open question is whether we can achieve linear communication complexity in the asynchronous setting with guaranteed output delivery and optimal resilience $t < n/3$.

1.1 Asynchronous Complete Secret Sharing

An asynchronous complete secret sharing (ACSS) protocol is a basic tool in building AMPC protocols. It allows a dealer to share a group of degree-t Shamir sharings to all parties which ensures that (1) if the dealer is honest, all (honest) parties will eventually receive correct shares, and (2) even if the dealer is corrupted, either no honest party receives/accepts his shares or all honest parties eventually obtain correct shares. With an ACSS protocol, a typical approach to achieving AMPC [CP17, CP23] with optimal resilience $t < n/3$ is to first prepare random Beaver triples in the offline phase, and then rely on the technique of circuit randomization [Bea92] to evaluate the circuit in the online phase. Following this approach, the work [CP17] has shown that any circuit C can be securely computed by using an ACSS protocol in a black-box way with $\mathcal{O}(n^2\kappa)$ bits of communication plus sharing $\mathcal{O}(n)$ degree-t Shamir sharings via ACSS per multiplication gate. Combining with the best-known result of ACSS [CP23] against $t < n/3$ corruption, which requires communicating $\mathcal{O}(n^3\kappa)$ bits per sharing, it gives the best-known result of AMPC which requires $\mathcal{O}(n^4\kappa)$ bits per multiplication gate in this setting.

Thanks to the compiler in [CP17], any improvement in building ACSS protocols directly leads to improvement in AMPC. We note that in the synchronous setting with optimal resilience $t < n/2$, distributing degree-t Shamir sharings can be done with linear communication complexity $\mathcal{O}(n\kappa)$ bits per sharing [BFO12, CP17]. This leads to our following question.

Does there exist an optimal-resilient ($t < n/3$) information-theoretic ACSS protocol with amortized communication of $\mathcal{O}(n\kappa)$ bits per sharing?

1.2 Contributions

In this paper, we answer this question affirmatively. Our main contribution is an ACSS protocol with $\mathcal{O}(Nn\kappa + n^{12}\kappa^2)$-bit communication to share N degree-t Shamir sharings, where κ is the size of a field element. As a result, our ACSS protocol achieves an amortized communication of $\mathcal{O}(n\kappa)$ bits per sharing.

Theorem 1. *Let κ denote the security parameter. For a finite field $\mathbb{F}$ of size $2^{\Theta(\kappa)}$, there exists a fully malicious information-theoretic ACSS protocol against $t < n/3$ corrupted parties that shares N degree-t Shamir sharings over $\mathbb{F}$ with communication of $\mathcal{O}(Nn\kappa + n^{12}\kappa^2)$ bits and statistical error $\mathcal{O}((Nn + n^{15}\kappa^2)/2^\kappa)$. The round complexity is $\mathcal{O}(1)$ rounds via P2P channels plus $\mathcal{O}(1)$ rounds of invocations to ACast.*

To achieve our result,

- We extend the asynchronous information-checking protocol (AICP), which is introduced in [PCR09] and is used as an information-theoretic signature scheme, to support linear operations over signatures and verification by multiple receivers. This allows us to let each party efficiently verify the shares received from the dealer. This tool may be of independent interest.
- We extend the technique of authentication tags [BFO12] to the asynchronous setting. With authentication tags, we show that a set of $2t+1$ parties can help the rest of t parties recover their shares efficiently, thus achieving asynchronous complete secret sharing. However, adapting the technique of authentication tags to the asynchronous setting is not an easy task due to the challenge in the asynchronous setting we mentioned above.

In Section 2, we give an overview of our techniques.

Implications in AMPC. When applying the compiler from ACSS to AMPC in [CP17], we obtain an AMPC protocol with $\mathcal{O}(n^2\kappa)$ bits of communication per multiplication gate, which improves the best-known result [CP23] by a factor of n^2.

Theorem 2. ([CP17]) *Let $n = 3t + 1$. For any circuit C of size $|C|$ and depth D, there is a fully malicious asynchronous MPC protocol computing the circuit that is secure against at most t corrupted parties with guaranteed output delivery in the $\mathcal{F}_{\mathsf{ACSS}}$-hybrid model. The achieved communication complexity is $\mathcal{O}(|C| \cdot n^2\kappa + n^5\kappa)$ bits plus $\mathcal{O}(n)$ invocations of $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(|C| \cdot n)$ degree-t Shamir sharings in total.*

Corollary 1. *Let $n = 3t + 1$. For any circuit C of size $|C|$ and depth D, there is a fully malicious information-theoretic asynchronous MPC protocol that is secure against at most t corrupted parties with guaranteed output delivery. The total communication complexity is $O(|C| \cdot n^2\kappa + n^{13}\kappa^2)$ bits.*

We note that a concurrent work [GLS24] improves the compiler in [CP17] by a factor of n. I.e., the cost per multiplication gate is reduced to $O(n\kappa)$ bits plus sharing $O(1)$ degree-t Shamir sharings via ACSS. When applying the compiler in [GLS24], we obtain an AMPC protocol with $\mathcal{O}(n\kappa)$ bits of communication per multiplication gate, the first information-theoretic AMPC protocol with linear communication complexity and optimal resilience.

Theorem 3. ([GLS24]) *Let $n = 3t+1$. For any circuit C of size $|C|$ and depth D, there is a fully malicious asynchronous MPC protocol computing the circuit that is secure against at most t corrupted parties with guaranteed output delivery in the $\mathcal{F}_{\mathsf{ACSS}}$-hybrid model. The achieved communication complexity is $\mathcal{O}(|C| \cdot n\kappa + D \cdot n^2\kappa + n^8\kappa)$ bits plus $\mathcal{O}(n^2)$ invocations of $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(|C|)$ degree-t Shamir sharings in total.*

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

Guarantee Output v.s. Guarantee Termination. Our construction of ACSS ensures that if an honest party accepts his shares, then all honest parties will eventually obtain their shares. However, even if an honest party accepts his shares, we need this party to stay online to help other parties reconstruct their shares. As a result, our construction does not guarantee the termination of the protocol even if the dealer is honest. This is unlike the construction in [CHP13] where an honest party can terminate after he accepts his shares.

However, as observed in [CP23], this does not affect the termination of the MPC protocol. At a high level, when using our construction of ACSS, all honest parties are guaranteed to receive the function output (but may not terminate). Assuming that all parties should receive the same function output, it is sufficient to let all parties run a consensus protocol on the final output. The MPC protocol will terminate once the consensus is reach. We refer the readers to Appendix. F.2 for more details.

In Appendix. E, we show how to construct an ACSS protocol that guarantees termination with $\mathcal{O}(n^2)$ communication per secret. We leave the question of building information-theoretic ACSS in the optimal resilience ($t < n/3$) setting with both linear communication and guarantee of termination to future work.

1.3 Related Works

The question of designing a communication-efficient ACSS protocol is also studied in other settings.

In the setting of perfect security, it is known that $t < n/4$ is necessary [BCG93]. A line of works [SR00, BH07, CHP13, PCR15, CP17] has improved the communication complexity of perfect ACSS in this setting. The best-known result [CP17] has achieved linear communication complexity $O(n\kappa)$ bits per sharing.

In the setting of computational security against $t < n/3$ corrupted parties, ACSS with linear communication complexity is known in [AJM+23] relying on discrete logarithms and pairings. The work [SS23] also tries to only use lightweight cryptography such as collision-resistant hash functions and pseudo-random functions and achieves an amortized communication complexity of $\mathcal{O}(n^2\kappa)$ bits per sharing.

2 Technical Overview

We give a high-level overview of the main techniques used in this paper. In our setting, parties can access a complete network of point-to-point asynchronous and secure channels. Asynchronous channels only guarantee that messages sent by honest parties are eventually delivered, and the adversary can control the message scheduling. Let $P_1, \ldots, P_n$ denote the n parties in the protocol, who form a set $\mathcal{P}$.

An *Asynchronous Complete Secret Sharing* (ACSS) protocol enables a dealer to share degree-t Shamir secret sharings among n parties. Let $\mathbb{F}$ be a finite field and $\alpha_0, \ldots, \alpha_n \in \mathbb{F}$ be distinct field elements. A degree-t Shamir secret sharing of a secret $\omega \in \mathbb{F}$, denoted by $[\omega]_t$, is a vector $(\omega_1, \ldots, \omega_n)$ determined by a degree-t polynomial f where $f(\alpha_0) = \omega$ and $f(\alpha_i) = \omega_i$ for all $i \in [n]$. Each party P_i holds ω_i as his share. In our work, we use κ as the security parameter and assume that the size of $\mathbb{F}$ is $2^{\Theta(\kappa)}$. Then each field element is of length $\Theta(\kappa)$ bits.

An ACSS protocol satisfies the following two properties:

1. When the dealer is honest, the protocol must guarantee that all the honest parties will eventually receive their shares of the degree-t Shamir secret sharing.

2. When the dealer is corrupted, either all honest parties eventually get a valid degree-t Shamir secret sharing or no honest party gets/accepts his share.

Our goal is to construct an ACSS protocol with optimal resilience $n = 3t + 1$ and linear communication complexity $\mathcal{O}(n\kappa)$ bits per sharing.

2.1 Overview of Previous Approaches

The previous works on optimal-resilient statistically secure ACSS [CP23, PCR09] provide ACSS through a path AICP $\to$ AVSS[1] $\to$ ACSS. We will introduce each notion below. Here $X \to Y$ denotes that X is used as a sub-protocol in Y.

[1] In [PCR09], the authors add an intermediate protocol called AWSS between AICP and AVSS. In [CP23], AVSS is called AISS.

Asynchronous Information-Checking Protocol (AICP). The notion of AICP was first introduced in [PCR09] and can be considered as a signature scheme among a dealer D, an intermediary I, and a receiver R. It allows D to send a message to I with a signature on this message. When I passes the message and signature to R, R can use the signature to check whether this message is from D. The amortized communication complexity of AICP in [PCR09] is $\mathcal{O}(1)$ bits per bit of message, which is essentially at the same cost as sending this message directly.

From AICP to Asynchronous Verifiable Secret Sharing (AVSS). AVSS enables a dealer to share a secret among all parties which guarantees the success of the reconstruction of the secret. However, unlike ACSS, AVSS does not guarantee that all (honest) parties eventually obtain their shares. Instead, in the worst case, only $n-t$ parties can obtain their shares from an AVSS protocol.

In [CP23], an AVSS protocol is constructed as follows. Suppose the dealer D wants to share a degree-t Shamir sharing $[s]_t$. D first encodes $[s]_t$ into a random degree-(t,t) bivariate polynomials $F(x,y)$ such that the underlying polynomial of $[s]_t$ is stored at $F(x,\alpha_0)$. The goal is to let each party P_i learn $f_i(x) = F(x,\alpha_i)$ and $g_i(y) = F(\alpha_i, y)$. Note that if each party takes $g_i(\alpha_0)$, all parties together hold $[s]_t$.

At a high level, the sharing of $F(x,y)$ is achieved by the following three steps.

- **Step 1: Committing Bivariate Polynomial via Column Polynomials.** D sends the column polynomial $g_i(y) = F(\alpha_i, y)$ to each P_i and receives P_i's signature on $g_i = (g_i(\alpha_1), \ldots, g_i(\alpha_n))$ via AICP. Upon receiving $2t+1$ parties' signatures, D broadcasts the set $\mathcal{M}$ of these parties.

Note that $\mathcal{M}$ contains at least $t+1$ honest parties. The column polynomials of the first $t+1$ honest parties in $\mathcal{M}$ fully determine a bivariate polynomial F, which is viewed as the bivariate polynomial shared by D. However, when $\mathcal{M}$ contains more than $t+1$ honest parties, the column polynomials of the remaining honest parties may not lie on F.

- **Step 2: Reconstructing Row Polynomials.** Then, D sends $f_j(x)$ to P_j. In addition, for each $P_i \in \mathcal{M}$, D serves as the intermediary of AICP to send the signature of $g_i(\alpha_j) = f_j(\alpha_i)$ to P_j. Now each P_j verifies that for all $P_i \in \mathcal{M}$, $f_j(\alpha_i)$ indeed comes from P_i. If true, P_j accepts $f_j(x)$ and broadcasts a flag $\texttt{OK}_{P_j}$. After $2t+1$ parties accept their row polynomials, the sharing phase finishes.

Let $\mathcal{W}$ denote the set of parties that accept their row polynomials. By the property of AICP, each honest $P_i \in \mathcal{M}$ and each honest $P_j \in \mathcal{W}$ satisfy that $g_i(\alpha_j) = f_j(\alpha_i)$. In [CP23], the authors prove that in this case, both the column polynomials of honest parties in $\mathcal{M}$ and row polynomials of honest parties in $\mathcal{W}$ lie on the bivariate polynomial F determined in the first step.

- **Step 3: Reconstruction.** To reconstruct $F(x,y)$ to a party P_r, each party P_j in $\mathcal{W}$ sends $f_j(x)$ together with the signatures from parties in $\mathcal{M}$. P_r accepts $f_j(x)$ if the signatures from parties in $\mathcal{M}$ are all valid. Since $\mathcal{W}$ contains at least $t+1$ honest parties, P_r will eventually receive at least $t+1$ row polynomials and reconstruct the whole bivariate polynomial F.

Here following the same argument in [CP23], we can show that if P_r accepts $f_j(x)$, then $f_j(x)$ must lie on the bivariate polynomial F. Thus, P_r will eventually reconstruct the correct bivariate polynomial F.

One subtlety in the above approach is that AICP is not transferable. I.e., the signature is tied with the intermediary I and can only be used by I to convince a receiver R. In particular, the receiver R cannot use the same signature to convince a new receiver R'. However, in the above approach, the same set of signatures is used to first let D convince each $P_j \in \mathcal{W}$ about $f_j(x)$ in Step 2, and then let $P_j \in \mathcal{W}$ convince P_r about $f_j(x)$ in Step 3, which cannot be achieved by AICP. The work [CP23] manages to resolve this issue without introducing additional overhead in communication.

We note that in such an AVSS protocol, the dealer D needs to share a secret through a degree-(t,t) bivariate polynomial, which requires communication of at least $\mathcal{O}(n^2\kappa)$ bits.

From AVSS to ACSS. However, an AVSS protocol is not enough since in the worst case t honest parties may not get their shares, while an ACSS protocol requires all honest parties to get their shares. Patra, Choudhury, and Rangan [PCR09] provide a framework to construct ACSS by sharing and reconstructing each party's share via AVSS. Since the AVSS protocol needs to be executed n times, one time for each party, and each time requires communication of $\mathcal{O}(n^2\kappa)$ bits, the ACSS protocol has an amortized communication complexity of at least $\mathcal{O}(n^3\kappa)$ bits, as Choudhury and Patra reach in [CP23].

2.2 Our Solution to Achieve Linear Communication

In this part, we show how to build ACSS with linear communication complexity. In a nutshell, we manage to adapt the approach in [CP23] by using degree-$(t, 2t)$ bivariate polynomials, shaving the factor of n overhead in AVSS. Then we show how to directly upgrade from AVSS to ACSS, avoiding the factor of n overhead mentioned above. We elaborate on how each step is achieved below.

An Initial Attempt. Following the previous works, we let the dealer share degree-t Shamir sharings through bivariate polynomials. To reduce the amortized communication complexity, our starting point is to use degree-$(t, 2t)$ bivariate polynomials. Since a degree-$(t, 2t)$ bivariate polynomial is determined by $t + 1$ degree-$2t$ polynomials, any $t + 1$ honest parties with column polynomials can reconstruct the bivariate polynomial, but any t (corrupted) parties with their row and column polynomials need another $t + 1$ points on a column polynomial to determine the bivariate polynomial. By storing secrets at these $t+1$ points, each degree-$(t, 2t)$ bivariate polynomial can be used to store $t + 1 = \mathcal{O}(n)$ degree-t Shamir sharings. Hopefully, we can reduce the amortized communication complexity per sharing by $\mathcal{O}(n)$ in this way. We note that using bivariate polynomials to do secret sharing has has been used in several previous works such as [AAY22, AAPP22] in the synchronous setting and [CP17, AAPS23] in the asynchronous setting with $t < n/4$. However their approaches do not work in our setting.

To this end, we try to follow the AVSS protocol in [CP23].

1. D sends to each P_i the degree-$2t$ column polynomial $g_i(y) = F(\alpha_i, y)$ and receives the signature on g_i from each P_i via AICP. After receiving signatures from $2t+1$ parties, D broadcasts the set $\mathcal{M}$ of these parties.

2. D sends the degree-t row polynomial $f_j(x) = F(x, \alpha_j)$ to P_j together with the signature on $f_j(\alpha_i)$ for each $P_i \in \mathcal{M}$. Then each party P_j accepts $f_j(x)$ if it is of degree t and all signatures are valid.

Let $\mathcal{W}$ be the set of parties that accept their row polynomials. As in [CP23], we can only expect $|\mathcal{W}| = 2t+1$. At this moment, each honest party $P_i \in \mathcal{M}$ and each honest party $P_j \in \mathcal{W}$ satisfy that $g_i(\alpha_j) = f_j(\alpha_i)$. However, unlike [CP23], we cannot prove that the column polynomials of honest parties in $\mathcal{M}$ and the row polynomials of honest parties in $\mathcal{W}$ lie on a valid degree-$(t, 2t)$ bivariate polynomial due to the larger degree. What makes it even worse is that we cannot reconstruct this bivariate polynomial, in the same way, as [CP23] since we need $2t + 1$ correct row polynomials to reconstruct the whole bivariate polynomial while at the end of Step 2, only parties in $\mathcal{W}$ obtain their row polynomials which may only include $t + 1$ honest parties. In fact, if we want to follow the same way as [CP23], we have to ensure that all (honest) parties can receive their row polynomials! Unfortunately, this is impossible when using techniques in [CP23] since recovering the row polynomial to a party requires the help of the dealer. When the dealer is corrupted, he may choose to not send row polynomials to some honest parties.

Our Solution. Can we reconstruct the row polynomial without the help of the dealer? We note that parties in $\mathcal{M}$ have received their column polynomials. To let a party P_j learn $f_j(x)$, we may ask each party P_i in $\mathcal{M}$ to send $f_j(\alpha_i) = g_i(\alpha_j)$ to P_j. Since there are at least $t + 1$ honest parties in $\mathcal{M}$, P_j will receive enough number of correct shares to reconstruct $f_j(x)$. However, the issue is that P_j may also receive up to t incorrect shares. P_j can't reconstruct $f_j(x)$ when t out of $2t+1$ shares are incorrect *unless P_j can distinguish correct shares and incorrect shares.*

Our idea is to establish a way to allow every party P_j to verify the shares from P_i. In this way, P_j will only use correct shares to reconstruct $f_j(x)$. To this end, our idea is to make use of *authentication tags* introduced in [BFO12]. At a high level, for a vector of values $\boldsymbol{m} \in \mathbb{F}^k$ held by P_i, P_j will prepare a pair of authentication keys $(\boldsymbol{\mu}, \nu)$ where $\boldsymbol{\mu} \in \mathbb{F}^k$ and $\nu \in \mathbb{F}$. The authentication tag is defined by $\tau = \boldsymbol{\mu} \cdot \boldsymbol{m} + \nu$, where $\cdot$ denotes the inner-product operation, and we let P_i obtain τ. Later on, when P_i sends $\boldsymbol{m}$ to P_j, P_i also sends τ to P_j so that P_j can verify the correctness of $\boldsymbol{m}$. Note that when $\boldsymbol{\mu}, \nu$ are uniformly random, the probability that P_i can find a different $(\boldsymbol{m}', \tau')$ such that $\tau' = \boldsymbol{\mu} \cdot \boldsymbol{m}' + \nu$ is negligible.

So far the key size is linear in the message length. To reduce the key size, the authors in [BFO12] observe that $\boldsymbol{\mu}$ can be reused to authenticate multiple batches of messages as long as each time we use a uniformly random ν. Thus, $\boldsymbol{\mu}$ is first randomly sampled and used as the long-term key. For each batch of messages $\boldsymbol{m}$, a random ν is sampled and the tag of $\boldsymbol{m}$ is computed by $\tau = \boldsymbol{\mu} \cdot \boldsymbol{m} + \nu$. In this way to authenticate L messages, the key size is reduced to $k + L/k$, which is sublinear in the message length.

Given the above, our idea is to compute authentication tags for every pair of parties (P_i, P_j) where $P_i \in \mathcal{M}$. With more details, we will compute authentication tags for P_i's column polynomial $g_i(y)$ defined by the column polynomials of the first $t + 1$ honest parties in $\mathcal{M}$ under P_j's authentication keys. Later on, when reconstructing P_j's row polynomial $f_j(x)$, each party $P_i \in \mathcal{M}$ sends $g_i(y)$ together with the authentication tags to P_j. Then P_j only uses shares with correct authentication tags to reconstruct $f_j(x)$. We will elaborate on how to compute authentication tags in the next subsection. In the following, we show that with the help of authentication tags, we can allow each (honest) party to obtain both his row and column polynomials with a sublinear additive communication overhead. As a result, we manage to directly upgrade from AVSS to ACSS, avoiding the other $\mathcal{O}(n)$ overhead in [CP23].

Reconstructing Row Polynomials. A small issue in the above construction is that when reconstructing P_j's row polynomial, P_j actually learns $g_i(y)$ for all $P_i \in \mathcal{M}$, which allows him to recover not only $f_j(x)$, but the whole bivariate polynomial as well.

Can we only let P_j obtain his shares rather than reconstructing the whole bivariate polynomial to him? The issue is that the authentication tags only allow P_j to verify P_i's whole column polynomial $g_i(y)$ while what P_j should learn is just a single point $g_i(\alpha_j)$. We note that the authentication tags are linearly homomorphic: For $\boldsymbol{m}, \boldsymbol{m}'$, if P_i holds $\tau = \boldsymbol{\mu} \cdot \boldsymbol{m} + \nu$ and $\tau' = \boldsymbol{\mu} \cdot \boldsymbol{m}' + \nu'$, then $\tau + \tau'$ is an authentication tag of $\boldsymbol{m} + \boldsymbol{m}'$ under the keys $(\boldsymbol{\mu}, \nu + \nu')$. Utilizing this property, our solution is as follows.

1. Suppose D distributes L bivariate polynomials $F^{(1)}(x, y), \ldots, F^{(L)}(x, y)$. We ask D to distribute another random bivariate polynomial $F^{(0)}(x, y)$. Following the above steps, P_i holds $g_i^{(0)}(y), \ldots, g_i^{(L)}(y)$ together with authentication tags $\tau^{(0)}, \ldots, \tau^{(L)}$.
2. P_i sends $g_i^{(0)}(\alpha_j), \ldots, g_i^{(L)}(\alpha_j)$ to P_j, which are values P_j should learn.
3. To check the correctness of the values received from P_i, P_j samples a random challenge r and computes $g_i(\alpha_j) = g_i^{(0)}(\alpha_j) + r \cdot g_i^{(1)}(\alpha_j) + \cdots + r^L \cdot g_i^{(L)}(\alpha_j)$. Then P_j sends r to P_i.
4. P_i computes $g_i(y) = g_i^{(0)}(y) + r \cdot g_i^{(1)}(y) + \cdots + r^L \cdot g_i^{(L)}(y)$ and the authentication tag of $g_i(y)$, $\tau = \tau^{(0)} + r \cdot \tau^{(1)} + \cdots + r^L \cdot \tau^{(L)}$. Then P_i sends $g_i(y)$ and τ to P_j.
5. P_j verifies the correctness of $g_i(y)$ and τ, and checks whether $g_i(\alpha_j)$ computed by himself lies on $g_i(y)$. If the check passes, P_j accepts the values from P_i.

The intuition is that if some of $g_i^{(0)}(\alpha_j), \ldots, g_i^{(L)}(\alpha_j)$ are incorrect, then with overwhelming probability, $g_i(\alpha_j)$ computed by P_j does not lie on $g_i(y)$. Since P_i needs to provide the authentication tag associated with $g_i(y)$, P_i cannot lie about $g_i(y)$. As a result, with overwhelming probability, the check fails and P_j rejects P_i's values. As for secrecy, since $F^{(0)}(x, y)$ is used as a random mask, P_j only learns $g_i^{(1)}(\alpha_j), \ldots, g_i^{(L)}(\alpha_j)$.

Now P_j runs the above steps to request his shares from each party in $\mathcal{M}$. Each time, we will consume a random bivariate polynomial $F^{(0)}(x, y)$ as the random mask (so we will consume $\mathcal{O}(n)$ random bivariate polynomials to reconstruct P_j's row polynomials). When P_i and P_j are both honest, P_j will eventually receive the correct shares from P_i. Thus, P_j will eventually receive enough number of correct shares and reconstruct his row polynomials $f_j^{(1)}(x), \ldots, f_j^{(L)}(x)$.

Towards ACSS. So far, with the help of authentication tags, we have shown that each (honest) party can eventually obtain his row polynomial. However, to achieve ACSS, we need to let each (honest) party P_j obtain his column polynomial $g_j(y)$.

We can ask each party P_i sends $f_i(\alpha_j)$ to P_j. Since all honest parties will eventually send shares to P_j, P_j will receive at least $2t+1$ correct shares which allow him to reconstruct $g_j(y)$. However, similarly to the reconstruction of row polynomials above, P_j may also receive up to t incorrect shares from corrupted parties. To be able to reconstruct $g_i(y)$ which is of degree $2t$, P_j needs to be able to distinguish correct shares and incorrect shares. We again rely on authentication tags to achieve this task. However, the difference is that this time P_i does not hold the authentication tag of $f_i(x)$. We observe that parties in $\mathcal{M}$ can help P_j reconstruct the whole bivariate polynomial, thus allowing P_j to check the correctness of shares received from P_i.

1. Suppose D distributes L bivariate polynomials $F^{(1)}(x,y), \ldots, F^{(L)}(x,y)$. We ask D to distribute another random bivariate polynomial $F^{(0)}(x,y)$. Following the above steps, P_i holds $f_i^{(0)}(x), \ldots, f_i^{(L)}(x)$.
2. P_i sends $f_i^{(0)}(\alpha_j), \ldots, f_i^{(L)}(\alpha_j)$ to P_j, which are values P_j should learn.
3. To check the correctness of the values received from P_i, P_j samples a random challenge r and computes $f_i(\alpha_j) = f_i^{(0)}(\alpha_j) + r \cdot f_i^{(1)}(\alpha_j) + \cdots + r^L \cdot f_i^{(L)}(\alpha_j)$. Then P_j broadcasts r to all parties.
4. Each party $P_k \in \mathcal{M}$ computes $g_k(y) = g_k^{(0)}(y) + r \cdot g_k^{(1)}(y) + \cdots + r^L \cdot g_k^{(L)}(y)$ and the authentication tag of $g_k(y)$, denoted by τ_k. Then P_k sends $g_k(y)$ and τ_k to P_j.
5. P_j verifies the correctness of each $g_k(y)$ and τ_k, and uses the first $t+1$ correct polynomials to reconstruct the bivariate polynomial $F(x,y)$. Then P_j checks whether $f_i(\alpha_j)$ computed by himself lies on $F(x,y)$. If the check passes, P_j accepts the values from P_i.

Following the same intuition, with overwhelming probability, incorrect shares from P_i will be rejected by P_j. Due to the random mask $F^{(0)}(x,y)$, P_j only learns $f_i^{(1)}(\alpha_j), \ldots, f_i^{(L)}(\alpha_j)$.

Now P_j runs the above steps to request his shares from each party. Each time, we will consume a random bivariate polynomial $F^{(0)}(x,y)$ as the random mask (so we will consume $\mathcal{O}(n)$ random bivariate polynomials to reconstruct P_j's column polynomials). When P_i and P_j are both honest, P_j will eventually receive the correct shares from P_i. Thus, P_j will eventually receive enough number of correct shares and reconstruct his column polynomials $g_j^{(1)}(y), \ldots, g_j^{(L)}(y)$.

Note that when reconstructing row polynomials and column polynomials, we only send shares that P_j should learn together with verification whose communication cost is sublinear in the number of bivariate polynomials shared by D. Thus, we manage to upgrade from AVSS to ACSS with sublinear overhead, avoiding the other $\mathcal{O}(n)$ factor in [CP23].

2.3 Preparing Authentication Tags

In this section, we show how to compute authentication tags for every pair of parties (P_i, P_j).

Recall that in the beginning, we follow [CP23] and do the following:

- D sends to each P_i the degree-$2t$ column polynomial $g_i(y) = F(\alpha_i, y)$ and receives the signature on g_i from each P_i via AICP. After receiving signatures from $2t+1$ parties, D broadcasts the set $\mathcal{M}$ of these $2t+1$ parties.

As [CP23], the column polynomials of the first $t+1$ honest parties in $\mathcal{M}$ fully determine a bivariate polynomial $F(x,y)$. We view this bivariate polynomial as the one D distributes. When D is corrupted, however, the column polynomials of other honest parties may not lie on F.

Verifying Column Polynomials. Our first step is to let each (honest) party P_j check whether the column polynomial $g_j(y)$ he received lies on $F(x, y)$.

We observe that with the help of the dealer D, a party P_j can verifiably reconstruct the bivariate polynomial $F(x, y)$: D simply sends $F(x, y)$ together with the signatures from P_i for all $P_i \in \mathcal{M}$ to P_j, and P_j accepts $F(x, y)$ if all signatures are valid. Note that if all signatures are valid, the column polynomials of the first $t + 1$ honest parties in $\mathcal{M}$ agree with $F(x, y)$ sent by D. In this case, $F(x, y)$ must be the one determined by the column polynomials of the first $t + 1$ honest parties in $\mathcal{M}$.

We note that P_j's column polynomial can be verified if we let D reconstruct $F(x, y)$ to him. However, this would reveal the whole bivariate polynomial to P_j as well. Our idea is to follow a similar idea to that in Section 2.2 by letting P_j check a random linear combination of his column polynomials of a batch of bivariate polynomials distributed by D.

1. Suppose D distributes L bivariate polynomials $F^{(1)}(x, y), \ldots, F^{(L)}(x, y)$. We ask D to distribute another random bivariate polynomial $F^{(0)}(x, y)$. Following the above steps, D obtains signatures associated with each bivariate polynomial from parties in $\mathcal{M}$.

2. After P_j receives $g_j^{(0)}(y), \ldots, g_j^{(L)}(y)$ from D, P_j samples a random challenge r and computes $g_j(y) = g_j^{(0)}(y) + r \cdot g_j^{(1)}(y) + \cdots + r^L \cdot g_j^{(L)}(y)$. Then P_j sends r to D.

3. D computes $F(x, y) = F^{(0)}(x, y) + r \cdot F^{(1)}(x, y) + \cdots + r^L \cdot F^{(L)}(x, y)$. To allow P_j verifies $F(x, y)$, our hope is that D can compute the signatures associated with $F(x, y)$ from the signatures associated with $F^{(0)}(x, y), \ldots, F^{(L)}(x, y)$. Then D sends $F(x, y)$ as well as the signatures to P_j.

4. P_j verifies the signatures and checks whether $g_j(y)$ lies on $F(x, y)$. If the check passes, P_j accepts his column polynomials.

We note that Step 3 cannot be achieved by standard signature schemes since it essentially asks the dealer to forge signatures for messages that are not signed by the signer. However, in AICP, the verification of a signature is done in a distributed way. In our work, we extend AICP to support linear operations over signatures, making Step 3 possible. Another limitation of AICP is that the signature can only be verified by a single receiver. In our case, we need each party P_j to check his column polynomial. This requires the underlying AICP to support verification by multiple receivers. We refer the readers to Section 4 for our extension of AICP (which we refer to as APICP) that supports both (1) linear operations over signatures and (2) verification by multiple receivers.

In summary, after D distributes the bivariate polynomials to all parties, each party P_j runs the above steps to check his column polynomials. This ensures that the column polynomials of all honest parties that accept their checks lie on valid degree-$(t, 2t)$ bivariate polynomials. In the following, only parties with correct column polynomials participate. Note that if D is honest, all honest parties will accept the checks.

Computing Authentication Tags. For every pair of parties (P_i, P_j), we want to compute authentication tags for the column polynomial $g_i(y)$ under P_j's authentication keys. We first review how this is achieved in the synchronous setting [BFO12].

At a high level, the idea is to first secret-share the vector $g_i = (g_i(\alpha_1), \ldots, g_i(\alpha_n))$ by $[g_i]_t$, which is a vector of n degree-t Shamir sharings, and secret-share the authentication keys $(\boldsymbol{\mu}, \nu)$ by $[\boldsymbol{\mu}]_t, [\nu]_{2t}$ respectively. Here for simplicity, we assume that $\boldsymbol{\mu}$ is of length n. Then all parties locally compute

$$[\tau]_{2t} = [g_i]_t \cdot [\boldsymbol{\mu}]_t + [\nu]_{2t},$$

and send their shares to P_i to let P_i reconstruct the tag τ. The authors in [BFO12] observed that the vector g_i has already been shared by $(g_1, g_2, \ldots, g_n)$, where $g_k = (g_k(\alpha_1), \ldots, g_k(\alpha_n))$ is known by P_k, except that the secrets are stored at position α_i rather than α_0. To utilize this observation, the authors in [BFO12] introduced the notion of *twisted secret sharings*. A degree-t twisted secret sharing $[x]_t^i$ is a degree-t Shamir sharing whose secret is stored at position α_i and the i-th share is at position α_0[2]. Effectively, we switch positions of the secret and the i-th share. Then all parties hold $[g_i]_t^i$ except that P_i does not know his share, which should be g_0. Now P_j shares his authentication keys by twisted secret sharings $[\boldsymbol{\mu}]_t^i, [\nu]_{2t}^i$ such that the shares of $[\boldsymbol{\mu}]_t^i$ of P_i are 0. Note that when P_i, P_j are both honest, the adversary learns the shares of P_i and can recover the authentication key $\boldsymbol{\mu}$ of P_j. However this is fine since an honest P_i will never forge a tag. In this way, even P_i does not know his share of $[g_i]_t^i$, he can still compute his share of $[\tau]_{2t}^i$ by following

$$[\tau]_{2t}^i = [g_i]_t^i \cdot [\boldsymbol{\mu}]_t^i + [\nu]_{2t}^i$$

since P_i's shares of $[g_i]_t^i$ should be multiplied by his shares of $[\boldsymbol{\mu}]_t^i$, which are all 0. In this way, the communication cost of computing τ is only (1) sharing the authentication keys by P_j plus (2) reconstructing τ to P_i. Recall that the long-term key $\boldsymbol{\mu}$ can be reused. For each batch of messages, P_j only needs to share the short-term key ν. As a result, the communication cost of computing τ is sublinear in the message length.

Now we try to follow the above approach in the asynchronous setting. The first issue is that in the asynchronous setting, we cannot expect that each party P_j participates in the computation of the authentication tags. To address this issue, our solution is to let all parties prepare twisted secret sharings of authentication keys for P_j. Recall that a degree-t twisted secret sharing $[x]_t^i$ is just a standard degree-t Shamir sharing except that the secret is stored at position α_i while the i-th share is at position α_0.

- For $[\boldsymbol{\mu}]_t^i$, they are random degree-t Shamir sharings with the i-th shares to be 0. In Section 3.4 (see $\mathcal{F}_{\mathsf{RandShare}}^0$), we show that such random degree-t Shamir sharings can be efficiently prepared in the $\mathcal{F}_{\mathsf{ACSS}}$-hybrid model.
- For $[\nu]_{2t}^i$, it is just a random degree-$2t$ Shamir sharing. The authors in [EGPS22] observed that the preparation of a random degree-$2t$ Shamir sharing can be reduced to preparing $t+1$ random degree-t Shamir sharings. Then all parties can obtain a random degree-$2t$ Shamir sharing via local computation. We refer the readers to Section 5 (see Π_{Auth}) for more details. We note that random degree-t Shamir sharings can be efficiently prepared in the $\mathcal{F}_{\mathsf{ACSS}}$-hybrid model as well.

Recall that the size of the authentication keys is sublinear in the message length. Thus, we could afford to use the ACSS protocol in [CP23] to instantiate $\mathcal{F}_{\mathsf{ACSS}}$ for preparing $[\boldsymbol{\mu}]_t^i$ and $[\nu]_{2t}^i$. Note that a degree-t Shamir sharing can be robustly reconstructed. To let P_j learn his authentication keys, all parties simply send their shares of degree-t Shamir sharings to P_j.

The second issue is that, when reconstructing τ from $[\tau]_{2t}^i$, P_i receives not only $2t+1$ correct shares from honest parties, but also up to t incorrect shares from corrupted parties. To be able to reconstruct the correct τ, P_i needs to distinguish correct shares from incorrect shares. To be more concrete, for each party P_k, after P_k computes his share of $[\tau]_{2t}^i$ and sends it to P_i, P_i needs to check whether the share from P_k is correct or not. Our solution consists of two steps:

- We first consider a simple scenario where we do not need to protect the secrecy of the bivariate polynomial $F(x, y)$. In this case, we ask D to reconstruct $F(x, y)$ to all parties. Now our idea is to let all parties compute a degree-t Shamir sharing of the share of P_k. Then P_i can robustly reconstruct the share of P_k and check whether it matches the value received from P_k.
- Then we reduce the original problem to the simple scenario above. At a high level, we let P_i check a random linear combination of the shares from P_k for a batch of bivariate polynomials shared by D. By adding a random bivariate polynomial as a random mask, it is safe to reveal the bivariate polynomial after random linear combinations to all parties.

Step 1. For the first step, recall that the share of P_k is computed by following $[\tau]_{2t}^i = [g_i]_t^i \cdot [\boldsymbol{\mu}]_t^i + [\nu]_{2t}^i$. Our goal is to let all parties compute a degree-t Shamir sharing of the k-th share of $[\tau]_{2t}^i$. This problem can be reduced to compute

[2] In [BFO12], the authors require that the share of P_i is always 0.

- a degree-t Shamir sharing of the k-th share of $[g_i]_t^i \cdot [\boldsymbol{\mu}]_t^i$,
- and a degree-t Shamir sharing of the k-th share of $[\nu]_{2t}^i$.

For the k-th share of $[g_i]_t^i \cdot [\boldsymbol{\mu}]_t^i$, P_k will multiply g_k, which is his share of $[g_i]_t^i$, with the k-th share of $[\boldsymbol{\mu}]_t^i$. We note that if all parties compute $g_k \cdot [\boldsymbol{\mu}]_t^i$, the result is a degree-t Shamir sharing such that the k-th share is equal to the k-th share of $[g_i]_t^i \cdot [\boldsymbol{\mu}]_t^i$. To obtain a degree-$t$ Shamir sharing of the k-th share of $[\nu]_{2t}^i$, we rely on the observation in [EGPS22] again. Recall that $[\nu]_{2t}^i$ is prepared by first preparing $t+1$ random degree-t Shamir sharings and then computing $[\nu]_{2t}^i$ via local computation. In particular, each party just computes a linear combination of his shares. Now if all parties use the same coefficients as those used by P_k, they can obtain a degree-t Shamir sharing whose k-th share is equal to the k-th share of $[\nu]_{2t}^i$. In summary, when we do not need to protect the secrecy of $F(x,y)$, we can let D reconstruct $F(x,y)$ to all parties. Then all parties can compute a degree-t Shamir sharing of the k-th share of $[\tau]_{2t}^i$. Then P_i can robustly reconstruct the share of P_k and check whether it matches the value received from P_k.

An omitted security issue above is that directly reconstructing such a degree-t Shamir sharing of P_k's share to P_i may leak information about the authentication keys of P_j. In our construction, we will add a random mask, which is a random degree-t Shamir sharing with the k-th share to be 0.

Step 2. Now for the general case, suppose D distributes L bivariate polynomials $F^{(1)}(x,y), \ldots, F^{(L)}(x,y)$. We ask D to distribute another random bivariate polynomial $F^{(0)}(x,y)$ as a random mask. Then all parties hold $[g_i^{(0)}]_t^i, \ldots, [g_i^{(L)}]_t^i$, the long-term key $[\boldsymbol{\mu}]_t^i$, and the short-term keys $[\nu^{(0)}]_{2t}^i, \ldots, [\nu^{(L)}]_{2t}^i$. All parties locally compute $[\tau^{(0)}]_{2t}^i, \ldots, [\tau^{(L)}]_{2t}^i$ and each party P_k sends his shares to P_i.

To check P_k's shares, P_i samples a random challenge r and broadcasts it to all parties. Let

$$\begin{aligned}
F(x,y) &= F^{(0)}(x,y) + r \cdot F^{(1)}(x,y) + \cdots + r^L \cdot F^{(L)}(x,y), \\
[g_i]_t^i &= [g_i^{(0)}]_t^i + r \cdot [g_i^{(1)}]_t^i + \cdots + r^L \cdot [g_i^{(L)}]_t^i, \\
[\nu]_{2t}^i &= [\nu^{(0)}]_{2t}^i + r \cdot [\nu^{(1)}]_{2t}^i + \cdots + r^L \cdot [\nu^{(L)}]_{2t}^i, \\
[\tau]_{2t}^i &= [\tau^{(0)}]_{2t}^i + r \cdot [\tau^{(1)}]_{2t}^i + \cdots + r^L \cdot [\tau^{(L)}]_{2t}^i.
\end{aligned}$$

Then the problem becomes to check P_k's share of $[\tau]_{2t}^i$. Due to the random bivariate polynomial $F^{(0)}(x,y)$, we do not need to protect the secrecy of $F(x,y)$. Thus, the problem is reduced to the simple scenario considered in Step 1.

Summary. In summary, the above approach allows all parties to prepare the authentication keys for P_j and let P_i obtain the authentication tags of his column polynomials. We will do these steps for every pair of parties (P_i, P_j). When P_i accepts his column polynomials and obtains authentication tags for all other parties $P_1, \ldots, P_n$, P_i broadcasts Tag_i. The sharing phase finishes after $2t+1$ parties broadcast Tag_i.

2.4 Brief Outline of Our ACSS Protocol

We give a brief outline of our ACSS protocol. We divide the whole protocol into four phases: the sharing phase, the verification phase, the authentication phase, and the completion phase.

Sharing Phase. During the sharing phase, the dealer D distributes the shares to all parties. More concretely, D encodes each batch of $t+1$ degree-t Shamir secret sharings into a degree-$(t,2t)$ bivariate polynomial. Then we follow [CP23] to let D distribute the degree-$2t$ column polynomials to all parties and wait to receive signatures on these column polynomials (through our extension of AICP) from all parties. Upon receiving $2t+1$ parties' signatures, D creates a set $\mathcal{M}$ containing these parties and broadcasts it. The bivariate polynomials shared by D are fully determined by the column polynomials of the first $t+1$ honest parties in $\mathcal{M}$.

Verification Phase. During the verification phase, each party verifies whether his column polynomials are consistent with the bivariate polynomials determined by the first $t+1$ honest parties in $\mathcal{M}$. We refer the readers to Section 2.3 for the overview of the verification.

Authentication Phase. During the authentication phase, for every pair of parties (P_i, P_j), all parties help prepare the authentication keys for P_j and compute the authentication tags for P_i. We refer the readers to Section 2.3 for the overview of the computation of tags.

If an honest party P_i accepts his column polynomials and obtains authentication tags for all other parties $P_1, \ldots, P_n$, P_i broadcasts a flag Tag_i. Then D creates a set $\mathcal{W}$ containing parties whose Tag_i are received. Finally, D broadcasts $\mathcal{W}$ when $|\mathcal{W}| = 2t + 1$ for public verification.

Completion Phase. During the completion phase, with the help of authentication tags, each party reconstructs his row polynomials first and then his column polynomials to get his shares. We refer the readers to Section 2.2 for the overview of this step.

3 Preliminaries

Notation. For any $N \in \mathbb{Z}^+$, we denote $[N] = \{1, \ldots, N\}$. For $a, b \in \mathbb{Z}$ with $a < b$, we denote $[a, b] = \{a, a+1, \ldots, b\}$. We denote the security parameter by κ. $\mathbb{F}$ is used to denote a finite field where $|\mathbb{F}| = 2^{\Theta(\kappa)}$. We denote the inner product of two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}^L$ by $\boldsymbol{u} \cdot \boldsymbol{v}$.

We assume that $n = \mathsf{poly}(\kappa)$, where $\mathsf{poly}(\kappa)$ denotes a polynomial function of κ. We also use $\mathsf{negl}(\kappa)$ to denote a negligible function to κ, which means the function is smaller than any $\mathsf{poly}(\kappa)/2^\kappa$ for sufficiently large κ. All the polynomials we mention are over $\mathbb{F}$. A degree-d polynomial is of the form $f(x) = a_0 + \cdots + a_d x^d$, where each $a_i \in \mathbb{F}$. A degree-(ℓ, m) bivariate polynomial is of the form $F(x, y) = \sum_{i,j=0}^{i=\ell, j=m} r_{ij} x^i y^j$, where each $r_{ij} \in \mathbb{F}$.

3.1 The Security Model

In our work, we follow the security model in [CP23, Coh16].

The UC Framework. We use the UC framework introduced by Canetti [Can01] to define the security of our protocols, based on the *real and ideal world paradigm* [Can00]. Informally, we consider a protocol Π to be secure if its execution in the real world can also be done in the ideal world.

Real World. In the real world, there exists a set of n parties $P_1, \ldots, P_n$, an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$. The environment provides inputs to the honest parties, receives their outputs, and communicates with the adversary $\mathcal{A}$. We consider $\mathcal{A}$ to be *fully-malicious*. The adversary can corrupt up to t parties and completely control the behavior of the corrupted parties, where $t < n/3$. The parties not controlled by $\mathcal{A}$ are called honest. For simplicity, we consider a static adversary who selects the set of corrupted parties at the beginning of the protocol.

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

Parties have access to a network of point-to-point asynchronous and secure channels. Asynchronous channels guarantee *eventual* delivery [CR93], meaning that messages sent are eventually delivered. To model the worst-case scenario, the adversary is given the provision to decide the arrival time of each message exchanged between the parties. The adversary cannot drop, change, or inject messages from honest parties. Such channels have been modelized in UC using the *eventual-delivery secure message-transmission* ideal functionality, for example in [KMTZ13, CGHZ16]. The protocol completes once $\mathcal{Z}$ outputs a single bit.

We denote by $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z, \bar{r})$ the random variable containing the output of $\mathcal{Z}$ with input z, security parameter κ, and interacting with the parties $P_1, \ldots, P_n$ and the adversary A with random tapes $\bar{r} = (r_1, \ldots, r_n, r_\mathcal{A}, r_\mathcal{Z})$. We denote the random variable $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z, \bar{r})$ for uniformly random $\bar{r}$ by $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$.

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

The ideal functionality $\mathcal{F}$ models the desired behavior of the computation. $\mathcal{F}$ only receives inputs from the parties and $\mathcal{S}$ and provides outputs to them. $\mathcal{S}$ cannot see or delay the communication between the honest parties and $\mathcal{F}$. In order to model the fact that the adversary can decide when each honest party learns the output, we follow [KMTZ13] and model time via activations. We use a *request-based delay output* to model the output delivery from $\mathcal{F}$ to the honest parties, which is used in [Coh16, CFG$^+$23]. In this model, the functionality $\mathcal{F}$ doesn't directly send the output to the honest parties. Instead, honest parties need to send a "request" to the functionality to get the output. Moreover, the adversary can instruct $\mathcal{F}$ to delay the output for each party by ignoring the corresponding requests. The output can only be delayed for a polynomial number of times, which ensures that the output will eventually be delivered if an honest party sends sufficiently many requests.

We denote by $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z, \bar{r})$ the random variable containing the output of $\mathcal{Z}$ with input z, security parameter κ, and interacting with the parties $P_1, \ldots, P_n$ and the adversary S with random tapes $\bar{r} = (r_{\mathcal{S}}, r_{\mathcal{Z}})$. We denote the random variable $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z, \bar{r})$ for uniformly random $\bar{r}$ by $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z)$.

Perfect and Statistical Security. We say Π t-securely realizes $\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ in the ideal model such that for any adversary controlling up to t parties and any environment $\mathcal{Z}$, it holds that:

$$\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z) \equiv \mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z).$$

We say Π t-securely realizes $\mathcal{F}$ with statistical security if for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ in the ideal model such that for any adversary controlling up to t parties and any environment $\mathcal{Z}$, it holds that:

$$\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z) \equiv_{\epsilon} \mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z),$$

which means the output distributions of the real-world execution and the ideal-world execution are statistically close. I.e., the total variation distance between the two distributions is no more than $\epsilon = \mathsf{negl}(\kappa)$.

The Hybrid Model. In a $\mathcal{G}$-hybrid model, a protocol execution proceeds as in the real world except that the parties have access to an ideal functionality $\mathcal{G}$ for some specific task. During the protocol execution, the parties can communicate with $\mathcal{G}$ as in the ideal world. The UC framework guarantees that an ideal functionality in a hybrid model can be replaced with a protocol that UC-securely realizes $\mathcal{G}$. This is guaranteed by the following composition theorem from [Can01, Can20].

Theorem 4. ([Can01, Can20]) *Let Π be a protocol that UC-securely realizes a functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model and let ρ be a protocol that UC-securely realizes $\mathcal{G}$. Moreover, let Π^{ρ} denote the protocol that is obtained from Π by replacing every ideal call to $\mathcal{G}$ with the protocol ρ. Then protocol Π^{ρ} UC-securely realizes $\mathcal{F}$ in the model where the parties do not have access to the ideal functionality $\mathcal{G}$.*

Hybrid Arguments. We use hybrid arguments to prove that the distributions of $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$ and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z)$ are identical (or statistically close). In each hybrid, we use output distribution to denote the distribution of the random variable containing what $\mathcal{Z}$ outputs on input z and uniformly random $\bar{r}$. We construct a group of hybrids between the real-world scenario and the ideal-world scenario. If the output distributions of each two adjacent hybrids are identical (or statistically close), the distributions of $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$ and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z)$ are also identical (or statistically close).

3.2 Agreement Primitives

In our constructions of protocols, we need the *agree on a common set* (ACS) primitive to let the parties agree on a set of at least $n - t$ parties that satisfies a certain property Q (a so-called ACS property). Ben-Or, Kelmer, and Rabin provide an efficient ACS protocol Π_{ACS}^{Q} in [BKR94], which achieves communication complexity of $\mathcal{O}(n^3)$.

We also need an *A-Cast* protocol to enable a party to broadcast a message in the asynchronous network. From [Bra84], broadcasting an ℓ-bit message requires $\mathcal{O}(n^2\ell)$-bit communication.

For completeness, additional definitions of the agreement primitives are provided in Appendix A.

3.3 Secret Sharing

For fixed $(\alpha_0, \alpha_1, \ldots, \alpha_n) \in \mathbb{F}^{n+1}$, we introduce the secret sharing schemes of a value $\omega \in \mathbb{F}$ below.

- A *degree-t (or $2t$) Shamir sharing* [Sha79] of $\omega \in \mathbb{F}$ with respect to $(\alpha_0, \alpha_1, \ldots, \alpha_n) \in \mathbb{F}^{n+1}$ consists of n shares $\omega_1, \ldots, \omega_n \in \mathbb{F}$ of the following form: there exists a sharing polynomial $f(X) \in \mathbb{F}[X]$ of degree at most t (or $2t$) such that $\omega = f(\alpha_0)$ and $\omega_j = f(\alpha_j)$ for $j \in \{1, \ldots, n\}$. Furthermore, share ω_j is held by player P_j for $j \in \{1, \ldots, n\}$. We denote such a sharing as $[\omega]_t$ (or $[\omega]_{2t}$) with respect to $(\alpha_0, \alpha_1, \ldots, \alpha_n)$.

- A *degree-t (or $2t$) twisted sharing* [BFO12] of $\omega \in \mathbb{F}$ with respect to $P_i \in \mathcal{P}$ is $[\omega]_t$ (or $[\omega]_{2t}$) with respect to $(\alpha_i, \alpha_1, \ldots, \alpha_{i-1}, \alpha_0, \alpha_{i+1}, \ldots, \alpha_n)$, i.e. secret is now $f(\alpha_i)$, P_i's share is $f(\alpha_0)$. We denote this sharing by $[\omega]_t^i$ (or $[\omega]_{2t}^i$).

- We say a secret sharing $[\omega]_t$ (or $[\omega]_t^i$) is a t-sharing if the sharing polynomial is of degree-t. A t-sharing is called a complete t-sharing if every honest party holds his share of ω and all the parties' shares of ω lie on a degree-t polynomial.

- We denote a vector of degree-t sharings $([\omega^{(1)}]_t, \ldots, [\omega^{(m)}]_t)$ as $[\boldsymbol{\omega}]_t$, where $\boldsymbol{\omega} = (\omega^{(1)}, \ldots, \omega^{(m)})$.

- For fixed $(\alpha_{-t}, \ldots, \alpha_{-1})$, a *degree-$t$ degenerate packed secret sharing* of $(\omega^{(0)}, \ldots, \omega^{(t)}) \in \mathbb{F}^{t+1}$ with respect to $P_i \in \mathcal{P}$ consists of n shares $\omega_1, \ldots, \omega_n \in \mathbb{F}$ of the following form: there exists a sharing polynomial $f(X) \in \mathbb{F}[X]$ of degree at most t such that $\omega^{(j)} = f(\alpha_{-j})$ for $j \in [t]$, $\omega^{(0)} = f(\alpha_i)$, $\omega_i = f(\alpha_0)$ and $\omega_j = f(\alpha_j)$ for $j \in [n]\backslash\{i\}$. Furthermore, share ω_j is held by player P_j for $j \in [n]$. It's easy to see that $\boldsymbol{\omega}$ uniquely determines the polynomial $f(X)$, we denote such a sharing as $[\![\boldsymbol{\omega}]\!]_t^i$, where $\boldsymbol{\omega} = (\omega^{(0)}, \ldots, \omega^{(t)})$.

3.4 Sub-protocols

In this section, we introduce the sub-protocols we need in constructing our ACSS protocol. An ACSS protocol should satisfy that, if the dealer is honest, the protocol must distribute a complete t-sharing to all the honest parties. When the dealer is corrupted, once an honest party accepts his share as his output, each honest party must get his share of a t-sharing eventually. We give the functionality of ACSS in Fig. 1.

The state-of-the-art protocol that securely realizes $\mathcal{F}_{\mathsf{ACSS}}$ is given in [CP23], with an amortized communication cost of $\mathcal{O}(n^3\kappa)$ per sharing. Since we need to invoke $\mathcal{F}_{\mathsf{ACSS}}$ in constructing our sub-protocols, we state their result in Lemma 1.

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

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

Upon receiving $(\mathsf{Dealer}, \mathsf{ACSS}, \{q_1(\cdot), \ldots, q_N(\cdot)\})$ from $D \in \mathcal{P}$, the trusted party does the following:

1. If all the polynomials $q_1(\cdot), \ldots, q_N(\cdot)$ are degree-t polynomials, the trusted party sends an request-based delayed output $\{q_1(\alpha_i), \ldots, q_N(\alpha_i)\}$ to each party $P_i \in \mathcal{P}$.
2. If any of the $\{q_1(\cdot), \ldots, q_N(\cdot)\}$ is not a degree-t polynomial, the trusted party does nothing.

Figure 1: Ideal functionality for asynchronous complete secret sharing

Lemma 1. ([CP23]) *There exists a protocol that t-securely realizes $\mathcal{F}_{\mathsf{ACSS}}$ for any $N \in \mathbb{Z}^+$ with statistical security and $\mathcal{O}(N \cdot n^3\kappa + n^4\kappa^2 + n^5)$-bit communication.*

Note that in an ACSS protocol, all the parties agree on $\alpha_0, \ldots, \alpha_n$ and generate sharing with respect to these points. However, in sub-protocols, we may need to generate sharing with respect to different sets of field elements, so we use $\beta_0, \ldots, \beta_n$ instead of $\alpha_0, \ldots, \alpha_n$ in constructing sub-protocols in this section.

Private Reconstruction. Given some complete t-sharings, all parties can reconstruct the secrets to a single party privately through the *online error-correction* (OEC) process [Can96]. Besides, the OEC process enables a party to reconstruct the whole sharings, which is useful in our construction of the remaining protocols. We give the functionality of private reconstruction in Fig. 2. For completeness, we give the construction and the security proof of Π_{privRec} that realizes $\mathcal{F}_{\mathsf{privRec}}$ in Appendix B.1. The communication complexity of the protocol is $\mathcal{O}(Nn\kappa)$ bits.

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

Public Input: The receiver R, number N.

For complete t-sharing $[s_1]_t, \ldots, [s_N]_t$:

1: The trusted party receives the set of corrupted parties $\mathcal{C} \subset \mathcal{P}$.

2: Upon receiving a request $(\mathsf{Request}, \mathsf{privRec}, R, N)$ from an honest party, for all $i \in [N]$, the trusted party receives the shares of $[s_i]_t$ from all honest parties and sends the whole sharing $[s_i]_t$ as a request-based delayed output to R.

3: The trusted party sends the corrupted parties' shares to the ideal adversary.

Figure 2: Ideal functionality for private reconstruction

Preparing Random Sharings. The description of functionality $\mathcal{F}_{\mathsf{RandShare}}$ is in Fig. 3. The functionality can generate random degree-t sharings based on the corrupted parties' shares and then distribute the honest parties' shares to them. This functionality can be securely computed by letting each party run an ACSS protocol to randomly share some secrets. Then, the parties run an ACS protocol to decide a set in which each party correctly shares the secrets. Then the random sharings can be extracted from the sharings generated by these parties relying on known techniques in [DN07]. The concrete construction and security poof of $\Pi_{\mathsf{RandShare}}$ to realize $\mathcal{F}_{\mathsf{RandShare}}$ is present in Appendix B.2. The communication complexity of the protocol is $\mathcal{O}(N \cdot n^3\kappa + n^5\kappa^2 + n^6)$ bits.

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

Public Input: $(\beta_0, \ldots, \beta_n)$, number N.

1: The trusted party receives the set $\mathcal{C}$ of corrupted parties and waits to receive a request $(\mathsf{Request}, \mathsf{RandShare}, N)$ from an honest party, where $N \in \mathbb{Z}^+$.

2: For all $\ell \in [N]$, the trusted party randomly samples $r_\ell \in \mathbb{F}$.

3: For all $\ell \in [N]$, the trusted party receives a set of shares of corrupted parties from S and samples a random degree-t sharing $[r_\ell]_t$ with respect to $(\beta_0, \ldots, \beta_n)$ based on the shares of corrupted parties and the secret r_ℓ. (If not received, the trusted party sets the shares of corrupted parties to be 0.)

4: For all $\ell \in [N]$ and $P_i \in \mathcal{P}$, the trust party sends P_i's share of $[r_\ell]_t$ as a request-based delayed output to P_i.

Figure 3: Ideal functionality for preparing random t-sharings

Preparing Random Sharings with a Zero Share. We also need to prepare random sharings where a specific party P_i's shares are equal to 0. To prepare such random sharings, we provide a functionality $\mathcal{F}^0_{\mathsf{RandShare}}$ in Fig. 4. The protocol that realizes this functionality is similar to $\Pi_{\mathsf{RandShare}}$ except that each party needs to share secrets with P_i's shares equal to 0. We provide the protocol $\Pi^0_{\mathsf{RandShare}}$ to realize this functionality and the security proof in Appendix B.3. The communication complexity of the protocol is $\mathcal{O}(N \cdot n^3\kappa + n^5\kappa^2 + n^6)$ bits.

Functionality $\mathcal{F}^0_{\mathsf{RandShare}}$

Public Input: $(\beta_0, \ldots, \beta_n)$, the party P_i, number N.

1: The trusted party receives the set $\mathcal{C}$ of corrupted parties waits to receive a request

(Request, RandShare^0, N, P_i) from an honest party, where $N \in \mathbb{Z}^+$ and $P_i \in \mathcal{P}$.

2: For all $\ell \in [N]$, the trusted party randomly samples $r_\ell \in \mathbb{F}$[1].

3: For all $\ell \in [N]$, the trusted party receives a set of shares of corrupted parties from S. Then the trusted party sets P_i's shares to be 0 and samples a random degree-t sharing $[r_\ell]_t$ with respect to $(\beta_0, \ldots, \beta_n)$ based on the shares of corrupted parties and the secret r_ℓ. If not received, the trusted party sets the shares of corrupted parties to be 0.

4: For all $\ell \in [N]$ and $P_i \in \mathcal{P}$, the trust party sends P_i's share of $[r_\ell]_t$ as a request-based delayed output to P_i.

[1] If the number of corrupted parties is equal to t and P_i is honest, the adversary knows the whole sharing based on corrupted parties' shares. In this case, we don't need to sample this random value.

Figure 4: Ideal functionality for preparing random t-sharing with a zero share

4 The Asynchronous Packed Information-Checking Protocol (APICP)

In this section, we present our construction of APICP, which is extended from the previous work of AICP [PCR09]. We attach extra properties over AICP, the linear homomorphic property, and the support of multiple revelations (see Section 2.3 for why we need these properties). We note that an information-checking protocol that is linear homomorphic and supports multiple revelations has been constructed in the synchronous settings in [AKP23]. However, their construction does not achieve a constant overhead in the message length as AICP achieves, which will lead to additional overhead in communication.

Overview of AICP. AICP is a signature scheme among a dealer D, an intermediary I, and a receiver R. The dealer D wants to sign on a message he sends to I, and when I delivers this message together with the signature to R, R can check the signature to know whether this message is from D.

We encode the message sent by D to a vector $\boldsymbol{s} \in \mathbb{F}^L$. At a high level, the previous AICP [PCR09] is achieved by the following three steps.

- **Step 1: Generating a Signature on the Vector.** D samples a random degree-$(L + t\kappa)$ polynomial $f(x)$ whose L highest coefficients form the vector $\boldsymbol{s}$. For each party P_i, D randomly samples κ elements $\alpha_1^{(i)}, \ldots, \alpha_\kappa^{(i)}$ in $\mathbb{F}$ as base points and computes their corresponding verification points $(\alpha_1^{(i)}, f(\alpha_1^{(i)})), \ldots, (\alpha_\kappa^{(i)}, f(\alpha_\kappa^{(i)}))$ on $f(x)$. Then, D sends $f(x)$ to I and distributes the verification points to each party.

The polynomial $f(x)$ together with the verification points can be considered as a signature on the vector $\boldsymbol{s}$. However, the signature may not be correctly sent, so I should run a verification process to verify the validity of the signature.

- **Step 2: Verifying the Validity of the Signature.** Upon receiving verification points from D, each party randomly sends half of them to I. When I receives $f(x)$, for each party who sends verification points to him, he checks whether all of the points lie on $f(x)$. Once $2t+1$ parties satisfy this condition, I accepts $f(x)$. Then, the signature is valid and can be sent to the receiver.

Since each honest party's verification points are grouped into two sets randomly, if one of them consists of all correct points, there will be a correct verification point in another set with high probability.

- **Step 3: Revelation and Signature Checking.** When I accepts $f(x)$, he can send it to R. All parties send the rest of their verification points to R. When R receives $f(x)$, for each party who sends verification points to him, he checks whether at least one of them lies on $f(x)$. If $t + 1$ parties satisfy this condition, R obtains $\boldsymbol{s}$ from $f(x)$ and believes that it comes from D.

Since at least $t + 1$ honest parties' verification points are verified by I, their verification points can be accepted by R with high probability if I doesn't change the polynomial sent by D. This shows that once

I accepts the signatures, D can't deny that the message is sent by him when I is honest. In addition, if I reveals a different polynomial from what he receives from an honest dealer D, he must correctly guess a verification point held by an honest party to ensure that there exists a point from an honest polynomial lies on the polynomial he sends. This error probability is also negligible since the field is sufficiently large.

Note that each party only has two sets of verification points for a signature, one set for I and the other set for the receiver. As a result, the signature can only be used to convince a single receiver.

The Functionality of APICP. Now we explain the two extra properties of APICP over AICP.

- *Linear Homomorphism:* A linear combination of the signatures can be used to check the same linear combination of the messages.
- *Multiple Revelations:* A valid signature can be revealed to different receivers multiple times.

We give the functionality of our APICP in Fig. 5. We divide APICP into two phases, the initialization phase and the revelation phase. Once the initialization phase is invoked by D, the revelation phase can be invoked for at most T times. Before each invocation of the revelation phase, we assume that all parties agree on the request ($\mathsf{Request}, \mathsf{APICP}, R, \boldsymbol{c}$). This will be guaranteed by our ACSS protocol.

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

For fixed dealer D and intermediary I:

Initialization Phase: Init$(T, (\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}))$

1. The trusted party receives the identities of corrupted parties $\mathcal{C} \subset \mathcal{P}$.
2. Upon receiving $(\mathsf{Init}, \mathsf{APICP}, T, (\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}))$ from D, the trusted party sends a request-based delayed output $(D, \mathsf{APICP}, (\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}))$ to I and sets $\mathsf{count} = T$.

Revelation Phase: Rev$(R, \boldsymbol{c}, (\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}))$

3. Each time the trusted party receives a request $(\mathsf{Request}, \mathsf{APICP}, R, \boldsymbol{c})$ from an honest party, where $\boldsymbol{c} = (c_1, \ldots, c_m)$, If $\mathsf{count} > 0$, the trusted party does the following things and replaces count by $\mathsf{count} - 1$.
   - If $I \in \mathcal{C}$, the trusted party waits to receive an instruction from $\mathcal{S}$.
     - If $\mathcal{S}$ sends Ignore, the trusted party does nothing.
     - If $\mathcal{S}$ sends $\mathsf{Proceed}$, if $D \in \mathcal{C}$, the trusted party waits to receive $\boldsymbol{s}'$ from $\mathcal{S}$ and sends a request-based delayed output $\boldsymbol{s}'$ to the receiver R. Otherwise, the trusted party sends a request-based delayed output $\boldsymbol{s} = \sum_{k=1}^{m} c_k \cdot \boldsymbol{s}^{(k)}$ to the receiver R.
   - If $I \notin \mathcal{C}$, the trusted party sends a request-based delayed output $\boldsymbol{s} = \sum_{k=1}^{m} c_k \cdot \boldsymbol{s}^{(k)}$ to the receiver R.
4. If R is honest, R outputs the results received from the trusted party. Corrupted parties may output anything they want.

Figure 5: Ideal functionality for APICP

Overview of Our APICP Construction. Now we explain the high-level ideas about how to realize the APICP functionality.

In the beginning, like the previous AICP construction, D sends vectors together with the signatures. In APICP, D should send a batch of m vectors. To let our protocol satisfy the property of linear homomorphism, for each party, we let D choose the same base points for each vector. If the m vectors are sent via polynomials $f^{(1)}(x), \ldots, f^{(m)}(x)$ (where each vector is still the highest L coefficients of the corresponding polynomial), each verification point is of the form $(\alpha, f^{(1)}(\alpha), \ldots, f^{(m)}(\alpha))$. Then, if we need I to reveal a linear combination of the vectors, he just needs to send the linear combination of the polynomials. Each party can compute the same linear combination of $f^{(1)}(\alpha), \ldots, f^{(m)}(\alpha)$ for each base point α of his verification points and generate a new verification point for the linear combination of the polynomials.

To let our protocol support multiple revelations, we use the simple idea of letting D send more verification points, and each party can divide them into more sets. To be more concrete, to support T times of revelations, the verification points are randomly divided into $T+1$ sets. The first set is used for verification of the validity of the signatures. Then each time, we use a fresh set for the revelation. To maintain a negligible error probability, we let D send $(T+1)^2\kappa$ verification points to each party. In Theorem 5, we show that this allows us to achieve negligible error probability.

4.1 Our Instantiation of APICP

Our Π_{APICP} consists of Π_{Init} and Π_{Rev}, which correspond to the initialization phase and revelation phase respectively.

Π_{Init} is present in Fig. 6. In this protocol, D samples the polynomials to store the vectors and creates the signature by randomly sampling verification points on the polynomials and distributing them to all the parties. D then sends the polynomials to I. Each party chooses a set of $(T+1)\kappa$ verification points and sends it to I. When I receives the polynomials, he checks whether at least $2t+1$ parties' verification points are on the polynomials. If true, I accepts the polynomials, which means that I receives the signatures on the vectors. The communication complexity of Π_{Init} is $\mathcal{O}(mL\kappa + mnT^2\kappa^2)$ bits.

Protocol Π_{Init}

Protocol Init(D, I, T, m, L, κ)

Parameter: The identity of D and I, revelation times T, length of each secret vector L, number of secret vectors m, and security parameter κ.

1. D receives his input $(\boldsymbol{s}^{(1)}, \dots, \boldsymbol{s}^{(m)})$ from the environment.
2. D picks $n(T+1)^2\kappa$ random elements from $\mathbb{F}$, denoted by $\alpha_1^i, \cdots, \alpha_{(T+1)^2\kappa}^i$, where $i \in [n]$.
3. For each $\boldsymbol{s}^{(k)} = (s_1^{(k)}, \cdots, s_L^{(k)})$, where $k \in [m]$, D selects a random degree-$(L + t(T+1)^2\kappa)$ polynomial $f^{(k)}(x)$ whose the L highest coefficients are elements in $\boldsymbol{s}^{(k)}$.
4. D sends $f^{(1)}(x), \dots, f^{(m)}(x)$ to I and verification point $z_j^i = (\alpha_j^i, f^{(1)}(\alpha_j^i), \dots, f^{(m)}(\alpha_j^i))$ to party P_i for $j \in \left[(T+1)^2\kappa\right]$.
5. Each party P_i randomly divides $\{z_j^i\}_{j\in[(T+1)^2\kappa]}$ into $T+1$ disjoint sets, where each set is of size $(T+1)\kappa$, denoted by $Z_1^i, \dots, Z_{T+1}^i$.
6. Each party P_i sends Z_{T+1}^i to I.
7. I checks whether $\{f^{(k)}(x)\}_{k\in[m]}$ are all of degree $(L + t(T+1)^2\kappa)$. If true, I does the following computation:
   (a) Upon receiving Z_{T+1}^i from P_i and $|Z_{T+1}^i| = (T+1)\kappa$, I checks whether the verification points in Z_{T+1}^i are all consistent with $f^{(1)}(x), \dots, f^{(m)}(x)$.
   (b) If for at least $2t+1$ parties, the above condition is satisfied, then I accepts $\{f^{(k)}(x)\}_{k\in[m]}$.

Figure 6: The protocol of the Π_{Init}

Π_{Rev} is present in Fig. 7. We assume that all parties agree on the coefficients of a linear combination before the protocol is executed, which will be guaranteed by our ACSS protocol. When the protocol begins, I sends the linear combination of the polynomials to R. Each party sends the linear combination of $(T+1)\kappa$ verification points to R. R then checks the signature by checking each party's verification points. Suppose at least $t+1$ party sends a verification point that lies on the polynomial he receives from I. In that case, he can believe that the linear combination of the vectors implied from the polynomial is sent from D. Π_{Rev} can be executed t times per initialization of APICP, the communication complexity is $\mathcal{O}(L\kappa + nT^2\kappa^2)$ bits for each execution.

Protocol Π_{Rev}

Protocol Rev$(I, R, \boldsymbol{c} = (c_1, \dots, c_m), \mathsf{count}, T, L, \kappa)$

Parameter: The identity of I, R, vector $\boldsymbol{c}$, counter count, revelation times T, packed secrets number m, each secret vector length L and security parameter κ.

1. If I accepts $\{f^{(k)}(x)\}_{k\in[m]}$, he sends $f(x) = \sum_{k=1}^m c_k f^{(k)}(x)$ to R.
2. P_i sends $Z_{\mathsf{count}}^{i,\boldsymbol{c}} = \{(\alpha_j^i, \sum_{k=1}^m c_k f^{(k)}(\alpha_j^i))\}_{z_j^i \in Z_{\mathsf{count}}^i}$ to R, where $j \in [(T+1)\kappa]$.
3. R does the following computation:
   (a) Upon receiving the set $Z_{\mathsf{count}}^{i,\boldsymbol{c}}$ from P_i and $|Z_{\mathsf{count}}^{i,\boldsymbol{c}}| = (T+1)\kappa$, R checks whether there exists at least one point $(\alpha_j^i, \beta_j^i) \in Z_{\mathsf{count}}^{i,\boldsymbol{c}}$ satisfies $f(\alpha_j^i) = \beta_j^i$.
   (b) If for at least $t+1$ parties, the above condition is satisfied, let $\boldsymbol{s}$ be the L highest coefficients of $f(x)$, R outputs $\boldsymbol{s}$.

Figure 7: The protocol of the Π_{Rev}

Π_{APICP} is present in Fig. 8. Its communication complexity is $\mathcal{O}(mL\kappa + mnT^2\kappa^2 + LT\kappa + nT^3\kappa^2)$ bits, we will give a detailed complexity analysis in Appendix C.

Protocol Π_{APICP}

Parameter: The identity of dealer D and intermediary I, packed secrets number m, each secret vector length L and security parameter κ:

Initialization Phase: Init(T)

1. All parties participate in Π_{Init}.
2. All parties initialize a counter $\mathsf{count} = T$.

Revelation Phase: Rev$(R, \boldsymbol{c})$

3. All parties participate in Π_{Rev} with inputs $R, \boldsymbol{c}$ and replace count by $\mathsf{count} - 1$.

Figure 8: The protocol of the Π_{APICP}

Theorem 5. *The protocol Π_{APICP} realizes $\mathcal{F}_{\mathsf{APICP}}$ with statistically security and $\mathcal{O}(mL\kappa + mnT^2\kappa^2 + LT\kappa + nT^3\kappa^2)$-bit communication.*

We prove Theorem 5 in Appendix C.

5 The Asynchronous Secret Sharing Protocol (ACSS)

In this section, we provide our ACSS protocol Π_{ACSS} with linear communication complexity. Recall that we have present $\mathcal{F}_{\mathsf{ACSS}}$ in Section 3.4. A dealer is allowed to send degree-t sharing polynomials $q_1(x), \ldots, q_N(x)$ to $\mathcal{F}_{\mathsf{ACSS}}$, and $\mathcal{F}_{\mathsf{ACSS}}$ will distribute each honest party's shares if the polynomials are valid.

5.1 Our Instantiation of ACSS

All parties execute Π_{Sh}, Π_{Ver}, Π_{Auth} and Π_{Comp} protocols in sequence to realize our Π_{ACSS}. Our Π_{ACSS} (see Fig. 26) consists of four different phases as we have described in Section 2.4, and we present them one by one. The parameters used in our protocols are defined at the beginning of the first phase.

The sharing phase Π_{Sh} is present in Fig. 9. The dealer D distributes shares of secrets to all parties in this phase. Firstly, D encodes each batch of $t+1$ degree-t polynomials into a degree-$(t, 2t)$ bivariate polynomial and distributes the degree-$2t$ column polynomials to all the parties. Each party's shares of secrets are $g(\alpha_{-t}), \ldots, g(\alpha_0)$ for each column polynomial g held by him. Each party then invokes $\mathcal{F}_{\mathsf{APICP}}$ to create signatures on each of his column polynomials received from D and sends them to D together with his column polynomials. If they are received, D includes the party into a set $\mathcal{M}$. When the size of $\mathcal{M}$ reaches $2t+1$, D broadcasts the set to let all the parties verify it. Regardless of the communication cost of APICP, the communication complexity of Π_{Sh} is $\mathcal{O}(mLn\kappa + n^3 \log n)$ bits.

Protocol Π_{Sh}

Parameter: All parties agree on distinct public field elements $\alpha_{-t}, \ldots, \alpha_{-1}, \alpha_0, \alpha_1, \ldots, \alpha_n$ in $\mathbb{F}$, number of polynomials N. Let $\mathcal{F}_{\mathsf{APICP}}(S, I)$ denote $\mathcal{F}_{\mathsf{APICP}}$ with dealer S and intermediary I, and let L denote the vector length in $\mathcal{F}_{\mathsf{APICP}}$.

Initialization: Let $L' = L/n$ be the number of polynomials packed in a single vector. All polynomials are divided into $m' = \frac{N}{L'(t+1)}$ groups of size $L'(t+1)$.

Let $T = n^3 + n$ and $T' = 2n^2$. Later on, T will be the number of revelations in $\mathcal{F}_{\mathsf{APICP}}$, and T' will be the number of reconstruction times in the Completing Phase. Let $m = m' + T + T'$.

Sharing Phase

Distributing column polynomials: Upon receiving his input degree-t polynomials $q_1(x), \ldots, q_N(x)$ from the environment, D executes the following code:

For each $k \in [m]$ and $\ell \in [L']$:

1. Compute $\mathsf{idx} = ((k-1) \cdot L' + \ell - 1) \cdot (t+1) + 1$.
2. If $k \in [m']$, select a random degree-$(t, 2t)$ bivariate polynomial $F_\ell^{(k)}(x, y)$ s.t. for each $i \in [0, t]$,

$F_\ell^{(k)}(x, \alpha_{-i}) = q_{\mathsf{idx}+i}(x)$. Otherwise, select a random degree-$(t, 2t)$ bivariate polynomial $F_\ell^{(k)}(x, y)$.

3. Send the column polynomial $g_{\ell,i}^{(k)}(y) = F_\ell^{(k)}(\alpha_i, y)$ to each $P_i \in \mathcal{P}$.

Signing the column polynomials: Each $P_i \in \mathcal{P}$ executes the following code:

1. Wait to receive $\{g_{\ell,i}^{(k)}(y)\}_{\ell \in [L'], k \in [m]}$ from D.
2. If $\{g_{\ell,i}^{(k)}(y)\}_{\ell \in [L'], k \in [m]}$ are all of degree $2t$, broadcast OK_i and set $\boldsymbol{g}_{*,i}^{(k)} = (g_{1,i}^{(k)}, \ldots, g_{L',i}^{(k)})$ for each $k \in [m]$. Here we abuse the notation to also use $g_{\ell,i}^{(k)}$ to represent the evaluation vector $(g_{\ell,i}^{(k)}(\alpha_1), \ldots, g_{\ell,i}^{(k)}(\alpha_n))$. Then $\boldsymbol{g}_{*,i}^{(k)}$ is a vector of size $n \cdot L' = L$.
3. Send $(\mathsf{Init}, \mathsf{APICP}, T, (\boldsymbol{g}_{*,i}^{(1)}, \ldots, \boldsymbol{g}_{*,i}^{(m)}))$ to $\mathcal{F}_{\mathsf{APICP}}(P_i, D)$.

Identifying column polynomials: D executes the following code:

1. Initialize a set $\mathcal{M}$ to $\emptyset$.
2. If $|\mathcal{M}| < 2t + 1$, include P_i into $\mathcal{M}$ when:
   1) OK_i is received from P_i.
   2) $(P_i, \mathsf{APICP}, (\boldsymbol{g}_{*,i}^{(1)}, \ldots, \boldsymbol{g}_{*,i}^{(m)}))$ is received from $\mathcal{F}_{\mathsf{APICP}}(P_i, D)$ and $\boldsymbol{g}_{*,i}^{(k)}(y) = \boldsymbol{F}_*^{(k)}(\alpha_i, y)$ for each $k \in [m]$. Here $\boldsymbol{F}_*^{(k)}(\alpha_i, y) = (F_1^{(k)}(\alpha_i, y), \ldots, F_{L'}^{(k)}(\alpha_i, y))$.
3. Broadcast $\mathcal{M}$ when $|\mathcal{M}| = 2t + 1$.

Verifying the $\mathcal{M}$ set: Each party moves to the next phase if the following conditions are met:

(1). $\mathcal{M}$ is received from D and $|\mathcal{M}| = 2t + 1$.

(2). OK_h is received from all $P_h \in \mathcal{M}$.

Figure 9: The protocol of the Π_{Sh}

The verification phase Π_{Ver} is present in Fig. 10. In this phase, each party P_i does a verification on his column polynomials. When P_i receives his column polynomials from D, he verifies a random linear combination of the bivariate polynomials chosen by D to verify whether his column polynomials are consistent with them. This is realized by the revelation phases of $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ for all the parties $P_h \in \mathcal{M}$. Regardless of the communication cost of APICP, the communication complexity of Π_{Ver} is $\mathcal{O}(n^3\kappa)$ bits.

Protocol Π_{Ver}

Verification Phase

For each $P_i \in \mathcal{P}$:

1. Upon receiving $\{g_{\ell,i}^{(k)}(y)\}_{\ell \in [L'], k \in [m]}$ from D and these polynomials are all of degree $2t$, P_i broadcasts a random value $r_i \in \mathbb{F}$ and computes $g_{\ell,i}(y) = \sum_{k=1}^{m} g_{\ell,i}^{(k)}(y) \cdot r_i^k$ for each $\ell \in [L']$.
2. Upon receiving r_i from P_i, each party sends $(\mathsf{Request}, \mathsf{APICP}, P_i, (r_i, r_i^2, \ldots, r_i^m))$ to $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ for all $P_h \in \mathcal{M}$.
3. Upon receiving $\boldsymbol{g}_{*,h}$ from $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ for all $P_h \in \mathcal{M}$, P_i accepts $\{g_{\ell,i}^{(k)}(y)\}_{\ell \in [L'], k \in [m]}$ if the following hold for each $\ell \in [L']$.
   1) For each $P_h \in \mathcal{M}$, parse $\boldsymbol{g}_{*,h}$ into $\{g_{\ell,h}\}_{\ell \in [L']}$, each $g_{\ell,h}$ is a degree-$2t$ polynomial.
   2) There exists a degree-$(t, 2t)$ bivariate polynomial $F_\ell(x, y)$ s.t. $F_\ell(\alpha_h, y) = g_{\ell,h}(y)$ for all $P_h \in \mathcal{M}$ and $F_\ell(\alpha_i, y) = g_{\ell,i}(y)$.

Figure 10: The protocol of the Π_{Ver}

The authentication phase Π_{Auth} is present in Fig. 11. In this phase, each P_i who has accepted his column polynomials prepares the authentication tags on them. The authentication tags are prepared for each pair of (P_i, P_v). All the parties invoke the functionality of sub-protocols to prepare random shares of authentication keys and random masks. Then, each P_i follows the process we have discussed in Section 2.3 to prepare the tags. When P_i gets the authentication tags for all $P_v \in \mathcal{P}$, he broadcasts Tag_i. Each P_i who has broadcast Tag_i will be included in a set $\mathcal{W}$ created by D. Then, $\mathcal{W}$ will be publicly verified by all the parties. If the public verification does not pass, all the honest parties won't get/accept their shares. If the public verification passes, the ACSS protocol guarantees that all the honest parties will obtain their shares. Regardless of the communication cost of APICP, the communication complexity of Π_{Auth} is $\mathcal{O}(Ln^5\kappa + mn^6\kappa + n^7\kappa^2 + n^8)$ bits (including the communication to realize the functionalities of sub-protocols except for $\mathcal{F}_{\mathsf{APICP}}$).

Protocol Π_{Auth}

Authentication Phase

For each $P_i \in \mathcal{P}$ and $P_v \in \mathcal{P}$, do the following:

- **Preparing random shares:** For public value $(\beta_0, \ldots, \beta_n) = (\alpha_i, \alpha_1, \ldots, \alpha_{i-1}, \alpha_0, \alpha_{i+1}, \ldots, \alpha_n)$, $P_i \in \mathcal{P}$ executes the following code:
  (1). Send $(\mathsf{Request}, \mathsf{RandShare}^0, L, P_i)$ to $\mathcal{F}^0_{\mathsf{RandShare}}$ to prepare $[\boldsymbol{\mu}_{i\to v}]^i_t$ where $\boldsymbol{\mu}_{i\to v}$ is a vector in $\mathbb{F}^L$.
  (2). Send $(\mathsf{Request}, \mathsf{RandShare}, m)$ to $\mathcal{F}_{\mathsf{RandShare}}$ to prepare $[\nu^{(1)}_{i\to v}]^i_t, \ldots, [\nu^{(m)}_{i\to v}]^i_t$.
  (3). Send $(\mathsf{Request}, \mathsf{RandShare}, m \cdot t)$ to $\mathcal{F}_{\mathsf{RandShare}}$ to prepare $[r^{(k)}_u]^i_t$ for each $k \in [m]$ and $u \in [t]$.
  (4). Send $(\mathsf{Request}, \mathsf{RandShare}^0, n, P_j)$ to $\mathcal{F}^0_{\mathsf{RandShare}}$ to prepare $[\mathsf{mask}_j]^i_t$ for each $j \in [n]$.

For each $P_i \in \mathcal{P}$, do the following:

- **Preparing shares of tags $\{\tau^{(k)}_{i\to v}\}_{k\in[m],P_v\in\mathcal{P}}$ for P_i:** For each P_j who has accepted $\{g^{(k)}_{\ell,j}(y)\}_{k\in[m],\ell\in[L']}$:
  (1). For each $k \in [m]$ and $P_v \in \mathcal{P}$, P_j computes his share of $[\tau^{(k)}_{i\to v}]^i_{2t}$ (denoted by $\tau^{(k)}_{i\to v,j}$) by:

$$[\tau^{(k)}_{i\to v}]^i_{2t} = [\boldsymbol{g}^{(k)}_{*,i}]^i_t \cdot [\boldsymbol{\mu}_{i\to v}]^i_t + [\nu^{(k)}_{i\to v}]^i_t + \sum_{u=1}^{t} [\![\boldsymbol{e}_u]\!]^i_t \cdot [r^{(k)}_u]^i_t$$

  Here $[\![\boldsymbol{e}_u]\!]^i_t$ is the degenerate packed secret sharing (defined in Section 3.3) of $\boldsymbol{e}_u = (e^{(i)}_u, e^{(1)}_u, \ldots, e^{(t)}_u)$ with $e^{(u)}_u = 1$, $e^{(i)}_u = 0$ and $e^{(k)}_u = 0$ for each $k \in [t]\backslash\{u\}$. Here each $P_j \in \mathcal{P}$ except P_i has his share of $[g^{(k)}_{\ell,i}]^i_t$ equals to $g^{(k)}_{\ell,j}$ for each $\ell \in [L']$. Thus, each P_j who accepts his column polynomials gets his $\boldsymbol{g}^{(k)}_{*,j}$ in the verification phase, which is also his share of $[\boldsymbol{g}^{(k)}_{*,i}]^i_t$. Especially, P_i doesn't have his share of $[\boldsymbol{g}_{*,i}]^i_t$, but he can still compute his share of $[\tau^{(k)}_{i\to v}]^i_{2t}$ because his share of $[\boldsymbol{\mu}_{i\to v}]^i_t$ is equal to 0.

  (2). P_j sends $\{\tau^{(k)}_{i\to v,j}\}_{k\in[m],P_v\in\mathcal{P}}$ to P_i.
  (3). Upon receiving $\{\tau^{(k)}_{i\to v,j}\}_{k\in[m],P_v\in\mathcal{P}}$ from P_j, P_i broadcasts a random element $r_{i,j} \in \mathbb{F}$ and computes $\tau_{i\to v,j} = \sum_{k=1}^m r^k_{i,j} \cdot \tau^{(k)}_{i\to v,j}$ for each $P_v \in \mathcal{P}$.
  – **Verifying P_j's shares of tags:** Upon receiving $r_{i,j}$ from P_i, for each $P_\alpha, P_v \in \mathcal{P}$:
    1). For each $P_h \in \mathcal{M}$, all parties send $(\mathsf{Request}, \mathsf{APICP}, P_\alpha, (r_{i,j}, r^2_{i,j} \ldots, r^m_{i,j}))$ to $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$.
    2). Upon receiving $\boldsymbol{g}_{*,h}$ from $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ for each $P_h \in \mathcal{M}$, P_α accepts $\{\boldsymbol{g}_{*,h}\}_{P_h\in\mathcal{M}}$ if the following hold for each $\ell \in [L']$:
      (a) For each $P_h \in \mathcal{M}$, parse $\boldsymbol{g}_{*,h}$ into $\{g_{\ell,h}\}_{\ell\in[L']}$, each $g_{\ell,h}$ is a degree-$2t$ polynomial.
      (b) There exists a degree-$(t, 2t)$ bivariate polynomial $F_\ell(x, y)$ s.t. $F_\ell(\alpha_h, y) = g_{\ell,h}(y)$ for all $P_h \in \mathcal{M}$.
    3). Upon accepting $\{\boldsymbol{g}_{*,h}\}_{P_h\in\mathcal{M}}$, P_α computes $g_{\ell,j} = (F_\ell(\alpha_j, \alpha_1), \ldots, F_\ell(\alpha_j, \alpha_n))$ for each $\ell \in [L']$ and $\boldsymbol{g}_{*,j} = (g_{1,j}, \ldots, g_{L',j})$.
    4). All the parties jointly prepare a sharing $[\gamma_{i\to v,j}]^i_t$. P_α computes his share by:

$$[\gamma_{i\to v,j}]^i_t = \boldsymbol{g}_{*,j} \cdot [\boldsymbol{\mu}_{i\to v}]^i_t + \sum_{k=1}^{m} r^k_{i,j} \cdot \left([\nu^{(k)}_{i\to v}]^i_t + \sum_{u=1}^{t} e_{u,j} \cdot [r^{(k)}_u]^i_t\right) + [\mathsf{mask}_j]^i_t$$

    Here $e_{u,j}$ is P_j's share of $[\![\boldsymbol{e}_u]\!]^i_t$ which is public. Notice that P_j's share of $[\gamma_{i\to v,j}]^i_t$ should be $\tau_{i\to v,j}$.

    5). P_α sends his share of $[\gamma_{i\to v,j}]^i_t$ and $(\mathsf{Request}, \mathsf{privRec}, P_i, 1)$ to $\mathcal{F}_{\mathsf{privRec}}$.
  (4). Upon receiving the whole $[\gamma_{i\to v,j}]^i_t$ from $\mathcal{F}_{\mathsf{privRec}}$, P_i accepts $\{\tau^{(k)}_{i\to v,j}\}_{k\in[m],P_v\in\mathcal{P}}$ if P_j's share of $[\gamma_{i\to v,j}]^i_t$ is equal to $\tau_{i\to v,j}$ for all $P_v \in \mathcal{P}$.
- **Reconstructing P_i's tags:**
  (1). Upon accepting $2t+1$ different P_j's $\{\tau^{(k)}_{i\to v,j}\}_{k\in[m],P_v\in\mathcal{P}}$, P_i reconstructs $\{\tau^{(k)}_{i\to v}\}_{k\in[m],P_v\in\mathcal{P}}$ with these shares.

Preparing the $\mathcal{W}$ set: D executes the following code:

(1). Each $P_i \in \mathcal{P}$ broadcasts Tag_i after getting $\{\tau^{(k)}_{i\to v}\}_{k\in[m],P_v\in\mathcal{P}}$ and accepting $\{g^{(k)}_{\ell,i}(y)\}_{k\in[m],\ell\in[L']}$ during the verification phase.

(2). D initializes a set $\mathcal{W}$ to $\emptyset$.
(3). Upon receiving Tag_i from P_i, D includes P_i into $\mathcal{W}$.
(4). D broadcasts $\mathcal{W}$ when $|\mathcal{W}| = 2t+1$.

Verifying the $\mathcal{W}$ set: Each party executes the following code:
(1). $\mathcal{W}$ is received from D and $|\mathcal{W}| \geq 2t+1$.
(2). Tag_i is received from all $P_i \in \mathcal{W}$.
(3). Each party moves to the next phase if the above conditions are met.

Figure 11: The protocol of the Π_{Auth}

The completion phase Π_{Comp} is present in Fig. 12. This protocol describes how each honest party eventually obtains his output shares. Each party first reconstructs his degree-t row polynomials and then reconstructs his degree-$2t$ column polynomials to obtain his shares. The communication complexity of Π_{Comp} is $\mathcal{O}(mLn\kappa + mn^3\kappa + L'n^3\kappa)$ bits (including the communication to realize the functionalities of sub-protocols).

Protocol Π_{Comp}

Completion Phase

Reconstructing row polynomials:
For each $P_v \in \mathcal{P}$, do the following:
1. Each $P_i \in \mathcal{W}$ sends $\{g^{(k)}_{\ell,i}(\alpha_v)\}_{k\in[m],\ell\in[L']}$ to P_v.
2. For each $P_i \in \mathcal{W}$, do the following:
   (1). All parties send their shares of $[\boldsymbol{\mu}_{i\to v}]^i_t, [\nu^{(1)}_{i\to v}]^i_t, \ldots, [\nu^{(m)}_{i\to v}]^i_t$ and $(\mathsf{Request}, \mathsf{privRec}, P_v)$ to $\mathcal{F}_{\mathsf{privRec}}$. Upon receiving $[\boldsymbol{\mu}_{i\to v}]^i_t, \{[\nu^{(k)}_{i\to v}]^i_t\}_{k\in[m]}$ from $\mathcal{F}_{\mathsf{privRec}}$, P_v reconstructs the secrets $\boldsymbol{\mu}_{i\to v}, \{\nu^{(k)}_{i\to v}\}_{k\in[m]}$.
   (2). Upon receiving $\{g^{(k)}_{\ell,i}(\alpha_v)\}_{k\in[m],\ell\in[L']}$ from P_i, P_v sends a random element $r_{i\to v} \in \mathbb{F}$ to P_i.
   (3). Upon receiving $r_{i\to v}$, P_i sends $\tau_{i\to v} = \sum_{k=1}^m r^k_{i\to v} \cdot \tau^{(k)}_{i\to v}$ and $g_{\ell,i}(y) = \sum_{k=1}^m r^k_{i\to v} g^{(k)}_{\ell,i}(y)$ for each $\ell \in [L']$ to P_v.
   (4). Upon receiving $\tau_{i\to v}$ and $\{g_{\ell,i}(y)\}_{\ell\in[L']}$ from P_i, P_v computes $g_{\ell,i} = (g_{\ell,i}(\alpha_1), \ldots, g_{\ell,i}(\alpha_n))$ for each $\ell \in [L']$ and $\boldsymbol{g}_{*,i} = (g_{1,i}, \ldots, g_{L',i})$.
   (5). P_v accepts $\{g^{(k)}_{\ell,i}(\alpha_v)\}_{k\in[m],\ell\in[L']}$ if the following hold:
      1) For each $\ell \in [L']$, the degree of $g_{\ell,i}(y)$ is $2t$ and $g_{\ell,i}(\alpha_v) = \sum_{k=1}^m r^k_{i\to v} \cdot g^{(k)}_{\ell,i}(\alpha_v)$.
      2) $\tau_{i\to v} = \boldsymbol{g}_{*,i} \cdot \boldsymbol{\mu}_{i\to v} + \sum_{k=1}^m r^k_{i\to v} \cdot \nu^{(k)}_{i\to v}$.
3. Upon accepting $t+1$ different P_i's $\{g^{(k)}_{\ell,i}(\alpha_v)\}_{k\in[m],\ell\in[L']}$, P_v reconstructs a degree-t polynomial $f^{(k)}_{\ell,v}(x)$ s.t. $f^{(k)}_{\ell,v}(\alpha_i) = g^{(k)}_{\ell,i}(\alpha_v)$ for each $\ell \in [L']$ and $k \in [m]$.

Reconstructing column polynomials:
For each $P_w \in \mathcal{P}$, do the following:
1. Each $P_v \in \mathcal{P}$ sends $\{f^{(k)}_{\ell,v}(\alpha_w)\}_{k\in[m],\ell\in[L']}$ to P_w.
2. For each $P_v \in \mathcal{P}$:
   (1). Upon receiving $\{f^{(k)}_{\ell,v}(\alpha_w)\}_{k\in[m],\ell\in[L']}$ from P_v, P_w broadcasts a random value $r_{v\to w} \in \mathbb{F}$.
   (2). Upon receiving $r_{v\to w}$, each $P_i \in \mathcal{W}$ sends $\tau_{i\to w} = \sum_{k=1}^m r^k_{v\to w} \cdot \tau^{(k)}_{i\to w}$ and $g_{\ell,i}(y) = \sum_{k=1}^m r^k_{v\to w} g^{(k)}_{\ell,i}(y)$ for each $\ell \in [L']$ to P_w.
   (3). Upon receiving $\{g_{\ell,i}(y)\}_{\ell\in[L']}$, P_w computes $g_{\ell,i} = (g_{\ell,i}(\alpha_1), \ldots, g_{\ell,i}(\alpha_n))$ for each $\ell \in [L']$ and $\boldsymbol{g}_{*,i} = (g_{1,i}, \ldots, g_{L',i})$.
   (4). P_w accepts $\{g_{\ell,i}(y)\}_{\ell\in[L']}$ if the following hold.
      1) $g_{1,i}(y), \ldots, g_{L',i}(y)$ are all of degree $2t$.
      2) $\tau_{i\to w} = \boldsymbol{g}_{*,i} \cdot \boldsymbol{\mu}_{i\to w} + \sum_{k=1}^m r^k_{v\to w} \nu^{(k)}_{i\to w}$.
   (5). Upon accepting $t+1$ different P_i's $\{g_{\ell,i}(y)\}_{\ell\in[L']}$, P_w reconstructs a degree-$(t,2t)$ bivariate polynomial $F_\ell(x,y)$ s.t. $F_\ell(\alpha_i, y) = g_{\ell,i}(y)$ for each $\ell \in [L']$.
   (6). P_w accepts $\{f^{(k)}_{\ell,v}(\alpha_w)\}_{k\in[m],\ell\in[L']}$ if $\sum_{k=1}^m r^k_{v\to w} f^{(k)}_{\ell,v}(\alpha_w) = F_\ell(\alpha_w, \alpha_v)$ for each $\ell \in [L']$.
3. Upon accepting $2t+1$ different $\{f^{(k)}_{\ell,v}(\alpha_w)\}_{k\in[m],\ell\in[L']}$, P_w reconstructs a degree-$2t$ polynomial $g^{(k)}_{\ell,w}(y)$ s.t. $g^{(k)}_{\ell,w}(\alpha_v) = f^{(k)}_{\ell,v}(\alpha_w)$ for each $\ell \in [L']$ and $k \in [m]$.
4. P_w outputs $\{g^{(k)}_{\ell,w}(\alpha_i)\}_{\ell\in[L'],k\in[m'],i\in[-t,0]}$.

Figure 12: The protocol of the Π_{Comp}

Lemma 2. *The protocol* Π_{ACSS} *t-securely realizes* $\mathcal{F}_{\mathsf{ACSS}}$ *in the* $(\mathcal{F}_{\mathsf{APICP}}, \mathcal{F}_{\mathsf{privRec}}, \mathcal{F}_{\mathsf{RandShare}}, \mathcal{F}^0_{\mathsf{RandShare}})$*-hybrid model with statistical security.*

We now use the instances of $\mathcal{F}_{\mathsf{APICP}}$ (Π_{APICP} in section 4), $\mathcal{F}_{\mathsf{privRec}}$ (Π_{privRec} in Appendix B.1), $\mathcal{F}_{\mathsf{RandShare}}$ ($\Pi_{\mathsf{RandShare}}$ in Appendix B.2), and $\mathcal{F}^0_{\mathsf{RandShare}}$ ($\Pi^0_{\mathsf{RandShare}}$ in Appendix B.3) instead of the functionalities to realize $\mathcal{F}_{\mathsf{ACSS}}$. Taking $m = n^4$, we obtain our main Theorem 1.

Theorem 1. *Let* κ *denote the security parameter. For a finite field* $\mathbb{F}$ *of size* $2^{\Theta(\kappa)}$*, there exists a fully malicious information-theoretic ACSS protocol against* $t < n/3$ *corrupted parties that shares* N *degree-t Shamir sharings over* $\mathbb{F}$ *with communication of* $\mathcal{O}(Nn\kappa + n^{12}\kappa^2)$ *bits and statistical error* $\mathcal{O}((Nn+n^{15}\kappa^2)/2^{\kappa})$*. The round complexity is* $\mathcal{O}(1)$ *rounds via P2P channels plus* $\mathcal{O}(1)$ *rounds of invocations to ACast.*

The proof of Lemma 2 is given in Appendix D.2. Detailed analysis of the complexity of our Π_{ACSS} is given in Appendix D.3.

Construction of AMPC. The previous work [CP17] presents a framework using $\mathcal{F}_{\mathsf{ACSS}}$ to construct an AMPC protocol Π_{AMPC}. We give the ideal functionality $\mathcal{F}_{\mathsf{AMPC}}$ in Appendix F.1 and an overview of how to compile $\mathcal{F}_{\mathsf{ACSS}}$ to Π_{AMPC} in Appendix F.2.

Theorem 2. ([CP17]) *Let* $n = 3t + 1$*. For any circuit* C *of size* $|C|$ *and depth* D*, there is a fully malicious asynchronous MPC protocol computing the circuit that is secure against at most* t *corrupted parties with guaranteed output delivery in the* $\mathcal{F}_{\mathsf{ACSS}}$*-hybrid model. The achieved communication complexity is* $\mathcal{O}(|C| \cdot n^2\kappa + n^5\kappa)$ *bits plus* $\mathcal{O}(n)$ *invocations of* $\mathcal{F}_{\mathsf{ACSS}}$ *to share* $\mathcal{O}(|C| \cdot n)$ *degree-t Shamir sharings in total.*

Replacing $\mathcal{F}_{\mathsf{ACSS}}$ with our construction of Π_{ACSS}, we get Corollary 1.

Corollary 1. *Let* $n = 3t + 1$*. For any circuit* C *of size* $|C|$ *and depth* D*, there is a fully malicious information-theoretic asynchronous MPC protocol that is secure against at most* t *corrupted parties with guaranteed output delivery. The total communication complexity is* $O(|C| \cdot n^2\kappa + n^{13}\kappa^2)$ *bits.*

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.

References

[AAPP22] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Asymptotically free broadcast in constant expected time via packed VSS. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part I*, volume 13747 of *Lecture Notes in Computer Science*, pages 384–414. Springer, 2022.

[AAPS23] Ittai Abraham, Gilad Asharov, Arpita Patra, and Gilad Stern. Asynchronous agreement on a core set in constant expected time and more efficient asynchronous vss and mpc. Cryptology ePrint Archive, Paper 2023/1130, 2023.

[AAY22] Ittai Abraham, Gilad Asharov, and Avishay Yanai. Efficient perfectly secure computation with optimal resilience. *J. Cryptol.*, 35(4):27, 2022.

[ADS20] Ittai Abraham, Danny Dolev, and Gilad Stern. Revisiting asynchronous fault tolerant computation with optimal resilience. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20, page 139–148, New York, NY, USA, 2020. Association for Computing Machinery.

[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 Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 39–70. Springer, 2023.

[AKP23] Benny Applebaum, Eliran Kachlon, and Arpita Patra. The round complexity of statistical MPC with optimal resiliency. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1527–1536. ACM, 2023.

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 52–61. ACM, 1993.

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

[BFO12] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 663–680. Springer, 2012.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

[BH07] Zuzana Beerliová-Trubíniová and Martin Hirt. Simple and efficient perfectly-secure asynchronous MPC. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 376–392. Springer, 2007.

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

[BKR94] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In James H. Anderson, David Peleg, and Elizabeth Borowsky, editors, *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17, 1994*, pages 183–192. ACM, 1994.

[BOCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, page 52–61, New York, NY, USA, 1993. Association for Computing Machinery.

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

[Bra84] Gabriel Bracha. An asynchronou [(n-1)/3]-resilient consensus protocol. In Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro, editors, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 154–162. ACM, 1984.

[Can96] Ran Canetti. Studies in secure multiparty computation and applications. *Scientific Council of The Weizmann Institute of Science*, 1996.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptol.*, 13(1):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.

[Can20] Ran Canetti. Universally composable security. *J. ACM*, 67(5):28:1–28:94, 2020.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 11–19. ACM, 1988.

[CFG$^+$23] Ran Cohen, Pouyan Forghani, Juan A. Garay, Rutvik Patel, and Vassilis Zikas. Concurrent asynchronous byzantine agreement in expected-constant rounds, revisited. In Guy N. Rothblum and Hoeteck Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part IV*, volume 14372 of *Lecture Notes in Computer Science*, pages 422–451. Springer, 2023.

[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, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 998–1021, 2016.

[CHP13] Ashish Choudhury, Martin Hirt, and Arpita Patra. Asynchronous multiparty computation with linear communication complexity. In Yehuda Afek, editor, *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, volume 8205 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2013.

[Coh16] Ran Cohen. Asynchronous secure multiparty computation in constant time. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II*, volume 9615 of *Lecture Notes in Computer Science*, pages 183–207. Springer, 2016.

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

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

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 42–51. ACM, 1993.

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

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

[GLS24] Vipul Goyal, Chen-Da Liu-Zhang, and Yifan Song. Towards achieving asynchronous MPC with linear communication and optimal resilience. *IACR Cryptol. ePrint Arch.*, page 243, 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 V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 477–498. Springer, 2013.

[PCR09] Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. Efficient statistical asynchronous verifiable secret sharing with optimal resilience. In Kaoru Kurosawa, editor, *Information Theoretic Security, 4th International Conference, ICITS 2009, Shizuoka, Japan, December 3-6, 2009. Revised Selected Papers*, volume 5973 of *Lecture Notes in Computer Science*, pages 74–92. Springer, 2009.

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

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 73–85. ACM, 1989.

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

[SR00] K. Srinathan and C. Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In Bimal K. Roy and Eiji Okamoto, editors, *Progress in Cryptology - INDOCRYPT 2000, First International Conference in Cryptology in India, Calcutta, India, December 10-13, 2000, Proceedings*, volume 1977 of *Lecture Notes in Computer Science*, pages 117–129. Springer, 2000.

[SS23] Victor Shoup and Nigel P. Smart. Lightweight asynchronous verifiable secret sharing with optimal resilience. *IACR Cryptol. ePrint Arch.*, page 536, 2023.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 80–91. IEEE Computer Society, 1982.

A Additional Definitions of the Agreement Primitives

The *agree on a common set* (ACS) primitive allows the parties to agree on a set of at least $n-t$ parties that satisfies a certain property (a so-called ACS property). We give the formal definitions of agreement primitives here.

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 $(P_i, P_j) \in \mathcal{P}$ we have that P_i will eventually like P_j.*

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.*

Ben-Or, Kelmer, and Rabin provide an efficient ACS protocol in [BKR94]. We state their result as follows.

Lemma 3. ([BKR94]) *Given an ACS property Q, there exists a t-resilient ACS protocol Π^Q_{ACS} for Q with communication complexity $O(n^3)$ bits, for $t < n/3$ active corruptions.*

In [Bra84], an A-Cast protocol is provided, which enables a party to efficiently broadcast a message in an asynchronous network. If a message is broadcast, each party will eventually receive this message, but the arrival time is still controlled by the adversary. We state the formal functionality of A-Cast [Bra84] in Fig. 13. From [Bra84], broadcasting an ℓ-bit message requires $\mathcal{O}(n^2\ell)$-bit communication.

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

Upon receiving $(\mathsf{sender}, \mathsf{ACast}, m)$ from $P_S \in \mathcal{P}$, the trusted party sends an request-based delayed output (P_S, ACast, m) to each $P_i \in \mathcal{P}$.

Figure 13: Ideal functionality for broadcasting a message

B Constructions and Security Proofs for Sub-protocols

B.1 Construction of Π_{privRec}

We give our construction of Π_{privRec} in Fig. 14.

Protocol Π_{privRec}

1. Upon receiving the input shares of $[s_1]_t, \ldots, [s_N]_t$ from the environment, each party sends them to R.
2. For $r = 0, \ldots, t$, R executes the following code in iterating r:
   (a). Let $\mathcal{W}$ denote the set of parties in $\mathcal{P}$ from whom R has received the shares. Wait until $|\mathcal{W}| = 2t+1+r$.
   (b). Check whether there exists degree-t sharing polynomials $p_1(\cdot), \ldots, p_N(\cdot)$, such that for $2t+1$ parties in $\mathcal{W}$, their shares of $[s_1]_t, \ldots, [s_N]_t$ are on these polynomials respectively. If so, output the whole sharing $[s_1]_t, \ldots, [s_N]_t$. Otherwise, proceed to the next iteration.

Figure 14: The protocol for private reconstruction

Lemma 4. *Protocol* Π_{privRec} *t-securely realizes* $\mathcal{F}_{\mathsf{privRec}}$.

Proof. We prove this lemma by constructing a simulator $\mathcal{S}$. $\mathcal{S}$ needs to interact with the environment $\mathcal{Z}$ and with the ideal functionalities. $\mathcal{S}$ constructs virtual real-world honest parties and runs the real-world adversary $\mathcal{A}$. For simplicity, we just let $\mathcal{S}$ communicate with $\mathcal{A}$ on behalf of honest parties and the ideal functionality of sub-protocols in our proof. In order to simulate the communication with $\mathcal{Z}$, every message that $\mathcal{S}$ receives from $\mathcal{Z}$ is sent to $\mathcal{A}$, and likewise, every message sent from $\mathcal{A}$ sends to $\mathcal{Z}$ is forwarded by $\mathcal{S}$. Each time an honest party needs to send a message to another honest party, $\mathcal{S}$ will tell $\mathcal{A}$ that a message has been delivered such that $\mathcal{A}$ can tell $\mathcal{S}$ the arrival time of this message to help $\mathcal{S}$ instruct the functionalities to delay the outputs in the ideal world. For each request-based delayed output that needs to be sent to an honest party, we let $\mathcal{S}$ delay the output in default until we say $\mathcal{S}$ allows the functionality to send the output. We will show that the output in the ideal world is identically distributed to that in the real world by using hybrid arguments.

Construction of the ideal adversary $\mathcal{S}$.

If R is corrupted:

Simulator $\mathcal{S}$

- $\mathcal{S}$ receives the whole sharing $[s_1]_t, \dots, [s_N]_t$ from $\mathcal{F}_{\mathsf{privRec}}$. For each honest P_i, $\mathcal{S}$ sends the P_i's shares of $[s_1]_t, \dots, [s_N]_t$ to R on behalf of P_i.

Figure 15: Simulator for the $\mathcal{F}_{\mathsf{privRec}}$

Hybrid arguments:

$\mathbf{Hyb}_0$: In this hybrid, $\mathcal{S}$ learns honest parties' inputs, and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, for each honest party, $\mathcal{S}$ doesn't learn his shares of $s_1, \dots, s_N$ from him. Instead, he learns the shares from the output of $\mathcal{F}_{\mathsf{privRec}}$. Since $[s_1]_t, \dots, [s_N]_t$ are complete t-sharing, the honest parties' shares are contained in the whole sharing, so this doesn't change the output distribution. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

Note that $\mathbf{Hyb}_1$ is the ideal-world scenario, Π_{privRec} securely computes $\mathcal{F}_{\mathsf{pricRec}}$.

If R is honest:

Simulator $\mathcal{S}$

1. For each corrupted P_i, $\mathcal{S}$ receives his shares of $[s_1]_t, \dots, [s_N]_t$ from $\mathcal{F}_{\mathsf{privRec}}$.
2. For each corrupted P_i, $\mathcal{S}$ receives their messages $[s_1]'_t, \dots, [s_N]'_t$ sent to R. When the message arrives, $\mathcal{S}$ accept P_i's shares if his share of s_j equals to $[s_j]'_t$ for all $j \in [N]$. For each honest P_i, $\mathcal{S}$ regards that he accepts P_i's shares when the shares arrive.
3. After accepting $2t+1$ parties' shares, $\mathcal{S}$ allows $\mathcal{F}_{\mathsf{privRec}}$ to send the output to R.

Figure 16: Simulator for the $\mathcal{F}_{\mathsf{privRec}}$

Hybrid arguments:

$\mathbf{Hyb}_0$: In this hybrid, $\mathcal{S}$ learns honest parties' inputs, and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, R gets his output from $\mathcal{F}_{\mathsf{privRec}}$ instead of computing the output by himself. By the error-correction property of Reed-Solomon Codes, R will eventually get the correct secrets $s_1, \dots, s_N$. Note that R will also receive at least $t+1$ honest parties' shares, which fixes the whole sharing, so R can compute the whole sharing by himself. This shows that R gets the output after receiving $2t+1$ parties' correct shares. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

Note that $\mathbf{Hyb}_1$ is the ideal-world scenario, Π_{privRec} securely computes $\mathcal{F}_{\mathsf{pricRec}}$.

The protocol Π_{privRec} requires $\mathcal{O}(Nn\kappa)$-bit communication to send n parties' shares of the N complete t-sharing to R. $\square$

B.2 Construction of $\Pi_{\mathsf{RandShare}}$

We give our construction of $\Pi_{\mathsf{RandShare}}$ in the $(\mathcal{F}_{\mathsf{ACSS}}, \mathcal{F}_{\mathsf{privRec}})$-hybrid model in Fig. 17.

Protocol $\Pi_{\mathsf{RandShare}}$

On public parameters $N, (\beta_0, \ldots, \beta_n)$:

1. Each party $P_j \in \mathcal{P}$ samples $L = N/(t+1)$ random value $(s_j^{(1)}, \ldots, s_j^{(L)}) \in \mathbb{F}^L$ and chooses random degree-t polynomials $q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)$ such that for each $\ell \in [L]$, $q_j^{(\ell)}(\beta_0) = s_j^{(\ell)}$.
2. Each party P_j sends $(\mathsf{Dealer}, \mathsf{ACSS}, \{q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)\})$ to $\mathcal{F}_{\mathsf{ACSS}}$.
3. Let the ACS property Q defined by P_j likes P_k if P_j gets his shares from $\mathcal{F}_{\mathsf{ACSS}}$ whose dealer is P_k. Then all parties run Π_{ACS}^Q to get a set S of size $2t+1$. Let the sharing generated by $\mathcal{F}_{\mathsf{ACSS}}$ with dealers in S be $\{[s_{a_j}^{(\ell)}]_t\}_{j \in [2t+1], \ell \in [L]}$, where $S = \{P_{a_1}, \ldots, P_{a_{2t+1}}\}$.
4. Let M be the $(t+1) \times (2t+1)$ Vandermonde matrix

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & b_1 & \cdots & b_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & b_1^t & \cdots & b_{2t}^t \end{pmatrix},$$

   where $1, b_1, \ldots, b_{2t}$ are $2t+1$ distinct public elements in $\mathbb{F}$.
   Each party locally computes his shares by

$$\begin{pmatrix} [r_1^{(1)}]_t, \ldots, [r_1^{(L)}]_t \\ \vdots \\ [r_{t+1}^{(1)}]_t, \ldots, [r_{t+1}^{(L)}]_t \end{pmatrix} = M \cdot \begin{pmatrix} [s_{a_1}^{(1)}]_t, \ldots, [s_{a_1}^{(L)}]_t \\ \vdots \\ [s_{a_{2t+1}}^{(1)}]_t, \ldots, [s_{a_{2t+1}}^{(L)}]_t \end{pmatrix}$$

5. Each party outputs his shares of $\{[r_j^{(\ell)}]_t\}_{j \in [t+1], \ell \in [L]}$.

Figure 17: The protocol to prepare random t-sharing

Lemma 5. *Protocol $\Pi_{\mathsf{RandShare}}$ t-securely realizes $\mathcal{F}_{\mathsf{RandShare}}$ in the $(\mathcal{F}_{\mathsf{ACSS}}, \mathcal{F}_{\mathsf{privRec}})$-hybrid model.*

Proof. We prove this lemma by constructing a simulator $\mathcal{S}$. $\mathcal{S}$ needs to interact with the environment $\mathcal{Z}$ and with the ideal functionalities. $\mathcal{S}$ constructs virtual real-world honest parties and runs the real-world adversary $\mathcal{A}$. For simplicity, we just let $\mathcal{S}$ communicate with $\mathcal{A}$ on behalf of honest parties and the ideal functionality of sub-protocols in our proof. In order to simulate the communication with $\mathcal{Z}$, every message that $\mathcal{S}$ receives from $\mathcal{Z}$ is sent to $\mathcal{A}$, and likewise, every message sent from $\mathcal{A}$ sends to $\mathcal{Z}$ is forwarded by $\mathcal{S}$. Each time an honest party needs to send a message to another honest party, $\mathcal{S}$ will tell $\mathcal{A}$ that a message has been delivered such that $\mathcal{A}$ can tell $\mathcal{S}$ the arrival time of this message to help $\mathcal{S}$ instruct the functionalities to delay the outputs in the ideal world. For each request-based delayed output that needs to be sent to an honest party, we let $\mathcal{S}$ delay the output in default until we say $\mathcal{S}$ allows the functionality to send the output. We will show that the output in the ideal world is identically distributed to that in the real world by using hybrid arguments.

Construction of the ideal adversary $\mathcal{S}$.

Simulator $\mathcal{S}$

1. For each honest P_i, $\mathcal{S}$ follows the protocol to emulate $\mathcal{F}_{\mathsf{ACSS}}$ where the dealer is P_i such that corrupted parties get their shares of $[s_i^{(1)}]_t, \ldots, [s_i^{(L)}]_t$, where $L = N/(t+1)$. For each corrupted P_i, $\mathcal{S}$ receives the input of P_i from $\mathcal{A}$ and follows the protocol to simulate $\mathcal{F}_{\mathsf{ACSS}}$.
2. $\mathcal{S}$ follows the protocol to run Π_{ACS}^Q to get a set $S = \{P_{a_1}, \ldots, P_{a_{2t+1}}\}$ (assume that $P_{a_1}, \ldots, P_{a_k}$ are corrupted, where $k \leq t$) and sends it to each party in $\mathcal{P}$.
3. Suppose P_{a_j}'s input polynomials to $\mathcal{F}_{\mathsf{ACSS}}$ are $q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)$ for $j = 1, \ldots, k$. Take the share of

$[s_{a_j}^{(\ell)}]_t = [q_j^{(\ell)}(\beta_0)]_t$ for each P_i as $q_j^{(\ell)}(\beta_i)$ for all $j \in [k], \ell \in [L]$

4. For each corrupted party, $\mathcal{S}$ follows the protocol to compute his shares by

$$\begin{pmatrix} [r_1^{(1)}]_t, \ldots, [r_1^{(L)}]_t \\ \vdots \\ [r_{t+1}^{(1)}]_t, \ldots, [r_{t+1}^{(L)}]_t \end{pmatrix} = M \cdot \begin{pmatrix} [s_{a_1}^{(1)}]_t, \ldots, [s_{a_1}^{(L)}]_t \\ \vdots \\ [s_{a_{2t+1}}^{(1)}]_t, \ldots, [s_{a_{2t+1}}^{(L)}]_t \end{pmatrix}$$

and sends his shares of $\{[r_j^{(\ell)}]_t\}_{j \in [t+1], \ell \in [L]}$ to $\mathcal{F}_{\mathsf{RandShare}}$.

5. For each honest P_i, when P_i gets the set S after running Π_{ACS}^{Q}, $\mathcal{S}$ allows $\mathcal{F}_{\mathsf{RandShare}}$ to send the output to P_i.

Figure 18: Simulator for the $\mathcal{F}_{\mathsf{RandShare}}$

Hybrid arguments:

$\mathbf{Hyb}_0$: In this hybrid, $\mathcal{S}$ runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, honest parties get their shares from $\mathcal{F}_{\mathsf{RandShare}}$ instead of computing their shares by themselves. Note that in both $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$, the honest parties' shares are on degree-t polynomials, so we only need to show that in $\mathbf{Hyb}_1$, each $r_j^{(\ell)}$ is completely random. Take $t+1$ honest parties in S, let them form a set $\mathcal{H} = \{P_{h_1}, \ldots, P_{h_{t+1}}\}$ and let $S \setminus \mathcal{H} = \{P_{c_1}, \ldots, P_{c_t}\}$. For each $P_{a_j} \in \mathcal{H}$, take the j-th column of M out, and let the $t+1$ columns form a Vandermonde matrix $M_{\mathcal{H}}$, and let the other t columns of M form a matrix $M_{\mathcal{C}}$. Note that

$$\begin{aligned} \begin{pmatrix} r_1^{(1)}, \ldots, r_1^{(L)} \\ \vdots \\ r_{t+1}^{(1)}, \ldots, r_{t+1}^{(L)} \end{pmatrix} &= M \cdot \begin{pmatrix} s_{a_1}^{(1)}, \ldots, s_{a_1}^{(L)} \\ \vdots \\ s_{a_{2t+1}}^{(1)}, \ldots, s_{a_{2t+1}}^{(L)} \end{pmatrix} \\ &= M_{\mathcal{H}} \cdot \begin{pmatrix} s_{h_1}^{(1)}, \ldots, s_{h_1}^{(L)} \\ \vdots \\ s_{h_{t+1}}^{(1)}, \ldots, s_{h_{t+1}}^{(L)} \end{pmatrix} + M_{\mathcal{C}} \cdot \begin{pmatrix} s_{c_1}^{(1)}, \ldots, s_{c_1}^{(L)} \\ \vdots \\ s_{c_t}^{(1)}, \ldots, s_{c_t}^{(L)} \end{pmatrix}. \end{aligned}$$

Since $M_{\mathcal{H}}$ is invertible and each $s_{h_j}^{(\ell)}$ where $j \in [t+1]$, $\ell \in [L]$ is randomly sampled in $\mathbb{F}$ by $\mathcal{S}$ when emulating $\mathcal{F}_{\mathsf{ACSS}}$ where P_{h_j} is the dealer,

$$M_{\mathcal{H}} \cdot \begin{pmatrix} s_{h_1}^{(1)}, \ldots, s_{h_1}^{(L)} \\ \vdots \\ s_{h_{t+1}}^{(1)}, \ldots, s_{h_{t+1}}^{(L)} \end{pmatrix}$$

is completely random. Thus,

$$\begin{pmatrix} r_1^{(1)}, \ldots, r_1^{(L)} \\ \vdots \\ r_{t+1}^{(1)}, \ldots, r_{t+1}^{(L)} \end{pmatrix}$$

is also completely random. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

Note that $\mathbf{Hyb}_1$ is the ideal-world scenario, $\Pi_{\mathsf{RandShare}}$ securely computes $\mathcal{F}_{\mathsf{RandShare}}$ in the $(\mathcal{F}_{\mathsf{ACSS}}, \mathcal{F}_{\mathsf{privRec}})$-hybrid model.

We need to invoke $\mathcal{F}_{\mathsf{ACSS}}$ L times for each party. Thus the protocol $\Pi_{\mathsf{RandShare}}$ requires $\mathcal{O}(L \cdot n^4\kappa + n^5\kappa^2 + n^6) = \mathcal{O}(N \cdot n^3\kappa + n^5\kappa^2 + n^6)$-bit communication. □

B.3 Construction of $\Pi_{\mathsf{RandShare}}^0$

To construct our protocol $\Pi_{\mathsf{RandShare}}^0$, we first need a sub-protocol to prepare random coins in $\mathbb{F}$. For this, all parties can invoke $\mathcal{F}_{\mathsf{RandShare}}$ to generate a random share and then invoke $\mathcal{F}_{\mathsf{privRec}}$ to reconstruct the random value to all parties. The functionality is given in Fig. 19. The amortized communication complexity of generating per coin is $\mathcal{F}_{\mathsf{Coin}}$ is $\mathcal{O}(n^3\kappa)$ bits.

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

1: Upon receiving $(\mathsf{Request}, \mathsf{Coin})$ from an honest party, the trusted party samples a random value r.
2: The trusted party sends a request based delayed output r to each $P_i \in \mathcal{P}$.

Figure 19: Ideal functionality for generating a random value

Remark 1. *We need $\mathcal{O}(N \cdot n^3\kappa + n^5\kappa^2 + n^6)$-bit communication to generate N random sharings and $\mathcal{O}(Nn^2\kappa)$-bit communication to reconstruct the N coins to all parties. Thus, we generate N coins with a communication complexity of $\mathcal{O}(N \cdot n^3\kappa + n^5\kappa^2 + n^6)$ bits. We use an amortized cost when generating random coins since we can prepare a lot of random sharings first and reconstruct each coin when we need it.*

We give our construction of $\Pi^0_{\mathsf{RandShare}}$ in the $(\mathcal{F}_{\mathsf{ACSS}}, \mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{privRec}})$-hybrid model in Fig. 20.

Protocol $\Pi^0_{\mathsf{RandShare}}$

On public parameters $N, P_i, (\beta_0, \ldots, \beta_n)$:

1. Each party $P_j \in \mathcal{P}$ samples $L = N/(t+1)$ random value $(s_j^{(1)}, \ldots, s_j^{(L)}) \in \mathbb{F}^L$ and chooses random degree-t polynomials $q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)$ such that for each $\ell \in [L]$, $q_j^{(\ell)}(\beta_0) = s_j^{(L)}$ and $q_j^{(\ell)}(\beta_i) = 0$.
2. Each party P_j sends $(\mathsf{Dealer}, \mathsf{ACSS}, \{q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)\})$ to $\mathcal{F}_{\mathsf{ACSS}}$.
3. Upon terminating $\mathcal{F}_{\mathsf{ACSS}}$ whose dealer is P_j, each party sends $(\mathsf{Request}, \mathsf{Coin})$ to $\mathcal{F}_{\mathsf{Coin}}$ to get a random value r_j. Then each party sends his share of $[s_j]_t = \sum_{\ell=1}^{L} r_j^\ell \cdot [s_j^{(\ell)}]_t$ and $(\mathsf{Request}, \mathsf{privRec}, P_1), \ldots, (\mathsf{Request}, \mathsf{privRec}, P_n)$ to $\mathcal{F}_{\mathsf{privRec}}$ to reconstruct the whole sharing $[s_j]_t$.
4. Let the ACS property Q defined by P_k likes P_k if P_j gets his shares from $\mathcal{F}_{\mathsf{ACSS}}$ whose dealer is P_k and P_i's share of s_k is 0. Then all parties run Π^Q_{ACS} to get a set S of size $2t+1$. Let the sharing generated by $\mathcal{F}_{\mathsf{ACSS}}$ with dealers in S be $\{[s_{a_j}^{(\ell)}]_t\}_{j\in[2t+1],\ell\in[L]}$, where $S = \{P_{a_1}, \ldots, P_{a_{2t+1}}\}$.
4. Let M be the $(t+1) \times (2t+1)$ Vandermonde matrix

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & b_1 & \cdots & b_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & b_1^t & \cdots & b_{2t}^t \end{pmatrix},$$

where $1, b_1, \ldots, b_{2t}$ are $2t+1$ distinct public elements in $\mathbb{F}$.
Each party locally computes his shares by

$$\begin{pmatrix} [r_1^{(1)}]_t, \ldots, [r_1^{(L)}]_t \\ \vdots \\ [r_{t+1}^{(1)}]_t, \ldots, [r_{t+1}^{(L)}]_t \end{pmatrix} = M \cdot \begin{pmatrix} [s_{a_1}^{(1)}]_t, \ldots, [s_{a_1}^{(L)}]_t \\ \vdots \\ [s_{a_{2t+1}}^{(1)}]_t, \ldots, [s_{a_{2t+1}}^{(L)}]_t \end{pmatrix}$$

5. Each party outputs his shares of $\{[r_j^{(\ell)}]_t\}_{j\in[t+1],\ell\in[L]}$.

Figure 20: The protocol to prepare random t-sharing with a zero share

Lemma 6. *Protocol $\Pi^0_{\mathsf{RandShare}}$ t-securely realizes $\mathcal{F}^0_{\mathsf{RandShare}}$ in the $(\mathcal{F}_{\mathsf{ACSS}}, \mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{privRec}})$-hybrid model.*

Proof. We prove this lemma by constructing a simulator $\mathcal{S}$. $\mathcal{S}$ needs to interact with the environment $\mathcal{Z}$ and with the ideal functionalities. $\mathcal{S}$ constructs virtual real-world honest parties and runs the real-world adversary $\mathcal{A}$. For simplicity, we just let $\mathcal{S}$ communicate with $\mathcal{A}$ on behalf of honest parties and the ideal functionality of sub-protocols in our proof. In order to simulate the communication with $\mathcal{Z}$, every message that $\mathcal{S}$ receives from $\mathcal{Z}$ is sent to $\mathcal{A}$, and likewise, every message sent from $\mathcal{A}$ sends to $\mathcal{Z}$ is forwarded by $\mathcal{S}$.

Each time an honest party needs to send a message to another honest party, $\mathcal{S}$ will tell $\mathcal{A}$ that a message has been delivered such that $\mathcal{A}$ can tell $\mathcal{S}$ the arrival time of this message to help $\mathcal{S}$ instruct the functionalities to delay the outputs in the ideal world. For each request-based delayed output that needs to be sent to an honest party, we let $\mathcal{S}$ delay the output in default until we say $\mathcal{S}$ allows the functionality to send the output. We will show that the output in the ideal world is identically distributed to that in the real world by using hybrid arguments.

Construction of the ideal adversary $\mathcal{S}$.

Simulator $\mathcal{S}$

1. For each honest P_i, $\mathcal{S}$ follows the protocol to simulate $\mathcal{F}_{\mathsf{ACSS}}$ where the dealer is P_i such that corrupted parties get their shares of $[s_i^{(1)}]_t, \ldots, [s_i^{(L)}]_t$, where $L = N/(t+1)$. For each corrupted P_i, $\mathcal{S}$ receives the input of P_i from $\mathcal{A}$ and follows the protocol to simulate $\mathcal{F}_{\mathsf{ACSS}}$.
2. For each honest P_j or corrupted P_j whose input polynomials $q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)$ are all degree-t, $\mathcal{S}$ emulate $\mathcal{F}_{\mathsf{Coin}}$ to sample random coin r_j. Then $\mathcal{S}$ computes the shares of corrupted parties of $s_j = \sum_{\ell=1}^{L} r_j^{\ell} \cdot s_j^{(\ell)}$ and emulates $\mathcal{F}_{\mathsf{privRec}}$ to get the whole sharing $[s_j]_t$.
3. If there exists a corrupted P_j such that P_i's share of s_j equals to 0 for each $\ell \in [L]$ but for some $q_j^{(\ell)}(\cdot)$, $q_j^{(\ell)}(\beta_i) \neq 0$, then $\mathcal{S}$ aborts the simulation.
4. $\mathcal{S}$ follows the protocol to run Π_{ACS}^{Q} to get a set $S = \{P_{a_1}, \ldots, P_{a_{2t+1}}\}$ (assume that $P_{a_1}, \ldots, P_{a_k}$ are corrupted, where $k \leq t$) and sends it to each party in $\mathcal{P}$.
5. Suppose P_{a_j}'s input polynomials to $\mathcal{F}_{\mathsf{ACSS}}$ are $q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)$ for $j = 1, \ldots, k$. Take the share of $[s_{a_j}^{(\ell)}]_t = [q_j^{(\ell)}(\beta_0)]_t$ for each P_i as $q_j^{(\ell)}(\beta_i)$ for all $j \in [k], \ell \in [L]$
6. For each corrupted party, $\mathcal{S}$ follows the protocol to compute his shares

$$\begin{pmatrix} [r_1^{(1)}]_t, \ldots, [r_1^{(L)}]_t \\ \vdots \\ [r_{t+1}^{(1)}]_t, \ldots, [r_{t+1}^{(L)}]_t \end{pmatrix} = M \cdot \begin{pmatrix} [s_{a_1}^{(1)}]_t, \ldots, [s_{a_1}^{(L)}]_t \\ \vdots \\ [s_{a_{2t+1}}^{(1)}]_t, \ldots, [s_{a_{2t+1}}^{(L)}]_t \end{pmatrix}.$$

   and sends his shares of $\{[r_j^{(\ell)}]_t\}_{j \in [t+1], \ell \in [L]}$ to $\mathcal{F}_{\mathsf{RandShare}}^{0}$.
6. For each honest P_i, when P_i gets the set S after running Π_{ACS}^{Q}, $\mathcal{S}$ allows $\mathcal{F}_{\mathsf{RandShare}}$ to send the output to P_i.

Figure 21: Simulator for the $\mathcal{F}_{\mathsf{RandShare}}^{0}$

Hybrid arguments:

$\mathbf{Hyb}_0$: In this hybrid, $\mathcal{S}$ runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, $\mathcal{S}$ aborts the simulation if there exists a corrupted P_j such that P_i's share of s_j equals to 0 for each $\ell \in [L]$ but for some $q_j^{(\ell)}(\cdot)$, $q_j^{(\ell)}(\beta_i) \neq 0$. This happens only when the random r_j satisfies $\sum_{\ell=1}^{L} r_j^{\ell} \cdot s_j^{(\ell)} = 0$. Note that any non-zero polynomial $\sum_{\ell=1}^{L} s_j^{(\ell)} x^{\ell}$ has at most L roots in $\mathbb{F}$, so the output distribution only changes with probability

$$\epsilon \leq t \cdot \frac{L}{|\mathbb{F}|} = \frac{tL}{2^{\kappa}},$$

which is negligible. Thus, the output distributions of $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are statistically close.

$\mathbf{Hyb}_2$: In this hybrid, honest parties get their shares from $\mathcal{F}_{\mathsf{RandShare}}$ instead of computing their shares by themselves. Note that in both $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_2$, the honest parties' shares are on degree-t polynomials and P_i's share is 0. If the number of corrupted parties is equal to t and P_i is honest, the security and correctness are straightforward since $\mathcal{A}$ knows the random sharing of any honest party. For other cases, we only need to show that in $\mathbf{Hyb}_1$, each $r_j^{(\ell)}$ is completely random if P_i is corrupted. Take $t+1$ honest parties in S, let them form a set $\mathcal{H} = \{P_{h_1}, \ldots, P_{h_{t+1}}\}$ and let $S \backslash \mathcal{H} = \{P_{c_1}, \ldots, P_{c_t}\}$. For each $P_{a_j} \in \mathcal{H}$, take the j-th column of M out, and let the $t+1$ columns form a Vandermonde matrix $M_{\mathcal{H}}$, and let the other t columns of M form a matrix $M_{\mathcal{C}}$. Note that

$$\begin{pmatrix} r_1^{(1)}, \dots, r_1^{(L)} \\ \vdots \\ r_{t+1}^{(1)}, \dots, r_{t+1}^{(L)} \end{pmatrix} = M \cdot \begin{pmatrix} s_{a_1}^{(1)}, \dots, s_{a_1}^{(L)} \\ \vdots \\ s_{a_{2t+1}}^{(1)}, \dots, s_{a_{2t+1}}^{(L)} \end{pmatrix}$$

$$= M_{\mathcal{H}} \cdot \begin{pmatrix} s_{h_1}^{(1)}, \dots, s_{h_1}^{(L)} \\ \vdots \\ s_{h_{t+1}}^{(1)}, \dots, s_{h_{t+1}}^{(L)} \end{pmatrix} + M_{\mathcal{C}} \cdot \begin{pmatrix} s_{c_1}^{(1)}, \dots, s_{c_1}^{(L)} \\ \vdots \\ s_{c_t}^{(1)}, \dots, s_{c_t}^{(L)} \end{pmatrix}.$$

Since $M_{\mathcal{H}}$ is invertible and each $s_{h_j}^{(\ell)}$ where $j \in [t+1]$, $\ell \in [L]$ is randomly sampled in $\mathbb{F}$ by $\mathcal{S}$ when emulating $\mathcal{F}_{\mathsf{ACSS}}$ where P_{h_j} is the dealer,

$$M_{\mathcal{H}} \cdot \begin{pmatrix} s_{h_1}^{(1)}, \dots, s_{h_1}^{(L)} \\ \vdots \\ s_{h_{t+1}}^{(1)}, \dots, s_{h_{t+1}}^{(L)} \end{pmatrix}$$

is completely random. Thus,

$$\begin{pmatrix} r_1^{(1)}, \dots, r_1^{(L)} \\ \vdots \\ r_{t+1}^{(1)}, \dots, r_{t+1}^{(L)} \end{pmatrix}$$

is also completely random. Thus, $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ have the same output distribution.

Note that $\mathbf{Hyb}_2$ is the ideal-world scenario, $\Pi^0_{\mathsf{RandShare}}$ statistically-securely computes $\mathcal{F}^0_{\mathsf{RandShare}}$ in the $(\mathcal{F}_{\mathsf{ACSS}}, \mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{privRec}})$-hybrid model.

Similar with $\Pi_{\mathsf{RandShare}}$, the protocol $\Pi^0_{\mathsf{RandShare}}$ requires $\mathcal{O}(N \cdot n^3\kappa + n^5\kappa^2 + n^6)$-bit communication. □

C Security Proof of The Asynchronous Packed Information-Checking Protocol

Proof. We prove the security of the APICP protocol by constructing an ideal adversary $\mathcal{S}$. $\mathcal{S}$ needs to interact with the environment $\mathcal{Z}$ and with the ideal functionalities. $\mathcal{S}$ constructs virtual real-world honest parties and runs the real-world adversary $\mathcal{A}$. For simplicity, we just let $\mathcal{S}$ communicate with $\mathcal{A}$ on behalf of honest parties and the ideal functionality of sub-protocols in our proof. In order to simulate the communication with $\mathcal{Z}$, every message that $\mathcal{S}$ receives from $\mathcal{Z}$ is sent to $\mathcal{A}$, and likewise, every message sent from $\mathcal{A}$ sends to $\mathcal{Z}$ is forwarded by $\mathcal{S}$. Each time an honest party needs to send a message to another honest party, $\mathcal{S}$ will tell $\mathcal{A}$ that a message has been delivered such that $\mathcal{A}$ can tell $\mathcal{S}$ the arrival time of this message to help $\mathcal{S}$ instruct the functionalities to delay the outputs in the ideal world. For each request-based delayed output that needs to be sent to an honest party, we let $\mathcal{S}$ delay the output in default until we say $\mathcal{S}$ allows the functionality to send the output. We will show that the output in the ideal world is identically distributed to that in the real world by using hybrid arguments.

Construction of the ideal adversary $\mathcal{S}$ is as follows. If we say that $\mathcal{S}$ delivers a message, $\mathcal{S}$ just tells $\mathcal{A}$ that the message has been delivered. $\mathcal{S}$ may not be able to know the context of the message.

When D and I are honest:

Simulator $\mathcal{S}$

1. For each corrupted party P_i, $\mathcal{S}$ randomly samples $(T+1)T\kappa$ verification points $z = (\alpha, \beta_1, \ldots, \beta_m)$ (each verification point is corresponding to $(\alpha, f^{(1)}(\alpha), \ldots, f^{(m)}(\alpha))$, where $f^{(1)}(\cdot), \ldots, f^{(m)}(\cdot)$ are the polynomials generated by D) from $\mathbb{F}^{m+1}$ and send them to P_i on behalf of D. If the $(T+1)^2 t\kappa$ verification points are not distinct, $\mathcal{S}$ aborts the simulation.
2. For each $P_i \in \mathcal{P}$, if P_i is corrupted, $\mathcal{S}$ waits to receive a verification set Z^i_{T+1} from P_i and then checks whether each verification point in this set is in the set of all verification points he sent to corrupted parties. If P_i is honest, when I receives the verification set of P_i, $\mathcal{S}$ considers that I receives a correct verification set.
3. When I receives at least $2t+1$ correct verification sets, he initializes a counter $\mathsf{count} = T$ and a set $\mathcal{W} = \emptyset$. Then, $\mathcal{S}$ allows $\mathcal{F}_{\mathsf{APICP}}$ to send the output to I.
4. For each revelation, if $\mathsf{count} > 0$, $\mathcal{S}$ does the following things and replaces count by $\mathsf{count} - 1$:
   - If R is honest:
   (a). For each verification point $z = (\alpha, \beta_1, \ldots, \beta_m)$ $\mathcal{S}$ has sent to a corrupted party, $\mathcal{S}$ computes a new point $(\alpha, \sum_{k=1}^m c_k\beta_k)$. Assuming all these new points form a set $\mathcal{M}$. Then $\mathcal{S}$ checks $\mathcal{W} = \{(\boldsymbol{c}_1, f_1(x)), \ldots, (\boldsymbol{c}_k, f_k(x))\}$, if $\boldsymbol{c}, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$ are linear dependent, assume $\boldsymbol{c} = \sum_{j=1}^k a_j\boldsymbol{c}_j$, then $\mathcal{S}$ computes $f(x) = \sum_{j=1}^k a_j f_j(x)$.
   (b). $\forall P_i \in \mathcal{P}$, if P_i is corrupted, $\mathcal{S}$ waits to receive a verification set $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$ from $\mathcal{A}$. If $\boldsymbol{c}, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$ are linear independent, $\mathcal{S}$ checks whether at least one point in $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$ is in $\mathcal{M}$. Otherwise, $\mathcal{S}$ checks whether at least one point in $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$ on $f(x)$. If the check passes, $\mathcal{S}$ considers that R receives a correct verification set. If P_i is honest, $\mathcal{S}$ regards that R receives a correct verification set $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$ from P_i when the set arrives.
   (c). After receiving $t+1$ correct verification sets, $\mathcal{S}$ allows $\mathcal{F}_{\mathsf{APICP}}$ to send the output to R.
   - If R is corrupted:
   (a). $\mathcal{S}$ waits to receive $\boldsymbol{s}$ from $\mathcal{F}_{\mathsf{APICP}}$. For each verification point $z = (\alpha, \beta_1, \ldots, \beta_m)$ $\mathcal{S}$ sent to corrupted P_i, $\mathcal{S}$ computes a new point $(\alpha, \sum_{k=1}^m c_k\beta_k)$. Since the each α of these points are distinct, $\mathcal{S}$ has $(T+1)^2 t\kappa$ new points. Then $\mathcal{S}$ checks $\mathcal{W} = \{(\boldsymbol{c}_1, f_1(x)), \ldots, (\boldsymbol{c}_k, f_k(x))\}$.
     – If $\boldsymbol{c}, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$ are linear dependent, assume $\boldsymbol{c} = \sum_{j=1}^k a_j\boldsymbol{c}_j$, then $\mathcal{S}$ gets a new polynomial $f(x) = \sum_{j=1}^k a_j f_j(x)$.
     – If $\boldsymbol{c}, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$ are linear independent, $\mathcal{S}$ samples a random degree-$((T+1)^2 t\kappa + L)$ polynomial $f(x)$ whose L highest coefficients form vector $\boldsymbol{s}$ and all the $(T+1)^2 t\kappa$ new points are on this polynomial.
   (b). $\mathcal{S}$ reveals $f(x)$ to R. For each honest P_i, $\mathcal{S}$ samples $(T+1)\kappa$ random points $\alpha^i_1, \ldots, \alpha^i_{(T+1)\kappa}$ in $\mathbb{F}$ and sends $\{((\alpha^i_1, f(\alpha^i_1)), \ldots, (\alpha^i_{(T+1)\kappa}, f(\alpha^i_{(T+1)\kappa})))\}$ to R on behalf of P_i.

Figure 22: Simulator for the $\mathcal{F}_{\mathsf{APICP}}$ when both D and I are honest

Hybrid arguments:

$\mathbf{Hyb}_0$: In this hybrid, $\mathcal{S}$ receives honest parties' inputs and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, $\mathcal{S}$ first samples the verification points for corrupted parties randomly from $\mathbb{F}^{m+1}$ and then sample the polynomials $f^{(1)}(x), \ldots, f^{(m)}(x)$ based on the $(T+1)^2 t\kappa$ verification points and the secrets $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}$. Since the polynomials are degree-$(L+(T+1)^2 t\kappa)$ and each $\boldsymbol{s}^{(k)}$ is in $\mathbb{F}^L$, the verification points are $((T+1)^2 t\kappa + 1)$-wise independent in $\mathbf{Hyb}_0$, i.e. the $(T+1)^2 t\kappa$ verification points for corrupted parties are uniformly random. So we only change the order of generating corrupted parties' verification points and the polynomials, which doesn't change the output distribution. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

$\mathbf{Hyb}_2$: In this hybrid, if the verification points for corrupted parties are not distinct, then $\mathcal{S}$ aborts the simulation. Note that the probability that each two verification points are the same is $1/|\mathbb{F}|$ and there are $(T+1)^2 t\kappa$ points for corrupted parties, so the probability that the verification points are not distinct is

$$\epsilon_1 < \frac{1}{|\mathbb{F}|} \cdot \frac{((T+1)^2 t\kappa)((T+1)^2 t\kappa - 1)}{2} < \frac{(T+1)^4 t^2 \kappa^2}{2^\kappa},$$

which is negligible in κ. Thus, the output distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are statistically close.

$\mathbf{Hyb}_3$: In this hybrid, $\mathcal{S}$ doesn't check the verification points sent from honest parties by himself on behalf of I. Instead, $\mathcal{S}$ considers that I accepts P_i's verification set when I receives it. Since D and P_i are both honest, P_i always sends a correct verification set, so I always accepts it. Thus, $\mathbf{Hyb_3}$ and $\mathbf{Hyb_2}$ have the same output distribution.

$\mathbf{Hyb}_4$: In this hybrid, $\mathcal{S}$ doesn't check the verification points sent from corrupted parties by himself on behalf of I. Instead, $\mathcal{S}$ checks if the verification points are among those generated from D sent to corrupted parties. The distribution changes only when some verification point $(\alpha', \beta'_1, \ldots, \beta'_m)$ (different from the verification points sent from D to corrupted parties) sent from corrupted to I is just $(\alpha', f^{(1)}(\alpha'), \ldots, f^{(m)}(\alpha'))$. Since corrupted parties only have $(T+1)^2 t\kappa$ verification points, so at any point α' different from the first element of any of them, $(f^{(1)}(\alpha'), \ldots, f^{(m)}(\alpha'))$ is uniformly random in $\mathbb{F}^m$. So the probability that the verification point $(\alpha', \beta'_1, \ldots, \beta'_m)$ will be accepted by I in $\mathbf{Hyb_2}$ is at most $1/|\mathbb{F}|^m$ and $(T+1)t\kappa$ verification points are sent from corrupted parties to I. This means the probability that the output distribution changes is at most

$$\epsilon_2 < \frac{(T+1)t\kappa}{2^{m\kappa}},$$

which is negligible in κ. Thus, the output distributions of $\mathbf{Hyb_4}$ and $\mathbf{Hyb_3}$ are statistically close.

$\mathbf{Hyb}_5$: In this hybrid, $\mathcal{S}$ generate a set $\mathcal{W}$ with each element $(\boldsymbol{c}, f(x))$ as the adversary's knowledge of vectors $\boldsymbol{c}$ from previous revelations and their corresponding polynomial $f(x)$ sent from I to R. $\mathcal{W}$ is initialized to be $\emptyset$. For each revelation, if R is corrupted, $\mathcal{S}$ needs to add the $\boldsymbol{c}$ and the corresponding $f(x)$ to $\mathcal{W}$ if the $\boldsymbol{c}$ can't be generated by doing a linear combination of the vectors used in previous revelation to corrupted parties since $f(x)$ can't be determined based on previous polynomials revealed to corrupted parties. On the other hand, if $\boldsymbol{c}$ can be generated by doing a linear combination of the vectors used in previous revelation to corrupted parties, the adversary knows what polynomial will be sent from I from $\mathcal{W}$, so there is no need to add elements into $\mathcal{W}$. That is to say, $\mathcal{S}$ checks $\mathcal{W} = \{(\boldsymbol{c}_1, f_1(x)), \ldots, (\boldsymbol{c}_k, f_k(x))\}$, if $\boldsymbol{c}, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$ are linear independent, $\mathcal{S}$ includes $(\boldsymbol{c}, f(x))$ into $\mathcal{W}$, where $f(x) = \sum_{i=1}^m f^{(i)}(x)$. Generating this $\mathcal{W}$ doesn't change the output distribution. Thus, $\mathbf{Hyb_5}$ and $\mathbf{Hyb_4}$ have the same output distribution.

$\mathbf{Hyb}_6$: In this hybrid, $\mathcal{S}$ doesn't sample $f^{(1)}(x), \ldots, f^{(k)}(x)$ at the beginning. Instead, $\mathcal{S}$ only computes or samples $f(x)$ corresponding to $\boldsymbol{c}$ during each revelation. If $f(x)$ can be determined from $\mathcal{W} = \{(\boldsymbol{c}_1, f_1(x)), \ldots, (\boldsymbol{c}_k, f_k(x))\}$, i.e. $\boldsymbol{c}$ can be computed by doing linear combination of $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$, $\mathcal{S}$ directly computes $f(x)$ by doing the same linear combination of $f_1(x), \ldots, f_k(x)$. Otherwise, $\mathcal{S}$ samples a random degree-$((T+1)^2 t\kappa + L)$ polynomial $f(x)$ whose L highest coefficients form vector $\boldsymbol{s} = \sum_{i=1}^m c_i \boldsymbol{s}^{(i)}$ and all the $(T+1)^2 t\kappa$ new points $(\alpha, \sum_{i=1}^m c_i\beta_i)$ are on this polynomial. Since sampling each $f^{(i)}(x)$ based on $\boldsymbol{s}^{(i)}$ and those (α, β_i) can be regard as sampling a point (α'_i, β'_i) on $f^{(i)}(x)$ where α'_i is different from the α of any corrupted parties' verification points. Then $\mathcal{S}$ can compute $f^{(i)}(x)$ based on and $\boldsymbol{s}^{(i)}$ and the $(T+1)^2 t\kappa + 1$ distinct points on the polynomial. Since random sampling each β'_i in $\mathbb{F}$ and add up $\beta = \sum_{i=1}^m c_i \beta'_i$ can only get a uniformly random $\beta \in \mathbb{F}$ when $\boldsymbol{c}$ can't be computed by doing linear combination of $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$, which is equivalent to directly sampling $f(x)$ based on $\sum_{i=1}^m c_i \boldsymbol{s}^{(i)}$ and those $(\alpha, \sum_{i=1}^m c_i\beta_i)$. So this doesn't change the output distribution. Thus, $\mathbf{Hyb_6}$ and $\mathbf{Hyb_5}$ have the same output distribution.

$\mathbf{Hyb}_7$: In this hybrid, for each revelation if R is corrupted, for each honest party P_i, D doesn't compute $(\alpha, \sum_{k=1}^m c_k\beta_k)$ for each verification point P_i has. Instead, $\mathcal{S}$ sample $(T+1)\kappa$ random points $\alpha_1^i, \ldots, \alpha_{(T+1)\kappa}^i$ in $\mathbb{F}$ and sends $\{((\alpha_1^i, f(\alpha_1^i)), \ldots, (\alpha_{(T+1)\kappa}^i, f(\alpha_{(T+1)\kappa}^i)))\}$ to R on behalf of P_i. Since each α of the verification points P_i is randomly sampled and each (α, β) sent from P_i to R must be on the $f(x)$ sent from I to R. So this doesn't change the output distribution. Thus, $\mathbf{Hyb_7}$ and $\mathbf{Hyb_6}$ have the same output distribution.

$\mathbf{Hyb}_8$: In this hybrid, for each revelation if R is honest, for each honest P_i, $\mathcal{S}$ doesn't follow the protocol to check P_i's verification set on behalf of R. Instead, $\mathcal{S}$ considers that R accepts P_i's verification set when R receives it. Since D and P_i are both honest, P_i always sends a correct verification set, so R always accepts it. Thus, $\mathbf{Hyb_8}$ and $\mathbf{Hyb_7}$ have the same output distribution.

$\mathbf{Hyb}_9$: In this hybrid, for each revelation if R is honest, when $\mathbf{c}$ is not a linear combination of $\mathbf{c}_1, \ldots, \mathbf{c}_k$ in $\mathcal{W}$, $\mathcal{S}$ doesn't check whether there exists an verification point sent from corrupted P_i to R on the $f(x)$ sent from I to R. Instead, $\mathcal{S}$ checks if there exists an verification point correctly computed from the verification points sent from D to corrupted parties. The output distribution only changes when some verification point (α', β') sent from corrupted P_i to R satisfies that α' is different from any α of the corrupted parties' verification points sent from D, but $\beta' = f(\alpha)$. Since in this condition, even if $\mathcal{A}$ knows $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}$, he still need another point to compute $f(x)$. Since $f(x)$ is randomly generated based on $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}$ and the corrupted parties' verification points, for any α', $F(\alpha')$ is uniformly random in $\mathbb{F}$. Since there are $(T+1)t\kappa$ points sent from corrupted parties to R, the probability that the output distribution changes is

$$\epsilon_3 = \frac{(T+1)t\kappa}{2^\kappa},$$

which is negligible in κ. Thus, the distributions of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ are statistically close.

$\mathbf{Hyb}_{10}$: In this hybrid, $\mathcal{S}$ doesn't know $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}$ at the beginning and use them to compute $\boldsymbol{s}$ for each revelation to corrupted parties. Instead, $\mathcal{S}$ gets $\boldsymbol{s}$ from the functionality output if R is corrupted. Since $\mathcal{S}$ doesn't need to sample $f^{(1)}(x), \ldots, f^{(m)}(x)$ in $\mathbf{Hyb}_7$, he doesn't need $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}$ except using them to compute $\boldsymbol{s}$ when R is corrupted. Thus, $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ have the same output distribution.

Note that $\mathbf{Hyb}_{10}$ is the ideal-world scenario, Π_{APICP} statistically-securely computes $\mathcal{F}_{\mathsf{APICP}}$.

Remark 2. *When D and I are both honest, R is corrupted, $\mathcal{S}$ may not be able to compute the polynomial based on the verification points of the corrupted parties, the secret, and one other point on the polynomial if there are less than t corrupted parties. In this case, we can generate the corrupted parties and $(n-1)/3-t$ honest parties' verification points, and then $\mathcal{S}$ can still compute the polynomials.*

When D and I are corrupted:

Simulator $\mathcal{S}$

1. For each honest $P_i \in \mathcal{P}$, $\mathcal{S}$ receives $(T+1)^2\kappa$ verification points on behalf of P_i. Then $\mathcal{S}$ randomly divides them into $T+1$ disjoint sets. Each set is of size $(T+1)\kappa$, denoted by $Z_1^i, \ldots, Z_{T+1}^i$. Then $\mathcal{S}$ sends Z_{T+1}^i to I on behalf of P_i.
2. $\mathcal{S}$ sets $\boldsymbol{s}^{(1)} = \cdots = \boldsymbol{s}^{(m)} = \mathbf{0}$ and sends $(\mathsf{Init}, \mathsf{APICP}, T, (\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}))$ to $\mathcal{F}_{\mathsf{APICP}}$ on behalf of D. Here $\mathbf{0}$ is the zero vector in $\mathbb{F}^L$.
3. $\mathcal{S}$ initializes a counter $\mathsf{count} = T$.
4. For each revelation, if $\mathsf{count} > 0$, $\mathcal{S}$ does the following and replaces count by $\mathsf{count} - 1$:
   - If R is honest:
     (a). $\mathcal{S}$ receives $f'(x)$ from I.
     (b). For each corrupted $P_i \in \mathcal{P}$, $\mathcal{S}$ waits to receive a verification set $Z_{\mathsf{count}}^{i,\mathbf{c}}$. For each honest $P_i \in \mathcal{P}$, $\mathcal{S}$ uses Z_{count}^i to compute $Z_{\mathsf{count}}^{i,\mathbf{c}} = \{(r_0, \sum_{k=1}^m c_k r_k) \mid (r_0, \ldots, r_m) \in Z_{\mathsf{count}}^i\}$. In both cases, $\mathcal{S}$ checks whether at least one point in $Z_{\mathsf{count}}^{i,\mathbf{c}}$ is consistent with $f'(x)$. If true, $\mathcal{S}$ considers that R receives a correct verification set.
     (c). When $\mathcal{S}$ receives $t+1$ correct verification sets, let $\boldsymbol{s}'$ be the L highest coefficients of $f'(x)$, he sends $\mathsf{Proceed}$ and $\boldsymbol{s}'$ to $\mathcal{F}_{\mathsf{APICP}}$ and allows $\mathcal{F}_{\mathsf{APICP}}$ to send the output to R. When $\mathcal{S}$ receives $2t+1$ incorrect verification sets, he send Ignore to $\mathcal{F}_{\mathsf{APICP}}$.
   - If R is corrupted, for each honest P_i, $\mathcal{S}$ uses Z_{count}^i to compute $Z_{\mathsf{count}}^{i,\mathbf{c}}$ and sends $Z_{\mathsf{count}}^{i,\mathbf{c}}$ to R on behalf of P_i.

Figure 23: Simulator for the $\mathcal{F}_{\mathsf{APICP}}$ when both D and I are corrupted

Hybrid arguments:

$\mathbf{Hyb}_0$: In this hybrid, $\mathcal{S}$ receives honest parties' inputs and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, we change how R receives $\boldsymbol{s}'$. After receiving at least $t+1$ correct verification sets, $\mathcal{S}$ sends $\mathsf{Proceed}$ and $\boldsymbol{s}'$ to $\mathcal{F}_{\mathsf{APICP}}$ and lets R receives the output from $\mathcal{F}_{\mathsf{APICP}}$. The difference between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$ is R will wait to receive $\boldsymbol{s}'$ from $\mathcal{S}$ or $\mathcal{F}_{\mathsf{APICP}}$, which makes no difference to the output distribution. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

Note that $\mathbf{Hyb}_1$ is the ideal-world scenario, Π_{APICP} statistically-securely computes $\mathcal{F}_{\mathsf{APICP}}$.

When D is honest and I is corrupted:

Simulator $\mathcal{S}$

1. $\mathcal{S}$ waits to receive $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}$ from $\mathcal{F}_{\mathsf{APICP}}$. For each $k \in [m]$, $\mathcal{S}$ selects a random degree-$(L + t(T+1)^2\kappa)$ polynomial $f^{(k)}(x)$ whose the L highest coefficients form vector $\boldsymbol{s}^{(k)}$.
2. For each corrupted P_i, $\mathcal{S}$ randomly samples $(T+1)^2\kappa$ elements from $\mathbb{F}$. For each element α, $\mathcal{S}$ sends verification point $z = (\alpha, f^{(1)}(\alpha), \ldots, f^{(m)}(\alpha))$ to corrupted P_i on behalf of D.
3. $\mathcal{S}$ sends $f^{(1)}(x), \ldots, f^{(m)}(x)$ to I. For each honest P_i, $\mathcal{S}$ randomly samples $(T+1)\kappa$ elements $\alpha^i_1, \ldots, \alpha^i_{(T+1)\kappa}$ in $\mathbb{F}$. For each $j \in [(T+1)\kappa]$, $\mathcal{S}$ sends verification point $\{(\alpha^i_j, f^{(1)}(\alpha^i_j), \ldots, f^{(m)}(\alpha^i_j))\}$ to I on behalf of P_i.
4. $\mathcal{S}$ initializes a counter $\mathsf{count} = T$.
5. For each revelation, $\mathcal{S}$ computes $f(x) = \sum_{k=1}^{m} c_k f^{(k)}(x)$. If $\mathsf{count} > 0$, $\mathcal{S}$ does the following things and replaces count by $\mathsf{count} - 1$:
   - If R is honest:
     (a). $\mathcal{S}$ receives $f'(x)$ from I.
     (b). Let $\boldsymbol{s}$ be vector of the L highest coefficients of $f(x)$. $\mathcal{S}$ checks whether $f(x) = f'(x)$. If so, $\mathcal{S}$ sends $\mathsf{Proceed}$ and $\boldsymbol{s}$ to $\mathcal{F}_{\mathsf{APICP}}$. Otherwise, $\mathcal{S}$ sends Ignore to $\mathcal{F}_{\mathsf{APICP}}$.
     (c). If $f'(x) = f(x)$, for each corrupted $P_i \in \mathcal{P}$, $\mathcal{S}$ receives a verification set $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$ from P_i and follows the protocol to check the verification set from P_i on behalf of R. For each honest P_i, $\mathcal{S}$ considers that R receives a correct verification set when R receives it.
     (d). When $\mathcal{S}$ receives $t+1$ correct verification sets, he allows $\mathcal{F}_{\mathsf{APICP}}$ to send the output to R.
   - If R is corrupted, for each honest P_i, $\mathcal{S}$ samples $(T+1)\kappa$ random elements $\alpha^i_1, \ldots, \alpha^i_{(T+1)\kappa}$ from $\mathbb{F}$ and sends $Z^{i,\boldsymbol{c}}_{\mathsf{count}} = \{(\alpha^i_j, f(\alpha^i_j))\}_{j \in [(T+1)\kappa]}$ to R on behalf of P_i.

Figure 24: Simulator for the $\mathcal{F}_{\mathsf{APICP}}$ when D is honest and I is corrupted

Hybrid arguments:

$\mathbf{Hyb}_0$: In this hybrid, $\mathcal{S}$ receives honest parties' inputs and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, $\mathcal{S}$ will change how to compute the corrupted parties' verification points. Instead of using honest D's inputs, $\mathcal{S}$ waits to receive $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}$ from $\mathcal{F}_{\mathsf{APICP}}$. The only difference between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$ is how $\mathcal{S}$ gets $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}$, which doesn't change the output distribution. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

$\mathbf{Hyb}_2$: In this hybrid, for each honest party, $\mathcal{S}$ will not sample the whole $(T+1)^2\kappa$ random elements to compute verification points at the beginning. When each honest P_i needs to uses Z^i_{T+1} or $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$, $\mathcal{S}$ will randomly samples $(T+1)\kappa$ elements from $\mathbb{F}$ to compute Z^i_{T+1} or $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$. We only changed the order of creating random elements. Thus, $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ have the same output distribution.

$\mathbf{Hyb}_3$: In this hybrid, when R is honest, $\mathcal{S}$ adds an addition condition $f'(x) = f(x)$ to check whether $f'(x)$ received from I is acceptable. If so, $\mathcal{S}$ sends $\mathsf{Proceed}$ and $\boldsymbol{s}$ to $\mathcal{F}_{\mathsf{APICP}}$. Otherwise, $\mathcal{S}$ sends Ignore to $\mathcal{F}_{\mathsf{APICP}}$. The difference between $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ is when $f'(x) \neq f(x)$, $\mathcal{S}$ can still receive $t+1$ correct $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$. When $f'(x) \neq f(x)$ are accept by $\mathcal{S}$, that means at least one point in honest P_i's $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$ is consistent with $f'(x)$. For each honest P_i, in each revelation time, the probability ϵ_4 that this happens is equal to I correctly guesses one random element sampled by $\mathcal{S}$.

$$\epsilon_4 = \Pr\left[\mathcal{S} \text{ accepts } f'(x) | f'(x) \neq f(x)\right] = \prod_{j=0}^{(T+1)\kappa-1} \frac{\kappa - j}{|\mathbb{F}| - j} \leq (\frac{\kappa}{2^\kappa})^{(T+1)\kappa}$$

which is negligible.

Then we take the union bound for all honest parties and revelation times, the probability that I can fake a $f'(x)$ is $(2t+1)T\epsilon_4$, which is still negligible. Thus, the output distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are statistically close.

$\mathbf{Hyb}_4$: In this hybrid, for each revelation if R is honest, for each honest P_i, $\mathcal{S}$ doesn't follow the protocol to check P_i's verification set on behalf of R. Instead, $\mathcal{S}$ considers that R accepts P_i's verification set when R receives it. Since D and P_i are both honest, $f'(x) = f(x)$, P_i always sends a correct verification set, so R always accepts it. Thus, $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ have the same output distribution.

$\mathbf{Hyb}_5$: In this hybrid, we change how R receives $\boldsymbol{s}$. Upon receiving at least $t+1$ correct $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$ and $f'(x) = f(x)$, $\mathcal{S}$ lets R receives the output from $\mathcal{F}_{\mathsf{APICP}}$. Besides, $\boldsymbol{s}$ computed by $\mathcal{S}$ is equal to $\boldsymbol{s}$ computed by $\mathcal{F}_{\mathsf{APICP}}$. Thus, $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ have the same output distribution.

Note that $\mathbf{Hyb}_5$ is the ideal-world scenario, Π_{APICP} statistically-securely computes $\mathcal{F}_{\mathsf{APICP}}$.

When D is corrupted and I is honest:

Simulator $\mathcal{S}$

1. For each honest P_i, $\mathcal{S}$ waits to receive verification points from D. When P_i receives $(T+1)^2\kappa$ verification points, $\mathcal{S}$ randomly divides them into $T+1$ disjoint sets, where each set is of size $(T+1)\kappa$, denoted by $Z^i_1, \ldots, Z^i_{T+1}$.
2. $\mathcal{S}$ receives $f^{(1)}(x), \ldots, f^{(m)}(x)$ from D and does the following:
   (a). $\mathcal{S}$ checks whether all of these polynomials are degree-$(L + t(T+1)^2\kappa)$. If so, $\mathcal{S}$ lets $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}$ be the vector of the L highest coefficients of $f^{(1)}(x), \ldots, f^{(m)}(x)$. Otherwise, $\mathcal{S}$ aborts the simulation.
   (b). For each $P_i \in \mathcal{P}$:
   - If P_i is corrupted, $\mathcal{S}$ waits to receive a verification set Z^i_{T+1}. When verification points in Z^i_{T+1} are all consistent with $f^{(1)}(x), \ldots, f^{(m)}(x)$, $\mathcal{S}$ considers that I receives a correct verification set.
   - Otherwise, $\mathcal{S}$ uses the Z^i_{T+1} created by himself. When verification points in Z^i_{T+1} are all consistent with $f^{(1)}(x), \ldots, f^{(m)}(x)$ and at least $T(T+1)\kappa + 1$ verification points among $\{Z^i_j\}_{j\in[T+1]}$ are all consistent with $f^{(1)}(x), \ldots, f^{(m)}(x)$, $\mathcal{S}$ considers that R receives a correct verification set.

   (c). When I receives $2t+1$ correct verification sets, he initializes a counter $\mathsf{count} = T$ and sends $(\mathsf{Init}, \mathsf{APICP}, T, (\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}))$ to $\mathcal{F}_{\mathsf{APICP}}$. Then, $\mathcal{S}$ allows $\mathcal{F}_{\mathsf{APICP}}$ to send the output to I. Otherwise, $\mathcal{S}$ does not continue.
3. For each revelation, $\mathcal{S}$ computes $f(x) = \sum_{k=1}^m c_k f^{(k)}(x)$. When $\mathsf{count} > 0$, $\mathcal{S}$ does the following things and replaces count by $\mathsf{count} - 1$:
   - If R is honest:
     (a). For each $P_i \in \mathcal{P}$, if P_i is corrupted, $\mathcal{S}$ waits to receive a verification set $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$ from P_i. If P_i is honest, $\mathcal{S}$ uses $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$ created by himself. In both cases, $\mathcal{S}$ checks whether at least one point in $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$ is consistent with $f(x)$.
     (b). When $\mathcal{S}$ receives at least $t+1$ correct verification sets, let $\boldsymbol{s}$ be the vector of the L highest coefficients of $f(x)$, he allows $\mathcal{F}_{\mathsf{APICP}}$ to send the output to R.
   - If R is corrupted, $\mathcal{S}$ sends $f(x)$ to R on behalf of I. $\mathcal{S}$ uses Z^i_{count} to compute $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$ and sends $Z^{i,\boldsymbol{c}}_{\mathsf{count}}$ to R on behalf of each honest P_i.

Figure 25: Simulator for the $\mathcal{F}_{\mathsf{APICP}}$ when D is corrupted and I is honest

Hybrid arguments:

$\mathbf{Hyb}_0$: In this hybrid, $\mathcal{S}$ receives honest parties' inputs and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, we change how I receives $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}$. After receiving at least $2t+1$ correct verification sets, $\mathcal{S}$ sends $(\mathsf{Init}, \mathsf{APICP}, T, (\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}))$ to $\mathcal{F}_{\mathsf{APICP}}$. Then I will receive $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(m)}$ from $\mathcal{F}_{\mathsf{APICP}}$ instead of computing them by himself, which makes no difference to the output distribution. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

$\mathbf{Hyb}_2$: In this hybrid, $\mathcal{S}$ adds an additional verification condition for honest P_i's verification points. $\mathcal{S}$ will also check at least $(T+1)T\kappa+1$ verification points among $\{Z^i_j\}_{j\in[T+1]}$ are consistent with $f^{(1)}(x), \ldots, f^{(m)}(x)$. If true, $\mathcal{S}$ considers that Z^i_{T+1} is a correct verification set. The output distribution changes only when honest P_i has less than $(T+1)^2\kappa + 1$ correct verification points but he still provides a correct Z^i_{T+1} to I. The probability is

$$\epsilon_5 \le \prod_{j=0}^{(T+1)\kappa-1} \frac{(T+1)T\kappa - j}{(T+1)^2\kappa - j} \le \left(1 - \frac{1}{T+1}\right)^{(T+1)\kappa} \le e^{-\kappa}$$

which is negligible. Since $\mathcal{S}$ needs to receive at least $t+1$ honest P_i's correct Z^i_{T+1}, the output distribution only changes with probability $(t+1)\epsilon_5$, which is still negligible. Thus, the output distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are statistically close.

Note that $\mathbf{Hyb}_2$ is the ideal-world scenario, Π_{APICP} statistically-securely computes $\mathcal{F}_{\mathsf{APICP}}$.

Then we compute the communication complexity of our protocol.

Π_{APICP} takes m vectors as inputs and each vector is of size L, T is an input parameter. All parties execute the Initialization Phase only once, while the Revelation Phase can be executed for at most T times.

During the Initialization Phase: D sends m degree-$(L+t(T+1)^2\kappa)$ polynomials to I, which requires $\mathcal{O}(mL\kappa + mnT^2\kappa^2)$-bit communication. D also sends $(m+1)(T+1)^2\kappa$ evaluation points to each party, resulting in a communication of $\mathcal{O}(mnT^2\kappa^2)$. Each party sends a set of size $(T+1)\kappa^2$ to I, resulting in a communication of $\mathcal{O}(nT^2\kappa^2)$ bits. Therefore, the total communication cost is $\mathcal{O}(mL\kappa+mnT^2\kappa^2)$ bits during the Initialization Phase.

During the Revelation Phase: Each time sending a degree-$(L+t(T+1)^2\kappa)$ polynomial from I to R requires communication of $\mathcal{O}(L\kappa + nT^2\kappa^2)$ bits. Each party sends a set of $(T+1)\kappa$ field elements to I, resulting in a communication of $\mathcal{O}(nT^2\kappa^2)$ bits. Therefore, the total communication cost is $\mathcal{O}(L\kappa + nT^2\kappa^2)$ bits during the Revelation Phase. Since the Revelation Phase can be executed for at most T times, the total communication cost is $\mathcal{O}(LT\kappa + nT^3\kappa^2)$ bits.

Therefore, Π_{APICP} requires communication of $\mathcal{O}(mL\kappa + mnT^2\kappa^2 + LT\kappa + nT^3\kappa^2)$ bits. □

D Proof of the Main Theorem about Our ACSS Protocol

D.1 Construction of Π_{ACSS}

We present our construction of Π_{ACSS} as follows.

Protocol Π_{ACSS}

All parties execute Π_{Sh}, Π_{Ver}, Π_{Auth} and Π_{Comp} in order.

Figure 26: The protocol of the Π_{ACSS}

D.2 Proof of Lemma 2

Proof. We prove this theorem by constructing a simulator $\mathcal{S}$. $\mathcal{S}$ needs to interact with the environment $\mathcal{Z}$ and with the ideal functionalities. $\mathcal{S}$ constructs virtual real-world honest parties and runs the real-world adversary $\mathcal{A}$. For simplicity, we just let $\mathcal{S}$ communicate with $\mathcal{A}$ on behalf of honest parties and the ideal functionality of sub-protocols in our proof. In order to simulate the communication with $\mathcal{Z}$, every message that $\mathcal{S}$ receives from $\mathcal{Z}$ is sent to $\mathcal{A}$, and likewise, every message sent from $\mathcal{A}$ sends to $\mathcal{Z}$ is forwarded by $\mathcal{S}$. Each time an honest party needs to send a message to another honest party, $\mathcal{S}$ will tell $\mathcal{A}$ that a message has been delivered such that $\mathcal{A}$ can tell $\mathcal{S}$ the arrival time of this message to help $\mathcal{S}$ instruct the functionalities to delay the outputs in the ideal world. For each request-based delayed output that needs to be sent to an honest party, we let $\mathcal{S}$ delay the output in default until we say $\mathcal{S}$ allows the functionality to send the output. We will show that the output in the ideal world is identically distributed to that in the real world by using hybrid arguments.

Construction of the ideal adversary $\mathcal{S}$.

When D is honest:

Simulator $\mathcal{S}$

Sharing Phase

1. For each corrupted P_j, $\mathcal{S}$ receives $\{q_1(\alpha_j), \ldots, q_N(\alpha_j)\}$ from $\mathcal{F}_{\mathsf{ACSS}}$.
2. Define $\mathsf{idx} = ((k-1)\cdot L' + \ell - 1)\cdot(t+1)+1$. For each $\ell \in [L']$, $k \in [m']$ and corrupted P_j, $\mathcal{S}$ randomly selects a degree-$2t$ (column) polynomial $g_{\ell,j}^{(k)}(y)$ such that $g_{\ell,j}^{(k)}(\alpha_{-i}) = q_{\mathsf{idx}+i}(\alpha_j)$ for each $i \in [0,t]$. For each $\ell \in [L']$, $k \in [m'+1, m]$ and corrupted P_j, $\mathcal{S}$ randomly samples a degree-$2t$ polynomial as $g_{\ell,j}^{(k)}(y)$.
3. For each $\ell \in [L']$, $k \in [m]$ and corrupted P_i, $\mathcal{S}$ randomly selects a degree-t (row) polynomial $f_{\ell,i}^{(k)}(y)$ such

that $f_{\ell,i}^{(k)}(\alpha_j) = g_{\ell,j}^{(k)}(\alpha_i)$ for each corrupted party P_j on behalf of D.

4. $\mathcal{S}$ sends the polynomials $\{g_{\ell,j}^{(k)}\}_{\ell\in[L'],k\in[m]}$ to each corrupted P_j.
5. $\mathcal{S}$ initializes a set $\mathcal{M}$ to $\emptyset$. For each $P_i \in \mathcal{P}$:
   - If P_i is honest, when P_i receives his column polynomials, $\mathcal{S}$ broadcasts OK_i on behalf of P_i. Then $\mathcal{S}$ delivers an initialization request to $\mathcal{F}_{\mathsf{APICP}}(P_i, D)$ on behalf of P_i and emulates $\mathcal{F}_{\mathsf{APICP}}(P_i, D)$ to deliver the output to D. When D receives the output and OK_i, $\mathcal{S}$ includes P_i into $\mathcal{M}$.
   - If P_i is corrupted, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{APICP}}(P_i, D)$ to receive $(\mathsf{Init}, \mathsf{APICP}, T, (g_{*,i}^{(1)}, \ldots, g_{*,i}^{(m)}))$ from P_i. If OK_i is received from P_i, $\mathcal{S}$ checks if $g_{*,i}^{(k)}(y) = (g_{1,i}^{(k)}, \ldots, g_{\ell,i}^{(k)})$ for each $k \in [m]$. If so, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{APICP}}(P_i, D)$ to send an output $(P_i, \mathsf{APICP}, (g_{*,i}^{(1)}, \ldots, g_{*,i}^{(m)}))$ to D and includes P_i into $\mathcal{M}$ when D receives the output and OK_i.
6. When $|\mathcal{M}| = 2t + 1$, $\mathcal{S}$ broadcasts $\mathcal{M}$ on behalf of D.
7. For each honest P_j, $\mathcal{S}$ waits until P_j receives $\mathcal{M}$ and $\{\mathsf{OK}_i\}_{P_i\in\mathcal{M}}$ and then begins the simulation of P_j in the next phase.

Figure 27: Part-(1/4) of the simulator for the $\mathcal{F}_{\mathsf{ACSS}}$ when D is honest

Simulator $\mathcal{S}$

Verification Phase

1. $\mathcal{S}$ initializes a set $\mathcal{R} = \{(0, \boldsymbol{o})\}$ where $\boldsymbol{o}$ is a vector of L' bivariate polynomials that maps any $(x, y) \in \mathbb{F}^2$ to 0. Each element that will be included into $\mathcal{R}$ is of form $(r, \boldsymbol{F}) \in \mathbb{F} \times (\mathbb{F}[X,Y])^{L'}$.
2. For each honest $P_i \in \mathcal{P}$:
   (1). When P_i receives the column polynomials from D, $\mathcal{S}$ broadcasts a random $r_i \in \mathbb{F}$ on behalf of P_i.
   (2). For each honest party P_j and $P_h \in \mathcal{M}$, when P_j receives r_i, $\mathcal{S}$ sends $(\mathsf{Request}, \mathsf{APICP}, P_i, (r_i, r_i^2, \ldots, r_i^m))$ to $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ on behalf of P_j.
   (3). For each honest $P_h \in \mathcal{M}$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ to deliver an output to P_i. For each corrupted $P_h \in \mathcal{M}$, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$.
   (4). When P_i receives the output from $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ for all $P_h \in \mathcal{M}$, $\mathcal{S}$ considers that P_i accepts his $\{g_{\ell,i}^{(k)}\}_{k\in[m],\ell\in[L']}$ and begins the simulation of P_i in the next phase.
3. For each corrupted $P_i \in \mathcal{P}$:
   (1). For each honest P_j and $P_h \in \mathcal{M}$, if $\mathcal{S}$ receives r_i on behalf of P_j, $\mathcal{S}$ sends $(\mathsf{Request}, \mathsf{APICP}, P_i, (r_i, r_i^2, \ldots, r_i^m))$ to $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ on behalf of P_j.
   (2). For each corrupted $P_h \in \mathcal{M}$, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$.
   (3). For each honest $P_h \in \mathcal{M}$,
      - If $(r_i, \boldsymbol{F}) \in \mathcal{R}$ for some $\boldsymbol{F} = (F_1(x,y), \ldots F_{L'}(x,y)) \in (\mathbb{F}[X,Y])^{L'}$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ to send $\boldsymbol{g}_{*,h}$ to P_i, where $\boldsymbol{g}_{*,h} = (g_{1,h}(y), \ldots, g_{L',h}(y))$ with each $g_{\ell,h}(y) = F_\ell(\alpha_h, y)$.
      - Otherwise, $\mathcal{S}$ samples a random vector of L' degree-$(t, 2t)$ polynomial $\boldsymbol{F} = (F_1(x,y), \ldots, F_{L'}(x,y))$ with $F_\ell(\alpha_j, y) = \sum_{k=1}^m r_i^k \cdot g_{\ell,j}^{(k)}(y)$ and $F_\ell(x, \alpha_j) = \sum_{k=1}^m r_i^k \cdot f_{\ell,j}^{(k)}(x)$ for each corrupted P_j and $\ell \in [L']$. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ to send $\boldsymbol{g}_{*,h}$ to P_i for the corresponding $\boldsymbol{g}_{*,h}$ and includes $(r_i, \boldsymbol{F})$ into $\mathcal{R}$.

Figure 28: Part-(2/4) of the simulator for the $\mathcal{F}_{\mathsf{ACSS}}$ when D is honest

Simulator $\mathcal{S}$

Authentication Phase

1. For each $P_i \in \mathcal{P}$ and $P_v \in \mathcal{P}$:
   (1). $\mathcal{S}$ follows the protocol to send requests to $\mathcal{F}_{\mathsf{RandShare}}, \mathcal{F}^0_{\mathsf{RandShare}}$ on behalf of honest parties. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{RandShare}}, \mathcal{F}^0_{\mathsf{RandShare}}$ to receive corrupted parties' shares of $[\boldsymbol{\mu}_{i\to v}]_t^i, \{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}, \{[r_u^{(k)}]_t^i\}_{k\in[m],u\in[t]}, \{[\mathsf{mask}_j]_t^i\}_{j\in[n]}$ from $\mathcal{A}$ and sends them to corrupted parties.
   (2). $\mathcal{S}$ follows the protocol to compute the corrupted parties' shares of $\{[\tau_{i\to v}^{(k)}]_{2t}^i\}_{k\in[m]}$.
   (3). If P_i is honest:
      1) For each honest $P_j \in \mathcal{P}$, when P_i receives P_j's shares of tags, $\mathcal{S}$ broadcasts a random $r_{i,j} \in \mathbb{F}$ on behalf of P_i.

2) For each corrupted P_j, $\mathcal{S}$ receives $\{\tilde{\tau}^{(k)}_{i\to v,j}\}_{k\in[m]}$ from P_j and broadcasts a random $r_{i,j} \in \mathbb{F}$ on behalf of P_i. Then $\mathcal{S}$ computes $\tilde{\tau}_{i\to v,j} = \sum_{k=1}^{m} r^k_{i,j} \cdot \tilde{\tau}^{(k)}_{i\to v,j}$ and P_j's share of $[\tau_{i\to v}]^i_{2t} = \sum_{k=1}^{m} r^k_{i,j} \cdot [\tau^{(k)}_{i\to v}]^i_{2t}$. If there exists some $k \in [m]$ such that $\tilde{\tau}^{(k)}_{i\to v,j}$ is not equal to P_j's share of $[\tau^{(k)}_{i\to v}]^i_{2t}$ but $\tilde{\tau}_{i\to v,j}$ is equal to P_j's share of $[\tau_{i\to v}]^i_{2t}$, $\mathcal{S}$ aborts the simulation.

If P_i is corrupted:

1) $\mathcal{S}$ randomly samples honest parties' shares of $\{[\tau^{(k)}_{i\to v}]^i_{2t}\}_{k\in[m]}$ based on the corrupted parties shares.
2) For each honest P_j, $\mathcal{S}$ sends P_j's shares of $\{[\tau^{(k)}_{i\to v}]^i_{2t}\}_{k\in[m]}$ to P_i on behalf of P_j.
3) $\mathcal{S}$ receives $r_{i,j}$ from P_i.

(4). For each $P_j \in \mathcal{P}$:

1) For each P_α and $P_h \in \mathcal{M}$, after each honest party receives $r_{i,j}$, $\mathcal{S}$ sends $(\mathsf{Request}, \mathsf{APICP}, P_\alpha, (r_{i,j}, r^2_{i,j} \dots, r^m_{i,j}))$ to $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ on behalf of this honest party.
2) For each honest P_α and $P_h \in \mathcal{M}$:
   - If P_h is honest, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ to deliver an output to P_α.
   - If P_h is corrupted, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$.
3) For each corrupted P_α and $P_h \in \mathcal{M}$:
   - If P_h is corrupted, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$.
   - If P_h is honest:
     – If $(r_{i,j}, \boldsymbol{F}) \in \mathcal{R}$ for some $\boldsymbol{F} = (F_1(x, y), \dots F_{L'}(x, y)) \in (\mathbb{F}[X, Y])^{L'}$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ to send $\boldsymbol{g}_{*,h}$ to P_α, where $\boldsymbol{g}_{*,h} = (g_{1,h}(y), \dots, g_{L',h}(y))$ with each $g_{\ell,h}(y) = F_\ell(\alpha_h, y)$.
     – Otherwise, $\mathcal{S}$ samples a random vector of L' degree-$(t, 2t)$ polynomial $\boldsymbol{F} = (F_1(x, y), \dots, F_{L'}(x, y))$ with $F_\ell(\alpha_j, y) = \sum_{k=1}^{m} r^k_{i,j} \cdot g^{(k)}_{\ell,j}(y)$ and $F_\ell(x, \alpha_j) = \sum_{k=1}^{m} r^k_{i,j} \cdot f^{(k)}_{\ell,j}(x)$ for each corrupted P_j and $\ell \in [L']$. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ to send $\boldsymbol{g}_{*,h}$ to P_α for the corresponding $\boldsymbol{g}_{*,h}$ and includes $(r_{i,j}, \boldsymbol{F})$ into $\mathcal{R}$.
4) $\mathcal{S}$ follows the protocol to compute corrupted parties' shares of $[\gamma_{i\to v,j}]^i_t$. If P_i is corrupted, $\mathcal{S}$ computes P_j's share $\tau_{i\to v,j}$ and randomly samples the honest parties' shares of $[\gamma_{i\to v,j}]^i_t$ based on the corrupted parties' shares and P_j's share.
5) If P_i is corrupted, for each honest P_α, $\mathcal{S}$ sends P_α's share of $[\gamma_{i\to v,j}]^i_t$ and $(\mathsf{Request}, \mathsf{privRec}, P_i)$ to $\mathcal{F}_{\mathsf{privRec}}$ on behalf of P_α. If P_i is honest, for each honest P_α, $\mathcal{S}$ sends a request to $\mathcal{F}_{\mathsf{privRec}}$ on behalf of P_α.
6) If $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{privRec}}$ to send the corrupted parties shares of $[\gamma_{i\to v,j}]^i_t$ to $\mathcal{A}$. If P_i is corrupted, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{privRec}}$ to send the whole sharing $[\gamma_{i\to v,j}]^i_t$ to P_i. If P_i is honest, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{privRec}}$ to deliver an output sharing $[\gamma_{i\to v,j}]^i_t$ to P_i.

(5). If P_i is honest:

1) For each honest P_j, when P_i receives the sharing $[\gamma_{i\to v,j}]^i_t$, $\mathcal{S}$ considers that P_i accepts P_j's shares $\{\tau^{(k)}_{i\to v,j}\}_{k\in[m]}$.
2) For each corrupted P_j, $\mathcal{S}$ considers that P_i accepts P_j's shares $\{\tau^{(k)}_{i\to v,j}\}_{k\in[m]}$ if P_j's share of $[\gamma_{i\to v,j}]^i_t$ is equal to $\tau_{i\to v,j}$.
3) When P_i accepts $2t + 1$ different parties' shares, $\mathcal{S}$ reconstructs $\{\tau^{(k)}_{i\to v}\}_{k\in[m]}$.

2. For each honest P_i, $\mathcal{S}$ broadcasts Tag_i on behalf of P_i after reconstructing $\{\tau^{(k)}_{i\to v}\}_{k\in[m],v\in[n]}$. Then $\mathcal{S}$ follows the protocol to prepare the set $\mathcal{W}$ and broadcasts it on behalf of D.
3. For each honest party P_j, $\mathcal{S}$ waits until P_j receives $\mathcal{W}$ and $\{\mathsf{Tag}_i\}_{P_i\in\mathcal{W}}$ and then begins the simulation of P_j in the next phase.

Figure 29: Part-(3/4) of the simulator for the $\mathcal{F}_{\mathsf{ACSS}}$ when D is honest

Simulator $\mathcal{S}$

Completion Phase

Reconstructing row polynomials:

For each $P_v \in \mathcal{P}$:

- If P_v is honest:
  1. For each honest party, $\mathcal{S}$ follows the protocol to deliver his shares of $[\boldsymbol{\mu}_{i\to v}]^i_t$, $\{[\nu^{(k)}_{i\to v}]^i_t\}_{k\in[m]}$ and $(\mathsf{Request}, \mathsf{privRec}, P_v)$ for each $P_i \in \mathcal{W}$ to $\mathcal{F}_{\mathsf{privRec}}$.

2. For each corrupted $P_i \in \mathcal{W}$:
   (1). $\mathcal{S}$ receives $\{\tilde{g}^{(k)}_{\ell,i}(\alpha_v)\}_{k\in[m],\ell\in[L']}$ from P_i.
   (2). $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{privRec}}$ to deliver the whole sharings $[\boldsymbol{\mu}_{i\to v}]^i_t, [\nu^{(1)}_{i\to v}]^i_t, \ldots, [\nu^{(m)}_{i\to v}]^i_t$ to P_v and sends the corrupted parties' shares of them to $\mathcal{A}$.
   (3). $\mathcal{S}$ randomly samples $r_{i\to v} \in \mathbb{F}$ and sends it to P_i on behalf of P_v.
   (4). $\mathcal{S}$ receives $\tilde{\tau}_{i\to v}$ and $\{\tilde{g}_{\ell,i}\}_{\ell\in[L']}$ from P_i.
3. For each $P_i \in \mathcal{W}$:
   – If P_i is honest, when P_v receives P_i's $\{g_{\ell,i}\}_{\ell\in[L']}$ and $\tau_{i\to v}$, $\mathcal{S}$ considers that P_v accepts P_i's $\{g^{(k)}_{\ell,i}(\alpha_v)\}_{k\in[m],\ell\in[L']}$.
   – If P_i is corrupted, $\mathcal{S}$ considers that P_v accepts P_i's $\{g^{(k)}_{\ell,i}(\alpha_v)\}_{k\in[m],\ell\in[L']}$ if $\tilde{g}_{\ell,i}(y) = \sum_{k=1}^m r^k_{i\to v} g^{(k)}_{\ell,i}(y)$ for all $\ell \in [L']$, $\tilde{\tau}_{i\to v} = \sum_{k=1}^m r^k_{i\to v}\tau^{(k)}_{i\to v}$ and $\tilde{g}^{(k)}_{\ell,i}(\alpha_v) = g^{(k)}_{\ell,i}(\alpha_v)$ for each $k \in [m]$ and $\ell \in [L']$.
4. When P_v accepts $t+1$ different P_i's $\{g^{(k)}_{\ell,i}(\alpha_v)\}_{k\in[m],\ell\in[L']}$, $\mathcal{S}$ considers that P_v gets his $\{f^{(k)}_{\ell,v}\}_{k\in[m],\ell\in[L']}$.

- If P_v is corrupted:
  1. For each honest party $\mathcal{S}$ follows the protocol to deliver his shares of $[\boldsymbol{\mu}_{i\to v}]^i_t, \{[\nu^{(k)}_{i\to v}]^i_t\}_{k\in[m]}$ and $(\mathsf{Request}, \mathsf{privRec}, P_v)$ for each $P_i \in \mathcal{W}$ to $\mathcal{F}_{\mathsf{privRec}}$.
  2. For each corrupted $P_i \in \mathcal{W}$:
     (1). $\mathcal{S}$ randomly samples $\boldsymbol{\mu}_{i\to v} \in \mathbb{F}^L$ and computes $\nu^{(k)}_{i\to v} = \tau^{(k)}_{i\to v} - \boldsymbol{g}^{(k)}_{*,i} \cdot \nu_{i\to v}$.
     (2). $\mathcal{S}$ randomly samples honest parties' shares of $[\boldsymbol{\mu}_{i\to v}]^i_t, \{[\nu^{(k)}_{i\to v}]^i_t\}_{k\in[m]}$ based on the corrupted parties' shares and the secrets $\boldsymbol{\mu}_{i\to v}, \{\nu^{(k)}_{i\to v}\}_{k\in[m]}$.
     (3). $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{privRec}}$ to send the whole sharings $[\boldsymbol{\mu}_{i\to v}]^i_t, \{[\nu^{(k)}_{i\to v}]^i_t\}_{k\in[m]}$ to P_v.
  3. For each honest $P_i \in \mathcal{W}$:
     (1). $\mathcal{S}$ sends $\{f^{(k)}_{\ell,v}(\alpha_i)\}_{k\in[m],\ell\in[L']}$ to P_v on behalf of P_i.
     (2). $\mathcal{S}$ randomly samples honest parties' shares of $[\boldsymbol{\mu}_{i\to v}]^i_t, \{[\nu^{(k)}_{i\to v}]^i_t\}_{k\in[m]}$ based on the corrupted parties' shares.
     (3). $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{privRec}}$ to send the whole sharings $[\boldsymbol{\mu}_{i\to v}]^i_t, \{[\nu^{(k)}_{i\to v}]^i_t\}_{k\in[m]}$ to P_v.
     (4). $\mathcal{S}$ receives $r_{i\to v}$ from P_v.
       – If $(r_{i\to v}, \boldsymbol{F}) \in \mathcal{R}$ for some $\boldsymbol{F} = (F_1(x,y), \ldots F_{L'}(x,y)) \in (\mathbb{F}[X,Y])^{L'}$, $\mathcal{S}$ sends $\boldsymbol{g}_{*,i} = (g_{1,i}(y), \ldots, g_{L',i}(y))$ with each $g_{\ell,i}(y) = F_\ell(\alpha_i, y)$ to P_v.
       – Otherwise, $\mathcal{S}$ samples a random vector of L' degree-$(t,2t)$ polynomial $\boldsymbol{F} = (F_1(x,y), \ldots, F_{L'}(x,y))$ with $F_\ell(\alpha_j, y) = \sum_{k=1}^m r^k_{i\to v} \cdot g^{(k)}_{\ell,j}(y)$ and $F_\ell(x, \alpha_j) = \sum_{k=1}^m r^k_{i\to v} \cdot f^{(k)}_{\ell,j}(x)$ for each corrupted P_j and $\ell \in [L']$. Then $\mathcal{S}$ sends $\boldsymbol{g}_{*,i}$ to P_v and includes $(r_{i\to v}, \boldsymbol{F})$ into $\mathcal{R}$.
     (5). $\mathcal{S}$ computes $\tau_{i\to v} = \boldsymbol{g}_{*,i} \cdot \boldsymbol{\mu}_{i\to v} + \sum_{k=1}^m r^k_{i\to v}\nu^{(k)}_{i\to v}$ and sends it to P_v on behalf of P_i.

Reconstructing column polynomials:

For each $P_w \in \mathcal{P}$:

- If P_w is honest:
  1. For each honest $P_v \in \mathcal{P}$, when P_w receives $\{f^{(k)}_{\ell,v}(\alpha_w)\}_{k\in[m],\ell\in[L']}$, $\mathcal{S}$ randomly broadcasts $r_{v\to w} \in \mathbb{F}$ on behalf of P_w.
  2. For each corrupted $P_v \in \mathcal{P}$:
     (1). $\mathcal{S}$ receives $\{\tilde{f}^{(k)}_{\ell,v}(\alpha_w)\}_{k\in[m],\ell\in[L']}$ from P_v.
     (2). $\mathcal{S}$ randomly broadcasts $r_{v\to w} \in \mathbb{F}$ on behalf of P_w.
     (3). For each corrupted $P_i \in \mathcal{W}$, $\mathcal{S}$ receives $\tilde{\tau}_{i\to w}$ and $\{\tilde{g}_{\ell,i}\}_{\ell\in[L']}$ from P_i.
  3. For each $P_v \in \mathcal{P}$:
     (1). For each $P_i \in \mathcal{W}$:
       – If P_i is honest, when P_w receives P_i's $\{g_{\ell,i}\}_{\ell\in[L']}$ and $\tau_{i\to w}$, $\mathcal{S}$ considers that P_w accepts P_i's $\{g_{\ell,i}\}_{\ell\in[L']}$.
       – If P_i is corrupted, $\mathcal{S}$ considers that P_w accepts P_i's $\{g_{\ell,i}\}_{\ell\in[L']}$ if $\tilde{g}_{\ell,i}(y) = \sum_{k=1}^m r^k_{v\to w} g^{(k)}_{\ell,i}(y)$ for all $\ell \in [L']$ and $\tilde{\tau}_{i\to w} = \sum_{k=1}^m r^k_{v\to w}\tau^{(k)}_{i\to w}$.
     (2) When P_w accepts $t+1$ different P_i's $\{g_{\ell,i}\}_{\ell\in[L']}$,
       – If P_v is honest, $\mathcal{S}$ considers that P_w accepts P_v's $\{f^{(k)}_{\ell,v}(\alpha_w)\}_{k\in[m],\ell\in[L']}$.

– If P_v is corrupted, $\mathcal{S}$ considers that P_w accepts P_v's $\{f^{(k)}_{\ell,v}(\alpha_w)\}_{k\in[m],\ell\in[L']}$ if $f^{(k)}_{\ell,v}(\alpha_w) = \tilde{f}^{(k)}_{\ell,v}(\alpha_w)$ for each $k \in [m]$ and $\ell \in [L']$.

4. When P_w accepts $2t+1$ different P_v's $\{f^{(k)}_{\ell,v}(\alpha_w)\}_{k\in[m],\ell\in[L']}$, $\mathcal{S}$ allows $\mathcal{F}_{\mathsf{ACSS}}$ to send the output to P_v.

- If P_w is corrupted:
  1. For each honest $P_v \in \mathcal{P}$:
     (1). $\mathcal{S}$ sends $\{g^{(k)}_{\ell,w}(\alpha_v)\}_{k\in[m],\ell\in[L']}$ to P_w on behalf of P_v.
     (2). $\mathcal{S}$ receives $r_{v\to w}$ from P_w.
     (3). For each honest $P_i \in \mathcal{W}$:
     – If $(r_{v\to w}, \boldsymbol{F}) \in \mathcal{R}$ for some $\boldsymbol{F} = (F_1(x,y), \dots F_{L'}(x,y)) \in (\mathbb{F}[X,Y])^{L'}$, $\mathcal{S}$ sends $\{g_{\ell,i}\}_{\ell\in[L']}$ with each $g_{\ell,i}(y) = F_\ell(\alpha_i, y)$ to P_w.
     – Otherwise, $\mathcal{S}$ samples a random vector of L' degree-$(t,2t)$ polynomial $\boldsymbol{F} = (F_1(x,y), \dots, F_{L'}(x,y))$ with $F_\ell(\alpha_j, y) = \sum_{k=1}^m r^k_{v\to w} \cdot g^{(k)}_{\ell,j}(y)$ and $F_\ell(x, \alpha_j) = \sum_{k=1}^m r^k_{v\to w} \cdot f^{(k)}_{\ell,j}(x)$ for each corrupted P_j and $\ell \in [L']$. Then $\mathcal{S}$ sends $\{g_{\ell,i}\}_{\ell\in[L']}$ to P_w and includes $(r_{v\to w}, \boldsymbol{F})$ into $\mathcal{R}$.
     (4). For each honest $P_i \in \mathcal{W}$, $\mathcal{S}$ computes $\tau_{i\to w} = \boldsymbol{g}_{*,i} \cdot \boldsymbol{\mu}_{i\to w} + \sum_{k=1}^m r^k_{v\to w}\nu^{(k)}_{i\to w}$ and sends it to P_w on behalf of P_i, where $\boldsymbol{g}_{*,i} = (g_{1,i}(y), \dots, g_{L',i}(y))$.

Figure 30: Part-(4/4) of the simulator for the $\mathcal{F}_{\mathsf{ACSS}}$ when D is honest

Hybrid arguments:

$\mathbf{Hyb}_0$: In this hybrid, $\mathcal{S}$ receives honest parties' inputs and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, during the Sharing Phase, $\mathcal{S}$ samples corrupted parties' column polynomials and row polynomials first and then samples the bivariate polynomials based on D's input and corrupted parties' polynomials. We only change the order of generating bivariate polynomials and corrupted parties' polynomials, which doesn't change the output distribution. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

$\mathbf{Hyb}_2$: In this hybrid, $\mathcal{S}$ initializes a set $\mathcal{R} = \{(0, \boldsymbol{o})\}$ to record each random value r and the corresponding bivariate polynomials computed by $F_\ell(x,y) = \sum_{k=1}^m r^k_i \cdot F^{(k)}_\ell(x,y)$. Each time $\mathcal{S}$ needs to compute a linear combination of some honest parties' column polynomials, we regard $\mathcal{S}$ computes a linear combination of the bivariate polynomials generated in the Sharing Phase and then take the corresponding column polynomial as the result. Each time $\mathcal{S}$ needs to compute a linear combination of bivariate polynomials, if r_i is recorded before, $\mathcal{S}$ uses $\{F_\ell(x,y)\}_{\ell\in[L']}$ recorded before instead of computing them again. Initializing a set doesn't change the output distribution. Thus, $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ have the same output distribution.

Since $\mathcal{S}$ computes each F_ℓ by computing the linear combination of m' degree-$(t,2t)$ bivariate polynomials and $m - m' = T + T'$ completely random degree-$(t,2t)$ bivariate polynomials. We claim that doing at most $m - m' = T + T'$ times of linear combinations corresponding to $T + T'$ different random values $r_1, \dots, r_{T+T'} \neq 0$. Their corresponding vectors $\boldsymbol{F} = (F_1, \dots, F_\ell)$ of polynomials, denoted by $\boldsymbol{F}_1, \dots, \boldsymbol{F}_{T+T'}$, are random with the corrupted parties' row and column polynomials of each bivariate polynomial F_ℓ given. We prove this fact through hybrid arguments $\mathbf{Hyb}_{2.0}, \dots, \mathbf{Hyb}_{2.(T+T'+1)}$.

Note that we need $(2t+1)(t+1)$ evaluation points to determine each degree-$(t,2t)$ bivariate polynomial. Given corrupted parties' column and row polynomials, we have $t'(2t+1) + t'(t+1) - t'^2 < (2t+1)(t+1)$ evaluation points. Let $\delta = L' \cdot ((2t+1)(t+1) - t'(2t+1) - t'(t+1) + t'^2)$, we need another a randomness $r' \in \mathbb{F}^\delta$ to determine each $\boldsymbol{F}$. Here r' determines the vector of L' outputs of $\boldsymbol{F}$ with L' fixed inputs. Then, we can reconstruct $\boldsymbol{F}$ by r' and the polynomials of corrupted parties. Picking random $\mathbb{F}$ with column and row polynomials fixed is the same as randomly sampling $r' \in \mathbb{F}^\delta$. Suppose that we need randomness $r'_1, \dots, r'_{T+T'}$ to determine $\boldsymbol{F}_{(1)}, \dots, \boldsymbol{F}_{(T+T')}$. For a part added in $r'_1, \dots, r'_{T+T'}$, we let $\hat{r}_1, \dots, \hat{r}_{T+T'}$ be the randomness r' of the linear combination of the $T+T'$ vectors of completely random bivariate polynomials corresponds to $r_1, \dots, r_{T+T'}$. Besides, we let the randomness $r' \in \mathbb{F}^\delta$ of the $T+T'$ vectors of ℓ completely random degree-$(t,2t)$ bivariate polynomials be $\tilde{r}^{(1)}, \dots, \tilde{r}^{(T+T')}$, then $(\tilde{r}^{(1)}, \dots, \tilde{r}^{(T+T')})$ is completely random in $\mathbb{F}^{\delta(T+T')}$.

$\mathbf{Hyb}_{2.0}$: $\mathcal{S}$ computes $r'_1, \dots, r_{T+T'}$ honestly, uses them to compute $\boldsymbol{F}_1, \dots, \boldsymbol{F}_{T+T'}$ and outputs them.

$\mathbf{Hyb}_{2.1}$: In this hybrid, for the first time $\mathcal{S}$ needs to compute a linear combination of bivariate polynomials, he samples a random $\hat{r}_1 \in \mathbb{F}^\delta$ as the randomness r' of the linear combination of the $T+T'$ vectors of completely random bivariate polynomials. Then, $\mathcal{S}$ computes r'_1 with $\hat{r}'_1$ and the bivariate polynomials $\{F_\ell^{(k)}\}_{k\in[m'],\ell\in[L']}$. $\mathcal{S}$ then determines $\boldsymbol{F}_1$ using r'_1. $\hat{r}_1$ is computed by:

$$\hat{r}^{(1)} = \begin{pmatrix} r_1^{m'+1} & r_1^{m'+2} & \cdots & r_1^m \end{pmatrix} \cdot \begin{pmatrix} \tilde{r}^{(1)} \\ \tilde{r}^{(2)} \\ \vdots \\ \tilde{r}^{(T+T')} \end{pmatrix}.$$

Since $r_1 \neq 0$, $(\tilde{r}^{(1)}, \dots, \tilde{r}^{(T+T')})$ is completely random in $\mathbb{F}^{\delta(T+T')}$, $\hat{r}_1$ is completely random in $\mathbb{F}$. Thus, $\mathbf{Hyb}_{2.1}$ and $\mathbf{Hyb}_{2.0}$ have the same output distribution.

$\mathbf{Hyb}_{2.2}$: In this hybrid, for the second time $\mathcal{S}$ needs to compute a linear combination of bivariate polynomials, he samples a random $\hat{r}_2 \in \mathbb{F}^\delta$ as the randomness r' of the linear combination of the $T+T'$ vectors of completely random bivariate polynomials. Then, $\mathcal{S}$ computes r'_2 with $\hat{r}'_2$ and the bivariate polynomials $\{F_\ell^{(k)}\}_{k\in[m'],\ell\in[L']}$. $\mathcal{S}$ then determines $\boldsymbol{F}_2$ using r'_2. Given $\hat{r}_1$, $\hat{r}_2$ is computed by:

$$\begin{aligned}\hat{r}^{(2)} &= \begin{pmatrix} r_2^{m'+1} & r_2^{m'+2} & \cdots & r_2^m \end{pmatrix} \cdot \begin{pmatrix} \tilde{r}^{(1)} \\ \tilde{r}^{(2)} \\ \vdots \\ \tilde{r}^{(T+T')} \end{pmatrix} \\ &= \begin{pmatrix} \frac{r_2^{m'+1}}{r_1^{m'+1}} & \frac{r_2^{m'+2} - r_2^{m'+1} r_1^{m'+2}}{r_1^{m'+1}} & \cdots & \frac{r_2^{m} - r_2^{m'+1} r_1^{m}}{r_1^{m'+1}} \end{pmatrix} \cdot \begin{pmatrix} \hat{r}_1 \\ \tilde{r}^{(2)} \\ \vdots \\ \tilde{r}^{(T+T')} \end{pmatrix}.\end{aligned}$$

Since $r_1 \neq r_2$,

$$\mathrm{Det} \begin{pmatrix} r_1^{m'+1} & r_1^{m'+2} \\ r_2^{m'+1} & r_2^{m'+2} \end{pmatrix} \neq 0,$$

we have $\frac{r_2^{m'+2} - r_2^{m'+1} r_1^{m'+2}}{r_1^{m'+1}} \neq 0$. Since $(\tilde{r}^{(1)}, \dots, \tilde{r}^{(T+T')})$ is completely random in $\mathbb{F}^{\delta(T+T')}$, given $\hat{r}_1$, $\mathcal{S}$ can randomly sample $\tilde{r}^{(2)}, \dots, \tilde{r}^{(T+T')}$ and compute $\tilde{r}^{(1)}$ based on $\hat{r}_1$. This shows that $\tilde{r}^{(2)}$ is completely random in $\mathbb{F}$. Note that $\frac{r_2^{m'+2} - r_2^{m'+1} r_1^{m'+2}}{r_1^{m'+1}} \cdot \tilde{r}^{(2)}$ is added to $\hat{r}_2$, $\hat{r}_2$ is also completely random in $\mathbb{F}$. Thus, $\mathbf{Hyb}_{2.2}$ and $\mathbf{Hyb}_{2.1}$ have the same output distribution.

$\mathbf{Hyb}_{2.i} (i \in [3, T+T'])$: In this hybrid, for the i-th time $\mathcal{S}$ needs to compute a linear combination of bivariate polynomials, he samples a random $\hat{r}_i \in \mathbb{F}^\delta$ as the randomness r' of the linear combination of the $T+T'$ vectors of completely random bivariate polynomials. Then, $\mathcal{S}$ computes r'_i with $\hat{r}'_i$ and the bivariate polynomials $\{F_\ell^{(k)}\}_{k\in[m'],\ell\in[L']}$. $\mathcal{S}$ then determines $\boldsymbol{F}_i$ using r'_i. Given $\hat{r}_1, \dots, \hat{r}_{i-1}$, $\mathcal{S}$ can sample $\tilde{r}^{(i)}, \dots, \tilde{r}^{(T+T')}$ randomly and compute $\tilde{r}^{(1)}, \dots, \tilde{r}^{(i-1)}$ based on $\hat{r}_1, \dots, \hat{r}_{i-1}$. This shows that $\tilde{r}^{(i)}$ is completely random in $\mathbb{F}$. Like in $\mathbf{Hyb}_{2.2}$, $\hat{r}_i$ can be computed by a linear combination of $\hat{r}_1, \dots, \hat{r}_{i-1}$ and $\tilde{r}^{(i)}, \dots, \tilde{r}^{(T+T')}$, we only need to prove that the coefficient of $\tilde{r}^{(i)}$ is nonzero. Since

$$\begin{pmatrix} \hat{r}^{(1)} \\ \hat{r}^{(2)} \\ \vdots \\ \hat{r}^{(i)} \end{pmatrix} = \begin{pmatrix} r_1^{m'+1} & r_1^{m'+2} & \cdots & r_1^m \\ r_2^{m'+1} & r_2^{m'+2} & \cdots & r_2^m \\ \vdots & \vdots & \ddots & \vdots \\ r_i^{m'+1} & r_i^{m'+2} & \cdots & r_i^m \end{pmatrix} \cdot \begin{pmatrix} \tilde{r}^{(1)} \\ \tilde{r}^{(2)} \\ \vdots \\ \tilde{r}^{(T+T')} \end{pmatrix},$$

the coefficient of $\tilde{r}^{(i)}$ is nonzero if and only if

$$\mathrm{Det}\begin{pmatrix} r_1^{m'+1} & r_1^{m'+2} & \cdots & r_1^{m'+i} \\ r_2^{m'+1} & r_2^{m'+2} & \cdots & r_2^{m'+i} \\ \vdots & \vdots & \ddots & \vdots \\ r_i^{m'+1} & r_i^{m'+2} & \cdots & r_i^{m'+1} \end{pmatrix} \neq 0.$$

This is guaranteed because $r_1, \ldots, r_{T+T'}$ are all distinct. Hence, $\hat{r}_i$ is completely random in $\mathbb{F}$. Thus, $\mathbf{Hyb}_{2.i}$ and $\mathbf{Hyb}_{2.(i-1)}$ have the same output distribution.

$\mathbf{Hyb}_{2.(T+T'+1)}$: In this hybrid, $\mathcal{S}$ samples random vectors of L degree-$(t, 2t)$ polynomials as $\boldsymbol{F}_1, \ldots, \boldsymbol{F}_{T+T'}$ based on corrupted parties' polynomials instead of computing them based on $\hat{r}_1, \ldots, \hat{r}_{T+T'}$. Since $\hat{r}_1, \ldots, \hat{r}_{T+T'}$ are all randomly sampled in $\mathbf{Hyb}_{2,(T+T')}$, $r'_1, \ldots, r'_{T+T'}$ are also completely random. Thus, given corrupted parties' polynomials, $\boldsymbol{F}_1, \ldots, \boldsymbol{F}_{T+T'}$ are random, as in $\mathbf{Hyb}_{2,(T+T'+1)}$. Thus, $\mathbf{Hyb}_{2.(T+T'+1)}$ and $\mathbf{Hyb}_{2.(T+T')}$ have the same output distribution.

Then we complete the proof that with no more than $T + T'$ times of linear combinations with respect to different $r \neq 0$, we can sample random $\boldsymbol{F}$ based on corrupted parties' polynomials.

$\mathbf{Hyb}_3$: In this hybrid, during the Verification Phase, $\mathcal{S}$ doesn't compute the output of $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ for corrupted P_i if P_h is honest. Instead, he checks if r_i is recorded in $\mathcal{R}$. If r_i is recorded before, $\mathcal{S}$ only needs to use $\{F_\ell(x, y)\}_{\ell \in [L']}$ to compute the output of $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$. Otherwise $\mathcal{S}$ chooses random bivariate degree-$(t, 2t)$ polynomials that are consistent with corrupted parties' polynomials as $\{F_\ell(x, y)\}_{\ell \in [L']}$ corresponding to r_i and includes them into $\mathcal{R}$. Then $\mathcal{S}$ computes the output of $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ with the bivariate polynomials. As we have argued in $\mathbf{Hyb}_{2.0}, \ldots, \mathbf{Hyb}_{2.(T+T'+1)}$, with at most $n < T + T'$ revelation requests for each $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ in the Verification Phase, if some r_i hasn't been recorded before, each F_ℓ is random with corrupted parties' column and row polynomials given, as what $\mathcal{S}$ samples in $\mathbf{Hyb}_3$. Thus, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ have the same output distribution.

$\mathbf{Hyb}_4$: In this hybrid, during the Authentication Phase, for each honest P_i and corrupted P_j, $\mathcal{S}$ aborts the simulation if P_j's shares of $[\tau^{(k)}_{i \to v}]^i_{2t}$ are not correctly sent for some $P_v \in \mathcal{P}$ and $k \in [m]$ but $\tau_{i \to v, j}$ is correct. This happens only when the random $r_{i,j}$ satisfies $\sum_{k=1}^m r_{i,j}^k \cdot \tau^{(k)}_{i \to v, j} = \tau_{i \to v, j}$ where each $\tau^{(k)}_{i \to v, j}$ is the share of $[\tau^{(k)}_{i \to v}]^i_{2t}$ received from P_j. Note that any non-zero polynomial $\sum_{k=1}^m \tau^{(k)}_{i \to v, j} x^k - \tau_{i \to v, j}$ has at most m roots in $\mathbb{F}$, and there are at most $t(2t + 1)n$ pairs of (P_i, P_j, P_v), so the output distribution only changes with probability

$$\epsilon_1 \leq \frac{mt(2t + 1)n}{|\mathbb{F}|} < \frac{m \cdot n^3}{2^\kappa},$$

which is negligible. Thus, the output distributions of $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ are statistically close.

$\mathbf{Hyb}_5$: In this hybrid, during the Authentication Phase, for each $P_v \in \mathcal{P}$ and corrupted P_i, $\mathcal{S}$ samples honest parties' shares of $\{[\tau^{(k)}_{i \to v}]^i_{2t}\}_{k \in [m]}$ based on corrupted parties' shares instead of computing them by himself. Besides, $\mathcal{S}$ doesn't sample the honest parties' shares of $\{[\nu^{(k)}_{i \to v}]^i_t\}_{k \in [m]}$. Instead, $\mathcal{S}$ computes them by

$$[\nu^{(k)}_{i \to v}]^i_t = [\tau^{(k)}_{i \to v}]^i_{2t} - [\boldsymbol{g}^{(k)}_{*,i}]^i_t \cdot [\boldsymbol{\mu}_{i \to v}]^i_t - \sum_{u=1}^{t} [\![\boldsymbol{e}_u]\!]^i_t \cdot [r^{(k)}_u]^i_t.$$

Since $[\![\boldsymbol{e}_u]\!]^i_t$ is a fixed packed secret sharing, we can denote its degree-t sharing polynomial by $f_u(x)$, where $f_u(\alpha_{-u}) = 1$ and $f_u(\alpha_j) = 0$ for each $j \in \{-1, -2, \ldots, -t, i\} \backslash \{-u\}$. Let the sharing polynomial of $[r^{(k)}_u]^i_t$ be $g_u(x)$, then $g_u(x)$ is a random degree-t polynomial with $\{g_u(\alpha_j)\}_{P_j \in \mathcal{C}}$ fixed. Hence, $g_u(\alpha_{-u})$ is random in $\mathbb{F}$. Thus, the sharing polynomial $h(x)$ of $\sum_{u=1}^t [\![\boldsymbol{e}_u]\!]^i_t \cdot [r^{(k)}_u]^i_t$ is a degree-$2t$ polynomial such that for each $u \in [t]$, $h(\alpha_{-u}) = \sum_{j=1}^t f_j(\alpha_{-u}) g_j(\alpha_{-u}) = f_u(\alpha_{-u}) g_u(\alpha_{-u}) = g_u(\alpha_{-u})$ is a random value in $\mathbb{F}$. Since honest parties' shares of $[r^{(k)}_u]^i_t$ are randomly sampled based on corrupted parties' shares and $g_u(\alpha_{-u})$, the honest parties' shares of $\sum_{u=1}^t [\![\boldsymbol{e}_u]\!]^i_t \cdot [r^{(k)}_u]^i_t$ are also random with the corrupted parties' shares and $\{h(\alpha_{-u})\}_{u \in [t]}$ given. Since each $h(\alpha_{-u})$ is also random, sampling the honest parties' shares of $\sum_{u=1}^t [\![\boldsymbol{e}_u]\!]^i_t \cdot [r^{(k)}_u]^i_t$ based on corrupted parties' shares gives the same output distribution as computing them. Besides, since $f_u(\alpha_i) = 0$ for all $u \in [t]$, the secret of the $2t$-sharing $\sum_{u=1}^{t} [\![\boldsymbol{e}_u]\!]_t^i \cdot [r_u^{(k)}]_t^i$ is 0, i.e. it can be regarded as a $[0]_{2t}^i$ added in $\{[\tau_{i\to v}^{(k)}]_t^i\}_{j\in[m]}$. Note that the honest parties' shares of $[\boldsymbol{\mu}_{i\to v}]_t^i$ $\{[\nu_{i\to v}^{(k)}]_t^i\}_{j\in[m]}$ are also randomly sampled based on corrupted parties' shares, $\{[\tau_{i\to v}^{(k)}]_t^i\}_{j\in[m]}$ is a random $2t$-sharing given corrupted parties' shares, so sampling the honest parties' shares of $\{[\tau_{i\to v}^{(k)}]_t^i\}_{j\in[m]}$ instead of computing them doesn't change the output distribution. Hence, we only change the order of sampling the honest parties' shares of $\{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}$ and $\{[\tau_{i\to v}^{(k)}]_{2t}^i\}_{k\in[m]}$. Thus, $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ have the same output distribution.

$\mathbf{Hyb}_6$: In this hybrid, during the Authentication Phase, for each $P_j \in \mathcal{P}$, each corrupted P_α and each honest $P_h \in \mathcal{M}$, $\mathcal{S}$ doesn't compute the output of $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$. Instead, $\mathcal{S}$ checks whether $r_{i,j}$ has been recorded in $\mathcal{R}$ before. If $r_{i,j}$ is recorded before, $\mathcal{S}$ only needs to use $\{F_\ell(x,y)\}_{\ell\in[L']}$ to compute the output of $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$. Otherwise $\mathcal{S}$ chooses random bivariate degree-$(t, 2t)$ polynomials that are consistent with corrupted parties' polynomials as $\{F_\ell(x,y)\}_{\ell\in[L']}$ corresponding to $r_{i,j}$ and includes them into $\mathcal{R}$. Then, $\mathcal{S}$ computes the output of $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ with the bivariate polynomials. As we have argued in $\mathbf{Hyb}_{2.0}, \ldots, \mathbf{Hyb}_{2.(T+T'+1)}$, with at most $n + n^3 = T < T + T'$ revelation requests for each $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ in the Verification Phase and the Authentication Phase, if some $r_{i,j}$ hasn't been recorded before, each F_ℓ is random with corrupted parties' column and row polynomials given, as what $\mathcal{S}$ samples in $\mathbf{Hyb}_6$. Thus, $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ have the same distribution.

$\mathbf{Hyb}_7$: In this hybrid, in the Authentication Phase, for each $P_v, P_j \in \mathcal{P}$ and corrupted P_i, $\mathcal{S}$ samples honest parties' shares of $[\gamma_{i\to v,j}]_t^i$ based on corrupted parties' shares instead of computing them by himself. Since honest parties' shares of $[\mathsf{mask}_j]_t^i$ are randomly sampled based on the corrupted parties' shares and are added in $[\gamma_{i\to v,j}]_t^i$ as a mask, the honest parties' shares of $[\gamma_{i\to v,j}]_t^i$ are also random based on the corrupted parties' shares. Thus, $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ have the same output distribution.

$\mathbf{Hyb}_8$: In this hybrid, during the Authentication Phase, for each honest P_i and P_j, $\mathcal{S}$ doesn't follow the protocol to check P_j's shares. Instead, $\mathcal{S}$ considers that P_i accepts P_j's shares when P_i receives the sharing $[\gamma_{i\to v,j}]_t^i$. Since D and P_j are both honest, P_j always sends correct shares, so P_i always accepts P_j's shares. Thus, $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ have the same output distribution.

$\mathbf{Hyb}_9$: In this hybrid, during the Completion Phase, for each honest P_v and corrupted $P_i \in \mathcal{W}$, $\mathcal{S}$ doesn't follow the protocol to check P_i's polynomials $\{\tilde{g}_{\ell,i}\}_{\ell\in[L']}$ and the tag $\tilde{\tau}_{i\to v}$. Instead, $\mathcal{S}$ checks whether the polynomials are consistent with what $\mathcal{S}$ generates in the Sharing Phase and whether $\tilde{\tau}_{i\to v}$ is consistent with the tags P_i gets in the Authentication Phase. This changes the output distribution only if P_i sends $\{\tilde{g}_{\ell,i}\}_{\ell\in[L']}$ and $\tilde{\tau}_{i\to v}$ different from $\{g_{\ell,i}\}_{\ell\in[L']}$ and $\tau_{i\to v}$ but still satisfying $\tilde{\tau}_{i\to v} = \tilde{\boldsymbol{g}}_{*,i} \cdot \boldsymbol{\mu}_{i\to v} + \sum_{k=1}^{m} r_{i\to v}^{k} \cdot \nu_{i\to v}^{(k)}$, where $\tilde{\boldsymbol{g}}_{*,i} = (\tilde{g}_{1,i}, \ldots, \tilde{g}_{L',i})$. Then we know that $(\tilde{\boldsymbol{g}}_{*,i} - \boldsymbol{g}_{*,i}) \cdot \boldsymbol{\mu}_{i\to v} - (\tilde{\tau}_{i\to v} - \tau_{i\to v}) = 0$. Since $\boldsymbol{\mu}_{i\to v}$ is random in $\mathbb{F}^L$ when P_i is corrupted and we don't need it for any computation before P_v receives P_i's polynomials and tag, $\mathcal{S}$ can randomly sample it after $\{\tilde{g}_{\ell,i}\}_{\ell\in[L']}$ and $\tilde{\tau}_{i\to v}$ are received. Then, the output distribution changes only when a random $\boldsymbol{\mu}_{i\to v} \in \mathbb{F}^L$ satisfies a linear equation $\boldsymbol{a} \cdot \boldsymbol{\mu}_{i\to v} + b = 0$ with $\boldsymbol{a} \neq \boldsymbol{0} \in \mathbb{F}^L$ and $b \in \mathbb{F}$. This happens with probability $1/|\mathbb{F}| = 1/2^\kappa$. Now we take the union bound for t corrupted P_i and $2t+1$ honest P_v, the probability is at most $t(2t+1)/|\mathbb{F}| \leq n^2/|\mathbb{F}|$, which is negligible, which is negligible. Thus, the output distributions of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ are statistically close.

$\mathbf{Hyb}_{10}$: In this hybrid, during the Completion Phase, for each honest P_v and $P_i \in \mathcal{W}$, $\mathcal{S}$ doesn't follow the protocol to check P_i's polynomials and tags. Instead, $\mathcal{S}$ considers that P_v accepts P_i's $\{g_{\ell,i}^{(k)}(\alpha_v)\}_{k\in[m],\ell\in[L']}$ when P_v receives P_i's polynomials. Since D and P_i are both honest, P_i always sends correct polynomials and tags, so P_v always accepts P_i's $\{g_{\ell,i}^{(k)}(\alpha_v)\}_{k\in[m],\ell\in[L']}$. Thus, $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ have the same output distribution.

$\mathbf{Hyb}_{11}$: In this hybrid, for each corrupted P_v and $P_i \in \mathcal{W}$, $\mathcal{S}$ doesn't sample the honest parties' shares of $[\boldsymbol{\mu}_{i\to v}]_t^i, \{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}$ in the Authentication Phase. Instead, $\mathcal{S}$ samples $\boldsymbol{\mu}_{i\to v}$ in the Completion Phase and computes $\{\nu_{i\to v}^{(k)}\}_{k\in[m]}\}$ with $\boldsymbol{\mu}_{i\to v}$ and $\{\tau_{i\to v}^{(k)}\}_{k\in[m]}\}$. Then $\mathcal{S}$ samples the honest parties' shares of $[\boldsymbol{\mu}_{i\to v}]_t^i, \{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}$ based on $\boldsymbol{\mu}_{i\to v}, \{\nu_{i\to v}^{(k)}\}_{k\in[m]}$ and the corrupted parties' shares. Since we don't need $\boldsymbol{\mu}_{i\to v}$ and the honest parties' shares of $[\boldsymbol{\mu}_{i\to v}]_t^i$ we can sample them in the Completion Phase. Then $\{\nu_{i\to v}^{(k)}\}_{k\in[m]}\}$

is fixed and can be computed. For $\{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}$, computing the honest parties' shares of the $2t$-sharing $\sum_{u=1}^{t}[\![\boldsymbol{e}_u]\!]_t^i \cdot [r_{i\to v}^{(k)}]_t^i$ is the same with sampling a random sharing based on corrupted parties' shares and the secret 0, as we argued in $\mathbf{Hyb}_5$. Hence, sampling the honest parties' shares of $\{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}$ based on the secrets and the corrupted parties' shares is the same as computing them. Thus, $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ have the same output distribution.

$\mathbf{Hyb}_{12}$: In this hybrid, for each corrupted P_v and honest $P_i \in \mathcal{W}$, $\mathcal{S}$ doesn't sample the honest parties' shares of $[\boldsymbol{\mu}_{i\to v}]_t^i, \{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}$ in the Authentication Phase. Instead, $\mathcal{S}$ randomly samples them in the Completion Phase and emulates $\mathcal{F}_{\mathsf{PrivRec}}$ to send them to P_v. Since when P_i is honest, the honest parties' shares of $[\boldsymbol{\mu}_{i\to v}]_t^i, \{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}$ are not used to compute anything in the Authentication Phase, we can change the order of simulating the Authentication Phase and generating the honest parties' shares. Thus, $\mathbf{Hyb}_{12}$ and $\mathbf{Hyb}_{11}$ have the same output distribution.

$\mathbf{Hyb}_{13}$: In this hybrid, during the Completion Phase, for each corrupted P_v and honest $P_i \in \mathcal{W}$, $\mathcal{S}$ doesn't follow the protocol to compute P_i's $\boldsymbol{g}_{*,i}$ by doing a linear combination of polynomials and the corresponding tag $\tau_{i\to v}$. Instead, $\mathcal{S}$ checks whether $r_{i\to v}$ has been recorded in $\mathcal{R}$. If it has been recorded, $\boldsymbol{g}_{*,i}$ can be computed from the corresponding $\boldsymbol{F}$. Otherwise, $\mathcal{S}$ generates a random $\boldsymbol{F}$ based on corrupted parties' column and row polynomials. As we have argued in $\mathbf{Hyb}_{2.0}, \ldots, \mathbf{Hyb}_{2.(T+T'+1)}$, with at most $n + n^3 + n^2 < T + T'$ linear combinations we need to do, if some $r_{i\to v}$ hasn't been recorded before, each F_ℓ is random with corrupted parties' column and row polynomials given, as what $\mathcal{S}$ samples in $\mathbf{Hyb}_{13}$. Thus, $\mathbf{Hyb}_{13}$ and $\mathbf{Hyb}_{12}$ have the same output distribution.

$\mathbf{Hyb}_{14}$: In this hybrid, during the Completion Phase, for each honest P_w and corrupted $P_i \in \mathcal{W}$, $\mathcal{S}$ doesn't follow the protocol to check P_i's polynomials $\{\tilde{g}_{\ell,i}\}_{\ell\in[L']}$ and the tag $\tilde{\tau}_{i\to w}$. Instead, $\mathcal{S}$ checks whether the polynomials are consistent with what $\mathcal{S}$ generates in the Sharing Phase and whether $\tilde{\tau}_{i\to w}$ is consistent with the tags P_i gets in the Authentication Phase. This changes the output distribution only if P_i sends $\{\tilde{g}_{\ell,i}\}_{\ell\in[L']}$ and $\tilde{\tau}_{i\to w}$ different from $\{g_{\ell,i}\}_{\ell\in[L']}$ and $\tau_{i\to w}$ but still satisfying $\tilde{\tau}_{i\to v} = \tilde{\boldsymbol{g}}_{*,i} \cdot \boldsymbol{\mu}_{i\to w} + \sum_{k=1}^{m} r_{v\to w}^{k} \cdot \nu_{i\to w}^{(k)}$, where $\tilde{\boldsymbol{g}}_{*,i} = (\tilde{g}_{1,i}, \ldots, \tilde{g}_{L',i})$. Then we know that $(\tilde{\boldsymbol{g}}_{*,i} - \boldsymbol{g}_{*,i}) \cdot \boldsymbol{\mu}_{i\to w} - (\tilde{\tau}_{i\to w} - \tau_{i\to w}) = 0$. Since $\boldsymbol{\mu}_{i\to w}$ is random in $\mathbb{F}^L$ when P_i is corrupted and we don't need it for any computation before P_w receives P_i's polynomials and tag, $\mathcal{S}$ can randomly sample it after $\{\tilde{g}_{\ell,i}\}_{\ell\in[L']}$ and $\tilde{\tau}_{i\to w}$ are received. Then the output distribution changes only when a random $\boldsymbol{\mu}_{i\to w} \in \mathbb{F}^L$ satisfies a linear equation $\boldsymbol{a} \cdot \boldsymbol{\mu}_{i\to w} + b = 0$ with $\boldsymbol{a} \neq \boldsymbol{0} \in \mathbb{F}^L$ and $b \in \mathbb{F}$. This happens with probability $1/|\mathbb{F}| = 1/2^\kappa$. Now we take the union bound for t corrupted P_i and $2t+1$ honest P_w, the probability is at most $t(2t+1)/|\mathbb{F}| \le n^2/|\mathbb{F}|$, which is negligible. Thus, the output distributions of $\mathbf{Hyb}_{14}$ and $\mathbf{Hyb}_{13}$ are statistically close.

$\mathbf{Hyb}_{15}$: In this hybrid, during the Completion Phase, for each honest P_w and $P_i \in \mathcal{W}$, $\mathcal{S}$ doesn't follow the protocol to check P_i's polynomials and tags. Instead, $\mathcal{S}$ considers that P_w accepts P_i's $\{g_{\ell,i}\}_{\ell\in[L']}$ when P_w receives P_i's polynomials. Since D and P_i are both honest, P_i always sends correct polynomials and tags, so P_w always accepts P_i's $\{g_{\ell,i}\}_{\ell\in[L']}$. Thus, $\mathbf{Hyb}_{15}$ and $\mathbf{Hyb}_{14}$ have the same output distribution.

$\mathbf{Hyb}_{16}$: In this hybrid, during the Completion Phase, for each honest P_w, $\mathcal{S}$ doesn't compute the output for P_w by himself. Instead, when P_w accepts $2t+1$ different P_v's $\{f_{\ell,v}^{(k)}(\alpha_w)\}_{k\in[m],\ell\in[L']}$, $\mathcal{S}$ lets P_w receive the output from $\mathcal{F}_{\mathsf{ACSS}}$. Since the $2t+1$ different P_v's $\{f_{\ell,v}^{(k)}(\alpha_w)\}_{k\in[m],\ell\in[L']}$ are accepted only if P_v is honest or the values sent by P_v are consistent with what $\mathcal{S}$ generated on behalf of D in the Sharing Phase, P_w must get $2t+1$ different points on each polynomial in $\{g_{\ell,w}^{(k)}\}_{k\in[m],\ell\in[L']}$, which enables P_w to reconstruct $\{g_{\ell,w}^{(k)}\}_{k\in[m],\ell\in[L']}$ and compute the output correctly. Hence, changing how honest parties get outputs doesn't change the output distribution. Thus, $\mathbf{Hyb}_{16}$ and $\mathbf{Hyb}_{15}$ have the same output distribution.

$\mathbf{Hyb}_{17}$: In this hybrid, during the Completion Phase, for each corrupted P_w and honest $P_i \in \mathcal{W}$, $\mathcal{S}$ doesn't follow the protocol to compute P_i's $\boldsymbol{g}_{*,i}$ by doing a linear combination of polynomials and the corresponding tag $\tau_{i\to w}$. Instead, $\mathcal{S}$ checks whether $r_{v\to w}$ has been recorded in $\mathcal{R}$. If it has been recorded, $\boldsymbol{g}_{*,i}$ can be computed from the corresponding $\boldsymbol{F}$. Otherwise, $\mathcal{S}$ generates a random $\boldsymbol{F}$ based on corrupted parties' column and row polynomials. As we have argued in $\mathbf{Hyb}_{2.0}, \ldots, \mathbf{Hyb}_{2.(T+T'+1)}$, with at most $n + n^3 + 2n^2 = T + T'$ linear combinations we need to do if some $r_{i\to w}$ hasn't been recorded before, each F_ℓ is random with corrupted parties' column and row polynomials given, as what $\mathcal{S}$ samples in $\mathbf{Hyb}_{17}$. Thus,

$\mathbf{Hyb}_{17}$ and $\mathbf{Hyb}_{16}$ have the same output distribution.

Note that $\mathbf{Hyb}_{17}$ is the ideal-world scenario, Π_{ACSS} statistically-securely computes $\mathcal{F}_{\mathsf{ACSS}}$ when D is honest.

When D is corrupted:

Simulator $\mathcal{S}$

Sharing Phase

1. For each $P_i \in \mathcal{P}$:
   - If P_i is honest:
     (1). When P_i receives $\{g_{\ell,i}^{(k)}(y)\}_{\ell\in[L'],k\in[m]}$ from D, if these polynomials are all of degree $2t$, $\mathcal{S}$ broadcasts OK_i on behalf of P_i.
     (2). $\mathcal{S}$ follows the protocol to compute $\boldsymbol{g}_{*,i}^{(k)}$ for each $k \in [m]$.
     (3). $\mathcal{S}$ sends $(\mathsf{Init}, \mathsf{APICP}, T, (\boldsymbol{g}_{*,i}^{(1)}, \dots, \boldsymbol{g}_{*,i}^{(m)}))$ to $\mathcal{F}_{\mathsf{APICP}}(P_i, D)$ on behalf of P_i.
     (4). $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\mathsf{APICP}}(P_i, D)$.
   - If P_i is corrupted, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\mathsf{APICP}}(P_i, D)$.
2. For each honest party, upon this party receives $\mathcal{M}$ from D, $\mathcal{S}$ follows the protocol to verify the $\mathcal{M}$ set. If $\mathcal{S}$ succeeds, he begins the next simulation phase of this honest party.
3. Let $\mathcal{H}$ be the first $t+1$ honest parties in $\mathcal{M}$. For each $\ell \in [L']$ and $k \in [m]$, $\mathcal{S}$ reconstructs a degree-$(t, 2t)$ bivariate polynomial $F_\ell^{(k)}(x, y)$ such that $F_\ell^{(k)}(\alpha_i, y) = g_{\ell,i}^{(k)}(y)$ for each $P_i \in \mathcal{H}$.
4. For each $k \in [m]$ and $\ell \in [L']$, $\mathcal{S}$ computes each corrupted P_j's $\hat{g}_{\ell,j}^{(k)}(y) = F_\ell^{(k)}(\alpha_j, y)$. Then $\mathcal{S}$ sets $\hat{\boldsymbol{g}}_{*,j}^{(k)} = (\hat{g}_{1,j}^{(k)}, \dots, \hat{g}_{L',j}^{(k)})$ for each $k \in [m]$.

Figure 31: Part-(1/4) of the simulator for the $\mathcal{F}_{\mathsf{ACSS}}$ when D is corrupted

Simulator $\mathcal{S}$

Verification Phase

For each $P_i \in \mathcal{P}$:
- If P_i is honest:
  (1). When P_i receives $\{g_{\ell,i}^{(k)}(y)\}_{\ell\in[L'],k\in[m]}$ from D and these polynomials are all of degree $2t$, $\mathcal{S}$ broadcasts a random value $r_i \in \mathbb{F}$ and follows the protocol to compute $\{g_{\ell,i}(y)\}_{\ell\in[L']}$ on behalf of P_i.
  (2). For each $P_h \in \mathcal{M}$, when each honest party receives r_i, $\mathcal{S}$ sends a request $(\mathsf{Request}, \mathsf{APICP}, P_i, (r_i, r_i^2, \dots, r_i^m))$ to $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ on behalf of this party.
  (3). For each $P_h \in \mathcal{M}$, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$.
  (4). $\mathcal{S}$ does the following things:
    (a). $\mathcal{S}$ checks whether $F_\ell^{(k)}(\alpha_i, y) = g_{\ell,i}^{(k)}(y)$ for each $\ell \in [L']$ and $k \in [m]$.
    (b). When P_i receives $\{\boldsymbol{g}_{*,h}\}_{P_h\in\mathcal{M}}$, $\mathcal{S}$ follows the protocol to check $\{g_{\ell,i}(y)\}_{\ell\in[L']}$.
    (c). If both checks pass, $\mathcal{S}$ accepts $\{g_{\ell,i}^{(k)}(y)\}_{\ell\in[L'],k\in[m]}$. Otherwise, $\mathcal{S}$ rejects those polynomials.
- If P_i is corrupted:
  (1). For each $P_h \in \mathcal{M}$, when each honest party receives r_i, $\mathcal{S}$ sends a request $(\mathsf{Request}, \mathsf{APICP}, P_i, (r_i, r_i^2, \dots, r_i^m))$ to $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ on behalf of this party.
  (2). For each $P_h \in \mathcal{M}$, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$.

Figure 32: Part-(2/4) of the simulator for the $\mathcal{F}_{\mathsf{ACSS}}$ when D is corrupted

Simulator $\mathcal{S}$

Authentication Phase

1. For each $P_i \in \mathcal{P}$ and $P_v \in \mathcal{P}$:

Preparing random shares:

(1). $\mathcal{S}$ follows the protocol to send requests to $\mathcal{F}_{\mathsf{RandShare}}, \mathcal{F}^0_{\mathsf{RandShare}}$ on behalf of honest parties.
(2). $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{RandShare}}, \mathcal{F}^0_{\mathsf{RandShare}}$ to receive corrupted parties' shares of

$[\boldsymbol{\mu}_{i\to v}]_t^i, \{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}, \{[r_u^{(k)}]_t^i\}_{k\in[m],u\in[t]}, \{[\mathsf{mask}_j]_t^i\}_{j\in[n]}$ from $\mathcal{A}$ and sends them to corrupted parties.

(3). For each corrupted P_j and $k \in [m]$, $\mathcal{S}$ computes P_j's share of $\{[\tau_{i\to v}^{(k)}]_{2t}^i\}_{k\in[m]}$ (denoted by $\hat{\tau}_{i\to v,j}^{(k)}$) as follows:

$$[\tau_{i\to v}^{(k)}]_{2t}^i = [\hat{\boldsymbol{g}}_{*,i}^{(k)}]_t^i \cdot [\boldsymbol{\mu}_{i\to v}]_t^i + [\nu_{i\to v}^{(k)}]_t^i + \sum_{u=1}^{t} [\![\boldsymbol{e}_u]\!]_t^i \cdot [r_u^{(k)}]_t^i$$

2. For each $P_i \in \mathcal{P}$:

Preparing shares of tags $\{\tau_{i\to v}^{(k)}\}_{k\in[m],P_v\in\mathcal{P}}$ for P_i:

- If P_i is honest:
  (1). For each $P_j \in \mathcal{P}$:
  1). If P_j is honest, $\mathcal{S}$ honestly executes the protocol. If P_j is corrupted, when P_i receives $\{\tau_{i\to v,j}^{(k)}\}_{k\in[m],P_v\in\mathcal{P}}$ from P_j, $\mathcal{S}$ broadcasts a random element $r_{i,j} \in \mathbb{F}$ and computes $\tau_{i\to v,j} = \sum_{k=1}^{m} r_{i,j}^k \cdot \tau_{i\to v,j}^{(k)}$ for all $P_v \in \mathcal{P}$ on behalf of P_i.

  Verifying P_j's shares of tags:

  2). For each $P_\alpha, P_v \in \mathcal{P}$:
  (a) For each honest party and $P_h \in \mathcal{M}$, when this honest party receives $r_{i,j}$, $\mathcal{S}$ sends $(\mathsf{Request}, \mathsf{APICP}, P_\alpha, (r_{i,j}, r_{i,j}^2 \ldots, r_{i,j}^m))$ to $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ on behalf of this honest party.
  (b) For each $P_h \in \mathcal{M}$, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$.
  (c) If P_α is honest, when P_α receives $\{\boldsymbol{g}_{*,h}\}_{P_h\in\mathcal{M}}$, $\mathcal{S}$ follows the protocol to check these polynomials. If true, then $\mathcal{S}$ delivers P_α's share of $[\gamma_{i\to v,j}]_t^i$ and sends $(\mathsf{Request}, \mathsf{privRec}, P_i)$ to $\mathcal{F}_{\mathsf{privRec}}$ on behalf of P_α.

  3). If P_j is honest, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\mathsf{privRec}}$. Otherwise, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{privRec}}$ as follows:
  (a). Upon receiving a request from an honest party, $\mathcal{S}$ follows the protocol to compute each corrupted P_α's share of $[\gamma_{i\to v,j}]_t^i$ with $\hat{\boldsymbol{g}}_{*,h}$.
  (b). $\mathcal{S}$ randomly samples the whole $[\gamma_{i\to v,j}]_t^i$ based on corrupted parties' shares.
  (c). $\mathcal{S}$ sends corrupted parties' shares to $\mathcal{A}$.
  (d). $\mathcal{S}$ sends the whole $[\gamma_{i\to v,j}]_t^i$ to P_i.

  4). When P_i receives the output, if P_j is honest, $\mathcal{S}$ considers that P_i accepts P_j's shares. If P_j is corrupted, for each $k \in [m]$, $\mathcal{S}$ checks whether $\hat{\tau}_{i\to v}^{(k)} = \tau_{i\to v}^{(k)}$. If true, $\mathcal{S}$ accepts P_j's shares.

  (2). Upon accepting $2t+1$ different P_j's $\{\tau_{i\to v,j}^{(k)}\}_{k\in[m],P_v\in\mathcal{P}}$, $\mathcal{S}$ considers that P_i reconstructs $\{\tau_{i\to v}^{(k)}\}_{k\in[m]}$.
- If P_i is corrupted:
  (1). $\mathcal{S}$ randomly samples elements from $\mathbb{F}$ as honest parties' shares of $\{[\tau_{i\to v}^{(k)}]_{2t}^i\}_{k\in[m],P_v\in\mathcal{P}}$ based on corrupted parties shares. Then $\mathcal{S}$ computes $\tau_{i\to v}^{(k)}$ for all $P_v \in \mathcal{P}$ based on these shares.
  (2). For each honest P_j, if $\mathcal{S}$ accepts $\{g_{\ell,j}^{(k)}(y)\}_{k\in[m],\ell\in[L']}$, $\mathcal{S}$ sends P_j's shares of $\{[\tau_{i\to v}^{(k)}]_{2t}^i\}_{k\in[m],P_v\in\mathcal{P}}$ (denoted each one by $\tilde{\tau}_{i\to v,j}^{(k)}$) to P_i on behalf of P_j.
  (3). For each $P_\alpha, P_v \in \mathcal{P}$:
  (a) For each honest party and $P_h \in \mathcal{M}$, when this honest party receives $r_{i,j}$, $\mathcal{S}$ sends $(\mathsf{Request}, \mathsf{APICP}, P_\alpha, (r_{i,j}, r_{i,j}^2 \ldots, r_{i,j}^m))$ to $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ on behalf of this honest party. $\mathcal{S}$ computes $\tilde{\tau}_{i\to v,j} = \sum_{k=1}^{m} r_{i,j}^k \cdot \tilde{\tau}_{i\to v,j}^{(k)}$ for each honest P_j.
  (b) For each $P_h \in \mathcal{M}$, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$.
  (c) If P_α is honest, when P_α receives $\{\boldsymbol{g}_{*,h}\}_{P_h\in\mathcal{M}}$, $\mathcal{S}$ considers that P_α accepts these polynomials. Then $\mathcal{S}$ delivers P_α's share of $[\gamma_{i\to v,j}]_t^i$ and sends $(\mathsf{Request}, \mathsf{privRec}, P_i)$ to $\mathcal{F}_{\mathsf{privRec}}$ on behalf of P_α. If P_α is corrupted, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\mathsf{privRec}}$ to wait for P_α's request.
  (4). $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{privRec}}$ as follows:
  (a) For each $P_j, P_v \in \mathcal{P}$, if P_j is honest, $\mathcal{S}$ randomly samples the whole $[\gamma_{i\to v,j}]_t^i$ based on corrupted parties' shares and P_j's $\tilde{\tau}_{i\to v,j}$. If P_j is corrupted, $\mathcal{S}$ randomly samples the whole $[\gamma_{i\to v,j}]_t^i$ based on corrupted parties' shares.
  (b) $\mathcal{S}$ sends the whole $[\gamma_{i\to v,j}]_t^i$ to P_i.

3. For each honest P_i, when P_i reconstructs $\{\tau_{i\to v}^{(k)}\}_{k\in[m],P_v\in\mathcal{P}}$ and P_i accepts his column polynomials, $\mathcal{S}$ broadcasts Tag_i on behalf of P_i.
4. For each honest party, when this honest party receives $\mathcal{W}$, $\mathcal{S}$ follows the protocol to verify the $\mathcal{W}$ set. If $\mathcal{S}$ succeeds, he begins the next simulation phase of this honest party.
5. For each $k \in [m], \ell \in [L']$, $\mathcal{S}$ sets $\mathsf{idx} = ((k-1)\cdot L' + \ell - 1)\cdot(t+1) + 1$. Then for each $i \in [0, t]$, $\mathcal{S}$ computes

$q_{\mathsf{idx}+i}(x) = F_\ell^{(k)}(x, \alpha_{-i})$. Finally, $\mathcal{S}$ sends $(\mathsf{Dealer}, \mathsf{ACSS}, \{q_1(x), \ldots, q_N(x)\})$ to $\mathcal{F}_{\mathsf{ACSS}}$ on behalf of D.

Figure 33: Part-(3/4) of the simulator for the $\mathcal{F}_{\mathsf{ACSS}}$ when D is corrupted

Simulator $\mathcal{S}$

Completion Phase

Reconstructing row polynomials:

1. For each $P_v \in \mathcal{P}$, $\mathcal{S}$ does the following things:
   - If P_v is honest:
     (1). For each honest party, $\mathcal{S}$ delivers his shares of $[\boldsymbol{\mu}_{i\to v}]_t^i$, $\{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}$ and sends $(\mathsf{Request}, \mathsf{privRec}, P_v)$ to $\mathcal{F}_{\mathsf{privRec}}$ on behalf this honest party.
     (2). $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{privRec}}$ and delivers an output to P_v.
     (3). For each $P_i \in \mathcal{W}$:
       – If P_i is honest, $\mathcal{S}$ honestly executes the protocol.
       – If P_i is corrupted, $\mathcal{S}$ does the following things on behalf of P_v:
         1). When P_v receives $\{g_{\ell,i}^{(k)}(\alpha_v)\}_{k\in[m],\ell\in[L']}$ from P_i, $\mathcal{S}$ sends a random element $r_{i\to v} \in \mathbb{F}$ to P_i.
         2). When P_v receives $\tau_{i\to v}$ and $\{g_{\ell,i}(y)\}_{\ell\in[L']}$ from P_i, $\mathcal{S}$ does the following things:
           (a). $\mathcal{S}$ checks whether $\tau_{i\to v} = \sum_{k=1}^m r_{i\to v}^k \cdot \hat{\tau}_{i\to v}^{(k)}$.
           (b). For each $\ell \in [L']$, $\mathcal{S}$ checks whether the degree of $g_{\ell,i}(y)$ is $2t$ and $g_{\ell,i}(y) = \sum_{k=1}^m r_{i\to v}^k \cdot \hat{g}_{\ell,i}^{(k)}(y)$.
           (c). For each $k \in [m]$ and $\ell \in [L']$, $\mathcal{S}$ checks whether $\hat{g}_{\ell,i}^{(k)}(\alpha_v) = g_{\ell,i}^{(k)}(\alpha_v)$.
           (d). If $(a), (b), (c)$ are true, when P_v receives the output from $\mathcal{F}_{\mathsf{privRec}}$, $\mathcal{S}$ accepts P_i's $\{g_{\ell,i}^{(k)}(\alpha_v)\}_{k\in[m],\ell\in[L']}$.
     (4). When P_v accepts $t+1$ different P_i's $\{g_{\ell,i}^{(k)}(\alpha_v)\}_{k\in[m],\ell\in[L']}$, $\mathcal{S}$ follows the protocol to reconstruct $\{f_{\ell,v}^{(k)}(x)\}_{\ell\in[L'],k\in[m]}$ on behalf of P_v.
   - If P_v is corrupted:
     (1). For each honest party, $\mathcal{S}$ delivers his shares of $[\boldsymbol{\mu}_{i\to v}]_t^i$, $\{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}$ and sends $(\mathsf{Request}, \mathsf{privRec}, P_v)$ to $\mathcal{F}_{\mathsf{privRec}}$ on behalf this honest party.
     (2). $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{privRec}}$ as follows:
       (a) If P_i is honest, $\mathcal{S}$ randomly samples the whole $[\boldsymbol{\mu}_{i\to v}]_t^i, \{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}$ based on corrupted parties' shares. If P_i is corrupted, $\mathcal{S}$ randomly samples the whole $[\boldsymbol{\mu}_{i\to v}]_t^i$ based on corrupted parties' shares, then $\mathcal{S}$ computes $\nu_{i\to v}^{(k)} = \hat{\tau}_{i\to v}^{(k)} - \hat{\boldsymbol{g}}_{*,i}^{(k)} \cdot \boldsymbol{\mu}_{i\to v}$ for each $k \in [m]$. Finally, for each $k \in [m]$, $\mathcal{S}$ samples the whole $[\nu_{i\to v}^{(k)}]_t^i$ based on corrupted parties' shares and $\nu_{i\to v}^{(k)}$.
       (b) $\mathcal{S}$ sends the whole $[\boldsymbol{\mu}_{i\to v}]_t^i, \{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}$ to P_v.
     (2). $\mathcal{S}$ sends $\{g_{\ell,i}^{(k)}(\alpha_v)\}_{k\in[m],\ell\in[L']}$ to P_v on behalf of each honest $P_i \in \mathcal{W}$.
     (3). For each honest $P_i \in \mathcal{W}$, when P_i receives $r_{i\to v}$ from P_v, $\mathcal{S}$ computes $\tau_{i\to v} = \boldsymbol{g}_{*,i} \cdot \boldsymbol{\mu}_{i\to v} + \sum_{k=1}^m r_{i\to v}^k \cdot \nu_{i\to v}^{(k)}$. $\mathcal{S}$ follows the protocol to compute $\{g_{\ell,i}(y)\}_{\ell\in[L']}$.
     (4). $\mathcal{S}$ sends $\tau_{i\to v}$ and $\{g_{\ell,i}(y)\}_{\ell\in[L']}$ to P_v on behalf of each honest $P_i \in \mathcal{W}$.

Reconstructing column polynomials:

2. For each $P_w \in \mathcal{P}$, $\mathcal{S}$ does the following things:
   - If P_w is honest:
     (1). For each $P_v \in \mathcal{P}$:
       1). $\mathcal{S}$ does the following things:
         – If P_v is honest, $\mathcal{S}$ honestly executes the protocol.
         – If P_v is corrupted, when P_w receives $\{f_{\ell,v}^{(k)}(\alpha_w)\}_{k\in[m],\ell\in[L']}$ from P_v, $\mathcal{S}$ broadcasts a random value $r_{v\to w} \in \mathbb{F}$ on behalf of P_w.
       2). For each $P_i \in \mathcal{W}$:
         – If P_i is honest, when P_i receives $r_{v\to w}$, $\mathcal{S}$ delivers $\tau_{i\to w}$ and $\{g_{\ell,i}(y)\}_{\ell\in[L']}$ to P_w on behalf of P_i. When P_w receives them, $\mathcal{S}$ considers that P_w accepts $\{g_{\ell,i}(y)\}_{\ell\in[L']}$.
         – If P_i is corrupted, when P_w receives $\tau_{i\to w}$ and $\{g_{\ell,i}(y)\}_{\ell\in[L']}$ from P_i, $\mathcal{S}$ does the following things:
           (a). $\mathcal{S}$ checks whether $\tau_{i\to w} = \sum_{k=1}^m r_{v\to w}^k \cdot \hat{\tau}_{i\to w}^{(k)}$.

(b). For each $\ell \in [L']$, $\mathcal{S}$ checks whether the degree of $g_{\ell,i}(y)$ is $2t$ and $g_{\ell,i}(y) = \sum_{k=1}^{m} r_{v\to w}^{k} \cdot \hat{g}_{\ell,i}^{(k)}(y)$.

(c). If both (a) and (b) are true, $\mathcal{S}$ accepts P_i's $\{g_{\ell,i}(y)\}_{\ell\in[L']}$.

3). When P_w accepts $t+1$ different P_i's $\{g_{\ell,i}(y)\}_{\ell\in[L']}$, $\mathcal{S}$ follows the protocol to check $\{f_{\ell,v}^{(k)}(\alpha_w)\}_{k\in[m],\ell\in[L']}$.

(2). When P_w accepts $2t+1$ different P_v's $\{f_{\ell,v}^{(k)}(\alpha_w)\}_{k\in[m],\ell\in[L']}$, $\mathcal{S}$ allows $\mathcal{F}_{\mathsf{ACSS}}$ to send the output to P_w.

- If P_w is corrupted:.
  (1). $\mathcal{S}$ sends $\{f_{\ell,v}^{(k)}(\alpha_w)\}_{k\in[m],\ell\in[L']}$ to P_w on behalf of each honest P_v.
  (2). For each honest $P_i \in \mathcal{W}$, when P_i receives $r_{v\to w}$ from P_w, $\mathcal{S}$ computes $\tau_{i\to w} = \boldsymbol{g}_{*,i} \cdot \boldsymbol{\mu}_{i\to w} + \sum_{k=1}^{m} r_{i\to w}^{k} \cdot \nu_{i\to w}^{(k)}$. $\mathcal{S}$ follows the protocol to compute $\{g_{\ell,i}(y)\}_{\ell\in[L']}$.
  (3). $\mathcal{S}$ sends $\tau_{v\to w}$ and $\{g_{\ell,i}(y)\}_{\ell\in[L']}$ to P_w on behalf of each honest $P_i \in \mathcal{W}$.

3. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

Figure 34: Part-(4/4) of the simulator for the $\mathcal{F}_{\mathsf{ACSS}}$ when D is corrupted

Hybrid arguments:

$\mathbf{Hyb}_0$: In this hybrid, $\mathcal{S}$ runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, during the Sharing Phase, $\mathcal{S}$ additionally does the following things. Let $\mathcal{H}$ be the first $t+1$ honest parties in $\mathcal{M}$, for each $k \in [m]$ and $\ell \in [L']$, $\mathcal{S}$ reconstructs a bivariate polynomial $F_{\ell}^{(k)}(x,y)$ such that $F_{\ell}^{(k)}(\alpha_i, y) = g_{\ell,i}^{(k)}(y)$ for each $P_i \in \mathcal{H}$. Then for each corrupted P_j, $\mathcal{S}$ computes P_j's column polynomial $\hat{g}_{\ell,j}^{(k)}(y) = F_{\ell}^{(k)}(\alpha_j, y)$. Finally, $\mathcal{S}$ sets $\hat{\boldsymbol{g}}_{*,j}^{(k)} = (\hat{g}_{1,j}^{(k)}, \ldots, \hat{g}_{L',j}^{(k)})$. Since $|\mathcal{H}| = t+1$, $\mathcal{S}$ can reconstruct each $F_{\ell}^{(k)}(x,y)$. $\mathcal{S}$ does not use these polynomials to do anything. Thus, $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$ have the same output distribution.

$\mathbf{Hyb}_2$: In this hybrid, during the Verification Phase, for each honest P_i and $\ell \in [L']$, $\mathcal{S}$ computes $F_{\ell}(x,y) = \sum_{k=1}^{m} F_{\ell}^{(k)}(x,y) \cdot r_i^k$. If P_i accepts his $\{g_{\ell,i}^{(k)}\}_{\ell\in[L'],k\in[m]}$, for each $\ell \in [L']$, P_i can reconstruct a degree-$(t,2t)$ bivariate polynomial $\tilde{F}_{\ell}(x,y)$. Then $\mathcal{S}$ additionally checks whether $\tilde{F}_{\ell}(x,y) = F_{\ell}(x,y)$ for each $\ell \in [L']$. When an honest P_i accepts his column polynomials, $\tilde{F}_{\ell}(\alpha_h, y) = g_{\ell,h}(y)$ holds for all $P_h \in \mathcal{M}$ and $\ell \in [L']$. Since $\mathcal{H} \subset \mathcal{M}$ and $|\mathcal{H}| = t+1$, we always have $\tilde{F}_{\ell}(x,y) = F_{\ell}(x,y)$ if P_i accepts his column polynomials. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_2$ have the same output distribution.

$\mathbf{Hyb}_3$: In this hybrid, during the Verification Phase, for each $\ell \in [L'], k \in [m]$, $\mathcal{S}$ additionally checks whether $F_{\ell}^{(k)}(\alpha_i, y) = g_{\ell,i}^{(k)}(y)$. When P_i accepts his column polynomials, for each $\ell \in [L']$ it holds that $F_{\ell}(\alpha_i, y) = g_{\ell,i}(y)$. The output distribution only changes when there exists $\ell \in [L']$ or $k \in [m]$ such that $F_{\ell}^{(k)}(\alpha_i, y) \neq g_{\ell,i}^{(k)}(y)$, we consider equation $\sum_{k=1}^{m}(F_{\ell}^{(k)}(\alpha_i, y) - g_{\ell,i}^{(k)}(y)) \cdot x^k = 0$. Let $F_{\ell}^{(k)}(\alpha_i, y) - g_{\ell,i}^{(k)} = \sum_{j=0}^{2t} h_{\ell,j}^{(k)} \cdot y^j$, and there exists $k \in [m], \ell \in [L']$ and $j \in [0, 2t]$ such that $h_{\ell,j}^{(k)}$ is none zero. We need the polynomial to y $\sum\limits_{j=0}^{2t}(\sum\limits_{k=1}^{m} h_{\ell,j}^{(k)} \cdot x^k) \cdot y^j = 0$, which means all the coefficient are 0, i.e. $\forall j$, $\sum\limits_{k=1}^{m} h_{\ell,j}^{(k)} \cdot x^k = 0$. For each $j \in [0, 2t]$, since the degree of $\sum_{k=1}^{m} h_{\ell,j}^{(k)} \cdot x^k$ is at most m, which has at most m roots in $\mathbb{F}$. Since r_i is randomly sampled by honest P_i, if r_i is one of the roots, then the output will change, and the probability this happens is $m/|\mathbb{F}|$. Now we take the union bound for $2t+1$ honest P_i, $j \in [0, 2t]$ and $\ell \in [L']$. The probability that there exists some $F_{\ell}^{(k)}(\alpha_i, y) \neq g_{\ell,i}^{(k)}(y)$ but P_i accepts his $\{g_{\ell,i}^{(k)}(y)\}_{\ell\in[L'],k\in[m]}$ is at most:

$$\epsilon_2 = L' \cdot (2t+1)^2 \cdot \frac{m}{|\mathbb{F}|} \leq \frac{mL'n^2}{|\mathbb{F}|},$$

which is negligible. Thus, the output distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_3$ are statically close.

$\mathbf{Hyb}_4$: In this hybrid, during the Authentication Phase, for each corrupted P_j, $\mathcal{S}$ computes P_j's shares of $\{[\tau_{i\to v}^{(k)}]_{2t}^{i}\}_{k\in[m],P_v\in\mathcal{P}}$ with $\hat{\boldsymbol{g}}_{*,j}^{(k)}$, denoted each one by $\hat{\tau}_{i\to v,j}^{(k)}$. $\mathcal{S}$ does not use these $\hat{\tau}_{i\to v,j}^{(k)}$ to do anything else. Thus, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_4$ have the same output distribution.

$\mathbf{Hyb}_5$: In this hybrid, during the Authentication Phase, when P_i is corrupted, for each P_j, instead of following the protocol to compute each P_α's shares of $[\gamma_{i\to v,j}]_t^i$, $\mathcal{S}$ randomly samples the whole $[\gamma_{i\to v,j}]_t^i$ based on corrupted parties' shares and P_j's share of tag. For each honest P_α, $\mathcal{S}$ uses these random elements as honest parties' shares of $[\gamma_{i\to v,j}]_t^i$ and sends them to $\mathcal{F}_{\mathsf{privRec}}$. Then, $\mathcal{S}$ uses P_α's share of $[\gamma_{i\to v,j}]_t^i$ to compute his share of $[\mathsf{mask}_j]_t^i$. The difference between $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_5$ is $\mathcal{S}$ will not use honest P_α's share of $[\mathsf{mask}_j]_t^i$ to compute his share of $[\gamma_{i\to v,j}]_t^i$. Instead, $\mathcal{S}$ samples the whole $[\gamma_{i\to v,j}]_t^i$ based on corrupted parties' shares and P_j's share of tag. Then, $\mathcal{S}$ computes honest parties' share of $[\mathsf{mask}_j]_t^i$ with $[\gamma_{i\to v,j}]_t^i$. Since honest parties' shares of $[\gamma_{i\to v,j}]_t^i$ are randomly sampled, their shares of $[\mathsf{mask}_j]_t^i$ are also random when the corrupted parties' shares are fixed. Therefore, we only change the order of sampling the two sharings. Thus, $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_5$ have the same output distribution.

$\mathbf{Hyb}_6$: In this hybrid, during the Authentication Phase, when P_i is corrupted, instead of following the protocol to compute each P_j's shares of $\{[\tau_{i\to v}^{(k)}]_{2t}^i\}_{k\in[m],P_v\in\mathcal{P}}$ (denoted by $\tau_{i\to v,j}^{(k)}$), $\mathcal{S}$ randomly samples the whole $\{[\tau_{i\to v}^{(k)}]_{2t}^i\}_{k\in[m],P_v\in\mathcal{P}}$ based on corrupted parties' shares. For each honest P_j, $\mathcal{S}$ uses these random elements (denoted by $\tilde{\tau}_{i\to v,j}^{(k)}$) as P_j's shares of $\{[\tau_{i\to v}^{(k)}]_{2t}^i\}_{k\in[m],P_v\in\mathcal{P}}$ and sends them to P_i. Since $\sum_{u=1}^{t}[\![\boldsymbol{e}_u]\!]_t^i\cdot[r_u^{(k)}]_t^i$ can be considered as $[0]_{2t}^i$ as we have argued in the hybrid arguments when D is honest, due to the similar reason in $\mathbf{Hyb}_5$, $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_6$ have the same output distribution.

$\mathbf{Hyb}_7$: In this hybrid, during the Authentication Phase, when P_i is honest, for each corrupted P_j, $\mathcal{S}$ follows the protocol to compute each corrupted P_α's share of $[\gamma_{i\to v,j}]_t^i$ with $\hat{\boldsymbol{g}}_{*,h}$, then $\mathcal{S}$ randomly samples honest P_α's shares based on corrupted parties' shares and sends them to $\mathcal{F}_{\mathsf{privRec}}$. The difference between $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_7$ is how we let honest P_i get corrupted P_j's share of $[\gamma_{i\to v,j}]_t^i$. In $\mathbf{Hyb}_6$, honest P_i will get corrupted P_j's share of $[\gamma_{i\to v,j}]_t^i$ which is consistent with all honest parties' shares from $\mathcal{F}_{\mathsf{privRec}}$. In $\mathbf{Hyb}_7$, $\mathcal{S}$ uses $\hat{\boldsymbol{g}}_{*,h}$ to compute corrupted P_α's share of $[\gamma_{i\to v,j}]_t^i$ and then randomly samples honest parties' shares based on corrupted parties' shares. Since $\hat{\boldsymbol{g}}_{*,h}$ is determined by the first $t+1$ honest parties' column polynomials in $\mathcal{M}$, the results of P_j's share of $[\gamma_{i\to v,j}]_t^i$ in the two hybrids are the same. Thus, $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_7$ have the same output distribution.

$\mathbf{Hyb}_8$: In this hybrid, during the Authentication Phase, when P_i is honest, for each corrupted P_j, when P_i receives $\{\tau_{i\to v,j}^{(k)}\}_{k\in[m],P_v\in\mathcal{P}}$ from P_j, for each $k\in[m]$ and $P_v\in\mathcal{P}$, $\mathcal{S}$ checks whether $\tau_{i\to v,j}^{(k)}=\hat{\tau}_{i\to v,j}^{(k)}$. If true, $\mathcal{S}$ accepts P_j's shares. In $\mathbf{Hyb}_7$, honest P_i will accepts corrupted P_j's $\{\tau_{i\to v}^{(k)}\}_{k\in[m]}$ when P_j's share of $[\gamma_{i\to v,j}]_t^i$ is $\tau_{i\to v,j}=\sum_{k=1}^{m}\tau_{i\to v}^{(k)}\cdot r_{i\to v,j}^k$. The output distribution only changes when there exists $k\in[m]$ such that $\hat{\tau}_{i\to v}^{(k)}\neq\tau_{i\to v}^{(k)}$. Since P_i receives the whole $[\gamma_{i\to v,j}]_t^i$ from $\mathcal{F}_{\mathsf{privRec}}$, corrupted P_j's share of $[\gamma_{i\to v,j}]_t^i$ is equal to $\sum_{k=1}^{m}\hat{\tau}_{i\to v}^{(k)}\cdot r_{i\to v,j}^k$. Therefore, we consider polynomial $\sum_{k=1}^{m}(\hat{\tau}_{i\to v}^{(k)}-\tau_{i\to v}^{(k)})\cdot x^k$. The degree of it is at most m, which means this polynomial has at most m roots in $\mathbb{F}$. Since $r_{i,j}$ is randomly sampled by P_i, therefore, the probability that $r_{i,j}$ is a root is $m/|\mathbb{F}|$. Now we take the union bound for $2t+1$ honest P_i, t corrupted P_j and n P_v, the probability that the distribution changes is at most

$$\epsilon_3=t\cdot(2t+1)\cdot n\cdot\frac{m}{|\mathbb{F}|}\leq\frac{mn^3}{|\mathbb{F}|},$$

which is negligible. Thus, the output distributions of $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_8$ are statically close.

$\mathbf{Hyb}_9$: In this hybrid, during the Completion Phase, when P_v is honest, for each corrupted $P_i\in\mathcal{W}$, $\mathcal{S}$ additionally checks whether $\tau_{i\to v}=\sum_{k=1}^{m}r_{i\to v}^k\cdot\hat{\tau}_{i\to v}^{(k)}$. For each $\ell\in[L']$, instead of checking $g_{\ell,i}(\alpha_v)=\sum_{k=1}^{m}r_{i\to v}^k\cdot g_{\ell,i}^{(k)}(\alpha_v)$, $\mathcal{S}$ checks whether $g_{\ell,i}(y)=\sum_{k=1}^{m}r_{i\to v}^k\cdot\hat{g}_{\ell,i}^{(k)}(y)$. If both checks pass, $\mathcal{S}$ accepts P_i's $\{g_{\ell,i}^{(k)}(\alpha_v)\}_{\ell\in[L'],k\in[m]}$. The only difference between $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_9$ is when $\tau_{i\to v}\neq\sum_{k=1}^{m}r_{i\to v}^k\cdot\hat{\tau}_{i\to v}^{(k)}$, if there exists $\ell\in[L']$ such that $g_{\ell,i}(y)\neq\sum_{k=1}^{m}r_{i\to v}^k\cdot\hat{g}_{\ell,i}^{(k)}(y)$, P_v still accepts $\{g_{\ell,i}^{(k)}(\alpha_v)\}_{k\in[m],\ell\in[L']}$. We denote $\hat{\tau}_{i\to v}=\sum_{k=1}^{m}r_{i\to v}^k\cdot\hat{\tau}_{i\to v}^{(k)}$, according to the following equations:

$$\left.\begin{aligned}\tau_{i\to v}&=\boldsymbol{g}_{*,i}\cdot\boldsymbol{\mu}_{i\to v}+\textstyle\sum_{k=1}^{m}r_{i\to v}^k\cdot\nu_{i\to v}^{(k)}\\ \hat{\tau}_{i\to v}&=\hat{\boldsymbol{g}}_{*,i}\cdot\boldsymbol{\mu}_{i\to v}+\textstyle\sum_{k=1}^{m}r_{i\to v}^k\cdot\nu_{i\to v}^{(k)}\end{aligned}\right\}\Rightarrow\tau_{i\to v}-\hat{\tau}_{i\to v}=(\boldsymbol{g}_{*,i}-\hat{\boldsymbol{g}}_{*,i})\cdot\boldsymbol{\mu}_{i\to v},$$

when there exists $\ell\in[L']$ such that $g_{\ell,i}(y)\neq\sum_{k=1}^{m}r_{i\to v}^k\cdot\hat{g}_{\ell,i}^{(k)}(y)$ when $\boldsymbol{g}_{*,i}\neq\hat{\boldsymbol{g}}_{*,i}$. Since $\boldsymbol{\mu}_{i\to v}$ is randomly distributed, $(\boldsymbol{g}_{*,i}-\hat{\boldsymbol{g}}_{*,i})\cdot\boldsymbol{\mu}_{i\to v}$ is also randomly distributed. Therefore, the probability that $\tau_{i\to v}-\hat{\tau}_{i\to v}=$

$(\boldsymbol{g}_{*,i} - \hat{\boldsymbol{g}}_{*,i}) \cdot \boldsymbol{\mu}_{i\to v}$ is $1/|\mathbb{F}|$. Now we take the union bound for at most t corrupted $P_i \in \mathcal{W}$ and $2t+1$ honest P_v, the probability that the output distribution changes is at most

$$\epsilon_4 = t \cdot (2t+1) \cdot \frac{1}{|\mathbb{F}|} \le \frac{n^2}{|\mathbb{F}|},$$

which is negligible. Thus, the output distributions of $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_9$ are statically close.

$\mathbf{Hyb}_{10}$: In this hybrid, during the Completion Phase, when P_v is honest, for each corrupted $P_i \in \mathcal{W}$, $\ell \in [L']$ and $k \in [m]$, $\mathcal{S}$ additionally checks whether $\hat{g}^{(k)}_{\ell,i}(\alpha_v) = g^{(k)}_{\ell,i}(\alpha_v)$. In $\mathbf{Hyb}_9$, $\mathcal{S}$ only accepts the values sent by P_i when $g_{\ell,i}(y) = \sum_{k=1}^m r^k_{i\to v} \cdot \hat{g}^{(k)}_{\ell,i}(y)$ for each $\ell \in [L']$. If there exists $\ell \in [L'], k \in [m]$ such that $\hat{g}^{(k)}_{\ell,i}(\alpha_v) \ne g^{(k)}_{\ell,i}(\alpha_v)$, which means $\sum_{k=1}^m (\hat{g}^{(k)}_{\ell,i}(\alpha_v) - g^{(k)}_{\ell,i}(\alpha_v)) \cdot r^k_{i\to v} = 0$. Similarly, the probability is at most $m/|\mathbb{F}|$. Now we take the union bound for all $\ell \in [L']$, t corrupted P_i and $2t+1$ honest P_v, the probability that the output distribution changes is at most

$$L' \cdot t \cdot (2t+1) \cdot \frac{m}{|\mathbb{F}|} \le \frac{mL'n^2}{|\mathbb{F}|},$$

which is negligible. Thus, the output distributions of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_{10}$ are statically close.

$\mathbf{Hyb}_{11}$: In this hybrid, during the Completion Phase, when P_v is corrupted, instead of following the protocol to compute each honest P_i's $\tau_{i\to v}$, $\mathcal{S}$ does the following things:

(1). When $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{privRec}}$, if P_i is honest, $\mathcal{S}$ randomly samples the whole $[\boldsymbol{\mu}_{i\to v}]^i_t, \{[\nu^{(k)}_{i\to v}]^i_t\}_{k\in[m]}$ based on corrupted parties' shares. If P_i is corrupted, $\mathcal{S}$ randomly samples the whole $[\boldsymbol{\mu}_{i\to v}]^i_t$ based on corrupted parties' shares, then $\mathcal{S}$ computes $\nu^{(k)}_{i\to v} = \hat{\tau}^{(k)}_{i\to v} - \hat{\boldsymbol{g}}^{(k)}_{*,i} \cdot \boldsymbol{\mu}_{i\to v}$ for each $k \in [m]$. Finally, for each $k \in [m]$, $\mathcal{S}$ samples the whole $\{[\nu^{(k)}_{i\to v}]^i_t\}$ based on corrupted parties' shares and $\nu^{(k)}_{i\to v}$.

(2). For each honest $P_i \in \mathcal{W}$, when P_i receives $r_{i\to v}$ from P_v, $\mathcal{S}$ computes $\tau_{i\to v} = \boldsymbol{g}_{*,i} \cdot \boldsymbol{\mu}_{i\to v} + \sum_{k=1}^m r^k_{i\to v} \cdot \nu^{(k)}_{i\to v}$.

The only difference between $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_{11}$ the way of computing each honest P_i's $\tau_{i\to v}$. In $\mathbf{Hyb}_{10}$, $\mathcal{S}$ computes $\tau_{i\to v} = \sum_{k=1}^m r^k_{i\to v} \cdot \tau^{(k)}_{i\to v}$. In $\mathbf{Hyb}_{11}$, $\mathcal{S}$ computes $\tau_{i\to v} = \boldsymbol{g}_{*,i} \cdot \boldsymbol{\mu}_{i\to v} + \sum_{k=1}^m r^k_{i\to v} \cdot \nu^{(k)}_{i\to v}$, where $\boldsymbol{\mu}_{i\to v}$ and $\{\nu^{(k)}_{i\to v}\}_{k\in[m]}$ are randomly sampled based on corrupted parties' shares by $\mathcal{S}$. Therefore, $\tau_{i\to v}$ is also random when the corrupted parties' shares of it are fixed, as sampled in $\mathbf{Hyb}_{11}$. Thus, $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_{11}$ have the same the output distribution.

$\mathbf{Hyb}_{12}$: In this hybrid, during the Completion Phase, when P_w is honest, for each corrupted $P_i \in \mathcal{W}$, when P_w receives $\tau_{i\to w}$ and $\{g_{\ell,i}(y)\}_{\ell\in[L']}$, $\mathcal{S}$ doesn't follow the protocol to check the values. Instead, $\mathcal{S}$ checks whether $\tau_{i\to w} = \sum_{k=1}^m r^k_{v\to w} \cdot \hat{\tau}^{(k)}_{i\to w}$. For each $\ell \in [L']$, $\mathcal{S}$ also checks whether the degree of $g_{\ell,i}(y)$ is $2t$ and $g_{\ell,i}(y) = \sum_{k=1}^m r^k_{v\to w} \cdot \hat{g}^{(k)}_{\ell,i}(y)$. If true, $\mathcal{S}$ accepts P_i's $\{g_{\ell,i}(y)\}_{\ell\in[L']}$. Due to the same reason in $\mathbf{Hyb}_9$, the output distribution sof $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{12}$ are statically close.

$\mathbf{Hyb}_{13}$: In this hybrid, during the Completion Phase, when P_w is corrupted, for each honest $P_i \in \mathcal{W}$, instead of computing $\tau_{i\to w}$ with $\{\tau^{(k)}_{i\to w}\}_{k\in[m]}$, $\mathcal{S}$ computes $\tau_{i\to w} = \boldsymbol{g}_{*,i} \cdot \boldsymbol{\mu}_{i\to w} + \sum_{k=1}^m r^k_{v\to w} \cdot \nu^{(k)}_{i\to w}$. Due to the same reason as in $\mathbf{Hyb}_{11}$, the output distributions of $\mathbf{Hyb}_{12}$ and $\mathbf{Hyb}_{13}$ are statically close.

$\mathbf{Hyb}_{14}$: In this hybrid, during the Authentication Phase, if any honest party receives $\mathcal{W}$ from D and $\mathcal{S}$ checks $\mathcal{W}$ is correct, $\mathcal{S}$ reconstructs $q_1(x), \dots, q_N(x)$ and sends $(\mathsf{Dealer}, \mathsf{ACSS}, \{q_1(x), \dots, q_N(x)\})$ to $\mathcal{F}_{\mathsf{ACSS}}$. This doesn't affect the output. Thus, $\mathbf{Hyb}_{13}$ and $\mathbf{Hyb}_{14}$ have the same output distribution.

$\mathbf{Hyb}_{15}$: In this hybrid, during the Completion Phase, for each honest P_w, when P_w accepts $2t+1$ different P_v's $\{f^{(k)}_{\ell,v}(\alpha_w)\}_{k\in[m],\ell\in[L']}$, instead of computing $\{g^{(k)}_{\ell,w}(\alpha_i)\}_{\ell\in[L'],k\in[m'],i\in[-t,0]}$ by himself, $\mathcal{S}$ lets $\mathcal{F}_{\mathsf{ACSS}}$ send output to P_w. If the authentication phase doesn't terminate, all the parties won't pass the public verification of $\mathcal{W}$, all the honest parties won't get any output and the functionality $\mathcal{F}_{\mathsf{ACSS}}$ is not requested by D, so it won't affect the output distribution. If the authentication phase terminates, all the honest parties must eventually get their outputs. The outputs are fixed by the polynomials determined by the column polynomials of first $t+1$ parties in $\mathcal{M}$, and the output of from $\mathcal{F}_{\mathsf{ACSS}}$ is also determined by the row polynomials of those polynomials. Thus, $\mathbf{Hyb}_{14}$ and $\mathbf{Hyb}_{15}$ have the same output distribution.

Note that $\mathbf{Hyb}_{15}$ is the ideal-world scenario, Π_{ACSS} statistically-securely computes $\mathcal{F}_{\mathsf{ACSS}}$ when D is corrupted.

□

D.3 Analysis of the Communication Complexity and Statistical Error

We make a recap of the parameters in our Π_{ACSS}, D divides his N input polynomials into m' groups. For each group, every $t+1$ polynomial will be batched to be shared by a bivariate polynomial. Therefore, there are $L' = N/(m'(t+1))$ bivariate polynomials for each group. Taking security into account, we add $T+T'$ random groups, where $T = n^3+n$ and $T' = 2n^2$. As a result, there are $m = m'+T+T'$ groups in total. Our $\mathcal{F}_{\mathsf{APICP}}$ will take m vectors as inputs and each vector is of length $L = L' \cdot n$.

During the Sharing Phase: For each group, D sends a degree-$2t$ column polynomial to each $P_i \in \mathcal{P}$, since there are m groups, resulting in a total communication of $\mathcal{O}(mL'n^2\kappa)$ bits for all parties. Then each $P_i \in \mathcal{P}$ broadcasts OK_i (each OK_i can be encoded with $\mathcal{O}(\log n)$ bits), the total communication is $\mathcal{O}(n^3 \log n)$ bits. Finally, D broadcasts set $\mathcal{M}$, which requires $\mathcal{O}(n^3 \log n)$ bits. Here we omit the communication of each $\mathcal{F}_{\mathsf{APICP}}$, and we will compute it later. Therefore, we need communication of $\mathcal{O}(mL'n^2\kappa + n^3 \log n)$ bits during the Sharing Phase. Since $L' = L/n$, the communication cost is $\mathcal{O}(mLn\kappa + n^3 \log n)$ bits.

During the verification Phase: Each $P_i \in \mathcal{P}$ broadcasts a random element, resulting in a total communication of $\mathcal{O}(n^3\kappa)$ bits. The communication of each $\mathcal{F}_{\mathsf{APICP}}$ is still omitted. Therefore, we need $\mathcal{O}(n^3\kappa)$-bit communication during the verification phase.

During the Authentication Phase:

1. For each $P_i, P_v \in \mathcal{P}$, preparing random shares requires communication of $\mathcal{O}(Ln^3\kappa + mn^4\kappa + n^5\kappa^2 + n^6)$ bits:
   (1). Each $[\boldsymbol{\mu}_{i\to v}]_t^i$: $\mathcal{O}(Ln^3\kappa + n^5\kappa^2 + n^6)$ bits.
   (2). Each $\{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m]}$: $\mathcal{O}(mn^3\kappa + n^5\kappa^2 + n^6)$ bits.
   (3). Each $\{[r_u^{(k)}]_t^i\}_{u\in[t],k\in[m]}$: $\mathcal{O}(mn^4\kappa + n^5\kappa^2 + n^6)$ bits.
   (4). Each $\{[\mathsf{mask}_j]_t^i\}_{j\in[n]}$: $\mathcal{O}(n^5\kappa^2 + n^6)$ bits.
2. For each P_i, computing P_i's tags requires communication of $\mathcal{O}(mn^2\kappa + n^2\kappa + n^3 \log n)$ bits:
   (1). Each $P_j \in \mathcal{P}$ sends his shares of tags for all $P_v \in \mathcal{P}$, resulting in a total communication of $\mathcal{O}(mn^2\kappa)$ bits.
   (2). P_i broadcasts a random element, which requires communication of $\mathcal{O}(n^2\kappa)$ bits.
   (3). Executing Π_{privRec} for all $P_v \in \mathcal{P}$ requires communication of $\mathcal{O}(n^2\kappa)$ bits.
   (4). P_i broadcasts Tag_i, which requires $\mathcal{O}(n^2 \log n)$ bits.
   (5). D broadcasts $\mathcal{W}$, which requires communication of $\mathcal{O}(n^3 \log n)$ bits.

The communication of realizing each $\mathcal{F}_{\mathsf{APICP}}$ is still omitted here, and we will compute them later. The total communication cost is $\mathcal{O}(Ln^5\kappa + mn^6\kappa + n^7\kappa^2 + n^8)$ bits during the Authentication Phase.

During the Completion Phase, while reconstructing row polynomials, for each $P_v \in \mathcal{P}$ and $P_i \in \mathcal{W}$, we need $\mathcal{O}(mn\kappa + mL'\kappa + L'n\kappa)$-bit communication:

1. P_i sends $\{g_{\ell,i}^{(k)}(\alpha_v)\}_{k\in[m],\ell\in[L']}$ to P_v, which requires communication of $\mathcal{O}(mL'\kappa)$ bits.
2. Invoking $\mathcal{F}_{\mathsf{privRec}}$ requires communication of $\mathcal{O}(mn\kappa)$ bits.
3. P_v sends a random element to each P_i, which requires communication of $\mathcal{O}(\kappa)$ bits.
4. P_i sends $\tau_{i\to v}$ and $\{g_{\ell,i}(y)\}_{\ell\in[L']}$ to P_v, which requires communication of $\mathcal{O}(L'n\kappa)$ bits.

Therefore, reconstructing row polynomials requires $\mathcal{O}(mn^3\kappa + mL'n^2\kappa + L'n^3\kappa)$ bits.

During the Completion Phase, while reconstructing column polynomials, for each $P_w \in \mathcal{P}$:

1. Each $P_v \in \mathcal{P}$ sends $\{f_{\ell,v}^{(k)}(\alpha_w)\}_{k\in[m],\ell\in[L']}$ to P_w, resulting in a total communication of $\mathcal{O}(mL'n\kappa)$ bits.
2. P_w broadcasts a random element, which requires communication of $\mathcal{O}(n^2\kappa)$ bits.
3. Each $P_i \in \mathcal{W}$ sends $\tau_{i\to w}$ and $\{g_{\ell,i}(y)\}_{\ell\in[L']}$ to P_w, resulting in a total communication of $\mathcal{O}(L'n^2\kappa)$ bits.

Therefore, reconstructing column polynomials requires $\mathcal{O}(mL'n^2\kappa + L'n^3\kappa)$ bits.

Then, the total communication cost is $\mathcal{O}(mLn\kappa + mn^3\kappa + L'n^3\kappa)$ bits during the Completion Phase.

In the end, we consider the communication cost of APICP. For each P_h in $\mathcal{M}$, according to Theorem 5, the communication of realizing each $\mathcal{F}_{\mathsf{APICP}}(P_h, D)$ is $\mathcal{O}(mL\kappa + mnT^2\kappa^2 + LT\kappa + nT^3\kappa^2)$ bits. Since we have defined $m = m' + T + T'$ and $T = n^3 + n$, the total communication cost is $\mathcal{O}(mLn\kappa + mn^8\kappa^2)$ bits.

Therefore, the protocol Π_{ACSS} requires communication of $\mathcal{O}(mLn\kappa + mn^8\kappa^2 + Ln^5\kappa)$ bits. Since $mL = (m' + T + T') \cdot L'n = \mathcal{O}(N)$, when we take $m = n^4$, the protocol Π_{ACSS} requires communication of $\mathcal{O}(Nn\kappa + n^{12}\kappa^2)$ bits.

Statistical Error. The statistical error of Π_{ACSS} is $\mathcal{O}(mn^3/2^\kappa + mL'n^2/2^\kappa)$ in the hybrid model. Since we take $m = n^4$ and it holds that $mL'n = mL = \mathcal{O}(N)$, this statistical error is $\mathcal{O}((n^7 + Nn)/2^\kappa)$. In the plain model, each execution of Π_{APICP} leads to a statistical error of $(T+1)^4 t^2\kappa^2/2^\kappa$. Since we take $T = \mathcal{O}(n^3)$ and it holds that $t = \mathcal{O}(n)$ in our ACSS construction, this statistical error item is $\mathcal{O}(n^{14}\kappa^2/2^\kappa)$. We invoke $\mathcal{F}_{\mathsf{APICP}}$ n times in the ACSS construction, which leads to a statistical error of $n^{15}\kappa^2/2^\kappa$. Besides, each execution of $\Pi^0_{\mathsf{RandShare}}$ causes a statistical error of $\mathcal{O}(k/2^\kappa)$ to generate k sharings. In our construction of Π_{ACSS}, we need to invoke $\mathcal{F}^0_{\mathsf{RandShare}}$ two times with to generate L and n sharings respectively, with leads to a statistical error of $\mathcal{O}((L+n)/2^\kappa) = \mathcal{O}((N/n^4 + n)/2^\kappa)$. Thus, the total statistical error is $\mathcal{O}((Nn + n^{15}\kappa^2)/2^\kappa)$.

E ACSS Protocol with Guarantee of Termination

In this section, we show how to modify our ACSS protocol so that if an honest party accepts his shares, then all honest parties will eventually receive their shares and terminate. We refer to this property as the guarantee of termination. The amortized communication complexity of our construction is $\mathcal{O}(n^2)$ per sharing. This represents a factor of $\mathcal{O}(n)$ improvement compared with the previously best-known result [CP23], which requires $\mathcal{O}(n^3)$ communication per sharing.

We first show how to construct an *Asynchronous Verifiable Secret Sharing* (AVSS) protocol based on our techniques in Section 5, then give the construction of an ACSS with guarantee of termination.

E.1 Construction of Asynchronous Verifiable Secret Sharing

Functionality of AVSS. AVSS guarantees that, when a dealer successfully distributes shares to all parties, all parties can jointly reconstruct the secrets. However, unlike ACSS, AVSS does not ensure that all honest parties can eventually obtain their shares. To construct an ACSS protocol with guarantee of termination, we first extend the functionality of AVSS so that it supports reconstructing linear combinations of the secrets *shared by a single dealer*.

To be more concrete, for public inputs (N, m), $\mathcal{F}_{\mathsf{AVSS}}$ receives N degree-t polynomials from the dealer and divide them into N/m vectors, each of size m. In the reconstruction phase, for a public vector $\boldsymbol{c}$ of size N/m and a receiver R, $\mathcal{F}_{\mathsf{AVSS}}$ uses $\boldsymbol{c}$ as the coefficients, computes the linear combination of these N/m vectors, and sends the result (which is a vector of m degree-t polynomials) to R. The detailed description of $\mathcal{F}_{\mathsf{AVSS}}$ is in Fig. 35.

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

Sharing Phase(N, m)

Public Input: $(\alpha_1, \ldots, \alpha_n), N, m$.

Upon receiving $(\mathsf{Dealer}, \mathsf{AVSS}, N, m, \{q_1(\cdot), \ldots, q_N(\cdot)\})$ from $D \in \mathcal{P}$, the trusted party does the following:

1. The trusted party receives the set of corrupted parties $\mathcal{C} \subset \mathcal{P}$.
2. The trusted party sends $\{q_1(\alpha_i), \ldots, q_N(\alpha_i)\}_{P_i \in \mathcal{C}}$ to the ideal adversary.
3. If all the polynomials $q_1(\cdot), \ldots, q_N(\cdot)$ are degree-t polynomials, the trusted party sends a request-based delayed output message $\mathsf{success}$ to all parties. Otherwise, the trusted party does nothing.

Reconstruction Phase$(\boldsymbol{c} = (c_1, \ldots, c_{N/m}), R)$

Public Input: A vector $\boldsymbol{c} \in \mathbb{F}^{N/m}$, the identity of the receiver R.

Upon receiving $(\mathsf{Request}, \boldsymbol{c}, R)$ from an honest party, the trusted party sends a vector of degree-t polynomials

$\sum_{\ell=1}^{N/m} c_\ell \cdot (q_{(\ell-1)\cdot m+1}(x), \ldots, q_{\ell\cdot m}(x))$ as a request-based delayed output to the receiver R.

Figure 35: Ideal functionality for asynchronous verifiable secret sharing

Construction of AVSS. We show how to modify our ACSS protocol constructed in Section 5 to realize $\mathcal{F}_{\mathsf{AVSS}}$. We first recall the parameters and notations used in our ACSS protocol.

- We use N to denote the number of dealer's input degree-t polynomials.
- Let L be the vector length in $\mathcal{F}_{\mathsf{APICP}}$. We set $L' = L/n$ and $m' = N/(L'(t+1))$. Every $t+1$ degree-t polynomial is shared by a degree-$(t, 2t)$ bivariate polynomial. Thus, in total, there are $N/(t+1)$ bivariate polynomials, denoted by $\{F_\ell^{(k)}(x, y)\}_{k\in[m'],\ell\in[L']}$.
- Each party P_i should obtain his column polynomial of every bivariate polynomial, denoted by $\{g_{\ell,i}^{(k)}(y)\}_{k\in[m'],\ell\in[L']}$. We use $g_{\ell,i}^{(k)}$ to denote the vector $(g_{\ell,i}^{(k)}(\alpha_1), \ldots, g_{\ell,i}^{(k)}(\alpha_n))$. Let $\boldsymbol{g}_{*,i}^{(k)} = (g_{1,i}^{(k)}, \ldots, g_{L',i}^{(k)})$.
- For every pair of parties (P_i, P_v), we use $\boldsymbol{\mu}_{i\to v}$ and $\{\nu_{i\to v}^{(k)}\}_{k\in[m']}$ to denote the long-term and short-term keys respectively. The authentication tag $\tau_{i\to v}^{(k)}$ is defined by $\boldsymbol{\mu}_{i\to v} \cdot \boldsymbol{g}_{*,i}^{(k)} + \nu_{i\to v}^{(k)}$.

Then we show how $(\Pi_{\mathsf{AVSS\text{-}Share}}, \Pi_{\mathsf{AVSS\text{-}Rec}})$ realize $\mathcal{F}_{\mathsf{AVSS}}$. In $\Pi_{\mathsf{AVSS\text{-}Share}}$, all parties execute the protocols Π_{Sh}, Π_{Ver}, and Π_{Auth} in sequence. As a result, if the dealer successfully distributes shares, all parties will agree on a set $\mathcal{W}$ of size $n - t$ at the end of Π_{Auth}. Then all parties jointly help each party to reconstruct his long-term and short-term keys for each $P_i \in \mathcal{W}$. In $\Pi_{\mathsf{AVSS\text{-}Rec}}$, with a public input vector $\boldsymbol{c}$ of size m' and the identity of receiver R, each $P_i \in \mathcal{W}$ sends $\boldsymbol{g}_{*,i} = \sum_{k=1}^{m} c_k \cdot \boldsymbol{g}_{*,i}^{(k)}$ to R. P_i also sends $\tau_{i\to R} = \sum_{k=1}^{m} c_k \cdot \tau_{i\to R}^{(k)}$ to R to prove that he sends the correct $\boldsymbol{g}_{*,i}$. R will accept $\boldsymbol{g}_{*,i}$ if $\tau_{i\to R} = \boldsymbol{\mu}_{i\to R} \cdot \boldsymbol{g}_{*,i} + \sum_{k=1}^{m'} c_k \cdot \nu_{i\to R}^{(k)}$. Since there are at least $t+1$ honest parties in $\mathcal{W}$, R will eventually accept $\boldsymbol{g}_{*,i}$ from $t+1$ distinct P_i. Then R can reconstruct L' degree-$(t, 2t)$ bivariate polynomials, where each bivariate polynomial contains $t+1$ linear combinations of dealer's inputs. Thus the vector size m in $\mathcal{F}_{\mathsf{AVSS}}$ is equal to $L'(t+1)$.

Note that in $\Pi_{\mathsf{AVSS\text{-}Rec}}$, a party P_i terminates after he finishes the first step.

Protocol $\Pi_{\mathsf{AVSS\text{-}Share}}$

1. All parties execute Π_{Sh}, Π_{Ver}, Π_{Auth} in order.
2. For each $P_i \in \mathcal{W}, P_v \in \mathcal{P}$:
   (1). All parties send their shares of $[\boldsymbol{\mu}_{i\to v}]_t^i, [\nu_{i\to v}^{(1)}]_t^i, \ldots, [\nu_{i\to v}^{(m')}]_t^i$ and $(\mathsf{Request}, \mathsf{privRec}, v)$ to $\mathcal{F}_{\mathsf{privRec}}$.
   (2). Upon receiving $[\boldsymbol{\mu}_{i\to v}]_t^i, \{[\nu_{i\to v}^{(k)}]_t^i\}_{k\in[m']}$ from $\mathcal{F}_{\mathsf{privRec}}$, P_v reconstructs the secrets $\boldsymbol{\mu}_{i\to v}, \{\nu_{i\to v}^{(k)}\}_{k\in[m']}$.
3. All parties set their output to be $\mathsf{success}$.

Figure 36: The protocol of the $\Pi_{\mathsf{AVSS\text{-}Share}}$

Protocol $\Pi_{\mathsf{AVSS\text{-}Rec}}$

$\underline{\Pi_{\mathsf{AVSS\text{-}Rec}}(\boldsymbol{c} = (c_1 \ldots, c_{m'}), R)}$

1. For each $P_i \in \mathcal{W}$, he sets $g_{\ell,i}^{(k)} = (g_{\ell,i}^{(k)}(\alpha_1), \ldots, g_{\ell,i}^{(k)}(\alpha_n))$ for each $\ell \in [L'], k \in [m']$ and $\boldsymbol{g}_{*,i}^{(k)} = (g_{1,i}^{(k)}, \ldots, g_{L',i}^{(k)})$, then sends $\tau_{i\to R} = \sum_{k=1}^{m} c_k \cdot \tau_{i\to R}^{(k)}$ and $\boldsymbol{g}_{*,i} = \sum_{k=1}^{m} c_k \cdot \boldsymbol{g}_{*,i}^{(k)}$ to R.
2. The receiver R does the following things:
   (1). Upon receiving $\tau_{i\to R}$ and $\boldsymbol{g}_{*,i}$ from P_i, R checks whether $\tau_{i\to R} = \boldsymbol{\mu}_{i\to R} \cdot \boldsymbol{g}_{*,i} + \sum_{k=1}^{m'} c_k \cdot \nu_{i\to R}^{(k)}$. If true, R accepts P_i's $\boldsymbol{g}_{*,i}$.
   (2). Upon accepting $t+1$ different P_i's $\boldsymbol{g}_{*,i}$, for each $\boldsymbol{g}_{*,i}$, R reconstructs a vector of degree-$2t$ polynomials $(g_{1,i}(y), \ldots, g_{L',i}(y))$. Then P_v reconstructs a vector of degree-$(t, 2t)$ bivariate polynomials $\boldsymbol{F} = (F_1(x, y), \ldots, F_{L'}(x, y))$ such that $F_\ell(\alpha_i, y) = g_{\ell,i}(y)$ for all $\ell \in [L']$. Finally, R outputs

$\{F_\ell(x, \alpha_{-i})\}_{\ell \in [L'], i \in [0,t]}$.

Figure 37: The protocol of the $\Pi_{\mathsf{AVSS\text{-}Rec}}$

Lemma 7. *The protocols* $(\Pi_{\mathsf{AVSS\text{-}Share}}, \Pi_{\mathsf{AVSS\text{-}Rec}})$ *t-securely realizes* $\mathcal{F}_{\mathsf{AVSS}}$ *in the* $(\mathcal{F}_{\mathsf{APICP}}, \mathcal{F}_{\mathsf{privRec}}, \mathcal{F}_{\mathsf{RandShare}}, \mathcal{F}^0_{\mathsf{RandShare}})$*-hybrid model with statistical security.*

E.2 Construction of Asynchronous Complete Secret Sharing with Guarantee of Termination

In this subsection, we show how to construct an ACSS protocol with guarantee of termination in the $\mathcal{F}_{\mathsf{AVSS}}$-hybrid model.

At a high level, assuming that the dealer D uses degree-t polynomials $q_1(\cdot), \ldots, q_N(\cdot)$ as inputs of Π_{ACSS}, D will share each P_i's shares through an instance of $\mathcal{F}^{(i)}_{\mathsf{AVSS}}$. Later $\mathcal{F}^{(i)}_{\mathsf{AVSS}}$ can help P_i get his shares with guarantee of termination. The remaining problem is to make sure all honest parties' shares are consistent. To solve this problem, we check a random linear combination of the sharings distributed by D. We let all parties agree on a random vector $\boldsymbol{r}$ after receiving $\mathsf{success}$ from $\{\mathcal{F}^{(i)}_{\mathsf{AVSS}}\}_{i=1}^n$. Then each $\mathcal{F}^{(i)}_{\mathsf{AVSS}}$ uses $\boldsymbol{r}$ as the coefficients to compute and send a linear combination of P_i's shares to all parties. Intuitively, if all honest parties' shares lie on degree-t polynomials, the linear combination results should also lie on degree-t polynomials. Note that $\mathcal{F}^{(i)}_{\mathsf{AVSS}}$ will pack m degree-t polynomials together as a vector and use $\boldsymbol{r}$ to compute the linear combination of these N/m vectors.

The functionality $\mathcal{F}^{(i)}_{\mathsf{AVSS}}$ ensures that each party can get all parties' linear combinations of vectors. Then each party checks whether the values in these vectors lie on a vector of degree-t polynomials, if true, he considers that D honestly distributes shares and waits to receive his shares from the corresponding instance of $\mathcal{F}_{\mathsf{AVSS}}$ later. For more details, please refer to Fig. 38.

Protocol Π_{ACSS}

Parameter. Let N be the number of dealer's inputs, m be the vector length used in each instance of $\mathcal{F}_{\mathsf{AVSS}}$, $(\alpha_0, \ldots, \alpha_n)$ be distinct field elements in $\mathbb{F}$.

1. The dealer D receives degree-t polynomials $q_1(\cdot), \ldots, q_N(\cdot)$ from the environment. D randomly sample m degree-t polynomials $q_{-m+1}(\cdot), \cdots, q_0(\cdot)$.
2. For $i \in [n]$ and $k \in [-m+1, N]$, D randomly samples degree-t polynomials $q^{(i)}_k(x)$ such that $q^{(i)}_k(\alpha_0) = q_k(\alpha_i)$. Then for all $i \in [n]$, D sends request $(D, \mathsf{AVSS}, N+m, m, \{q^{(i)}_{-m+1}(x), \cdots, q^{(i)}_N(x)\})$ to $\mathcal{F}_{\mathsf{AVSS}_i}$. Denote $(q^{(i)}_{(\ell-1)\cdot m+1}(x), \ldots, q^{(i)}_{\ell \cdot m}(x))$ as $\boldsymbol{q}^{(i)}_\ell(x)$ for each $\ell \in [0, N/m]$.
3. Each party does the following things:
   (1). Upon receiving $\mathsf{success}$ from $\mathcal{F}^{(i)}_{\mathsf{AVSS}}$ for all $i \in [n]$, send request $(\mathsf{Request}, \mathsf{Coin})$ to $\mathcal{F}_{\mathsf{Coin}}$. Upon receiving r from $\mathcal{F}_{\mathsf{Coin}}$, set $\boldsymbol{r} = (r^0, \ldots, r^{N/m})$ and send request $(\mathsf{Request}, \boldsymbol{r}, P_j)$ to $\mathcal{F}^{(i)}_{\mathsf{AVSS}}$ for all $i \in [n]$ and $P_j \in \mathcal{P}$. Denote $\sum_{\ell=0}^{N/m} r^\ell \cdot \boldsymbol{q}^{(i)}_\ell(x)$ as $\boldsymbol{q}^{(i)}_*(x)$.
   (2). For $i \in [n]$, receive $\boldsymbol{q}^{(i)}_*(x)$ from $\mathcal{F}^{(i)}_{\mathsf{AVSS}}$ and compute $\boldsymbol{q}^{(i)}_*(\alpha_0)$. Then check whether $\{\boldsymbol{q}^{(i)}_*(\alpha_0)\}_{i=1}^n$ lie on a valid vector of m degree-t polynomials. If not, he outputs nothing.
   (3). Denote the unit vector of length $N/m+1$, with the ℓ^{th} value being 1 as $\boldsymbol{e}_\ell$. Send $(\mathsf{Request}, \boldsymbol{e}_\ell, P_i)$ to each $\mathcal{F}^{(i)}_{\mathsf{AVSS}}$ for all $i \in [n], \ell \in [0, N/m]$.
4. When P_i receives $\{\boldsymbol{q}^{(i)}_\ell(x)\}_{\ell=0}^{N/m}$ from $\mathcal{F}^{(i)}_{\mathsf{AVSS}}$, he outputs $\{\boldsymbol{q}^{(i)}_\ell(\alpha_0)\}_{\ell=1}^{N/m}$.

Figure 38: The protocol of the Π_{ACSS} with the guarantee of termination

Lemma 8. *The protocol* Π_{ACSS} *in Fig. 38 t-securely realizes* $\mathcal{F}_{\mathsf{ACSS}}$ *in the* $(\mathcal{F}_{\mathsf{AVSS}}, \mathcal{F}_{\mathsf{Coin}})$*-hybrid model with statistical security and guarantee of termination.*

When instantiating $\mathcal{F}_{\mathsf{AVSS}}$ by $(\Pi_{\mathsf{AVSS\text{-}Share}}, \Pi_{\mathsf{AVSS\text{-}Rec}})$ in section E.1, we obtain Theorem 6.

Theorem 6. *Let κ denote the security parameter. For a finite field $\mathbb{F}$ of size $2^{\Theta(\kappa)}$, there exists a fully malicious information-theoretic ACSS protocol against $t < n/3$ corrupted parties that shares N degree-t Shamir sharings over $\mathbb{F}$ with communication of $\mathcal{O}(Nn^2\kappa + n^{13}\kappa^2)$ bits, statistical error $\mathcal{O}((Nn^2 + n^{16}\kappa^2)/2^\kappa)$ and guarantee of termination. The round complexity is $\mathcal{O}(1)$ rounds via P2P channels plus $\mathcal{O}(1)$ rounds of invocations to ACast.*

F The Asynchronous MPC Protocol

F.1 Functionality $\mathcal{F}_{\mathsf{AMPC}}$

The ideal functionality $\mathcal{F}_{\mathsf{AMPC}}$ (see Fig. 39) from [CP23, Coh16] is as follows. Without loss of generality, assume that all parties receive the same function output from $\mathcal{F}_{\mathsf{AMPC}}$.

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

$\mathcal{F}_{\mathsf{AMPC}}$ proceeds as follows, running with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$ and parameterized by an 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 $y^{(i)} = \perp$.

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

Figure 39: Ideal functionality for asynchronous secure multiparty computation

F.2 Overview of the Construction of Π_{AMPC}

We recall the framework of constructing Π_{AMPC} in [CP17] here.

Let C be an arithmetic circuit over $\mathbb{F}$ with depth D and $|C|$ multiplication gates. The high-level idea of the construction is as follows:

- **Step 1: Preparing the Beaver Triples.** Each party generates $\mathcal{O}(|C|)$ completely t-shared random multiplication triples $([a]_t, [b]_t, [c]_t)$, where $c = a \cdot b$. For this, each party invokes $\mathcal{F}_{\mathsf{ACSS}}$ to share $\mathcal{O}(|C|)$ degree-t Shamir secret sharings. Since corrupted parties may provide invalid ($c \neq a \cdot b$) triples, all parties apply a *polynomial verification* process to check each party's triples. Due to the asynchronous network, the invocations of $\mathcal{F}_{\mathsf{ACSS}}$ by corrupted parties may never terminate. Therefore, all parties execute an ACS protocol to agree on a common subset of $n - t$ parties whose triples are correctly t-shared and valid. To prevent corrupted parties from providing valid but non-random triples, all parties apply a *triple extraction* procedure to output $|C|$ completely t-shared, truly random, and private triples. Each Beaver triple is used for the computation of one multiplication gate.

- **Step 2: Input Sharing.** Each party chooses degree-t polynomials to share his input $\boldsymbol{x}^{(i)}$ and invokes $\mathcal{F}_{\mathsf{ACSS}}$ to share his input. Since the corrupted parties may never share their inputs, all parties run an ACS protocol to agree on a common subset CoreSet of $n - t$ parties whose input is complete t-shared. All parties will use these $n - t$ parties' shares of inputs for the following computation and the remaining t parties' inputs will be set as 0.

- **Step 3: Circuit Evaluation.** When each party receives verified Beaver Triples in Step 1 and all shares from each one in CoreSet in Step 2, he starts to evaluate each gate in the circuit as follows, depending on the type of the gate. Let the inputs for each gate be x and y, and the output be z.

– **Linear Gate:** Each party gets his share of z by locally applying the linear function on his shares of x and y.

– **Multiplication Gate:** All parties use a Beaver triple (a, b, c) to compute their shares of the output. At a high level, each party computes his share of $[x-a]_t$ and $[y-b]_t$ and sends them to all parties. Then, each party uses OEC to reconstruct $x-a$ and $y-b$. Finally, each party computes his share of $[z]_t = (x-a)(y-b) + (x-a)[b]_t + (y-b)[a]_t + [c]_t$. The above procedures require each party to reconstruct the degree-t Shamir secret sharings. Considering the efficiency of reconstruction of degree-t Shamir secret sharings, a batch of $t+1$ reconstruction can be executed in parallel.

– **Output Gate:** All parties invoke $\mathcal{F}_{\mathsf{privRec}}$ with his t-shares of the output gate for every $P_i \in \mathcal{P}$ to help P_i reconstruct his output y.

- **Step 4: Termination Phase.** All parties do the following things:

  1. If a circuit-output y is computed, send y to all parties.
  2. If y is received from at least $t+1$ parties, send y to all parties.
  3. If y is received from at least $n-t$ parties, accept y and terminate.

The communication cost is summarized as follows: In Step 1, we invoke $\mathcal{F}_{\mathsf{ACSS}}$ n times to share $\mathcal{O}(|C| \cdot n)$ degree-t Shamir secret sharings in total and broadcast $\mathcal{O}(n^3)$ field elements. The *triple extraction* procedure requires $\mathcal{O}(|C| \cdot n^2\kappa)$ bits. In Step 2, we invoke $\mathcal{F}_{\mathsf{ACSS}}$ n times. In Step 3, the linear gates are free. We pack every $t+1$ multiplication gate in one layer, which requires communication of $\mathcal{O}(n^2\kappa)$ bits, and computing the whole multiplication gates requires communication $\mathcal{O}(|C| \cdot n\kappa + D \cdot n^2\kappa)$ bits. The output gates require n invocations of $\mathcal{F}_{\mathsf{privRec}}$. The main communication apart from the invocations of $\mathcal{F}_{\mathsf{ACSS}}$ comes from the invocations of *triple extraction*, broadcast field elements, and the circuit evaluation, which requires communication of $\mathcal{O}(|C| \cdot n^2\kappa + n^5\kappa)$ bits.

Therefore, when replacing $\mathcal{F}_{\mathsf{ACSS}}$ with our construction of Π_{ACSS} to realize $\mathcal{F}_{\mathsf{AMPC}}$, which requires communication of $O(|C| \cdot n^2\kappa + n^{13}\kappa^2)$ bits to share $\mathcal{O}(|C| \cdot n)$ degree-t Shamir secret sharings in total, the whole communication cost of Π_{AMPC} is $O(|C| \cdot n^2\kappa + n^{13}\kappa^2)$ bits.