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 $\kappa$ 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 [BH08, BGW88] 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 \( t < n/3 \) [ADS20, BOKR94]. 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 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 [CP23] still requires \( O(n^4 \kappa) \) bits of communication per multiplication gate, where \( \kappa \) 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.

### 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 terminates or all honest parties terminate with correct shares. With an ACSS protocol, a typical approach to achieving AMPC [CP17, CP23] 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 \( O(n^2 \kappa) \) bits of communication plus sharing \( O(n) \) degree-\( t \) Shamir sharings via ACSS per multiplication gate. Combining with the best-known result of ACSS [CP23], which requires communicating \( O(n^3 \kappa) \) bits per sharing, it gives the best-known result of AMPC which requires \( O(n^4 \kappa) \) bits per multiplication gate.

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, distributing degree-\( t \) Shamir sharings can be done with linear communication complexity \( O(n \kappa) \) bits per sharing [BFO12]. This leads to our following question.

*Does there exist an optimal-resilient information-theoretic ACSS protocol with amortized communication of \( O(n \kappa) \) bits per sharing?*

### 1.2 Contributions

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

**Theorem 1.** Let \( \kappa \) denote the security parameter. For a finite field \( \mathbb{F} \) of size \( 2^{O(\kappa)} \), there exists a fully malicious information-theoretic ACSS protocol against \( t < n/3 \) corrupted parties that shares \( N \) degree-\( t \)
Shamir sharings over $F$ with communication of $O(Nn\kappa + n^{12}\kappa^2)$ bits.

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 $O(n^2\kappa)$ 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 $\mathcal{F}_{\text{ACSS}}$-hybrid model. The achieved communication complexity is $O(|C| \cdot n^2\kappa + n^3\kappa)$ bits plus $O(n)$ invocations of $\mathcal{F}_{\text{ACSS}}$ to share $O(|C| \cdot n)$ 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 [GLZS24] 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 [GLZS24], we obtain an AMPC protocol with $O(n\kappa)$ bits of communication per multiplication gate, the first information-theoretic AMPC protocol with linear communication complexity and optimal resilience.

**Theorem 3.** [GLZS24] 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 $\mathcal{F}_{\text{ACSS}}$-hybrid model. The achieved communication complexity is $O(|C| \cdot n\kappa + D \cdot n^2\kappa + n^8\kappa)$ bits plus $O(n^2)$ invocations of $\mathcal{F}_{\text{ACSS}}$ to share $O(|C|)$ 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.

### 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 $O(n^2\kappa)$ 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 $\omega_i$ as his share. In our work, we use $\kappa$ 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 terminate. 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 terminate, or no honest party terminates. If all honest parties terminate, then the shares of honest parties lie on a valid degree-$t$ polynomial.

Our goal is to construct an ACSS protocol with optimal resilience $n = 3t + 1$ and linear communication complexity $O(n\kappa)$ 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 $\rightarrow$ AVSS $\rightarrow$ ACSS. We will introduce each notion below. Here $X \rightarrow Y$ denotes that $X$ is used as a sub-protocol in $Y$.

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 $O(1)$ 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 (if terminated). 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.

\footnote{In [PCR09], the authors add an intermediate protocol called AWSS between AICP and AVSS. In [CP23], AVSS is called AISS.}
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 0K\( P_j \). 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 \( P_i \in \mathcal{W} \) 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 \( O(n^2\kappa) \) 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 \( O(n^2\kappa) \) bits, the ACSS protocol has an amortized communication complexity of at least \( O(n^3\kappa) \) 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. At 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 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 each degree-\((t, 2t)\) bivariate polynomial can be used to store \( t + 1 = O(n) \) degree-\( t \) Shamir sharings, hopefully, we can reduce the amortized communication complexity per sharing by \( O(n) \) in this way.

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 $m \in \mathbb{F}^k$ held by $P_i$, $P_j$ will prepare a pair of authentication keys $(\mu, \nu)$ where $\mu \in \mathbb{F}^k$ and $\nu \in \mathbb{F}$. The authentication tag is defined by $\tau = \mu \cdot m + \nu$, where $\cdot$ denotes the inner-product operation, and we let $P_i$ obtain $\tau$. Later on, when $P_i$ sends $m$ to $P_j$, $P_i$ also sends $\tau$ to $P_j$ so that $P_j$ can verify the correctness of $m$. Note that when $\mu, \nu$ are uniformly random, the probability that $P_i$ can find a different $(m', \tau')$ such that $\tau' = \mu \cdot m' + \nu$ is negligible.

So far the key size is linear in the message length. To reduce the key size, the authors in [BFO12] observe that $\mu$ can be reused to authenticate multiple batches of messages as long as each time we use a uniformly random $\nu$. Thus, $\mu$ is first randomly sampled and used as the long-term key. For each batch of messages $m$, a random $\nu$ is sampled and the tag of $m$ is computed by $\tau = \mu \cdot m + \nu$. 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 $g_i(y)$ 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 sublinear communication overhead. As a result, we manage to directly upgrade from AVSS to ACSS, avoiding the other $O(n)$ 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 $m, m'$, if $P_i$ holds $\tau = \mu \cdot m + \nu$ and $\tau' = \mu \cdot m' + \nu'$, then $\tau + \tau'$ is an authentication tag of $m + m'$ under the keys $(\mu, \nu, \nu')$. 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 $\tau$ to $P_j$.

5. $P_j$ verifies the correctness of $g_i(y)$ and $\tau$, 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 $P_i$ 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 $O(n)$ 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 $g_j(y)$ 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 $P_j$ 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 $\tau_k$. Then $P_k$ sends $g_k(y)$ and $\tau_k$ to $P_j$. 

7
5. \( P_j \) verifies the correctness of each \( g_k(y) \) and \( r_k \), 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 \( M \) of these 2t + 1 parties.

As [CP23], the column polynomials of the first \( t+1 \) honest parties in \( M \) 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 \( P_i \in M \) 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 \( M \) 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 \( M \).

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 \( M \).

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 degree-\( t \) Shamir sharings, and secret-share the authentication keys \((\mu, \nu)\) by \([\mu]_t, [\nu]_{2t}\) respectively. Here for simplicity, we assume that \( \mu \) is of length \( n \). Then all parties locally compute

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

and send their shares to \( P_i \) to let \( P_i \) reconstruct the tag \( \tau \). 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 \( \alpha_i \) rather than \( \alpha_0 \). To utilize this observation, the authors in [BFO12] introduced the notion of *twisted secret sharings*. A degree-\( t \) twisted secret sharing \([x]_{i, t}\) is a degree-\( t \) Shamir sharing whose secret is stored at position \( \alpha_i \) and the \( i \)-th share is at position \( \alpha_0 \). Effectively, we switch positions of the secret and the \( i \)-th share. Then all parties hold \([g_i]_{i, t}\) 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 \([\mu]_{i, t}, [\nu]_{2t}\) such that the shares of \([\mu]_{i, t}\) of \( P_i \) are 0. In this way, even \( P_i \) does not know his share of \([g_i]_{i, t}\), he can still compute his share of \([\tau]_{2t}\) by following

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

since \( P_i \)'s shares of \([g_i]_{i, t}\) should be multiplied by his shares of \([\mu]_{i, t}\), which are all 0. In this way, the communication cost of computing \( \tau \) is only (1) sharing the authentication keys by \( P_j \) plus (2) reconstructing \( \tau \) to \( P_i \). Recall that the long-term key \( \mu \) can be reused. For each batch of messages, \( P_j \) only needs to share the short-term key \( \nu \). As a result, the communication cost of computing \( \tau \) 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 \([x]_{i, t}\) is just a standard degree-\( t \) Shamir sharing except that the secret is stored at position \( \alpha_i \) while the \( i \)-th share is at position \( \alpha_0 \).

- For \([\mu]_{i, t}\), they are random degree-\( t \) Shamir sharings with the \( i \)-th shares to be 0. In Section 3.4 (see \( F^{RandShare}_\text{ACSS} \)), we show that such random degree-\( t \) Shamir sharings can be efficiently prepared in the \( F^{ACSS}_\text{hybrid} \) model.

- For \([\nu]_{2t}\), 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 \( \Pi_{\text{Auth}} \)) for more details. We note that random degree-\( t \) Shamir sharings can be efficiently prepared in the \( F^{ACSS}_\text{hybrid} \) model as well.

\(^2\)In [BFO12], the authors require that the share of \( P_i \) is always 0.
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 $F_{ACSS}^k$ for preparing $[\mu]^i_t$ and $[\nu]^i_{2t}$. 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 $\tau$ from $[\tau]_{2t}$, $P_i$ 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 $\tau$, $P_i$ needs to distinguish correct shares from incorrect shares. To be more concrete, for each party $P_k$, after $P_k$ computes his share of $[\tau]_{2t}$ 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 $[\tau]_{2t} = [g]^i_t \cdot [\mu]^i_t + [\nu]^i_{2t}$. Our goal is to let all parties compute a degree-$t$ Shamir sharing of the $k$-th share of $[\tau]_{2t}$. This problem can be reduced to compute

- a degree-$t$ Shamir sharing of the $k$-th share of $[g]^i_t \cdot [\mu]^i_t$,
- and a degree-$t$ Shamir sharing of the $k$-th share of $[\nu]^i_{2t}$.

For the $k$-th share of $[g]^i_t \cdot [\mu]^i_t$, $P_k$ will multiply $g_k$, which is his share of $[g]^i_t$, with the $k$-th share of $[\mu]^i_t$. We note that if all parties compute $g_k \cdot [\mu]^i_t$, the result is a degree-$t$ Shamir sharing such that the $k$-th share is equal to the $k$-th share of $[g]^i_t \cdot [\mu]^i_t$. To obtain a degree-$t$ Shamir sharing of the $k$-th share of $[\nu]^i_{2t}$, we rely on the observation in [EGPS22] again. Recall that $[\nu]^i_{2t}$ is prepared by first preparing $t + 1$ random degree-$t$ Shamir sharings and then computing $[\nu]^i_{2t}$ 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 $[\nu]^i_{2t}$. 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 $[\tau]_{2t}$. 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 $P_i$. 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 $[g]^{(0)}_i, \ldots, [g]^{(L)}_i$, the long-term key $[\mu]_i$, and the short-term keys $[\nu^{(0)}]_{2t}, \ldots, [\nu^{(L)}]_{2t}$. All parties locally compute $[\tau^{(0)}]_{2t}, \ldots, [\tau^{(L)}]_{2t}$ 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

$$F(x, y) = F^{(0)}(x, y) + r \cdot F^{(1)}(x, y) + \ldots + r^L \cdot F^{(L)}(x, y),$$

$$[g]_i = [g]^{(0)}_i + r \cdot [g]^{(1)}_i + \ldots + r^L \cdot [g]^{(L)}_i,$$

$$[\nu]_{2t} = [\nu^{(0)}]_{2t} + r \cdot [\nu^{(1)}]_{2t} + \ldots + r^L \cdot [\nu^{(L)}]_{2t},$$

$$[\tau]_{2t} = [\tau^{(0)}]_{2t} + r \cdot [\tau^{(1)}]_{2t} + \ldots + r^L \cdot [\tau^{(L)}]_{2t}.$$ 

Then the problem becomes to check $P_k$’s share of $[\tau]_{2t}$. 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 \( \text{Tag}_i \). The sharing phase finishes after \( 2t + 1 \) parties broadcast \( \text{Tag}_i \).

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 \( M \) 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 \( M \).

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 \( M \). 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 \( \text{Tag}_i \). Then \( D \) creates a set \( W \) containing parties whose \( \text{Tag}_i \) are received. Finally, \( D \) broadcasts \( W \) when \( |W| = 2t + 1 \) 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 \( \kappa \). \( \mathbb{F} \) is used to denote a finite field where \(|\mathbb{F}| = 2^{\Theta(\kappa)}\). We denote the inner product of two vectors \( \mathbf{u}, \mathbf{v} \in \mathbb{F}^L \) by \( \mathbf{u} \cdot \mathbf{v} \).

We assume that \( n = \text{poly}(\kappa) \), where \( \text{poly}(\kappa) \) denotes a polynomial function of \( \kappa \). We also use \( \text{negl}(\kappa) \) to denote a negligible function to \( \kappa \), which means the function is smaller than any \( 1/\text{poly}(\kappa) \) for sufficiently large \( \kappa \). All the polynomials we mention are over \( \mathbb{F} \). A degree-\( d \) polynomial is of the form \( f(x) = a_0 + \cdots + a_dx^d \), where each \( a_i \in \mathbb{F} \). A degree-(\( \ell, m \)) bivariate polynomial is of the form \( F(x, y) = \sum_{i,j=0}^{\ell, m} r_{ij}x^i y^j \), 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 \( \Pi \) 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 \( A \), and an environment \( Z \). The environment provides inputs to the honest parties, receives their outputs, and communicates with
the adversary $A$. We consider $A$ 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 $A$ 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 modeled in UC using the eventual-delivery secure message-transmission ideal functionality, for example in [KMTZ13, CGHZ16]. The protocol completes once $Z$ outputs a single bit.

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

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

The ideal functionality $F$ models the desired behavior of the computation. $F$ only receives inputs from the parties and $S$ and provides outputs to them. $S$ cannot see or delay the communication between the honest parties and $F$. 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 $F$ to the honest parties, which is used in [Coh16, CFG+23]. In this model, the functionality $F$ 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 $F$ 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 $\text{IDEAL}_{F,S,Z}(\kappa,z,\bar{r})$ the random variable containing the output of $Z$ with input $z$, security parameter $\kappa$, and interacting with the parties $P_1,\ldots,P_n$ and the adversary $S$ with random tapes $\bar{r} = (r_S,r_Z)$. We denote the random variable $\text{IDEAL}_{F,S,Z}(\kappa,z,\bar{r})$ for uniformly random $\bar{r}$ by $\text{IDEAL}_{F,S,Z}(\kappa,z)$.

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

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

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

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

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 $\epsilon = \text{negl}(\kappa)$.

**The Hybrid Model.** In a $G$-hybrid model, a protocol execution proceeds as in the real world except that the parties have access to an ideal functionality $G$ for some specific task. During the protocol execution, the
parties can communicate with \( G \) 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 \( G \). This is guaranteed by the following composition theorem from [Can00, Can01].

**Theorem 4.** ([Can00, Can01]) Let \( \Pi \) be a protocol that UC-securely realizes a functionality \( F \) in the \( G \)-hybrid model and let \( \rho \) be a protocol that UC-securely realizes \( G \). Moreover, let \( \Pi' \) denote the protocol that is obtained from \( \Pi \) by replacing every ideal call to \( G \) with the protocol \( \rho \). Then protocol \( \Pi' \) UC-securely realizes \( F \) in the model where the parties do not have access to the ideal functionality \( G \).

**Hybrid Arguments.** We use hybrid arguments to prove that the distributions of \( \text{REAL}_{\Pi,A,Z}(\kappa,z) \) and \( \text{IDEAL}_{F,S,Z}(\kappa,z) \) are identical (or statistically close). In each hybrid, we use output distribution to denote the distribution of the random variable containing what \( Z \) outputs on input \( z \) and uniformly random \( \tilde{r} \). 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 \( \text{REAL}_{\Pi,A,Z}(\kappa,z) \) and \( \text{IDEAL}_{F,S,Z}(\kappa,z) \) 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 \( \Pi_{\text{ACS}}^Q \) in [BKR94], which achieves communication complexity of \( O(n^3) \).

We also need an *A-Cast* protocol to enable a party to broadcast a message in the asynchronous network. From [Bra84], broadcasting an \( \ell \)-bit message requires \( O(n^2\ell) \)-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 2\( t \)) 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 2\( t \)) such that \( \omega = f(\alpha_0) \) and \( \omega_j = f(\alpha_j) \) for \( j \in \{1, \ldots, n\} \). Furthermore, share \( \omega_j \) 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 2\( t \)) twisted sharing* [BFO12] of \( \omega \in \mathbb{F} \) with respect to \( P_t \in \mathbb{P} \) is \([\omega]_t \) (or \([\omega]_{2t}\) ) with respect to \((\alpha_t, \alpha_1, \ldots, \alpha_{t-1}, \alpha_0, \alpha_{t+1}, \ldots, \alpha_n) \), i.e. secret is now \( f(\alpha_t) \). \( P_t \)'s share is \( f(\alpha_0) \). We denote this sharing by \([\omega]_t^\alpha \) (or \([\omega]_{2t}^\alpha\) ).

- We say a secret sharing \([\omega]_t \) (or \([\omega]_{2t}\) ) 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 \( \omega \) and all the parties' shares of \( \omega \) lie on a degree-\( t \) polynomial.

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

- For fixed \((\alpha_{-t}, \ldots, \alpha_{-1}) \), a *degree-\( t \) packed secret sharing of* \((\omega^{(0)}, \ldots, \omega^{(t)}) \in \mathbb{F}^{t+1} \) with respect to \( P_t \in \mathbb{P} \) 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 \( j \in \{0\} \), \( \omega^{(0)} = f(\alpha_1), \omega_i = f(\alpha_0) \) and \( \omega_j = f(\alpha_j) \) for \( j \in [n] \setminus \{i\} \). Furthermore, share \( \omega_j \) is held by player \( P_j \) for \( j \in [n] \). It's easy to see that \( \omega \) uniquely determines the polynomial \( f(X) \), we denote such a sharing as \( [\omega]_t \), where \( \omega = (\omega^{(0)}, \ldots, \omega^{(t)}) \).
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 terminate and distribute a complete $t$-sharing to all the honest parties. When the dealer is corrupted, once an honest party terminates the protocol, each honest party must terminate the protocol and get his share of a $t$-sharing eventually. We give the functionality of ACSS in Fig. 1.

For public value $(\alpha_0, \ldots, \alpha_n)$, upon receiving $(\text{Dealer}, \text{ACSS}, \{q_1(\cdot), \ldots, q_n(\cdot)\})$ from $D \in \mathcal{P}$, do the following:

- If all the polynomials $q_1(\cdot), \ldots, q_n(\cdot)$ are degree-$t$ polynomials, the trusted party sends an request-based delayed output $\{q_i(\alpha_i), \ldots, q_N(\alpha_i)\}$ to each party $P_i \in \mathcal{P}$.
- 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.

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

**Lemma 1.** ([CP23]) There exists a protocol that $t$-securely realizes $\mathcal{F}_{\text{ACSS}}$ for any $N \in \mathbb{Z}_+^*$ with statistical security and $O(N \cdot n^3 \kappa + n^4 \kappa^2 + n^5)$-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 $\Pi_{\text{privRec}}$ that realizes $\mathcal{F}_{\text{privRec}}$ in Appendix B.1. The communication complexity of the protocol is $O(Nn\kappa)$ bits.

For complete $t$-sharing $[s_1], \ldots, [s_N]_t$ and a public party $R \in \mathcal{P}$:

1. The trusted party receives the number $N$ of degree-$t$ Shamir sharing to be reconstructed from all parties and the set of corrupted parties $\mathcal{C} \subset \mathcal{P}$.
2. Upon receiving a request $(\text{Request}, \text{privRec}, R)$ 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 $S$.

Figure 2: Ideal functionality for private reconstruction.

**Preparing Random Sharings.** The description of functionality $\mathcal{F}_{\text{RandShare}}$ 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 proof of $\Pi_{\text{RandShare}}$ to realize $\mathcal{F}_{\text{RandShare}}$ is present in Appendix B.2. The communication complexity of the protocol is $O(N \cdot n^3 \kappa + n^4 \kappa^2 + n^6)$ bits.
achieved by the following three steps.

1. The trusted party receives the set \(C\) of corrupted parties and waits to receive a request \((\text{Request}, \text{RandShare}, N)\) 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 \(r_\ell\). (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 \(F_{\text{RandShare}}^0\) in Fig. 4. The protocol that realizes this functionality is similar to \(F_{\text{RandShare}}\) except that each party needs to share secrets with \(P_i\)'s shares equal to 0. We provide the protocol \(F_{\text{RandShare}}^0\) to realize this functionality and the security proof in Appendix B.3. The communication complexity of the protocol is \(\mathcal{O}(N \cdot n^3\kappa + n^2\kappa^2 + n^6)\) bits.

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).

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 \(s \in \mathbb{F}^L\). 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 \(s\). For each party \(P_i\), \(D\) randomly samples \(\kappa\) 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 $s$. 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 $s$ 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.

<table>
<thead>
<tr>
<th><strong>Functionality</strong> $\mathcal{F}_{\text{APICP}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialization Phase: Init</strong> $(T, (s^{(1)}, \ldots, s^{(m)}))$</td>
</tr>
<tr>
<td>1. The trusted party receives the identities of corrupted parties $C \subseteq P$.</td>
</tr>
<tr>
<td>2. Upon receiving $(\text{Init}, \text{APICP}, T, (s^{(1)}, \ldots, s^{(m)}))$ from $D$, the trusted party sends a request-based delayed output $(D, \text{APICP}, (s^{(1)}, \ldots, s^{(m)}))$ to $I$ and sets count $= T$.</td>
</tr>
<tr>
<td>3. Each time the trusted party receives a request $(\text{Request}, \text{APICP}, R, c)$ from an honest party, where $c = (c_1, \ldots, c_m)$. If count $&gt; 0$, the trusted party does the following things and replaces count by $\text{count} - 1$.</td>
</tr>
<tr>
<td>- If $I \in C$, the trusted party waits to receive an instruction from the ideal adversary $S$.</td>
</tr>
<tr>
<td>- If $S$ sends Ignore, the trusted party does nothing.</td>
</tr>
<tr>
<td>- If $S$ sends Proceed, if $D \in C$, the trusted party waits to receive $s'$ from $S$ and sends a request-based delayed output $s'$ to the receiver $R$. Otherwise, the trusted party sends a request-based delayed output $s = \sum_{k=1}^{m} c_k \cdot s^{(k)}$ to the receiver $R$.</td>
</tr>
<tr>
<td>- If $I \notin C$, the trusted party sends a request-based delayed output $s = \sum_{k=1}^{m} c_k \cdot s^{(k)}$ to the receiver $R$.</td>
</tr>
<tr>
<td>4. If $R$ is honest, $R$ outputs the results received from the trusted party. Corrupted parties may output anything they want.</td>
</tr>
</tbody>
</table>
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 $\alpha$ 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 $\Pi_{APICP}$ consists of $\Pi_{init}$ and $\Pi_{rev}$, which correspond to the initialization phase and revelation phase respectively.

$\Pi_{init}$ 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 $\Pi_{init}$ is $O(mL\kappa + mnT^2\kappa^2)$ bits.

$\Pi_{rev}$ is present in Fig. 7. All parties agree on the coefficients of a linear combination before the protocol is executed. 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. If at least \( t + 1 \) party sends a verification point that lies on the polynomial he receives from \( I \), he can believe that the linear combination of the vectors implied from the polynomial is sent from \( D \). \( \Pi_{\text{Rev}} \) can be executed \( t \) times per initialization of APICP, the communication complexity is \( O(LK + nT^2K^2) \) bits for each execution.

\[ \text{Parameter: The identity of } T, R, \text{ vector } c, \text{ counter } \text{count}, \text{ revelation times } \kappa, \text{ constant } L \text{ and security parameter } \kappa. \]

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 \( Z_{\text{count}}^{c} = \{(\alpha'_j, \sum_{k=1}^{m} c_k f^{(k)}(\alpha'_j))\}_{j \in Z_{\text{count}}} \) to \( R \), where \( j \in [(T + 1)\kappa] \).
3. \( R \) does the following computation:
   (a) Upon receiving the set \( Z_{\text{count}}^{c} \) from \( P_i \) and \( |Z_{\text{count}}^{c}| = (T + 1)\kappa \), \( R \) checks whether there exists at least one point \((\alpha'_j, \beta'_j) \in Z_{\text{count}}^{c} \) satisfies \( f(\alpha'_j) = \beta'_j \).
   (b) If for at least \( t + 1 \) parties, the above condition is satisfied, let \( s \) be the \( L \) highest coefficients of \( f(x) \), \( R \) outputs \( s \).

Theorem 5. The protocol \( \Pi_{\text{APICP}} \) realizes \( F_{\text{APICP}} \) with statistically security and \( O(mLk + mnT^2K^2 + LTk + nT^3K^2) \) bit communication.

We prove Theorem 5 in Appendix C.

5 The Asynchronous Secret Sharing Protocol (ACSS)

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

5.1 Our Instantiation of ACSS

All parties execute \( \Pi_{\text{Sh}}, \Pi_{\text{Ver}}, \Pi_{\text{Auth}} \) and \( \Pi_{\text{Comp}} \) protocols in sequence to realize our \( \Pi_{\text{ACSS}} \). Our \( \Pi_{\text{ACSS}} \) (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 \( \Pi_{\text{Sh}} \) 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 \( g(\alpha_{-1}), \ldots, g(\alpha_0) \) for each column polynomial \( g \) held by him. Each party then invokes \( \mathcal{F}_{\text{APICP}} \) 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 it be verified by all the parties. Regardless of the communication cost of \( \text{APICP} \), the communication complexity of \( \Pi_{\text{Sh}} \) is \( O(mL\kappa + n^3 \log n) \) bits.

**Protocol \( \Pi_{\text{Sh}} \)**

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

**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 \( \mathcal{F}_{\text{APICP}} \), 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:

1. For each \( k \in [m] \) and \( \ell \in [L'] \) do:
   
   1. Compute \( \text{idx} = ((k - 1) \cdot L' + \ell - 1) \cdot (t + 1) + 1 \).
   
   2. If \( k \in [m'] \), select a random degree-(\( t, 2t \)) bivariate polynomial \( F_l^{(k)}(x, y) \) s.t. for each \( i \in [0, \ell] \), \( F_l^{(k)}(x, \alpha_{-i}) = q_{\text{idx} + i}(x) \). Otherwise, select a random degree-(\( t, 2t \)) bivariate polynomial \( F_l^{(k)}(x, y) \).

   3. Send the column polynomial \( g_{\ell,i}^{(k)}(y) = F_l^{(k)}(x, y) \) 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 \( \text{OK}_i \) and set \( g_{\ell,i}^{(k)} = (g_{\ell,i}^{(k)}(y))_{k \in [m]} \).

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) \( \text{OK}_i \) is received from \( P_i \).
   
   2) \( (P_i, \text{APICP}, (g_{\ell,i}^{(1)}, \ldots, g_{\ell,i}^{(m)})) \) is received from \( \mathcal{F}_{\text{APICP}}(P_i, D) \) and \( g_{\ell,i}^{(k)}(y) = F_l^{(k)}(x, y) \) for each \( k \in [m] \). Here \( F_l^{(k)}(x, y) = (F_l^{(k)}(x, y))_{k \in [L']} \).

   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) \( \text{OK}_i \) is received from all \( P_j \in \mathcal{M} \).

**Figure 9:** The protocol of the \( \Pi_{\text{Ver}} \)

The verification phase \( \Pi_{\text{Ver}} \) 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 \( \mathcal{F}_{\text{APICP}}(P_i, D) \) for all the parties \( P_i \in \mathcal{M} \). Regardless of the communication cost of \( \text{APICP} \), the communication complexity of \( \Pi_{\text{Ver}} \) is \( O(n^3 \kappa) \) bits.

**Protocol \( \Pi_{\text{Ver}} \)**

**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 $(\text{Request}, \text{APICP}, P_i, (r_{i1}, r_{i2}, \ldots, r_{in}))$ to $\mathcal{F}_{\text{APICP}}(P_h, D)$ for all $P_h \in \mathcal{M}$.

3. Upon receiving $g_{*,h}$ from $\mathcal{F}_{\text{APICP}}(P_h, D)$ for all $P_h \in \mathcal{M}$, $P_i$ accepts $\{g_{i,h}^{(k)}(y)\}_{k \in [m]}$ if the following hold for each $\ell \in [L]$:
   1. For each $P_h \in \mathcal{M}$, parse $g_{*,h}$ into $\{g_{\ell,k}(y)\}_{k \in [m]}$, each $g_{\ell,k}$ is a degree-2t polynomial.
   2. There exists a degree-($t, 2t$) bivariate polynomial $F_\ell(x, y)$ s.t. $F_\ell(\alpha_i, y) = g_{\ell,h}(y)$ for all $P_h \in \mathcal{M}$ and $F_\ell(\alpha, y) = g_{\ell,i}(y)$.

---

The authentication phase $\Pi_{\text{Auth}}$ 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 $(P_i, P_j)$. 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 $P_j \in \mathcal{P}$, 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, the protocol won’t terminate in the end and all the honest parties won’t get their shares. If the public verification passes, the ACSS protocol is guaranteed to terminate eventually, and all the honest parties will obtain their shares. Regardless of the communication cost of APICP, the communication complexity of $\Pi_{\text{Auth}}$ is $O(Ln^3\kappa + mn^6\kappa + n^7\kappa^2 + n^8)$ bits (including the communication to realize the functionalities of sub-protocols except for $\mathcal{F}_{\text{APICP}}$).
$P_h \in \mathcal{M}$.  

3. Upon accepting $\{g_{\ast,h}\}_{P_h \in \mathcal{M}}$, $P_{\alpha}$ computes $g_{\ell,j} = (F_\ell(\alpha_j, \alpha_1), \ldots, F_\ell(\alpha_j, \alpha_n))$ for each $\ell \in [L']$ and $g_{\ast,j} = (g_1, \ldots, g_{L'}, j)$.  

4. All the parties jointly prepare a sharing $[\gamma_{i \rightarrow v,j}]_t$. $P_{\alpha}$ computes his share by:

$$[\gamma_{i \rightarrow v,j}]_t = g_{\ast,j} \cdot [\mu_{i \rightarrow v}]_t + \sum_{k=1}^{m} r_{i \rightarrow v,j}^k \cdot \left([\nu_{i \rightarrow v}]_t + \sum_{u=1}^{t} e_{u,j} \cdot [\tau_{u}^{(k)}]_t\right)$$

$$+ \text{[mask]}_t$$

Here $e_{u,j}$ is $P_j$’s share of $[e_u]_t$ which is public. Notice that $P_j$’s share of $[\gamma_{i \rightarrow v,j}]_t$ should be $\tau_{i \rightarrow v,j}$.  

5. $P_{\alpha}$ sends his share of $[\gamma_{i \rightarrow v,j}]_t$ and (Request, privRec, $P_i$) to $F_{\text{privRec}}$.  

(4). Upon receiving the whole $[\gamma_{i \rightarrow v,j}]_t$ from $F_{\text{privRec}}$, $P_i$ accepts $\{\tau_{i \rightarrow v,j}\}_{k \in [m]}$ if $P_j$’s share of $[\gamma_{i \rightarrow v,j}]_t$ is equal to $\tau_{i \rightarrow v,j}$.

3. **Reconstructing $P_i$’s tags**:  

1. Upon accepting $2t + 1$ different $P_j$’s $\{\tau_{j \rightarrow i}^{(k)}\}_{k \in [m]}$, $P_i$ reconstructs $\{\tau_{j \rightarrow i}^{(k)}\}_{k \in [m]}$ with these shares.

**Preparing the $W$ set**: $D$ executes the following code:  

1. Each $P_i \in P$ broadcasts $\text{Tag}_i$ after getting $\{\tau_{i \rightarrow v,j}\}_{k \in [m]}$ for all $P_v \in P$ and accepting $\{g_{i \rightarrow v}^{(k)}(y)\}_{k \in [m], \ell \in [L']}$ during the verification phase.  

2. $D$ initializes a set $W$ to $\emptyset$.  

3. Upon receiving $\text{Tag}_i$ from $P_i$, $D$ includes $P_i$ into $W$.  

4. $D$ broadcasts $W$ when $|W| \geq 2t + 1$.

**Verifying the $W$ set**: Each party executes the following code:  

1. $W$ is received from $D$ and $|W| \geq 2t + 1$.  

2. $\text{Tag}_i$ is received from all $P_i \in W$.  

3. Each party moves to the next phase if the above conditions are met.

Figure 11: The protocol of the $\Pi_{\text{auth}}$  

The completion phase $\Pi_{\text{Comp}}$ 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 $\Pi_{\text{Comp}}$ is $O(mLnk + mn^3k + L'n^3k)$ bits (including the communication to realize the functionalities of sub-protocols).

**Protocol $\Pi_{\text{Comp}}$**

**Completion Phase**

**Reconstructing row polynomials**:  

For each $P_v \in P$, do the following:  

1. Each $P_i \in W$ sends $\{g_{i \rightarrow v}^{(k)}(\alpha_e)\}_{k \in [m], \ell \in [L']}$ to $P_v$.  

2. For each $P_i \in W$, do the following:  

   1. Each party sends his shares of $[\mu_{i \rightarrow v}]_t$, $[r_{i \rightarrow v}]_t$, $[\nu_{i \rightarrow v}]_t$ and (Request, privRec, $P_i$) to $F_{\text{privRec}}$.  

      Upon receiving $[\mu_{i \rightarrow v}]_t$, $[\nu_{i \rightarrow v}]_t$, $[\tau_{i \rightarrow v}]_t$ from $F_{\text{privRec}}$, $P_v$ reconstructs the secrets $\mu_{i \rightarrow v}$, $\{\nu_{i \rightarrow v}\}_{k \in [m]}$.  

   2. Upon receiving $\{g_{i \rightarrow v}^{(k)}(\alpha_e)\}_{k \in [m], \ell \in [L']}$ from $P_i$, $P_v$ sends a random element $r_{i \rightarrow v} \in \mathbb{F}$ to $P_i$.  

   3. Upon receiving $r_{i \rightarrow v}$, $P_i$ sends $\tau_{i \rightarrow v} = \sum_{k=1}^{m} r_{i \rightarrow v}^k \cdot \tau_{i \rightarrow v}^{(k)}$ and $g_{\ell,i}(y) = \sum_{k=1}^{m} r_{i \rightarrow v}^k \cdot g_{\ell,i}^{(k)}(y)$ for each $\ell \in [L']$ to $P_v$.  

   4. Upon receiving $\tau_{i \rightarrow v}$ and $\{g_{\ell,i}(y)\}_{\ell \in [L']}$ from $P_v$, $P_v$ computes $g_{\ell,i} = (g_{\ell,i}(\alpha_1), \ldots, g_{\ell,i}(\alpha_n))$ for each $\ell \in [L']$ and $g_{\ast,i} = (g_{1,i}, \ldots, g_{L', i})$.  

   5. $P_i$ accepts $\{g_{\ell,i}^{(k)}(\alpha_e)\}_{k \in [m], \ell \in [L']}$ if the following hold:  

      1. For each $\ell \in [L']$, the degree of $g_{\ell,i}(y)$ is $2t$ and $g_{\ell,i}(\alpha_e) = \sum_{k=1}^{m} r_{i \rightarrow v}^k \cdot g_{\ell,i}^{(k)}(\alpha_e)$.  

      2. $\tau_{i \rightarrow v} = g_{\ell,i} - \mu_{i \rightarrow v} + \sum_{k=1}^{m} r_{i \rightarrow v}^k \cdot \nu_{i \rightarrow v}^{(k)}$.  

3. Upon accepting $t + 1$ different $P_i$’s $\{g_{\ell,i}^{(k)}(\alpha_e)\}_{k \in [m], \ell \in [L']}$, $P_v$ reconstructs a degree-$t$ polynomial $f_{\ell,i}^{(k)}(x)$ s.t. $f_{\ell,i}^{(k)}(\alpha_i) = g_{\ell,i}^{(k)}(\alpha)$ 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 $(f_{t,v}^{k}(\alpha))_{k \in [m], \ell \in [L']}$ to $P_w$.
2. For each $P_v \in \mathcal{P}$:
   1. Upon receiving $(f_{t,v}^{k}(\alpha))_{k \in [m], \ell \in [L']}$, from $P_v$, $P_w$ broadcasts a random value $r_{v \to w} \in \mathbb{F}$.
   2. Upon receiving $r_{v \to w}$, each $P_v \in \mathcal{W}$ sends $\tau_{v \to w} = \sum_{k=1}^{m} r_{v \to w}^{k} f_{t,v}^{k}(\alpha)$ and $g_{t,\ell}(y) = \sum_{k=1}^{m} r_{v \to w}^{k} g_{t,\ell}^{k}(y)$ for each $\ell \in [L']$ to $P_w$.
   3. Upon receiving $(g_{t,\ell}(y))_{\ell \in [L']}, P_w$ computes $g_{t,\ell}(y) = (g_{t,\ell}(\alpha_1), \ldots, g_{t,\ell}(\alpha_n))$ for each $\ell \in [L']$ and $g_{*,\ell} = (g_{*,\ell}(1), \ldots, g_{*,\ell}(L'))$.
   4. $P_w$ accepts $(g_{t,\ell}(y))_{\ell \in [L']}$ if the following hold.
      1. $g_{t,1}(y), \ldots, g_{t,L'}(y)$ are all of degree $2t$.
      2. $\tau_{v \to w} = g_{*,\ell} + \mu_{v \to w} + \sum_{k=1}^{m} r_{v \to w}^{k} f_{t,v}^{k}(\alpha)$.
   5. Upon accepting $t + 1$ different $P_v$’s $(g_{t,\ell}(y))_{\ell \in [L']}, P_w$ reconstructs a degree-$(t,2t)$ bivariate polynomial $F_\ell(x,y)$ such that $F_\ell(\alpha, y) = g_{t,\ell}(y)$ for each $\ell \in [L']$.
   6. $P_w$ accepts $(f_{t,v}^{k}(\alpha))_{k \in [m], \ell \in [L']} if $\sum_{v=1}^{m} r_{v \to w} f_{t,v}^{k}(\alpha) = F_\ell(\alpha, w)$ for each $\ell \in [L']$.
3. Upon accepting $2t + 1$ different $(f_{t,v}^{k}(\alpha))_{k \in [m], \ell \in [L']}, P_w$ reconstructs a degree-$2t$ polynomial $g_{t,\ell}^{k}(y)$ such that $g_{t,\ell}^{k}(\alpha) = f_{t,v}^{k}(\alpha)$ for each $\ell \in [L']$ and $k \in [m]$.
4. $P_w$ outputs $(g_{t,w}^{k}(\alpha))_{\ell \in [L'], k \in [m], \ell \in [t,0]}$.

Figure 12: The protocol of the $\Pi_{\text{Comp}}$

**Lemma 2.** The protocol $\Pi_{\text{ACSS}}$ t-securely realizes $F_{\text{ACSS}}$ in the $(F_{\text{APICP}}, F_{\text{privRec}}, F_{\text{RandShare}}, F_{\text{RandShare}}^n)$-hybrid model with statistical security.

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

**Theorem 1.** Let $\kappa$ denote the security parameter. For a finite field $\mathbb{F}$ of size $2^{O(\kappa)}$, 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 $O(N\kappa + n^2 \kappa)$ bits.

The proof of Lemma 2 is given in Appendix D.2. Detailed analysis of the communication complexity of our $\Pi_{\text{ACSS}}$ is given in Appendix D.3.

**Construction of AMPC.** The previous work [CP17] presents a framework using $F_{\text{ACSS}}$ to construct an AMPC protocol $\Pi_{\text{AMPC}}$. We give the ideal functionality $F_{\text{AMPC}}$ in Appendix E.1 and an overview of how to compile $F_{\text{ACSS}}$ to $\Pi_{\text{AMPC}}$ in Appendix E.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 $F_{\text{ACSS}}$-hybrid model. The achieved communication complexity is $O(|C| \cdot n^2 \kappa + n^3 \kappa)$ bits plus $O(n)$ invocations of $F_{\text{ACSS}}$ to share $O(|C| \cdot n)$ degree-$t$ Shamir sharings in total.

Replacing $F_{\text{ACSS}}$ with our construction of $\Pi_{\text{ACSS}}$, 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 $O(|C| \cdot n^2 \kappa + n^3 \kappa^2)$ bits.

**References**


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 \( P \) be a set of \( n \) parties and let \( Q \) be a property that can be influenced by multiple protocols running in parallel. Every party \( P_i \in P \) can decide for every party \( P_j \in P \) 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 \((P_i, P_j) \in P\) we have that \( P_i \) will eventually like \( P_j \).

**Definition 2.** Let \( \Pi \) 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 \( \Pi \) 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 \( \Pi_{ACS}^Q \) 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 \( \ell \)-bit message requires \( O(n^2 \ell) \)-bit communication.

---

**Functionality \( F_{ACast} \)**

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

---

B Constructions and Security Proofs for Sub-protocols

B.1 Construction of \( \Pi_{privRec} \)

We give our construction of \( \Pi_{privRec} \) in Fig. 14.

---

**Protocol \( \Pi_{privRec} \)**

1. Upon receiving the input shares of \([s_1], \ldots, [s_N]\) 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 \( W \) denote the set of parties in \( P \) from whom \( R \) has received the shares. Wait until \( |W| = 2t + 1 + r \).
   (b). Check whether there exists degree-\( t \) sharing polynomials \( p_1(\cdot), \ldots, p_N(\cdot) \), such that for \( 2t + 1 \) parties in \( W \), their shares of \([s_1], \ldots, [s_N]_i \) are on these polynomials respectively. If so, output the whole sharing \([s_1], \ldots, [s_N]_i \). Otherwise, proceed to the next iteration.

---

Figure 14: The protocol for private reconstruction
Lemma 4. Protocol $\Pi_{\text{privRec}}$ t-securely realizes $F_{\text{privRec}}$.

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

Each time an honest party needs to send a message to another honest party, $S$ will tell $A$ that a message has been delivered such that $A$ can tell $S$ the arrival time of this message to help $S$ 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 $S$ delay the output in default until we say $S$ 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 $S$.

If $R$ is corrupted:

<table>
<thead>
<tr>
<th>Simulator $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$ receives the whole sharing $[s_1], \ldots, [s_N]$ from $F_{\text{privRec}}$. For each honest $P_i$, $S$ sends the $P_i$’s shares of $[s_1], \ldots, [s_N]$ to $R$ on behalf of $P_i$.</td>
</tr>
</tbody>
</table>

Hybrid arguments:

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

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

Note that $\text{Hyb}_1$ is the ideal-world scenario, $\Pi_{\text{privRec}}$ securely computes $F_{\text{privRec}}$.

If $R$ is honest:

<table>
<thead>
<tr>
<th>Simulator $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. For each corrupted $P_i$, $S$ receives his shares of $[s_1], \ldots, [s_N]$ from $F_{\text{privRec}}$.</td>
</tr>
<tr>
<td>2. For each corrupted $P_i$, $S$ receives their messages $[s_1], \ldots, [s_N]$ sent to $R$. When the message arrives, $S$ accept $P_i$’s shares if his share of $s_j$ equals to $[s_j]$ for all $j \in [N]$. For each honest $P_i$, $S$ regards that he accepts $P_i$’s shares when the shares arrive.</td>
</tr>
<tr>
<td>3. After accepting $2t + 1$ parties’ shares, $S$ allows $F_{\text{privRec}}$ to send the output to $R$.</td>
</tr>
</tbody>
</table>

Hybrid arguments:

$\text{Hyb}_1$: In this hybrid, $R$ gets his output from $F_{\text{privRec}}$ instead of computing the output by himself. By the error-correction property of Reed-Solomon Codes, $R$ will eventually get the correct secrets $s_1, \ldots, s_N$. 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, $\text{Hyb}_1$ and $\text{Hyb}_0$ have the same output distribution.

Note that $\text{Hyb}_1$ is the ideal-world scenario, $\Pi_{\text{privRec}}$ securely computes $F_{\text{privRec}}$.

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

Figure 15: Simulator for the $F_{\text{privRec}}$

Figure 16: Simulator for the $F_{\text{privRec}}$
B.2 Construction of $\Pi_{\text{RandShare}}$

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

**Protocol $\Pi_{\text{RandShare}}$**

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

1. Each party $P_i \in \mathcal{P}$ samples $L = N/(t + 1)$ random value $(s_{j_1}^{(1)}, \ldots, s_{j_1}^{(L)}) \in \mathbb{F}^L$ and chooses random degree-$t$ polynomials $q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)$ such that for each $\ell \in [L]$, $q_j^{(\ell)}(\beta_0) = s_{j_1}^{(\ell)}$.
2. Each party $P_i$ sends $(\text{Dealer}, \mathcal{ACSS}, \{q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)\})$ to $\mathcal{F}_{\text{ACSS}}$.
3. Let the ACS property $Q$ defined by $P_i$ be $P_k$ if $P_i$ terminates $\mathcal{F}_{\text{ACSS}}$ whose dealer is $P_k$. Then all parties run $\Pi_{\text{ACSS}}^Q$ to get a set $S$ of size $2t + 1$. Let the sharing generated by $\mathcal{F}_{\text{ACSS}}$ with dealers in $S$ be $\{[s_j^{(1)}]_t\}_{j \in [2t+1], i \in [L]}$, where $S = \{P_{a_1}, \ldots, P_{a_{2t+1}}\}$.
4. Let $M$ be the $(t + 1) \times (2t + 1)$ Vandermonde matrix

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

where $1, b_1, \ldots, b_{2t+1}$ are $2t + 1$ distinct public elements in $\mathbb{F}$.

Each party locally computes his shares by

$$
\begin{pmatrix}
[r_1^{(1)}]_t \\
\vdots \\
r_t^{(1)} \\
[r_1^{(L)}]_t \\
\vdots \\
r_t^{(L)}
\end{pmatrix} = M \cdot
\begin{pmatrix}
[s_1^{(1)}]_t, \ldots, [s_1^{(L)}]_t \\
\vdots \\
[s_t^{(1)}]_t, \ldots, [s_t^{(L)}]_t
\end{pmatrix}
$$

5. Each party outputs his shares of $\{[r_j^{(1)}]_t\}_{j \in [2t+1], i \in [L]}$.

**Figure 17**: The protocol to prepare random $t$-sharing

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

1. For each honest $P_i$, $\mathcal{S}$ follows the protocol to emulate $\mathcal{F}_{\text{ACSS}}$ where the dealer is $P_i$, such that corrupted parties get their shares of $[s_j^{(1)}]_t, \ldots, [s_j^{(L)}]_t$, 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 $\mathcal{F}_{\text{ACSS}}$.
2. $\mathcal{S}$ follows the protocol to run $\Pi_{\text{ACSS}}^Q$ 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 $\mathcal{F}_{\text{ACSS}}$ are $q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)$ for $j = 1, \ldots, k$. Take the share of
\[ [s^{(i)}_{st}] = [q^{(i)}_{st} (\beta_t)]_s \text{ for each } P_t \text{ as } q^{(i)}_{st} (\beta_t) \text{ for all } j \in [k], \ell \in [L] \]

4. For each corrupted party, \( S \) follows the protocol to compute his shares by

\[
\begin{pmatrix}
[r^{(1)}_1, \ldots, r^{(L)}_1] \\
\vdots \\
[r^{(1)}_{t+1}, \ldots, r^{(L)}_{t+1}]
\end{pmatrix}
= M \cdot
\begin{pmatrix}
[s^{(1)}_{st}, \ldots, s^{(L)}_{st}] \\
\vdots \\
[s^{(1)}_{st+1, \ell}, \ldots, s^{(L)}_{st+1, \ell}]
\end{pmatrix}
\]

and sends his shares of \( \{r^{(i)}_{st}\}_{\ell \in [L]} \) to \( F_{\text{RandShare}} \).

5. For each honest \( P_t \), when \( P_t \) gets the set \( S \) after running \( \Pi^*_{\text{ACS}} \), \( S \) allows \( F_{\text{RandShare}} \) to send the output to \( P_t \).

Figure 18: Simulator for the \( F_{\text{RandShare}} \)

Hybrid arguments:

\textbf{Hyb}_1: In this hybrid, \( S \) runs the protocol honestly. This corresponds to the real-world scenario.

\textbf{Hyb}_0: In this hybrid, honest parties get their shares from \( F_{\text{RandShare}} \) instead of computing their shares by themselves. Note that in both \textbf{Hyb}_1 and \textbf{Hyb}_0, the honest parties' shares are on degree-\( t \) polynomials, so we only need to show that in \textbf{Hyb}_1, each \( r^{(i)}_a \) 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 \( S \backslash \mathcal{H} = \{P_c_1, \ldots, P_{c_{t}}\} \). For each \( P_{h_j} \in \mathcal{H} \), take the \( j \)-th column of \( M \) out, and let the \( t + 1 \) columns form a Vandermonde matrix \( M_H \), and let the other \( t \) columns of \( M \) form a matrix \( M_C \). Note that

\[
\begin{pmatrix}
r^{(1)}_1, \ldots, r^{(L)}_1 \\
\vdots \\
r^{(1)}_{t+1}, \ldots, r^{(L)}_{t+1}
\end{pmatrix}
= M_H \cdot
\begin{pmatrix}
s^{(1)}_{h_1}, \ldots, s^{(L)}_{h_1} \\
\vdots \\
s^{(1)}_{h_{t+1}}, \ldots, s^{(L)}_{h_{t+1}}
\end{pmatrix}
+ M_C \cdot
\begin{pmatrix}
s^{(1)}_{c_1}, \ldots, s^{(L)}_{c_1} \\
\vdots \\
s^{(1)}_{c_{t}}, \ldots, s^{(L)}_{c_{t}}
\end{pmatrix}.
\]

Since \( M_H \) is invertible and each \( s^{(i)}_{h_j} \) where \( j \in [t+1], \ell \in [L] \) is randomly sampled in \( F \) by \( S \) when emulating \( F_{\text{ACS}} \) where \( P_{h_j} \) is the dealer,

\[
M_H \cdot
\begin{pmatrix}
s^{(1)}_{h_1}, \ldots, s^{(L)}_{h_1} \\
\vdots \\
s^{(1)}_{h_{t+1}}, \ldots, s^{(L)}_{h_{t+1}}
\end{pmatrix}
\]

is completely random. Thus,

\[
\begin{pmatrix}
r^{(1)}_1, \ldots, r^{(L)}_1 \\
\vdots \\
r^{(1)}_{t+1}, \ldots, r^{(L)}_{t+1}
\end{pmatrix}
\]

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

Note that \textbf{Hyb}_1 is the ideal-world scenario, \( \Pi_{\text{RandShare}} \) securely computes \( F_{\text{RandShare}} \) in the \( (F_{\text{ACS}}, F_{\text{privRec}}) \)-hybrid model.

We need to invoke \( F_{\text{ACS}} \) \( L \) times for each party. Thus the protocol \( \Pi_{\text{RandShare}} \) requires \( \mathcal{O}(L \cdot n^4 K + n^5 K + n^6) = \mathcal{O}(N \cdot n^3 K + n^5 K + n^6) \)-bit communication.

\[ \square \]

\textbf{B.3 Construction of} \( \Pi^0_{\text{RandShare}} \)

To construct our protocol \( \Pi^0_{\text{RandShare}} \), we first need a sub-protocol to prepare random coins in \( F \). For this, all parties can invoke \( F_{\text{RandShare}} \) to generate a random share and then invoke \( F_{\text{privRec}} \) to reconstruct the random
value to all parties. The functionality is given in Fig. 19. The amortized communication complexity of generating per coin is $\mathcal{F}_\text{Coin}$ is $\mathcal{O}(n^2 \kappa)$ bits.

**Functionality $\mathcal{F}_\text{Coin}$**

1. Upon receiving (Request, Coin) from $2t + 1$ parties, 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 $\mathcal{O}(N(n^3 \kappa + n^5 \kappa^2 + n^6))$-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 $\Pi^0_{\text{RandShare}}$ in the $(\mathcal{F}_\text{ACSS}, \mathcal{F}_\text{Coin}, \mathcal{F}_\text{privRec})$-hybrid model in Fig. 20.

**Protocol $\Pi^0_{\text{RandShare}}$**

On public parameters $N$, $\mathcal{P}$, $(\beta_0, \ldots, \beta_n)$:

1. Each party $P_i \in \mathcal{P}$ samples $L = N/(t+1)$ random value $(s_1^{(1)}, \ldots, s_L^{(1)}) \in \mathbb{F}^L$ and chooses random degree-$t$ polynomials $q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)$ such that for each $\ell \in [L]$, $q_j^{(\ell)}(\beta_0) = s_j^{(\ell)}(L)$ and $q_j^{(\ell)}(\beta_i) = 0$.
2. Each party $P_j$ sends $(\text{Dealer, ACSS}, \{q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)\})$ to $\mathcal{F}_\text{ACSS}$.
3. Upon terminating $\mathcal{F}_\text{ACSS}$ whose dealer is $P_j$, each party sends (Request, Coin) to $\mathcal{F}_\text{Coin}$ to get a random value $r_j$. Then each party sends his share of $[s_j]_L = \sum_{\ell=1}^L r_j \cdot [s_j]^{(\ell)}_L$ and $(\text{Request, privRec, } P_1, \ldots, \text{Request, privRec, } P_n)$ to $\mathcal{F}_\text{privRec}$ to reconstruct the whole sharing $[s_j]_L$.
4. Let the ACS property $Q$ defined by $P_h$ likes $P_h$ if $P_j$ terminates $\mathcal{F}_\text{ACSS}$ whose dealer is $P_h$ and $P_i$'s share of $s_h$ is 0. Then all parties run $\Pi^0_{\text{ACSS}}$ to get a set $S$ of size $2t + 1$. Let the sharing generated by $\mathcal{F}_\text{ACSS}$ with dealers in $S$ be $\{s_i^{(1)}\}_{i \in [2t+1], \ell \in [L]}$, where $S = \{P_{a_1}, \ldots, P_{a_{2t+1}}\}$.
5. 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

\[
\begin{pmatrix}
[r_1^{(1)}]_L, \ldots, [r_1^{(L)}]_L \\
\vdots \\
[r_{t+1}^{(1)}]_L, \ldots, [r_{t+1}^{(L)}]_L
\end{pmatrix} = M \cdot \begin{pmatrix}
[s_1^{(1)}]_L, \ldots, [s_1^{(L)}]_L \\
\vdots \\
[s_{2t+1}^{(1)}]_L, \ldots, [s_{2t+1}^{(L)}]_L
\end{pmatrix}
\]

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

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

**Lemma 6.** Protocol $\Pi^0_{\text{RandShare}}$ $t$-securely realizes $\mathcal{F}^0_{\text{RandShare}}$ in the $(\mathcal{F}_\text{ACSS}, \mathcal{F}_\text{Coin}, \mathcal{F}_\text{privRec})$-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 $A$. For simplicity, we just let $\mathcal{S}$ communicate with $A$ 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 $A$, and likewise, every message sent from $A$ sends to $\mathcal{Z}$ is forwarded by $\mathcal{S}$.
Each time an honest party needs to send a message to another honest party, $S$ will tell $A$ that a message has been delivered such that $A$ can tell $S$ the arrival time of this message to help $S$ 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 $S$ delay the output in default until we say $S$ 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 $S$.

**Simulator $S$**

1. For each honest $P_i$, $S$ follows the protocol to simulate $F_{ACSS}$ where the dealer is $P_i$ such that corrupted parties get their shares of $[s_1^j], \ldots, [s_{L}^j]$, where $L = N/(t+1)$. For each corrupted $P_i$, $S$ receives the input of $P_i$ from $A$ and follows the protocol to simulate $F_{ACSS}$.

2. For each honest $P_j$ or corrupted $P_j$ whose input polynomials $q_1^{(j)}(\cdot), \ldots, q_{L}^{(j)}(\cdot)$ are all degree-$t$, $S$ emulate $F_{Coin}$ to sample random coin $r_j$. Then $S$ computes the shares of corrupted parties of $s_j = \sum_{\ell = 1}^{L} r_{j}^{(\ell)} s_{j}^{(\ell)}$ and emulates $F_{privRec}$ to get the whole sharing $[s_j]_t$.

3. If there exists a corrupted $P_j$ such that $P_j$’s share of $s_j$ equals to 0 for each $\ell \in [L]$ but for some $q_{j}^{(\ell)}(\cdot)$, $q_{j}^{(\ell)}(\beta_j) \neq 0$, then $S$ aborts the simulation.

4. $S$ follows the protocol to run $\Pi_{ACSS}^Q$ 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 $P$.

5. Suppose $P_{a_j}$’s input polynomials to $F_{ACSS}$ are $q_j^{(1)}(\cdot), \ldots, q_j^{(L)}(\cdot)$ for $j = 1, \ldots, k$. Take the share of $[s_{a_j}]_t = [q_j^{(\ell)}(\beta_j)]_t$ for each $P_j$ as $q_{j}^{(\ell)}(\beta_j)$ for all $j \in [k], \ell \in [L]$

6. For each corrupted party, $S$ follows the protocol to compute his shares

$$
\begin{bmatrix}
[r_1^{(1)}]_t, \ldots, [r_1^{(L)}]_t \\
\vdots \\
[r_t^{(1)}]_t, \ldots, [r_t^{(L)}]_t
\end{bmatrix} = M \cdot 
\begin{bmatrix}
[s_1^{(1)}]_t, \ldots, [s_1^{(L)}]_t \\
\vdots \\
[s_{2t+1}^{(1)}]_t, \ldots, [s_{2t+1}^{(L)}]_t
\end{bmatrix},
$$

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

6. For each honest $P_i$, when $P_i$ gets the set $S$ after running $\Pi_{ACSS}^Q$, $S$ allows $F_{RandShare}$ to send the output to $P_i$.

**Figure 21:** Simulator for the $F_{RandShare}$

Hybrid arguments:

**Hyb$_0$**: In this hybrid, $S$ runs the protocol honestly. This corresponds to the real-world scenario.

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

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

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

**Hyb$_2$**: In this hybrid, honest parties get their shares from $F_{RandShare}$ instead of computing their shares by themselves. Note that in both $\text{Hyb}_1$ and $\text{Hyb}_2$, 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 $A$ knows the random sharing of any honest party. For other cases, we only need to show that in $\text{Hyb}_1$, 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 $S \setminus \mathcal{H} = \{P_{c_1}, \ldots, P_{c_t}\}$. 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 $M_C$. Note that
\[
\begin{pmatrix}
  r^{(1)}_1, \ldots, r^{(L)}_1 \\
  \vdots \\
  r^{(1)}_{t+1}, \ldots, r^{(L)}_{t+1}
\end{pmatrix}
= M \cdot
\begin{pmatrix}
  s^{(1)}_{a_1}, \ldots, s^{(L)}_{a_1} \\
  \vdots \\
  s^{(1)}_{a_{2t+1}}, \ldots, s^{(L)}_{a_{2t+1}}
\end{pmatrix}
+ M_H \cdot
\begin{pmatrix}
  s^{(1)}_{h_1}, \ldots, s^{(L)}_{h_1} \\
  \vdots \\
  s^{(1)}_{h_{t+1}}, \ldots, s^{(L)}_{h_{t+1}}
\end{pmatrix}
+ M_C \cdot
\begin{pmatrix}
  s^{(1)}_{c_1}, \ldots, s^{(L)}_{c_1} \\
  \vdots \\
  s^{(1)}_{c_L}, \ldots, s^{(L)}_{c_L}
\end{pmatrix}.
\]
Since $M_H$ is invertible and each $s^{(\ell)}_{h_j}$ where $j \in [t+1]$, $\ell \in [L]$ is randomly sampled in $\mathbb{F}$ by $S$ when emulating $F_{ACSS}$ where $P_{h_j}$ is the dealer,

\[
M_H \cdot
\begin{pmatrix}
  s^{(1)}_{h_1}, \ldots, s^{(L)}_{h_1} \\
  \vdots \\
  s^{(1)}_{h_{t+1}}, \ldots, s^{(L)}_{h_{t+1}}
\end{pmatrix}
\]
is completely random. Thus,

\[
\begin{pmatrix}
  r^{(1)}_1, \ldots, r^{(L)}_1 \\
  \vdots \\
  r^{(1)}_{t+1}, \ldots, r^{(L)}_{t+1}
\end{pmatrix}
\]
is also completely random. Thus, $\text{Hyb}_2$ and $\text{Hyb}_1$ have the same output distribution.

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

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

\section{Security Proof of The Asynchronous Packed Information-Checking Protocol}

\textbf{Proof.} We prove the security of the APICP protocol by constructing an ideal adversary $S$. $S$ needs to interact with the environment $Z$ and with the ideal functionalities. $S$ constructs virtual real-world honest parties and runs the real-world adversary $A$. For simplicity, we just let $S$ communicate with $A$ on behalf of honest parties and the ideal functionality of sub-protocols in our proof. In order to simulate the communication with $Z$, every message that $S$ receives from $Z$ is sent to $A$, and likewise, every message sent from $A$ sends to $Z$ is forwarded by $S$. Each time an honest party needs to send a message to another honest party, $S$ will tell $A$ that a message has been delivered such that $A$ can tell $S$ the arrival time of this message to help $S$ 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 $S$ delay the output in default until we say $S$ 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 $S$ is as follows. If we say that $S$ delivers a message, $S$ just tells $A$ that the message has been delivered. $S$ may not be able to know the context of the message.

When $D$ and $I$ are honest:
1. For each corrupted party $P_i$, $S$ randomly samples $(T + 1)T\kappa$ verification points $z = (\alpha, \beta_1, \ldots, \beta_m)$ (each verification point is corresponding to $c, f(1)(\alpha), \ldots, f(m)(\alpha)$, 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 $(T + 1)^2\kappa$ verification points are not distinct, $S$ aborts the simulation.

2. For each $P_i \in \mathcal{P}$, if $P_i$ is corrupted, $S$ waits to receive a verification set $Z_{T^2 + 1}$ 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 $P_i$, $S$ considers that $I$ receives a correct verification set.

3. When $I$ receives at least $2t + 1$ correct verification sets, he initializes a counter $count = T$ and a set $W = \emptyset$. Then, $S$ allows $F_{\text{APICP}}$ to send the output to $I$.

4. For each revelation, if $count > 0$, $S$ does the following things and replaces $count$ by $count - 1$:
   - If $R$ is honest:
     a. For each verification point $z = (\alpha, \beta_1, \ldots, \beta_m)$ $S$ has sent to a corrupted party, $S$ computes a new point $(\alpha, \sum_{k=1}^{m} c_k \beta_k)$. Assuming all these new points form a set $M$. Then $S$ checks
       $$\forall (b). W = \{(c_1, f_1(x)), \ldots, (c_k, f_k(x))\},$$
     where $c_1, \ldots, c_k$ are linear dependent, assume $c = \sum_{j=1}^{k} a_j c_j$, then $S$ computes $f(x) = \sum_{j=1}^{k} a_j f_j(x)$.
   b. $\forall P_i \in \mathcal{P}$, if $P_i$ is corrupted, $S$ waits to receive a verification set $Z_{\text{count}}$ from $A$. If $c, c_1, \ldots, c_k$ are linear independent, $S$ checks whether at least one point in $Z_{\text{count}}$ is in $M$. Otherwise, $S$ checks whether at least one point in $Z_{\text{count}}$ on $f(x)$. If the check passes, $S$ considers that $R$ receives a correct verification set. If $P_i$ is honest, $S$ regards that $R$ receives a correct verification set $Z_{\text{count}}$ from $P_i$ when the set arrives.
   c. After receiving $t + 1$ correct verification sets, $S$ allows $F_{\text{APICP}}$ to send the output to $R$.
   - If $R$ is corrupted:
     a. $S$ waits to receive $s$ from $F_{\text{APICP}}$. For each verification point $z = (\alpha, \beta_1, \ldots, \beta_m)$ $S$ sent to corrupted $P_i$, $S$ computes a new point $(\alpha, \sum_{k=1}^{m} c_k \beta_k)$. Since the each $\alpha$ of these points are distinct, $S$ has $(T + 1)^2\kappa$ new points. Then $S$ checks $W = \{(c_1, f_1(x)), \ldots, (c_k, f_k(x))\}$.
       - If $c, c_1, \ldots, c_k$ are linear dependent, assume $c = \sum_{j=1}^{k} a_j c_j$, then $S$ gets a new polynomial $f(x) = \sum_{j=1}^{k} a_j f_j(x)$.
       - If $c, c_1, \ldots, c_k$ are linear independent, $S$ samples a random degree-$(T + 1)^2\kappa + L$ polynomial $f(x)$ whose $L$ highest coefficients form vector $s$ and all the $(T + 1)^2\kappa$ new points are on this polynomial.
   b. $S$ reveals $f(x)$ to $R$. For each honest $P_i$, $S$ samples $(T + 1)\kappa$ 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 $P_i$.

Figure 22: Simulator for the $F_{\text{APICP}}$ when both $D$ and $I$ are honest.

Hybrid arguments:

Hyb$_1$: In this hybrid, $S$ receives honest parties’ inputs and runs the protocol honestly. This corresponds to the real-world scenario.

Hyb$_2$: In this hybrid, $S$ 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 $(T + 1)^2\kappa$ verification points and the secrets $s^{(1)}, \ldots, s^{(m)}$. Since the polynomials are degree-$(L+(T+1)^2\kappa)$ and each $s^{(k)}$ is in $\mathbb{F}^L$, the verification points are $(T + 1)^2\kappa + 1$-wise independent in $\text{Hyb}_1$, i.e. the $(T + 1)^2\kappa$ 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, $\text{Hyb}_1$ and $\text{Hyb}_2$ have the same output distribution.

Hyb$_3$: In this hybrid, if the verification points for corrupted parties are not distinct, then $S$ aborts the simulation. Note that the probability that each two verification points are the same is $1/|\mathbb{F}|$ and there are $(T + 1)^2\kappa$ points for corrupted parties, so the probability that the verification points are not distinct is
\[
\epsilon_1 < \frac{1}{|\mathbb{F}|} \cdot \frac{(T + 1)^2\kappa}{(T + 1)^2\kappa - 1} < \frac{(T + 1)^4\kappa^2}{2^\kappa},
\]
which is negligible in $\kappa$. Thus, the output distributions of $\text{Hyb}_2$ and $\text{Hyb}_1$ are statistically close.
Hyb₁: In this hybrid, \( S \) doesn’t check the verification points sent from honest parties by himself on behalf of \( I \). Instead, \( S \) 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, Hyb₃ and Hyb₂ have the same output distribution.

Hyb₂: In this hybrid, \( S \) doesn’t check the verification points sent from corrupted parties by himself on behalf of \( I \). Instead, \( S \) 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 \( (T + 1)^2tk \) verification points, so at any point \( \alpha' \) 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 Hyb₂ is at most \( 1/|\mathbb{F}|^m \) and \((T + 1)tk \) verification points are sent from corrupted parties to \( I \). This means the probability that the output distribution changes is at most

\[ e_2 < \frac{(T + 1)tk}{\epsilon m} \]

which is negligible in \( \kappa \). Thus, the output distributions of Hyb₁ and Hyb₃ are statistically close.

Hyb₃: In this hybrid, \( S \) generate a set \( W \) with each element \((c, f(x))\) as the adversary’s knowledge of vectors \( c \) from previous revelations and their corresponding polynomial \( f(x) \) sent from \( I \) to \( R \). \( W \) is initialized to be \( \emptyset \). For each revelation, if \( R \) is corrupted, \( S \) needs to add the \( c \) and the corresponding \( f(x) \) to \( W \) if the \( c \) 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 \( c \) 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 \( W \), so there is no need to add elements into \( W \). That is to say, \( S \) checks \( W = \{(c_1, f_1(x)), \ldots, (c_k, f_k(x))\} \), if \( c, c_1, \ldots, c_k \) are linear independent, \( S \) includes \((c, f(x))\) into \( W \), where \( f(x) = \sum_{i=1}^{m} f(i)(x) \). Generating this \( W \) doesn’t change the output distribution. Thus, Hyb₃ and Hyb₂ have the same output distribution.

Hyb₄: In this hybrid, \( S \) doesn’t sample \( f(1)(x), \ldots, f(k)(x) \) at the beginning. Instead, \( S \) only computes or samples \( f(x) \) corresponding to \( c \) during each revelation. If \( f(x) \) can be determined from \( W = \{(c_1, f_1(x)), \ldots, (c_k, f_k(x))\} \), i.e. \( c \) can be computed by doing linear combination of \( c_1, \ldots, c_k \), \( S \) directly computes \( f(x) \) by doing the same linear combination of \( f_1(x), \ldots, f_k(x) \). Otherwise, \( S \) samples a random degree-\((T + 1)^2tk + L \) polynomial \( f(x) \) whose \( L \) highest coefficients form vector \( s = \sum_{i=1}^{m} c_is(i) \) and all the \( (T + 1)^2tk \) new points \((\alpha, \sum_{i=1}^{m} c_i\beta_i) \) are on this polynomial. Since sampling each \( f(i)(x) \) based on \( s(i) \) and those \((\alpha, \beta_i) \) can be regard as sampling a point \((\alpha', \beta'_i) \) on \( f(i)(x) \) where \( \alpha' \) is different from the \( \alpha \) of any corrupted parties’ verification points. Then \( S \) can compute \( f(i)(x) \) based on \( s(i) \) and the \((T + 1)^2tk + 1 \) distinct points on the polynomial. Since random sampling each \( \beta'_i \) 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 \( c \) can’t be computed by doing linear combination of \( c_1, \ldots, c_k \), which is equivalent to directly sampling \( f(x) \) based on \( \sum_{i=1}^{m} c_is(i) \) and those \((\alpha, \sum_{i=1}^{m} c_i\beta_i) \). So this doesn’t change the output distribution. Thus, Hyb₄ and Hyb₂ have the same output distribution.

Hyb₅: 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, \( S \) sample \((T + 1)\kappa \) random points \( \alpha_1^1, \ldots, \alpha_1^{|\mathbb{F}|^\kappa} \) in \( \mathbb{F} \) and sends \( \{(\alpha_1^1, f(1)(\alpha_1^1)), \ldots, (\alpha_1^{|\mathbb{F}|^\kappa}, f(m)(\alpha_1^{|\mathbb{F}|^\kappa}))\} \) to \( R \) on behalf of \( P_i \). Since each \( \alpha \) of the verification points \( P_i \) is randomly sampled and each \((\alpha, \beta) \) 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, Hyb₂ and Hyb₅ have the same output distribution.

Hyb₆: In this hybrid, for each revelation if \( R \) is honest, for each honest \( P_i \), \( S \) doesn’t follow the protocol to check \( P_i \)'s verification set on behalf of \( R \). Instead, \( S \) 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, Hyb₄ and Hyb₅ have the same output distribution.

Hyb₇: In this hybrid, for each revelation if \( R \) is honest, when \( c \) is not a linear combination of \( c_1, \ldots, c_k \) in \( W \), \( S \) doesn’t check whether there exists an verification point sent from corrupted \( P_t \) to \( R \) on the \( f(x) \) sent from \( I \) to \( R \). Instead, \( S \) 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 \((\alpha', \beta')\) sent from corrupted \(P_i\) to \(R\) satisfies that \(\alpha'\) is different from any \(\alpha\) of the corrupted parties’ verification points sent from \(D\), but \(\beta' = f(\alpha)\). Since in this condition, even if \(A\) knows \(s^{(1)}, \ldots, s^{(m)}\), he still need another point to compute \(f(x)\). Since \(f(x)\) is randomly generated based on \(s^{(1)}, \ldots, s^{(m)}\) and the corrupted parties’ verification points, for any \(\alpha'\), \(F(\alpha')\) is uniformly random in \(F\). Since there are \((T + 1)t\kappa\) points sent from corrupted parties to \(R\), the output distribution changes is

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

which is negligible in \(\kappa\). Thus, the distributions of \(\text{Hyb}_7\) and \(\text{Hyb}_8\) are statistically close.

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

Note that \(\text{Hyb}_{10}\) is the ideal-world scenario, \(\Pi_{\text{APICP}}\) statistically-securely computes \(F_{\text{APICP}}\).

**Remark 2.** When \(D\) and \(I\) are both honest, \(R\) is corrupted, \(S\) 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 \(S\) can still compute the polynomials.

When \(D\) and \(I\) are corrupted:

**Simulator \(S\)**

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

Figure 23: Simulator for the \(F_{\text{APICP}}\) when both \(D\) and \(I\) are corrupted

Hybrid arguments:

**Hyb_5**: In this hybrid, \(S\) receives honest parties’ inputs and runs the protocol honestly. This corresponds to the real-world scenario.

**Hyb_1**: In this hybrid, we change how \(R\) receives \(s'\). After receiving at least \(t + 1\) correct verification sets, \(S\) sends \(\text{Proceed}\) and \(s'\) to \(F_{\text{APICP}}\) and lets \(R\) receives the output from \(F_{\text{APICP}}\). The difference between \(\text{Hyb}_1\) and \(\text{Hyb}_1\) is \(R\) will wait to receive \(s'\) from \(S\) or \(F_{\text{APICP}}\), which makes no difference to the output distribution. Thus, \(\text{Hyb}_1\) and \(\text{Hyb}_1\) have the same output distribution.

Note that \(\text{Hyb}_1\) is the ideal-world scenario, \(\Pi_{\text{APICP}}\) statistically-securely computes \(F_{\text{APICP}}\).

When \(D\) is honest and \(I\) is corrupted:

35
Simulator $S$

1. $S$ waits to receive $s^{(1)}, \ldots, s^{(m)}$ from $\mathcal{F}_{\text{APICP}}$. For each $k \in [m]$, $S$ selects a random degree-$(L + t(T + 1)^2 \kappa)$ polynomial $f^{(k)}(x)$ whose the $L$ highest coefficients form vector $s^{(k)}$.

2. For each corrupted $P_i$, $S$ randomly samples $(T + 1)^2 \kappa$ elements from $F$. For each element $\alpha$, $S$ sends verification point $z = (\alpha, f^{(1)}(\alpha), \ldots, f^{(m)}(\alpha))$ to corrupted $P_i$.

3. $S$ sends $f^{(1)}(x), \ldots, f^{(m)}(x)$ to $I$. For each honest $P_i$, $S$ randomly samples $(T + 1)\kappa$ elements $\alpha_1, \ldots, \alpha_{(T+1)\kappa}$ in $F$. For each $j \in [(T + 1)\kappa]$, $S$ sends verification point $\{ (\alpha_j, f^{(1)}(\alpha_j), \ldots, f^{(m)}(\alpha_j)) \}$ to $I$ on behalf of $P_i$.

4. $S$ initializes a counter $\text{count} = T$.

5. For each revelation, $S$ computes $f(x) = \sum_{k=1}^{m} c_k f^{(k)}(x)$. If $\text{count} > 0$, $S$ does the following things and replaces $\text{count}$ by $\text{count} - 1$:
   - If $R$ is honest:
     a. $S$ receives $f'(x)$ from $I$.
     b. Let $s$ be vector of the $L$ highest coefficients of $f(x)$. $S$ checks whether $f(x) = f'(x)$. If so, $S$ sends $\text{Proceed}$ and $s$ to $\mathcal{F}_{\text{APICP}}$. Otherwise, $S$ sends $\text{Ignore}$ to $\mathcal{F}_{\text{APICP}}$.
     c. If $f'(x) = f(x)$, for each corrupted $P_i \in P$, $S$ receives a verification set $Z^{\text{count}}_t$ from $P_i$ and follows the protocol to check the verification set from $P_i$ on behalf of $R$. For each honest $P_i$, $S$ considers that $R$ receives a correct verification set when $R$ receives it.
     d. When $S$ receives $t + 1$ correct verification sets, he allows $\mathcal{F}_{\text{APICP}}$ to send the output to $R$.
   - If $R$ is corrupted, for each honest $P_i$, $S$ samples $(T + 1)\kappa$ random elements $\alpha_1, \ldots, \alpha_{(T+1)\kappa}$ from $F$ and sends $Z^{\text{count}} = \{ (\alpha_j, f(\alpha_j)) \}_{j \in [(T+1)\kappa]}$ to $R$ on behalf of $P_i$.

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

Hybrid arguments:

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

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

$\text{Hyb}_2$: In this hybrid, for each honest party, $S$ 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 use $Z^{\text{count}}_t$ or $Z^{\text{count}}_c$, $S$ will randomly samples $(T + 1)\kappa$ elements from $F$ to compute $Z^{\text{count}}_t$ or $Z^{\text{count}}_c$. We only changed the order of creating random elements. Thus, $\text{Hyb}_2$ and $\text{Hyb}_1$ have the same output distribution.

$\text{Hyb}_3$: In this hybrid, when $R$ is honest, $S$ adds an addition condition $f'(x) = f(x)$ to check whether $f'(x)$ received from $I$ is acceptable. If so, $S$ sends $\text{Proceed}$ and $s$ to $\mathcal{F}_{\text{APICP}}$. Otherwise, $S$ sends $\text{Ignore}$ to $\mathcal{F}_{\text{APICP}}$. The difference between $\text{Hyb}_2$ and $\text{Hyb}_3$ is when $f'(x) \neq f(x)$, $S$ can still receive $t + 1$ correct $Z^{\text{count}}_t$. When $f'(x) \neq f(x)$ are accept by $S$, that means at least one point in honest $P_i$’s $Z^{\text{count}}_c$ is consistent with $f'(x)$. For each honest $P_i$, in each revelation time, the probability $\epsilon_4$ that this happens is equal to $I$ correctly guesses one random element sampled by $S$.

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

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 $\text{Hyb}_2$ and $\text{Hyb}_1$ are statistically close.

$\text{Hyb}_4$: In this hybrid, for each revelation if $R$ is honest, for each honest $P_i$, $S$ doesn’t follow the protocol to check $P_i$’s verification set on behalf of $R$. Instead, $S$ 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, $\text{Hyb}_4$ and $\text{Hyb}_3$ have the same output distribution.

36
Hyb₁: In this hybrid, we change how $R$ receives $s$. Upon receiving at least $t+1$ correct $Z^{i,e}_\text{count}$ and $f'(x) = f(x)$, $S$ lets $R$ receive the output from $F_{\text{APICP}}$. Besides, $s$ computed by $S$ is equal to $s$ computed by $F_{\text{APICP}}$. Thus, Hyb₁ and Hyb₂ have the same output distribution. 

Note that Hyb₂ is the ideal-world scenario, $\Pi_{\text{APICP}}$ statistically-securely computes $F_{\text{APICP}}$. When $D$ is corrupted and $I$ is honest:

**Simulator $S$**

1. For each honest $P_i$, $S$ waits to receive verification points from $D$. When $P_i$ receives $(T + 1)^2\kappa$ verification points, $S$ randomly divides them into $T + 1$ disjoint sets, where each set is of size $(T + 1)\kappa$, denoted by $Z_1, \ldots, Z_{T+1}^i$.
2. $S$ receives $f^{(1)}(x), \ldots, f^{(m)}(x)$ from $D$ and does the following:
   (a) $S$ checks whether all of these polynomials are degree-$(L + t(T + 1)^2\kappa)$. If so, $S$ lets $s^{(1)}, \ldots, s^{(m)}$ be the vector of the $L$ highest coefficients of $f^{(1)}(x), \ldots, f^{(m)}(x)$. Otherwise, $S$ aborts the simulation.
   (b) For each $P_i \in P$:
      • If $P_i$ is corrupted, $S$ waits to receive a verification set $Z_{T+1}^i$. When verification points in $Z_{T+1}^i$ are all consistent with $f^{(1)}(x), \ldots, f^{(m)}(x)$, $S$ considers that $I$ receives a correct verification set.
      • Otherwise, $S$ uses the $Z_{T+1}^i$ created by himself. When verification points in $Z_{T+1}^i$ are all consistent with $f^{(1)}(x), \ldots, f^{(m)}(x)$ and at least $T(T + 1)^2\kappa + 1$ verification points among $\{Z_j\}_{j \in [T+1]}$ are all consistent with $f^{(1)}(x), \ldots, f^{(m)}(x)$, $S$ considers that $R$ receives a correct verification set.
   (c) When $I$ receives $2t + 1$ correct verification sets, he initializes a counter $\text{count} = T$ and sends $(\text{Init}, \text{APICP}, T, (s^{(1)}, \ldots, s^{(m)}))$ to $F_{\text{APICP}}$. Then, $S$ allows $F_{\text{APICP}}$ to send the output to $I$. Otherwise, $S$ does not continue.
3. For each revelation, $S$ computes $f(x) = \sum_{k=1}^m c_k f^{(k)}(x)$. When $\text{count} > 0$, $S$ does the following things and replaces $\text{count} - 1$:
   • If $R$ is honest:
      (a) For each $P_i \in P$, if $P_i$ is corrupted, $S$ waits to receive a verification set $Z^{i,e}_\text{count}$ from $P_i$. If $P_i$ is honest, $S$ uses $Z^{i,e}_\text{count}$ created by himself. In both cases, $S$ checks whether at least one point in $Z^{i,e}_\text{count}$ is consistent with $f(x)$.
      (b) When $S$ receives at least $t+1$ correct verification sets, let $s$ be the vector of the $L$ highest coefficients of $f(x)$, he allows $F_{\text{APICP}}$ to send the output to $R$.
   • If $R$ is corrupted, $S$ sends $f(x)$ to $R$ on behalf of $I$. $S$ uses $Z^{i,e}_\text{count}$ to compute $Z^{i,e}_\text{count}$ and sends $Z^{i,e}_\text{count}$ to $R$ on behalf of each honest $P_i$.

Figure 25: Simulator for the $F_{\text{APICP}}$ when $D$ is corrupted and $I$ is honest

Hybrid arguments:

Hyb₂: In this hybrid, $S$ receives honest parties’ inputs and runs the protocol honestly. This corresponds to the real-world scenario.

Hyb₁: In this hybrid, we change how $I$ receives $s^{(1)}, \ldots, s^{(m)}$. After receiving at least $2t+1$ correct verification sets, $S$ sends $(\text{Init}, \text{APICP}, T, (s^{(1)}, \ldots, s^{(m)}))$ to $F_{\text{APICP}}$. Then $I$ will receive $s^{(1)}, \ldots, s^{(m)}$ from $F_{\text{APICP}}$ instead of computing them by himself, which makes no difference to the output distribution. Thus, Hyb₁ and Hyb₂ have the same output distribution.

Hyb₂: In this hybrid, $S$ adds an additional verification condition for honest $P_i$’s verification points. $S$ will also check at least $(T+1)T\kappa + 1$ verification points among $\{Z_j\}_{j \in [T+1]}$ are consistent with $f^{(1)}(x), \ldots, f^{(m)}(x)$. If true, $S$ considers that $Z_{T+1}^i$ 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 $Z_{T+1}^i$ to $I$. The probability is

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

which is negligible. Since $S$ needs to receive at least $t+1$ honest $P_i$’s correct $Z_{T+1}^i$, the output distribution only changes with probability $(t+1)\epsilon_5$, which is still negligible. Thus, the output distributions of Hyb₂ and Hyb₁ are statistically close.
Note that $Hyb_2$ is the ideal-world scenario, $\Pi_{APICP}$ statistically-securely computes $F_{APICP}$.

Then we compute the communication complexity of our protocol.

$\Pi_{APICP}$ 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 $O(mL\kappa + mnT^2\kappa^2)$-bit communication. $D$ also sends $(m + 1)(T + 1)^2\kappa$ evaluation points to each party, resulting in a communication of $O(mnT^2\kappa^3)$. Each party sends a set of size $(T + 1)\kappa^2$ to $I$, resulting in a communication of $O(nT^2\kappa^2)$ bits. Therefore, the total communication cost is $O(mL\kappa + mnT^2\kappa^2)$ 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 $O(L\kappa + nT^2\kappa^2)$ bits. Each party sends a set of $(T + 1)\kappa$ field elements to $I$, resulting in a communication of $O(nT^2\kappa^2)$ bits. Therefore, the total communication cost is $O(L\kappa + nT^2\kappa^2)$ bits during the Revelation Phase. Since the Revelation Phase can be executed for at most $T$ times, the total communication cost is $O(LT\kappa + nT^3\kappa^2)$ bits.

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

\section{Proof of the Main Theorem about Our ACSS Protocol}

\subsection{Construction of $\Pi_{ACSS}$}

We present our construction of $\Pi_{ACSS}$ as follows.

\begin{figure}[h]
\centering
\begin{tabular}{|c|}
\hline
\textbf{Protocol $\Pi_{ACSS}$} \\
\hline
All parties execute $\Pi_{Sh}, \Pi_{Ver}, \Pi_{Auth}$ and $\Pi_{Comp}$ in order. \\
\hline
\end{tabular}
\caption{The protocol of the $\Pi_{ACSS}$}
\end{figure}

\subsection{Proof of Lemma 2}

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

When $D$ is honest:

\begin{enumerate}
\item For each corrupted $P_j$, $S$ receives $\{q_1(\alpha_j), \ldots, q_N(\alpha_j)\}$ from $F_{ACSS}$.
\item Define $\text{id} = ((k - 1) \cdot L' + \ell - 1) \cdot (t + 1) + 1$. For each $\ell \in [L']$, $k \in [m']$ and corrupted $P_j$, $S$ randomly selects a degree-$2t$ (column) polynomial $g_{\ell,j}(\alpha)$ such that $g_{\ell,j}(\alpha_i) = g_{\alpha_i+1}(\alpha_j)$ for each $i \in [0, t]$. For each $\ell \in [L']$, $k \in [m' + 1, m]$ and corrupted $P_j$, $S$ randomly samples a degree-$2t$ polynomial as $g_{\ell,j}(y)$.
\item For each $\ell \in [L']$, $k \in [m]$ and corrupted $P_i$, $S$ randomly selects a degree-$t$ (row) polynomial $f_{\ell,i}(y)$ such
\end{enumerate}

\end{proof}

38
that \( f^{(k)}_i(\alpha_j) = g^{(k)}_i(\alpha_j) \) for each corrupted party \( P_j \).

4. \( S \) sends the polynomials \( \{g^{(k)}_i\}_{i \in [L'], k \in [m]} \) to each corrupted \( P_j \).

5. \( S \) initializes a set \( M = \emptyset \). For each \( P_i \in P \):
   - If \( P_i \) is honest, when \( P_i \) receives his column polynomials, \( S \) broadcasts \( \text{OK}_i \) on behalf of \( P_i \). Then \( S \) delivers an initialization request to \( F_{\text{APICP}}(P_i, D) \) on behalf of \( P_i \) and emulates \( F_{\text{APICP}}(P_i, D) \) to deliver the output to \( D \). When \( D \) receives the output and \( \text{OK}_i \), \( S \) includes \( P_i \) into \( M \).
   - If \( P_i \) is corrupted, \( S \) emulates \( F_{\text{APICP}}(P_i, D) \) to receive \((\text{Init}, \text{APICP}, T, (g^{(1)}_{*, i}, \ldots, g^{(m)}_{*, i})) \) from \( P_i \). If \( \text{OK}_i \) is received from \( P_i \), \( S \) checks if \( g^{(k)}_{*, i}(y) = (g^{(k)}_{i, 1}, \ldots, g^{(k)}_{i, m}) \) for each \( k \in [m] \). If so, \( S \) emulates \( F_{\text{APICP}}(P_i, D) \) to send an output \((P_i, \text{APICP}, (g^{(1)}_{*, i}, \ldots, g^{(m)}_{*, i})) \) to \( D \) and includes \( P_i \) into \( M \) when \( D \) receives the output and \( \text{OK}_i \).

6. When \(|M| = 2t + 1\), \( S \) broadcasts \( M \) on behalf of \( D \).

7. For each honest \( P_j \), \( S \) waits until \( P_j \) receives \( M \) and \( \{\text{OK}_i\}_{P_i \in M} \) and then begins the simulation of \( P_j \) in the next phase.

Figure 27: Part-(1/4) of the simulator for the \( F_{\text{ACSS}} \) when \( D \) is honest

**Verification Phase**

1. \( S \) initializes a set \( \mathcal{R} = \{(0, \mathbf{o})\} \) where \( \mathbf{o} \) 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 \((r, F) \in \mathbb{F} \times (\mathbb{F}[X, Y])^{L'} \).

2. For each honest \( P_i \in P \):
   1. When \( P_i \) receives the column polynomials from \( D \), \( S \) broadcasts a random \( r_i \in \mathbb{F} \) on behalf of \( P_i \).
   2. For each honest party \( P_j \) and \( P_h \in M \), when \( P_j \) receives \( r_i \), \( S \) sends \((\text{Request}, \text{APICP}, P_i, (r_i, r_{i1}^r, \ldots, r_{im}^r)) \) to \( F_{\text{APICP}}(P_i, D) \) on behalf of \( P_j \).
   3. For each honest \( P_h \in M \), \( S \) emulates \( F_{\text{APICP}}(P_h, D) \) to deliver an output to \( P_i \). For each corrupted \( P_h \in M \), \( S \) faithfully emulates \( F_{\text{APICP}}(P_h, D) \).
   4. When \( P_i \) receives the output from \( F_{\text{APICP}}(P_h, D) \) for all \( P_h \in M \), \( S \) considers that \( P_i \) accepts his \( \{g^{(k)}_{*, i}\}_{k \in [m], i \in [L']} \) and begins the simulation of \( P_i \) in the next phase.

3. For each corrupted \( P_i \in P \):
   1. For each honest \( P_j \) and \( P_h \in M \), if \( S \) receives \( r_i \) on behalf of \( P_j \), \( S \) sends \((\text{Request}, \text{APICP}, P_i, (r_i, r_{i1}^r, \ldots, r_{im}^r)) \) to \( F_{\text{APICP}}(P_i, D) \) on behalf of \( P_j \).
   2. For each corrupted \( P_h \in M \), \( S \) faithfully emulates \( F_{\text{APICP}}(P_h, D) \).
   3. For each honest \( P_h \in M \),
      - If \((r_i, F) \in \mathcal{R} \) for some \( F = (F_1(x, y), \ldots, F_L(x, y)) \in (\mathbb{F}[X, Y])^{L'} \), \( S \) emulates \( F_{\text{APICP}}(P_h, D) \) to send \( g_{*, h} \) to \( P_i \), where \( g_{*, h} = (g_{i, h}(y), \ldots, g_{L', h}(y)) \) with each \( g_{i, h}(y) = F_i(\alpha_h, y) \).
      - Otherwise, \( S \) samples a random vector of \( L' \) degree-\((t, 2t)\) polynomial \( F = (F_1(x, y), \ldots, F_L(x, y)) \) with \( F_i(\alpha_h, y) = \sum_{k=1}^{m} r_{i, k}^r \cdot g^{(k)}_{i, h}(y) \) and \( F_i(x, \alpha_j) = \sum_{k=1}^{m} r_{i, k}^r \cdot f^{(k)}_{i, j}(x) \) for each corrupted \( P_j \) and \( \ell \in [L'] \). Then \( S \) emulates \( F_{\text{APICP}}(P_h, D) \) to send \( g_{*, h} \) to \( P_i \) for the corresponding \( g_{*, h} \) and includes \((r_i, F) \) into \( \mathcal{R} \).

Figure 28: Part-(2/4) of the simulator for the \( F_{\text{ACSS}} \) when \( D \) is honest

**Authentication Phase**

1. For each \( P_i \in P \) and \( P_v \in P \):
   1. \( S \) follows the protocol to send requests to \( F_{\text{RandShare}}, F^0_{\text{RandShare}} \) on behalf of honest parties. Then \( S \) emulates \( F_{\text{RandShare}}, F^0_{\text{RandShare}} \) to receive corrupted parties’ shares of \([\mu_{i-v}], \{(r_{i-v,j}^{(k)})_{k \in [m]}: i \in [0], \{[r_{i-v,j}^{(k)}]_{k \in [m]}: u \in [0]\}, \{[\text{mask}]_j\}_{j \in [n]} \} \) from \( A \) and sends them to corrupted parties.
   2. \( S \) follows the protocol to compute the corrupted parties’ shares of \([\{r_{i-v,j}^{(k)}\}_{k \in [m]}: i \in [0], \{[r_{i-v,j}^{(k)}]_{k \in [m]}: u \in [0]\}, \{[\text{mask}]_j\}_{j \in [n]} \} \).
   3. If \( P_i \) is honest:
      1. For each honest \( P_j \in P \), when \( P_i \) receives \( P_j \)'s shares of tags, \( S \) broadcasts a random \( r_{i-v,j} \in \mathbb{F} \) on behalf of \( P_i \).
2. For each corrupted $P_j$, $S$ receives $\{x_i^{(k)}\}_{k \in [m]}$ from $P_j$ and broadcasts a random $r_{i \rightarrow v,j} \in \mathbb{F}$ on behalf of $P_j$. Then $S$ computes $\tilde{\tau}_{i \rightarrow v,j} = \sum_{k=1}^{m} r_{i \rightarrow v,j} \cdot x_i^{(k)}$ and $P_j$’s share of $[\tau_{i \rightarrow v,j}]_{2t} = \sum_{k=1}^{m} r_{i \rightarrow v,j} \cdot [x_i^{(k)}]_{2t}$. If there exists some $k \in [m]$ such that $x_i^{(k)}$ is not equal to $P_j$’s share of $[\tau_{i \rightarrow v,j}]_{2t}$ but $\tilde{\tau}_{i \rightarrow v,j}$ is equal to $P_j$’s share of $[\tau_{i \rightarrow v,j}]_{2t}$, $S$ aborts the simulation.

If $P_i$ is corrupted:
1) $S$ randomly samples honest parties’ shares of $\{[\tau_{i \rightarrow v,j}]_{2t}\}_{k \in [m]}$ based on the corrupted parties shares.
2) For each honest $P_j$, $S$ sends $P_j$’s shares of $\{[\tau_{i \rightarrow v,j}]_{2t}\}_{k \in [m]}$ to $P_i$ on behalf of $P_j$.
3) $S$ receives $r_{i \rightarrow v,j}$ from $P_j$.

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

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

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

Figure 29: Part-(3/4) of the simulator for the $\mathcal{F}_{\text{CSS}}$ when $D$ is honest.
1. For each honest party, $S$ follows the protocol to deliver his shares of $\{\mu_{\ell,v}\}_t; \{\nu_{\ell,v}\}_t\}_{k \in [m]}$ and $(\text{Request}, \text{privRec}, P_t)$ for each $P_t \in W$ to $F_{\text{privRec}}$.

2. For each corrupted $P_t \in W$: 
   (1) $S$ receives $\{\tilde{g}_{\ell,t}^{(k)} (\alpha_v)\}_{k \in [m], \ell \in [L]}$ from $P_t$.
   (2) $S$ emulates $F_{\text{privRec}}$ to deliver the whole sharings $\{\mu_{\ell,v}\}_t; \{\nu_{\ell,v}\}_t\}_{k \in [m]}$ to $P_v$ and sends the corrupted parties' shares of them to $A$.
   (3) $S$ randomly samples $r_{\ell,v} \in \mathbb{F}$ and sends it to $P_t$ on behalf of $P_v$.
   (4) $S$ receives $\tilde{\tau}_{\ell,v}$ and $\{\tilde{g}_{\ell,t}^{(k)}\}_{\ell \in [L]}$ from $P_t$.

3. For each $P_t \in W$:
   - If $P_t$ is honest, when $P_v$ receives $P_t$'s $\{g_{\ell,v}^{(k)}\}_{\ell \in [L]}$ and $\tilde{\tau}_{\ell,v}$, $S$ considers that $P_v$ accepts $P_t$'s $\{g_{\ell,t}^{(k)} (\alpha_v)\}_{k \in [m], \ell \in [L]}$.
   - If $P_t$ is corrupted, $S$ considers that $P_v$ accepts $P_t$'s $\{g_{\ell,t}^{(k)} (\alpha_v)\}_{k \in [m], \ell \in [L]}$ if 
     $$\tilde{g}_{\ell,v}(y) = \sum_{k=1}^{m} r_{\ell,v}^{k} g_{\ell,t}^{(k)}(y)$$ 
     for all $\ell \in [L']$, $\tilde{\tau}_{\ell,v} = \sum_{k=1}^{m} r_{\ell,v}^{k} \tau_{\ell,v}^{(k)}$ and $\tilde{g}_{\ell,t}^{(k)}(\alpha_v) = g_{\ell,t}^{(k)} (\alpha_v)$ for each $k \in [m]$ and $\ell \in [L']$.

4. When $P_v$ accepts $\ell + 1$ different $P_t$'s $\{g_{\ell,t}^{(k)} (\alpha_v)\}_{k \in [m], \ell \in [L]}$, $S$ considers that $P_v$ gets his 
   $$\{f_{\ell,v}^{(k)}\}_{k \in [m], \ell \in [L']}$$.

   • If $P_t$ is corrupted:
     1. For each honest party $S$ follows the protocol to deliver his shares of $\{\mu_{\ell,v}\}_t; \{\nu_{\ell,v}\}_t\}_{k \in [m]}$ and $(\text{Request}, \text{privRec}, P_t)$ for each $P_t \in W$ to $F_{\text{privRec}}$.
     2. For each corrupted $P_t \in W$: 
        (1) $S$ randomly samples $\mu_{\ell,v} \in \mathbb{F}$ and computes $\nu_{\ell,v} = u_{\ell,v} - g_{\ell,v}^{(k)} \times \nu_{\ell,v}$.
        (2) $S$ randomly samples honest parties' shares of $\{\mu_{\ell,v}\}_t; \{\nu_{\ell,v}\}_t\}_{k \in [m]}$ based on the corrupted parties' shares and the secrets $\mu_{\ell,v}, \{\nu_{\ell,v}\}_t\}_{k \in [m]}$.
        (3) $S$ emulates $F_{\text{privRec}}$ to send the whole sharings $\{\mu_{\ell,v}\}_t; \{\nu_{\ell,v}\}_t\}_{k \in [m]}$ to $P_v$.
     3. For each corrupted $P_t \in W$:
        (1) $S$ sends $\{f_{\ell,v}^{(k)} (\alpha_v)\}_{k \in [m], \ell \in [L']} \text{ to } P_v$ on behalf of $P_t$.
        (2) $S$ randomly samples honest parties' shares of $\{\mu_{\ell,v}\}_t; \{\nu_{\ell,v}\}_t\}_{k \in [m]}$ based on the corrupted parties' shares.
        (3) $S$ emulates $F_{\text{privRec}}$ to send the whole sharings $\{\mu_{\ell,v}\}_t; \{\nu_{\ell,v}\}_t\}_{k \in [m]}$ to $P_v$.
        (4) $S$ receives $r_{\ell,v}$ from $P_v$.
           - If $r_{\ell,v} \in \mathbb{F}, P_t \in R$ for some $P_t = (F_t(x,y) \ldots F_{L'}(x,y)) \in (\mathbb{F}[X,Y])^{L'}$, $S$ sends $g_{\ell,v} = (g_{\ell,v}(y), \ldots g_{\ell,v}(y))$ with each $g_{\ell,v}(y) = F_t(\alpha_v, y)$ to $P_v$.
           - Otherwise, $S$ samples a random vector of $L'$ degree-$(t, 2t)$ polynomial $F = (F_1(x,y) \ldots F_{L'}(x,y))$ with $F_t(\alpha_v, y) = \sum_{k=1}^{m} r_{\ell,v}^{k} \cdot g_{\ell,v}^{(k)}(y)$ and 
             $$F_t(x, \alpha_v) = \sum_{k=1}^{m} r_{\ell,v}^{k} \cdot f_{\ell,v}^{(k)}(x)$$ 
             for each corrupted $P_j$ and $\ell \in [L']$. Then $S$ sends $g_{\ell,v}$ to $P_v$ and includes $(r_{\ell,v}, F)$ into $R$.
        (5) $S$ computes $\tilde{\tau}_{\ell,v} = g_{\ell,v} ; + \sum_{k=1}^{m} r_{\ell,v}^{k} \tilde{\tau}_{\ell,v}^{(k)}$ and sends it to $P_v$ on behalf of $P_t$.

**Reconstructing column polynomials:**

For each $P_w \in P$: 

• If $P_w$ is honest:
   1. For each honest $P_v \in P$, when $P_w$ receives $\{ f_{\ell,v}^{(k)} (\alpha_v) \}_{k \in [m], \ell \in [L']}$, $S$ randomly broadcasts $r_{\ell,v} \in \mathbb{F}$ on behalf of $P_w$.
   2. For each corrupted $P_v \in P$: 
      (1) $S$ receives $\{ f_{\ell,v}^{(k)} (\alpha_v) \}_{k \in [m], \ell \in [L']} \text{ from } P_v$.
      (2) $S$ randomly broadcasts $r_{\ell,v} \in \mathbb{F}$ on behalf of $P_w$.
   (3) For each corrupted $P_t \in W$, $S$ receives $\tilde{\tau}_{\ell,v}$ and $\{\tilde{g}_{\ell,t}^{(k)}\}_{\ell \in [L]}$ from $P_t$.
   3. For each $P_v \in P$:
      (1) For each $P_t \in W$: 
         - If $P_t$ is honest, when $P_v$ receives $P_t$'s $\{g_{\ell,t}^{(k)}\}_{\ell \in [L]}$ and $\tilde{\tau}_{\ell,v}$, $S$ considers that $P_v$ accepts $P_t$'s $\{\tilde{g}_{\ell,t}^{(k)}\}_{\ell \in [L]}$.
         - If $P_t$ is corrupted, $S$ considers that $P_v$ accepts $P_t$'s $\{g_{\ell,t}^{(k)}\}_{\ell \in [L]}$ if $\tilde{g}_{\ell,v}(y) = \sum_{k=1}^{m} r_{\ell,v}^{k} \cdot \tilde{g}_{\ell,v}^{(k)}(y)$ for all $\ell \in [L'$] and $\tilde{\tau}_{\ell,v} = \sum_{k=1}^{m} r_{\ell,v}^{k} \tilde{\tau}_{\ell,v}^{(k)}$. 

41
(2) When $P_w$ accepts $t + 1$ different $P_i$’s $\{g_{\ell,i}\}_{\ell \in [L]}$, 
- If $P_i$ is honest, $S$ considers that $P_w$ accepts $P_i$’s $\{f_{\ell,v}^{(k)}(\alpha_v)\}_{k \in [m], \ell \in [L']}$. 
- If $P_i$ is corrupted, $S$ considers that $P_w$ accepts $P_i$’s $\{f_{\ell,v}^{(k)}(\alpha_v)\}_{k \in [m], \ell \in [L']}$ if $f_{\ell,v}^{(k)}(\alpha_v) = f_{\ell,v}^{(k)}(\alpha_w)$ for each $k \in [m]$ and $\ell \in [L']$.

4. When $P_w$ accepts $2t + 1$ different $P_i$’s $\{f_{\ell,v}^{(k)}(\alpha_w)\}_{k \in [m], \ell \in [L']}$, $S$ allows $F_{\text{ACSS}}$ to send the output to $P_e$.

- If $P_w$ is corrupted:
  1. For each honest $P_i \in P$:
     - $(1). S$ sends $\{g_{\ell,i}^{(k)}(\alpha_v)\}_{k \in [m], \ell \in [L]}$ to $P_w$ on behalf of $P_e$.
     - $(2). S$ receives $r_{\ell,v-w}$ from $P_w$.
  2. For each honest $P_i \in \mathcal{W}$:
     - If $(r_{\ell,v-w}, F) \in \mathcal{R}$ for some $F = (F_1(x, y), \ldots, F_{L'}(x, y)) \in [F[X, Y]]^{L'}$, $S$ sends $\{g_{\ell,i}(\alpha_v)\}_{\ell \in [L]}$ with each $g_{\ell,i}(y) = F_1(\alpha_v, y)$ to $P_w$.
     - Otherwise, $S$ samples a random vector of $L'$ degree-$(t, 2t)$ polynomial $F = (F_1(x, y), \ldots, F_{L'}(x, y))$ with $F_1(\alpha_v, y) = \sum_{k=1}^{m} r_{\ell,v-w}^{(k)} g_{\ell,i}^{(k)}(y)$ and $F_i(x, \alpha_v) = \sum_{k=1}^{m} r_{\ell,v-w}^{(k)} f_{\ell,v}^{(k)}(x)$ for each corrupted $P_i$ and $\ell \in [L']$. Then $S$ sends $\{g_{\ell,i}(\alpha_v)\}_{\ell \in [L]}$ to $P_w$ and includes $(r_{\ell,v-w}, F)$ into $\mathcal{R}$.
  4. For each honest $P_i \in \mathcal{W}$, $S$ computes $r_{\ell,v-w} = w_{\ell,v-w} \cdot \alpha_{\ell,v-w} + \sum_{k=1}^{m} r_{\ell,v-w}^{(k)}$.

**Figure 30:** Part-(4/4) of the simulator for the $F_{\text{ACSS}}$ when $D$ is honest

Hybrid arguments:

**Hyb$_3$:** In this hybrid, $S$ receives honest parties’ inputs and runs the protocol honestly. This corresponds to the real-world scenario.

**Hyb$_1$:** In this hybrid, during the Sharing Phase, $S$ 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, **Hyb$_1$** and **Hyb$_0$** have the same output distribution.

**Hyb$_2$:** In this hybrid, $S$ initializes a set $\mathcal{R} = \{(0, o)\}$ to record each random value $r$ and the corresponding bivariate polynomials computed by $F_1(x, y) = \sum_{k=1}^{m} r_i^{(k)} F_{\ell,v}^{(k)}(x, y)$. Each time $S$ needs to compute a linear combination of some honest parties’ column polynomials, we regard $S$ 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 $S$ needs to compute a linear combination of bivariate polynomials, if $r_i$ is recorded before, $S$ 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, **Hyb$_2$** and **Hyb$_1$** have the same output distribution.

Since $S$ computes each $F_\ell$ 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 $r_1, \ldots, r_T + T' \neq 0$. Their corresponding vectors $F = (F_1, \ldots, F_{T+T'})$ of polynomials, denoted by $F_1, \ldots, F_{T+T'}$, are random with the corrupted parties’ row and column polynomials of each bivariate polynomial $F_\ell$ given. We prove this fact through hybrid arguments **Hyb$_{1,0}$**, **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 $d = L' \cdot ((2t+1)(t+1) - t'(2t+1) - t'(t+1) + t'^2)$, we need another a randomness $r'$ in $F^d$ to determine each $F$. Here $r'$ determines the vector of $L'$ outputs of $F$ with $L'$ fixed inputs. Then, we can reconstruct $F$ by $r'$ and the polynomials of corrupted parties. Picking random $F$ with column and row polynomials fixed is the same as randomly sampling $r' \in F^d$. Suppose that we need randomness $r'_1, \ldots, r'_{T+T'}$, to determine $F_{(1)} \ldots, F_{(T+T')}$. For a part added in $r'_1 \ldots, r'_{T+T'}$, we let $\tilde{r}_1, \ldots, \tilde{r}_{T+T'}$ be the randomness $r'$ of the linear combination of the $T + T'$ vectors of completely random bivariate polynomials corresponds to $r_1, \ldots, r_{T+T'}$. Besides, we let the randomness $r' \in F^d$ of the $T + T'$ vectors of $\ell$ completely random degree-
(t, 2t) bivariate polynomials be \( \hat{\varphi}(1), \ldots, \hat{\varphi}(T + T') \), then \( (\hat{\varphi}(1), \ldots, \hat{\varphi}(T + T')) \) is completely random in \( \mathbb{F}^{\delta(T + T')} \).

\textbf{Hyb}_{2.0}: \( S \) computes \( r_1', \ldots, r_{T + T'} \) honestly, uses them to compute \( F_1', \ldots, F_{T + T'} \) and outputs them.

\textbf{Hyb}_{2.1}: In this hybride, for the first time \( S \) needs to compute a linear combination of bivariate polynomials, he samples a random \( \tilde{r}_1 \in \mathbb{F}^d \) as the randomness \( r' \) of the linear combination of the \( T + T' \) vectors of completely random bivariate polynomials. Then, \( S \) computes \( r_1' \) with \( \tilde{r}_1 \) and the bivariate polynomials \( \{F_{k}^{(k)}\}_{k \in [m'], \ell \in [L']} \). \( S \) then determines \( F_1 \) using \( r_1' \). \( \tilde{r}_1 \) is computed by:

\[
\hat{\varphi}(1) = \begin{pmatrix} r_1^{m' + 2} & r_1^{m' + 2} & \cdots & r_1^{m} \\
\end{pmatrix} \cdot \begin{pmatrix} \hat{\varphi}(1) \\
\hat{\varphi}(2) \\
\vdots \\
\hat{\varphi}(T + T') 
\end{pmatrix},
\]

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

\textbf{Hyb}_{2.2}: In this hybride, for the second time \( S \) needs to compute a linear combination of bivariate polynomials, he samples a random \( r_2 \in \mathbb{F}^d \) as the randomness \( r' \) of the linear combination of the \( T + T' \) vectors of completely random bivariate polynomials. Then, \( S \) computes \( r_2' \) with \( r_2' \) and the bivariate polynomials \( \{F_{k}^{(k)}\}_{k \in [m'], \ell \in [L']} \). \( S \) then determines \( F_2 \) using \( r_2' \). Given \( \tilde{r}_1, \tilde{r}_2 \) is computed by:

\[
\hat{\varphi}(2) = \begin{pmatrix} r_2^{m' + 2} & r_2^{m' + 2} & \cdots & r_2^{m} \\
\end{pmatrix} \cdot \begin{pmatrix} \hat{\varphi}(1) \\
\hat{\varphi}(2) \\
\vdots \\
\hat{\varphi}(T + T') 
\end{pmatrix} = \begin{pmatrix} r_2^{m' + 2} - r_2^{m' + 1} r_1^{m' + 2} \\
\frac{r_2^{m' + 2} - r_2^{m' + 1} r_1^{m' + 2}}{r_1^{m' + 1}} \\
\vdots \\
\frac{r_2^{m' + 2} - r_2^{m' + 1} r_1^{m' + 2}}{r_1^{m' + 1}} 
\end{pmatrix} \cdot \begin{pmatrix} \hat{\varphi}_1 \\
\hat{\varphi}(2) \\
\vdots \\
\hat{\varphi}(T + T') 
\end{pmatrix},
\]

Since \( r_1 \neq r_2 \),

\[
\begin{vmatrix} r_1^{m' + 1} & r_1^{m' + 2} \\
r_2^{m' + 2} & r_2^{m' + 2} 
\end{vmatrix} \neq 0,
\]

we have \( \frac{r_2^{m' + 2} - r_2^{m' + 1} r_1^{m' + 2}}{r_1^{m' + 1}} \neq 0 \). Since \( (\hat{\varphi}(1), \ldots, \hat{\varphi}(T + T')) \) is completely random in \( \mathbb{F}^{\delta(T + T')} \), given \( \tilde{r}_1, S \) can randomly sample \( \tilde{r}(2), \ldots, \tilde{r}(T + T') \) and compute \( \tilde{r}(1) \) based on \( \tilde{r}_1 \). 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 \( \tilde{r}_2 \), \( \tilde{r}_2 \) is also completely random in \( \mathbb{F} \). Thus, \textbf{Hyb}_{2.2} and \textbf{Hyb}_{2.1} have the same output distribution.

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

\[
\begin{pmatrix}
\hat{p}^{(1)} \\
\hat{p}^{(2)} \\
\vdots \\
\hat{p}^{(i)} \\
\end{pmatrix} =
\begin{pmatrix}
r_1^{m+1} & r_1^{m+2} & \ldots & r_1^m \\
r_2^{m+1} & r_2^{m+2} & \ldots & r_2^m \\
\vdots & \vdots & \ddots & \vdots \\
r_i^{m+1} & r_i^{m+2} & \ldots & r_i^m \\
\end{pmatrix}
\begin{pmatrix}
\hat{p}^{(1)} \\
\hat{p}^{(2)} \\
\vdots \\
\hat{p}^{(T+T')} \\
\end{pmatrix},
\]

the coefficient of \( \hat{p}^{(i)} \) is nonzero if and only if

\[
\det \begin{pmatrix}
r_1^{m+1} & r_1^{m+2} & \ldots & r_1^m \\
r_2^{m+1} & r_2^{m+2} & \ldots & r_2^m \\
\vdots & \vdots & \ddots & \vdots \\
r_i^{m+1} & r_i^{m+2} & \ldots & r_i^m \\
\end{pmatrix} \neq 0.
\]

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

**Hyb\(_{2,(T+T')}\):** In this hybrid, \( S \) samples random vectors of \( L \) degree-\( (t, 2t) \) polynomials as \( F_1, \ldots, F_{T+T'} \) based on corrupted parties’ polynomials instead of computing them based on \( r_1, \ldots, \hat{r}_{T+T'} \). Since \( \hat{r}_1, \ldots, \hat{r}_{T+T'} \) are all randomly sampled in \( \text{Hyb}_{1,(T+T')} \), \( r_1, \ldots, r_{T+T'} \) are also completely random. Thus, given corrupted parties’ polynomials, \( F_1, \ldots, F_{T+T'} \) are random, as in \( \text{Hyb}_{2,(T+T'+1)} \). Thus, \( \text{Hyb}_{2,(T+T')} \) and \( \text{Hyb}_{2,(T+T'+1)} \) 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 \( F \) based on corrupted parties’ polynomials.

**Hyb\(_3\):** In this hybrid, during the Verification Phase, \( S \) doesn’t compute the output of \( \mathcal{F}_{\text{APICP}}(P_h, D) \) for corrupted \( P_i \) if \( P_i \) is honest. Instead, when \( r_i \) is recorded in \( R \), if \( r_i \) is recorded before, \( S \) only needs to use \( \{F_t(x, y)\}_{t \in [L]} \) to compute the output of \( \mathcal{F}_{\text{APICP}}(P_h, D) \). Otherwise \( S \) chooses random bivariate degree-\( (t, 2t) \) polynomials that are consistent with corrupted parties’ polynomials as \( \{F_t(x, y)\}_{t \in [L]} \) corresponding to \( r_i \) and includes them into \( R \). Then \( S \) computes the output of \( \mathcal{F}_{\text{APICP}}(P_h, D) \) with the bivariate polynomials. As we have argued in **Hyb\(_{2,0}\), \ldots, Hyb\(_{2,(T+T'+1)}\)**, with at most \( n < T + T' \) revelation requests for each \( \mathcal{F}_{\text{APICP}}(P_h, D) \) in the Verification Phase, if some \( r_i \) hasn’t been recorded before, each \( F_t \) is random with corrupted parties’ column and row polynomials given, as what \( S \) samples in **Hyb\(_3\)**. Thus, **Hyb\(_3\)** and **Hyb\(_2\)** have the same output distribution. **Hyb\(_4\):** In this hybrid, during the Authentication Phase, for each honest \( P_i \) and corrupted \( P_j \), \( S \) aborts the simulation if \( P_j \)’s shares of \( r_{i-v,j}^{(k)} \) are not correctly sent for some \( P_v \in \mathcal{P} \) and \( k \in [m] \) but \( \tau_{i-v,j}^{(k)} \) is correct. This happens only when the random \( r_{i-v,j} \) satisfies \( \sum_{k=1}^{m} r_{i-v,j}^{(k)} = \tau_{i-v,j} \), where each \( \tau_{i-v,j}^{(k)} \) is the share of \( r_{i-v,j}^{(k)} \) received from \( P_j \). Note that any non-zero polynomial \( \sum_{k=1}^{m} \tau_{i-v,j}^{(k)} x^k - \tau_{i-v,j} \) 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

\[
eq 1 \leq \frac{mt(2t+1)n}{|F|} < \frac{m \cdot n^3}{2^\nu},
\]

which is negligible. Thus, the output distributions of **Hyb\(_4\)** and **Hyb\(_3\)** are statistically close.

**Hyb\(_5\):** In this hybrid, during the Authentication Phase, for each \( P_v \in \mathcal{P} \) and corrupted \( P_i \), \( S \) samples honest parties’ shares of \( \{r_{i-v,j}^{(k)}\}_{k \in [m]} \) based on corrupted parties’ shares instead of computing them by himself. Besides, \( S \) doesn’t sample the honest parties’ shares of \( \{r_{i-v,j}^{(k)}\}_{k \in [m]} \). Instead, \( S \) computes them by

\[
\left[ r_{i-v,j}^{(k)} \right] = \left[ r_{i-v,j}^{(k)} \right] - \left[ g_{i,j}^{(k)} \right] \cdot \left[ \mu_{i-v} \right] - \sum_{u=1}^{t} e_u \cdot \left[ r_{i-v,j}^{(k)} \right].
\]

Since \( [e_u]_i \) is a fixed packed secret sharing, we can denote its degree-\( t \) sharing polynomial by \( f_u(x) \), where \( f_u(-u) = 1 \) and \( f_u(\alpha_j) = 0 \) for each \( j \in \{-1, -2, \ldots, -t, i\} \). Let the sharing polynomial of \( [r_{i-v,j}^{(k)}]_i \)
be $g_u(x)$, then $g_u(x)$ is a random degree-$t$ polynomial with $\{g_u(\alpha_j)\}_{j \in C}$ fixed. Hence, $g_u(\alpha-u)$ is random in $\mathbb{F}$. Thus, the sharing polynomial $h(x)$ of $\sum_{u=1}^{t}[e_u] \cdot [r_u^{(k)}]_u$ is a degree-$2t$ polynomial such that for each $u \in [t]$, $h(\alpha-u) = \sum_{j=1}^{t}f_j(\alpha-u)g_u(\alpha-u) = f_u(\alpha-u)g_u(\alpha-u) = g_u(\alpha-u)$ is a random value in $\mathbb{F}$. Since honest parties' shares of $[r_u^{(k)}]_u$ are randomly sampled based on corrupted parties' shares and $g_u(\alpha-u)$, the honest parties' shares of $\sum_{u=1}^{t}[e_u] \cdot [r_u^{(k)}]_u$ 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 $\sum_{u=1}^{t}[e_u] \cdot [r_u^{(k)}]_u$ 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 $\sum_{u=1}^{t}[e_u] \cdot [r_u^{(k)}]_u$ is 0, i.e. it can be regarded as a $[0]_2^{t}$ added in $\{[r_i^{(k)}]_j\}_{i \in [m]}$. Note that the honest parties' shares of $[\mu_i]_i \cdot \{[r_i^{(k)}]_i\}_{j \in [m]}$ are also randomly sampled based on corrupted parties' shares, since honest parties' shares of $\{[r_i^{(k)}]_i\}_{j \in [m]}$ is a random 2t-sharing given corrupted parties' shares, so sampling the honest parties' shares of $\{[r_i^{(k)}]_i\}_{j \in [m]}$ instead of computing them doesn't change the output distribution. Hence, we only change the order of sampling the honest parties' shares of $\{[r_i^{(k)}]_i\}_{k \in [m]}$ and $\{[r_i^{(k)}]_{2t}\}_{k \in [m]}$. Thus, $\text{Hyb}_5$ and $\text{Hyb}_4$ have the same output distribution.

$\text{Hyb}_5$: In this hybrid, during the Connection Phase, for each $P_j \in \mathcal{P}$, each corrupted $P_b$ and each honest $P_h \in \mathcal{H}$, $S$ doesn't compute the output of $\mathcal{F}_{\text{APCP}}(P_h, D)$. Instead, $S$ checks whether $r_{i-v,j}$ has been recorded in $\mathcal{R}$ before. If $r_{i-v,j}$ is recorded before, $S$ only needs to use $\{F_i(x,y)\}_{i \in [L]}$ to compute the output of $\mathcal{F}_{\text{APCP}}(P_h, D)$. Otherwise $S$ chooses random bivariate degree-$(t,2t)$ polynomials that are consistent with corrupted parties' polynomials as $\{F_i(x,y)\}_{i \in [L]}$ corresponding to $r_{i-v,j}$ and includes them into $\mathcal{R}$. Then, $S$ computes the output of $\mathcal{F}_{\text{APCP}}(P_h, D)$ with the bivariate polynomials. As we have argued in $\text{Hyb}_3$, $\text{Hyb}_4, \ldots, \text{Hyb}_6$, with at most $n + n^3 = T + T' + T^2$ revelation requests for each $\mathcal{F}_{\text{APCP}}(P_h, D)$ in the Verification Phase and the Authentication Phase, if some $r_{i-v,j}$ hasn't been recorded before, each $F_i$ is random with corrupted parties' column and row polynomials given, as what $S$ samples in $\text{Hyb}_6$. Thus, $\text{Hyb}_5$ and $\text{Hyb}_4$ have the same distribution.

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

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

$\text{Hyb}_8$: In this hybrid, during the Completion Phase, for each honest $P_i$ and corrupted $P_j \in \mathcal{W}$, $S$ doesn't follow the protocol to check $P_i$'s polynomials $\{g_{i,j}^{(k)}\}_{i \in [L]}$ and the tag $\tilde{\tau}_{i-v}$. Instead, $S$ checks whether the polynomials are consistent with what $S$ generates in the Sharing Phase and whether $\tilde{\tau}_{i-v}$ is consistent with the tags $P_i$ gets in the Authentication Phase. This changes the output distribution only if $P_i$ sends $\{g_{i,j}^{(k)}\}_{i \in [L]}$ and $\tilde{\tau}_{i-v}$ different from $\{g_{i,j}^{(k)}\}_{i \in [L]}$ and $\tau_{i-v}$ but still satisfying $\tilde{\tau}_{i-v} = \tilde{g}_{i,j}^{(k)} \cdot \mu_{i-v} + \sum_{k=0}^{m} \tau_{i-v} = v_{i-v}$, where $\tilde{g}_{i,j}^{(k)} = \tilde{g}_{i}^{(k)}, \tilde{g}_{j}^{(k)}$. Then we know that $(\tilde{g}_{i,j}^{(k)} - g_{i,j}^{(k)}) \cdot \mu_{i-v} - (\tilde{\tau}_{i-v} - \tau_{i-v}) = 0$. Since $\mu_{i-v}$ is random in $\mathbb{F}_L$ when $P_i$ is corrupted and we don't need it for any computation before $P_i$ receives $P_i$'s polynomials and tag, $S$ can randomly sample it after $\{g_{i,j}^{(k)}\}_{i \in [L]}$ and $\tilde{\tau}_{i-v}$ are received. Then, the output distribution changes only when a random $\mu_{i-v} \in \mathbb{F}_L$ satisfies a linear equation $a \cdot \mu_{i-v} + b = 0$ with $a \neq 0 \in \mathbb{F}_L$ and $b \in \mathbb{F}$. This happens with probability $1/|\mathbb{F}| = 1/2^n$. 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 $\text{Hyb}_6$ and $\text{Hyb}_8$ are statistically close.

$\text{Hyb}_{10}$: In this hybrid, during the Completion Phase, for each honest $P_i$ and corrupted $P_j \in \mathcal{W}$, $S$ doesn't follow the protocol to check $P_i$'s polynomials and tags. Instead, $S$ considers that if $P_j$ accepts $P_i$'s $\{g_{i,j}^{(k)}(\alpha_v)\}_{k \in [m], i \in [L]}$ 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 $\{g_{i,j}^{(k)}(\alpha_v)\}_{k \in [m], i \in [L]}$. Thus, $\text{Hyb}_{10}$ and $\text{Hyb}_8$ have the same output distribution.
distribution.

**Hyb_{11}:** In this hybrid, for each corrupted \( P_u \) and \( P_t \in \mathcal{W} \), \( S \) doesn’t sample the honest parties’ shares of \([\mu_{i-v}], \{v_{i-v}^{(k)}\}_{k \in [m]}\) in the Authentication Phase. Instead, \( S \) samples \( \mu_{i-v} \) in the Completion Phase and computes \( \{v_{i-v}^{(k)}\}_{k \in [m]} \) with \( \mu_{i-v} \) and \( \{r_{i-v}^{(k)}\}_{k \in [m]} \). Then \( S \) samples the honest parties’ shares of \([\mu_{i-v}], \{v_{i-v}^{(k)}\}_{k \in [m]}\) based on \( \mu_{i-v}, \{v_{i-v}^{(k)}\}_{k \in [m]} \) and the corrupted parties’ shares. Since we don’t need \( \mu_{i-v} \) and the honest parties’ shares of \([\mu_{i-v}], \{v_{i-v}^{(k)}\}_{k \in [m]}\) we can sample them in the Completion Phase. Then \( \{v_{i-v}^{(k)}\}_{k \in [m]} \) is fixed and can be computed. For \( \{v_{i-v}^{(k)}\}_{k \in [m]} \), computing the honest parties’ shares of the 2ℓ-sharing \( \sum_{u=1}^{t} [\epsilon_u] \cdot [r_{i-v}^{(k)}] \) is the same with sampling a random sharing based on corrupted parties’ shares and the secret 0, as we argued in **Hyb_9**. Hence, sampling the honest parties’ shares of \([\mu_{i-v}], \{v_{i-v}^{(k)}\}_{k \in [m]}\) based on the secrets and the corrupted parties’ shares is the same as computing them. Thus, **Hyb_{11}** and **Hyb_{10}** have the same output distribution.

**Hyb_{12}:** In this hybrid, for each corrupted \( P_u \) and honest \( P_t \in \mathcal{W} \), \( S \) doesn’t sample the honest parties’ shares of \([\mu_{i-v}], \{v_{i-v}^{(k)}\}_{k \in [m]}\) in the Authentication Phase. Instead, \( S \) randomly samples them in the Completion Phase and emulates \( F_{PrivRec} \) to send them to \( P_v \). Since when \( P_t \) is honest, the honest parties’ shares of \([\mu_{i-v}], \{v_{i-v}^{(k)}\}_{k \in [m]}\) 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, **Hyb_{12}** and **Hyb_{11}** have the same output distribution.

**Hyb_{13}:** In this hybrid, during the Completion Phase, for each corrupted \( P_u \) and honest \( P_t \in \mathcal{W} \), \( S \) doesn’t follow the protocol to compute \( P_t \)’s \( g_{s,i} \) by doing a linear combination of polynomials and the corresponding tag \( r_{i-v} \). Instead, \( S \) checks whether \( r_{i-v} \) has been recorded in \( R \). If it has been recorded, \( g_{s,i} \) can be computed from the corresponding \( F \). Otherwise, \( S \) generates a random \( F \) based on corrupted parties’ column and row polynomials. As we have argued in **Hyb_{2,0, \ldots, Hyb_{2,(T+T')}},** with at most \( n + n^3 + n^2 < T + T' \) linear combinations we need to do, if some \( r_{i-v} \) hasn’t been recorded before, each \( F_\ell \) is random with corrupted parties’ column and row polynomials given, as what \( S \) samples in **Hyb_{13}**. Thus, **Hyb_{13}** and **Hyb_{12}** have the same output distribution.

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

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

**Hyb_{16}:** In this hybrid, during the Completion Phase, for each honest \( P_w \), \( S \) doesn’t compute the output for \( P_w \) by himself. Instead, when \( P_w \) accepts \( 2t + 1 \) different \( P_t \)’s \( \{f_{\ell,v}^{(k)}(\alpha_w)\}_{k \in [m], \ell \in [L']} \), \( S \) lets \( P_w \) receive the output from \( F_{\mathcal{ACS}} \). Since the \( 2t + 1 \) different \( P_t \)’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 \( S \) generated on behalf of \( D \) in the Sharing Phase, \( P_w \) must get \( 2t + 1 \) different points on each polynomial in \( \{g_{t,v}^{(k)}\}_{k \in [m], \ell \in [L']} \), which enables \( P_w \) to reconstruct \( \{g_{t,v}^{(k)}\}_{k \in [m], \ell \in [L']} \) and compute the output correctly. Hence, changing how honest parties get outputs doesn’t change the output distribution. Thus, **Hyb_{16}** and **Hyb_{15}** have the same output distribution.
Hyb_{17}: In this hybrid, during the Completion Phase, for each corrupted \( P_w \) and honest \( P_i \in \mathcal{W}, \) \( S \) doesn’t follow the protocol to compute \( P_i \)'s \( g_{s,i} \) by doing a linear combination of polynomials and the corresponding tag \( \tau_{t \leftarrow w} \). Instead, \( S \) checks whether \( r_{t \leftarrow w} \) has been recorded in \( \mathcal{R} \). If it has been recorded, \( g_{s,i} \) can be computed from the corresponding \( F \). Otherwise, \( S \) generates a random \( F \) based on corrupted parties’ column and row polynomials. As we have argued in Hyb doesn’t follow the protocol to compute \( n \) can be computed from the corresponding \( F \).

Note that Hyb_{2,0}, \ldots, Hyb_{2,(T+T'+1)} with at most \( n + n^2 + 2n^2 = T + T' \) linear combinations we need to do if some \( r_{t \leftarrow w} \) hasn’t been recorded before, each \( F_t \) is random with corrupted parties’ column and row polynomials given, as what \( S \) samples in Hyb_{17}. Thus, Hyb_{17} and Hyb_{16} have the same output distribution.

When \( D \) is corrupted:

**Sharing Phase**

1. For each \( P_i \in \mathcal{P} \):
   - If \( P_i \) is honest:
     1. When \( P_i \) receives \( \{g_{t,i}^{(k)}(y)\}_{t \in [L'],k \in [m]} \) from \( D \), if these polynomials are all of degree \( 2t \), \( S \) broadcasts \( \emptyset \) on behalf of \( P_i \).
     2. \( S \) follows the protocol to compute \( g_{s,i}^{(k)} \) for each \( k \in [m] \).
     3. \( S \) sends \((\text{Init, APICP, } T, (g_{t,i}^{(1)}, \ldots, g_{t,i}^{(m)}))\) to \( \mathcal{F}_{\text{APICP}}(P_i, D) \) on behalf of \( P_i \).
     4. \( S \) faithfully emulates \( \mathcal{F}_{\text{APICP}}(P_i, D) \).
   - If \( P_i \) is corrupted, \( S \) faithfully emulates \( \mathcal{F}_{\text{APICP}}(P_i, D) \).
2. For each honest party, upon this party receives \( \mathcal{M} \) from \( D \), \( S \) follows the protocol to verify the \( \mathcal{M} \) set. If \( S \) 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 \( t \in [L'] \) and \( k \in [m] \), \( S \) reconstructs a degree-(\( t, 2t \)) bivariate polynomial \( F_t^{(k)}(x,y) \) such that \( F_t^{(k)}(\alpha, y) = g_{t,i}^{(k)}(y) \) for each \( P_i \in \mathcal{H} \).
4. For each \( k \in [m] \) and \( t \in [L'] \), \( S \) computes each corrupted \( P_j \)'s \( g_{t,i}^{(k)}(y) = F_t^{(k)}(\alpha_j, y) \). Then \( S \) sets \( g_{s,j}^{(k)} = (g_{t,i}^{(1)} \ldots, g_{t,i}^{(m)}) \) for each \( k \in [m] \).

**Verification Phase**

For each \( P_i \in \mathcal{P} \):
- If \( P_i \) is honest:
  1. When \( P_i \) receives \( \{g_{t,i}^{(k)}(y)\}_{t \in [L'],k \in [m]} \) from \( D \) and these polynomials are all of degree \( 2t \), \( S \) broadcasts a random value \( r_i \in F \) and follows the protocol to compute \( \{g_{t,i}(y)\}_{t \in [L']} \) on behalf of \( P_i \).
  2. For each \( P_h \in \mathcal{M} \), when each honest party receives \( r_i \), \( S \) sends a request \((\text{Request, APICP, } P_i, (r_1, r_2^2, \ldots, r_m^m))\) to \( \mathcal{F}_{\text{APICP}}(P_h, D) \) on behalf of this party.
  3. For each \( P_h \in \mathcal{M} \), \( S \) faithfully emulates \( \mathcal{F}_{\text{APICP}}(P_h, D) \).
  4. \( S \) does the following things:
     a. \( S \) checks whether \( F_t^{(k)}(\alpha, y) = g_{t,i}^{(k)}(y) \) for each \( t \in [L'] \) and \( k \in [m] \).
     b. When \( P_i \) receives \( \{g_{s,h}^{(k)}\}_{P_h \in \mathcal{M}} \), \( S \) follows the protocol to check \( \{g_{t,i}(y)\}_{t \in [L']} \).
     c. If both checks pass, \( S \) accepts \( \{g_{t,i}^{(k)}(y)\}_{t \in [L'],k \in [m]} \). Otherwise, \( S \) rejects those polynomials.
- If \( P_i \) is corrupted:
  1. For each \( P_h \in \mathcal{M} \), when each honest party receives \( r_i \), \( S \) sends a request \((\text{Request, APICP, } P_i, (r_1, r_2^2, \ldots, r_m^m))\) to \( \mathcal{F}_{\text{APICP}}(P_h, D) \) on behalf of this party.
  2. For each \( P_h \in \mathcal{M} \), \( S \) faithfully emulates \( \mathcal{F}_{\text{APICP}}(P_h, D) \).
1. For each $P_i \in \mathcal{P}$ and $P_h \in \mathcal{P}$:

**Preparing random shares:**

(1). $\mathcal{S}$ follows the protocol to send requests to $\mathcal{F}_{\text{RandShare}}$, $\mathcal{F}'_{\text{RandShare}}$ on behalf of honest parties.

(2). $\mathcal{S}$ emulates $\mathcal{F}_{\text{RandShare}}, \mathcal{F}'_{\text{RandShare}}$ to receive corrupted parties’ shares of \( \{\mu_{i-v,j}\}, \{\nu_{\ell,i-v,j}\} \}_{i-v,j} \in [m] \), \( \{\nu_{\ell,i-v,j}\} \in [m], \{\{\text{mask}\}\} \}_{i-v,j} \in \mathcal{A} \) and sends them to corrupted parties.

(3). For each corrupted $P_j$ and $k \in [m]$, $\mathcal{S}$ computes $P_j$’s share of \( \{\nu_{\ell,i-v,j}\}_{k \in [m]} \) (denoted by $\hat{\tau}_{i-v,j}$) as follows:

\[
[\hat{\tau}_{i-v,j}]_{2t} = [g_{i-v,j}]_{2t} \cdot [\nu_{\ell,i-v,j}]_{2t} + \sum_{u=1}^{t} [e_u]_{2t} \cdot [\nu_{(k)}]_{2t}
\]

**Preventing shares of tags \( \{\tau_{i-v,j}\}_{k \in [m]} \) for \( P_i \):**

- If $P_i$ is honest:
  
  (1). For each $P_i \in \mathcal{P}$:
  
  1). If $P_i$ is honest, $\mathcal{S}$ honestly executes the protocol. If $P_i$ is corrupted, when $P_i$ receives \( \{\tau_{i-v,j}\} \in [m], \mathcal{S} $ sends \( \mathcal{F}_{\text{APIPCP}}(P_h, D) \) on behalf of this honest party.

- If $P_i$ is corrupted, $\mathcal{S}$ randomly samples the whole \( \{\nu_{\ell,i-v,j}\} \) based on corrupted parties’ shares.

**Verifying $P_j$’s shares of tags:**

(2). For each $P_h \in \mathcal{M}$, $\mathcal{S}$ emulates $\mathcal{F}_{\text{APIPCP}}(P_h, D)$.

(3). If $P_h$ is honest, when $P_h$ receives \( \{g_{i-v,j}\} \in \mathcal{M}, \mathcal{S}$ follows the protocol to check these polynomials. If true, then $\mathcal{S}$ delivers $P_h$’s share of \( \{\gamma_{i-v,j}\} \) and sends \( \mathcal{F}_{\text{privRec}}(P_i) \) to $\mathcal{F}_{\text{RandShare}}$ on behalf of $P_h$.

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

(2). Upon accepting $2t + 1$ different $P_j$’s \( \{\tau_{i-v,j}\} \in [m], \mathcal{S}$ considers that $P_i$ reconstructs \( \{\tau_{i-v,j}\} \in [m] \).

- If $P_i$ is corrupted:
  
  (1). $\mathcal{S}$ randomly samples elements from $\mathcal{F}$ as honest parties’ shares of \( \{\nu_{\ell,i-v,j}\} \in [m], \mathcal{S}$ ensures these shares are consistent with the corrupted parties shares. Then $\mathcal{S}$ computes $\tau_{i-v,j}$ based on these shares.

  (2). For each honest $P_j$, if $P_j$ accepts \( \{g_{i-v,j}(y)\} \in [m], \mathcal{S}$ sends $P_j$’s shares of \( \{\nu_{\ell,i-v,j}\} \in [m], \) (denoted each one by $\hat{\tau}_{i-v,j}$) to $P_j$ on behalf of $P_j$.

  (3). For each $P_h \in \mathcal{P}$:

  (a) For each honest party and $P_h \in \mathcal{M}$, when this honest party receives $r_{i-v,j}$, $\mathcal{S}$ sends \( \mathcal{F}_{\text{APIPCP}}(P_h, D) \) on behalf of this honest party. $\mathcal{S}$ computes $\tau_{i-v,j} = \sum_{u=1}^{t} [e_u]_t \cdot [\nu_{\ell,i-v,j}]_t$ for each honest $P_j$.

  (b) For each $P_h \in \mathcal{M}$, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\text{APIPCP}}(P_h, D)$.

  (c) If $P_h$ is honest, when $P_h$ receives \( \{g_{i-v,j}\} \in \mathcal{M}, \mathcal{S}$ considers that $P_h$ accepts these polynomials.

  Then $\mathcal{S}$ delivers $P_h$’s share of $\gamma_{i-v,j}$ and sends \( \mathcal{F}_{\text{privRec}}(P_i) \) to $\mathcal{F}_{\text{RandShare}}$ on behalf of $P_h$. If $P_h$ is corrupted, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\text{RandShare}}$ to wait for $P_h$’s request.

(4). $\mathcal{S}$ emulates $\mathcal{F}_{\text{privRec}}$ as follows:

(a) For each $P_i \in \mathcal{P}$, if $P_i$ is honest, $\mathcal{S}$ randomly samples the whole $\{\gamma_{i-v,j}\} \in [m]$ based on corrupted parties’ shares and $P_i$’s $\tau_{i-v,j}$. If $P_i$ is corrupted, $\mathcal{S}$ randomly samples the whole $\{\gamma_{i-v,j}\} \in [m]$ based on corrupted parties’ shares.
(b) $S$ sends the whole $[\tau_{i-v,j}]^t$ to $P_t$.
2. For each honest $P_v$, when $P_t$ reconstructs $\{\tau^{(k)}_{i-v}\} \in [m]$ for all $P_v \in P$ and $P_t$ accepts its column polynomials, $S$ broadcasts $Tag$ on behalf of $P_t$.
3. For each honest party, when this honest party receives $W$, $S$ follows the protocol to verify the $W$ set. If $S$ succeeds, he begins the next simulation phase of this honest party.
4. For each $k \in [m]$, $\ell \in [L']$, $S$ sets $id \ell = ((k-1) \cdot L' + \ell - 1) \cdot (t + 1) + 1$. Then for each $i \in [0, t]$, $S$ computes $q_{id + i}(x) = F^{(k)}_i(x, \alpha_i)$. Finally, $S$ sends (Dealer, ACSS, $\{q_1(x), \ldots, q_N(x)\}$) to $F_{ACSS}$ on behalf of $D$.

Figure 33: Part-(3/4) of the simulator for the $F_{ACSS}$ when $D$ is corrupted

Simulator $S$

Reconstructing row polynomials:
1. For each $P_v \in P$, $S$ does the following things:
   - If $P_v$ is honest:
     1. For each honest party, $S$ delivers his shares of $[\mu_{i-v}]_\ell \cdot \{v^{(k)}_{i-v}\} \in [m]$ and sends (Request, privRec, $P_v$) to $F_{privRec}$ on behalf this honest party.
     2. $S$ emulates $F_{privRec}$ and delivers an output to $P_v$.
   - For each $P_v \in W$:
     1. If $P_v$ is honest, $S$ honestly executes the protocol.
     2. If $P_v$ is corrupted, $S$ does the following things on behalf of $P_v$:
       - When $P_v$ receives $\{g_{i,v}^{(k)}(\alpha_v)\} \in [m], \ell \in [L']$ from $P_t$, $S$ sends a random element $\tau_{i-v} \in F$ to $P_v$.
       - If $P_v$ accepts the output from $F_{privRec}$, $S$ accepts $P_v$'s $\{g_{i,v}^{(k)}(\alpha_v)\} \in [m], \ell \in [L']$.
   - If $P_v$ is corrupted:
     1. For each honest party, $S$ delivers his shares of $[\mu_{i-v}]_\ell \cdot \{v^{(k)}_{i-v}\} \in [m]$ and sends (Request, privRec, $P_v$) to $F_{privRec}$ on behalf this honest party.
     2. $S$ emulates $F_{privRec}$ on behalf of each honest party.
   - For each honest party, $S$ randomly samples the whole $[\mu_{i-v}]_\ell \cdot \{v^{(k)}_{i-v}\} \in [m]$ based on corrupted parties’ shares. If $P_v$ is corrupted, $S$ randomly samples the whole $[\mu_{i-v}]_\ell \cdot \{\nu^{(k)}_{i-v}\}$ based on corrupted parties’ shares, then $S$ computes $\tau_{i-v} = \tau^{(k)}_{i-v} - \check{g}_{i,v}^{(k)} \cdot \mu_{i-v}$ for each $k \in [m]$. Finally, for each $k \in [m]$, $S$ samples the whole $[\check{\nu}_{i,v}^{(k)}]_\ell \cdot \{\nu^{(k)}_{i-v}\}$ based on corrupted parties’ shares and $\nu^{(k)}_{i-v}$.

Reconstructing column polynomials:
2. For each $P_v \in P$, $S$ does the following things:
   - If $P_v$ is honest:
     1. For each $P_v \in P$:
       - $S$ does the following things:
         - If $P_v$ is honest, $S$ honestly executes the protocol.
         - If $P_v$ is corrupted, when $P_v$ receives $\{f_{\ell,v}(\alpha_v)\} \in [m], \ell \in [L']$ from $P_v$, $S$ broadcasts a random value $r_{v-v} \in F$ on behalf of $P_v$. 
   - If $P_v$ is corrupted:
     1. For each $P_v \in P$:
       - $S$ does the following things:
         - If $P_v$ is honest, $S$ honestly executes the protocol.
         - If $P_v$ is corrupted, when $P_v$ receives $\{f_{\ell,v}(\alpha_v)\} \in [m], \ell \in [L']$ from $P_v$, $S$ broadcasts a random value $r_{v-v} \in F$ on behalf of $P_v$. 

49
For each $P_i \in W$:
- If $P_i$ is honest, when $P_i$ receives $\tau_{v \rightarrow w}$, $S$ delivers $\tau_{w \rightarrow u}$ and $\{g_{\ell,i}(y)\}_{\ell \in [L']}$. Since $P_w$ receives $\tau_{w \rightarrow u}$, $S$ considers that $P_w$ accepts $\{g_{\ell,i}(y)\}_{\ell \in [L']}$. 
- If $P_i$ is corrupted, when $P_w$ receives $\tau_{w \rightarrow u}$ and $\{g_{\ell,i}(y)\}_{\ell \in [L']}$, $S$ does the following things:
  (a). $S$ checks whether $\tau_{w \rightarrow u} = \sum_{k=1}^{m} r_{w \leftarrow v}^{(k)} \cdot \tau_{w \rightarrow u}^{(k)}$.
  (b). For each $\ell \in [L']$, $S$ checks whether the degree of $g_{\ell,i}(y)$ is $2t$ and $g_{\ell,i}(y) = \sum_{k=1}^{m} r_{w \leftarrow v}^{(k)} \cdot g_{\ell,i}^{(k)}(y)$.
  (c). If both (a) and (b) are true, $S$ accepts $P_i$’s $\{g_{\ell,i}(y)\}_{\ell \in [L']}$. 

3. When $P_w$ accepts $2t + 1$ different $P_i$’s $\{g_{\ell,i}(y)\}_{\ell \in [L']}$, $S$ follows the protocol to check $\{F_{\ell,i}^{(k)}(\alpha_{w})\}_{k \in [m], \ell \in [L']}$.

(2). When $P_w$ accepts $2t + 1$ different $P_i$’s $\{f_{\ell,i}^{(k)}(\alpha_{w})\}_{k \in [m], \ell \in [L']}$, $S$ allows $F_{\text{ACSS}}$ to send the output to $P_w$.

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


Figure 34: Part-(4/4) of the simulator for $F_{\text{ACSS}}$ when $D$ is corrupted.

Hybrid arguments:

**Hyb$_0$**: In this hybrid, $S$ runs the protocol honestly. This corresponds to the real-world scenario.

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

**Hyb$_2$**: In this hybrid, during the Verification Phase, for each honest $P_i$ and $\ell \in [L']$, $S$ computes $F_{\ell}(x, y) = \sum_{k=1}^{m} F_{\ell}^{(k)}(x, y) \cdot r_{w}^{(k)}$. 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 $\hat{F}_{\ell}(x, y)$. Then $S$ additionally checks whether $\hat{F}_{\ell}(x, y) = F_{\ell}(x, y)$ for each $\ell \in [L']$. When an honest $P_i$ accepts his column polynomials, $\hat{F}_{\ell}(\alpha_{i}, y) = g_{\ell,h}(y)$ holds for all $P_h \in M$ and $\ell \in [L']$. Since $H \subseteq M$ and $|H| = t + 1$, we always have $\hat{F}_{\ell}(x, y) = F_{\ell}(x, y)$ if $P_i$ accepts his column polynomials. Thus, $\text{Hyb}_1$ and $\text{Hyb}_2$ have the same output distribution.

**Hyb$_3$**: In this hybrid, during the Verification Phase, for each $\ell \in [L']$, $k \in [m]$, $S$ 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 $F_{\ell}(\alpha_{i}, y) = g_{\ell,i}^{(k)} = \sum_{j=0}^{2t} h_{\ell,j}^{(k)} \cdot y^{j}$, 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 $y \sum_{j=0}^{2t} (\sum_{k=1}^{m} h_{\ell,j}^{(k)} \cdot x^{k}) \cdot y^{j} = 0$, which means all the coefficient are 0, i.e., $\forall j, \sum_{k=1}^{m} h_{\ell,j}^{(k)} \cdot x^{k} = 0$. 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 that 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 $\text{Hyb}_2$ and $\text{Hyb}_3$ are statistically close.

**Hyb$_4$:** In this hybrid, during the Authentication Phase, for each corrupted $P_i$ and $k \in [m]$, $S$ computes $P_i$'s shares of $\{r_{i,v,j}^{(k)}\}_{k \in [m]}$ with $g_{i,v,j}^{(k)}$, denoted each one by $\hat{r}_{i,v,j}^{(k)}$. $S$ does not use these $\hat{r}_{i,v,j}^{(k)}$, to do anything else. Thus, $\text{Hyb}_3$ and $\text{Hyb}_4$ have the same output distribution.

**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 $P_i$'s $\{s_{i,v,j}^{(k)}\}_{k \in [m]}$, $S$ randomly samples the whole $\{s_{i,v,j}^{(k)}\}_{k \in [m]}$ based on corrupted parties' shares and $P_i$'s share of tag. For each honest $P_i$, $S$ uses these random elements as honest parties' shares of $\{s_{i,v,j}^{(k)}\}_{k \in [m]}$ and sends them to $F_{\text{privRec}}$. Then, $S$ uses $P_i$'s share of $\{s_{i,v,j}^{(k)}\}_{k \in [m]}$ to compute his share of $\{\text{mask}^{(k)}_j\}_v$. The difference between $\text{Hyb}_4$ and $\text{Hyb}_5$ is $S$ will not use honest $P_i$'s share of $\{\text{mask}^{(k)}_j\}_v$ to compute his share of $\{s_{i,v,j}^{(k)}\}_{k \in [m]}$. Instead, $S$ samples the whole $\{s_{i,v,j}^{(k)}\}_{k \in [m]}$ based on corrupted parties' shares and $P_i$'s share of tag. Then, $S$ computes honest parties' share of $\{\text{mask}^{(k)}_j\}_v$ with $\{s_{i,v,j}^{(k)}\}_{k \in [m]}$. Since honest parties' shares of $\{s_{i,v,j}^{(k)}\}_{k \in [m]}$ are randomly sampled, their shares of $\{\text{mask}^{(k)}_j\}_v$ are also random when the corrupted parties' shares are fixed. Therefore, we only change the order of sampling the two sharings. Thus, $\text{Hyb}_4$ and $\text{Hyb}_5$ have the same output distribution.

**Hyb$_6$:** In this hybrid, during the Authentication Phase, when $P_i$ is corrupted, instead of following the protocol to compute each $P_i$'s share of $\{s_{i,v,j}^{(k)}\}_{k \in [m]}$ denoted by $\tilde{r}_{i,v,j}^{(k)}$, $S$ randomly samples the whole $\{s_{i,v,j}^{(k)}\}_{k \in [m]}$ based on corrupted parties' shares. For each honest $P_j$, $S$ uses these random elements (denoted by $\hat{r}_{i,v,j}^{(k)}$) as $P_j$'s shares of $\{s_{i,v,j}^{(k)}\}_{k \in [m]}$ and sends them to $P_i$. Since $\sum_{k=1}^{m} [e_u^{(k)} \cdot r_{i,v,j}^{(k)}]$ can be considered as $[0]^{2t}$, as we have argued in the hybrid arguments when $D$ is honest, due to the similar reason in $\text{Hyb}_2$, $\text{Hyb}_3$ and $\text{Hyb}_4$ have the same output distribution.

**Hyb$_7$:** In this hybrid, during the Authentication Phase, when $P_i$ is honest, for each corrupted $P_j$, $S$ follows the protocol to compute each corrupted $P_j$'s share of $\{s_{i,v,j}^{(k)}\}_v$ with $g_{i,v,j}^{(k)}$, then $S$ randomly samples honest $P_i$'s shares based on corrupted parties' shares and sends them to $F_{\text{privRec}}$. The difference between $\text{Hyb}_6$ and $\text{Hyb}_7$ is how we let honest $P_i$ get corrupted $P_j$'s share of $\{s_{i,v,j}^{(k)}\}_v$. In $\text{Hyb}_6$, honest $P_i$ will get corrupted $P_j$'s share of $\{s_{i,v,j}^{(k)}\}_v$ which is consistent with all honest parties' shares from $F_{\text{privRec}}$. In $\text{Hyb}_7$, $S$ uses $g_{i,v,j}^{(k)}$ to compute corrupted $P_j$'s share of $\{s_{i,v,j}^{(k)}\}_v$ and then randomly samples honest $P_i$'s shares based on corrupted parties' shares. Since $g_{i,v,j}^{(k)}$ is determined by the first $t + 1$ honest parties' column polynomials in $M$, the results of $P_j$'s share of $\{s_{i,v,j}^{(k)}\}_v$ in the two hybrids are the same. Thus, $\text{Hyb}_6$ and $\text{Hyb}_7$ have the same output distribution.

**Hyb$_8$:** In this hybrid, during the Authentication Phase, when $P_i$ is honest, for each corrupted $P_j$, when $P_i$ receives $\{\tau_{i,v,j}^{(k)}\}_{k \in [m]}$ from $P_j$, for each $k \in [m]$, $S$ checks whether $\tau_{i,v,j}^{(k)} = \tilde{r}_{i,v,j}^{(k)}$. If true, $S$ accepts $P_j$'s shares. In $\text{Hyb}_7$, honest $P_i$ will accept corrupted $P_j$'s $\{\tau_{i,v,j}^{(k)}\}_{k \in [m]}$ when $P_j$'s share of $\{s_{i,v,j}^{(k)}\}_v$ is $\tau_{i,v,j} = \sum_{k=1}^{m} \tilde{r}_{i,v,j}^{(k)} \cdot r_{i,v,j}^{(k)}$. The output distribution only changes when there exists $k \in [m]$ such that $\tilde{r}_{i,v,j}^{(k)} \neq \tilde{r}_{i,v,j}^{(k)}$. Since $P_i$ receives the whole $\{s_{i,v,j}^{(k)}\}_v$ from $F_{\text{privRec}}$, corrupted $P_j$'s share of $\{s_{i,v,j}^{(k)}\}_v$ is equal to $\sum_{k=1}^{m} \tilde{r}_{i,v,j}^{(k)} \cdot r_{i,v,j}^{(k)}$. Therefore, we consider polynomial $\sum_{k=1}^{m} (\tilde{r}_{i,v,j}^{(k)} - \tilde{r}_{i,v,j}^{(k)}) \cdot x^k$. The degree of it is at most $m$, which means this polynomial has at most $m$ roots in $F$. Since $r_{i,v,j}$ is randomly sampled by $P_i$, therefore, the probability that $r_{i,v,j}$ is a root is $m/|F|$. 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

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

which is negligible. Thus, the output distributions of $\text{Hyb}_6$ and $\text{Hyb}_8$ are statistically close.

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

according to the following equations:

\[
\begin{align*}
\tau_{i-v} &= g_{s,i} \cdot \mu_{i-v} + \sum_{k=1}^{m} r_{v}^{k} \cdot \nu_{i-v}^{(k)} \\
\hat{\tau}_{i-v} &= g_{s,i} \cdot \hat{\mu}_{i-v} + \sum_{k=1}^{m} r_{v}^{k} \cdot \nu_{i-v}^{(k)}
\end{align*}
\]

\[ \Rightarrow \tau_{i-v} - \hat{\tau}_{i-v} = (g_{s,i} - \hat{g}_{s,i}) \cdot \mu_{i-v}, \]

when there exists \( \ell \in [L'] \) such that \( g_{i,v}(y) \neq \sum_{k=1}^{m} r_{v}^{k} \cdot g_{i,v}^{(k)}(y) \) when \( g_{s,i} \neq \hat{g}_{s,i} \). Since \( \mu_{i-v} \) is randomly distributed, \( (g_{s,i} - \hat{g}_{s,i}) \cdot \mu_{i-v} \) is also randomly distributed. Therefore, the probability that \( \tau_{i-v} - \hat{\tau}_{i-v} = (g_{s,i} - \hat{g}_{s,i}) \cdot \mu_{i-v} \) is at most \( 1/|F| \). Now we take the union bound for at most \( t \) corrupted \( P_{i} \in W \) and \( 2t + 1 \) honest \( P_{v} \), the probability that the output distribution changes is at most

\[ \epsilon_{4} = t \cdot (2t + 1) \cdot \frac{1}{|F|} \leq \frac{n^{2}}{|F|}, \]

which is negligible. Thus, the output distributions of \( \text{Hyb}_{a} \) and \( \text{Hyb}_{b} \) are statically close.

**Hyb_{10}**: In this hybrid, during the Completion Phase, when \( P_{v} \) is honest, for each corrupted \( P_{i} \in W, \ell \in [L'] \) and \( k \in [m], S \) additionally checks whether \( \hat{g}_{i,v}^{(k)}(\alpha_{v}) = g_{i,v}^{(k)}(\alpha_{v}) \). In \( \text{Hyb}_{10} \), \( S \) only accepts the values sent by \( P_{i} \) when \( g_{i,v}(y) = \sum_{k=1}^{m} r_{v}^{k} \cdot \hat{g}_{i,v}^{(k)}(y) \) for each \( \ell \in [L'] \). If there exists \( \ell \in [L'], k \in [m] \) such that \( \hat{g}_{i,v}^{(k)}(\alpha_{v}) \neq g_{i,v}^{(k)}(\alpha_{v}), \) which means \( \sum_{k=1}^{m} (g_{i,v}^{(k)}(\alpha_{v}) - \hat{g}_{i,v}^{(k)}(\alpha_{v}))) \cdot r_{v}^{k} = 0. \) Similarly, the probability is at most \( m/|F| \). 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

\[ L' \cdot t \cdot (2t + 1) \cdot \frac{m}{|F|} \leq \frac{mL^{2}n^{2}}{|F|}, \]

which is negligible. Thus, the output distributions of \( \text{Hyb}_{a} \) and \( \text{Hyb}_{10} \) are statically close.

**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 \( \tau_{i-v} \), \( S \) does the following things:

1. When \( S \) emulates \( F_{\text{priv-rec}} \), if \( P_{i} \) is honest, \( S \) randomly samples the whole \( [\mu_{i-v}^{(k)}]_{k\in[m]} \) based on corrupted parties' shares. If \( P_{i} \) is corrupted, \( S \) randomly samples the whole \( [\mu_{i-v}^{(k)}]_{k\in[m]} \) based on corrupted parties' shares, then \( S \) computes \( \nu_{i-v}^{(k)} = \hat{\nu}_{i-v}^{(k)} - \hat{\nu}_{i-v}^{(k)} \cdot \mu_{i-v} \) for each \( k \in [m] \). Finally, for each \( k \in [m], S \) samples the whole \( \{\nu_{i-v}^{(k)}\}_{k\in[m]} \) based on corrupted parties' shares and \( \nu_{i-v}^{(k)} \).

2. For each honest \( P_{i} \in W, \) when \( P_{i} \) receives \( r_{i-v} \) from \( P_{v}, S \) computes \( \tau_{i-v} = g_{s,i} \cdot \mu_{i-v} + \sum_{k=1}^{m} r_{v}^{k} \cdot \nu_{i-v}^{(k)} \).

The only difference between \( \text{Hyb}_{10} \) and \( \text{Hyb}_{11} \) is the way of computing each honest \( P_{i} \)'s \( \tau_{i-v} \). In \( \text{Hyb}_{10}, S \) computes \( \tau_{i-v} = g_{s,i} \cdot \mu_{i-v} + \sum_{k=1}^{m} r_{v}^{k} \cdot \nu_{i-v}^{(k)}, \) where \( \mu_{i-v} \) and \( \{\nu_{i-v}^{(k)}\}_{k\in[m]} \) are randomly sampled based on corrupted parties' shares by \( S \). Therefore, \( \tau_{i-v} \) is also random when the corrupted parties' shares of it are fixed, as sampled in \( \text{Hyb}_{11} \). Thus, \( \text{Hyb}_{10} \) and \( \text{Hyb}_{11} \) have the same the output distribution.

**Hyb_{12}**: In this hybrid, during the Completion Phase, when \( P_{w} \) is honest, for each corrupted \( P_{i} \in W, \) when \( P_{w} \) receives \( \tau_{i-w} \) and \( \{g_{i,v}(y)\}_{\ell\in[L]} \), \( S \) doesn't follow the protocol to check the values. Instead, \( S \) checks whether \( \tau_{i-w} = \sum_{k=1}^{m} r_{v}^{k} \cdot \tau_{i-w}^{(k)} \). For each \( \ell \in [L], S \) also checks whether the degree of \( g_{i,v}(y) \) is \( 2t \) and \( g_{i,v}(y) = \sum_{k=1}^{m} r_{v}^{k} \cdot \hat{g}_{i,v}^{(k)}(y) \). If true, \( S \) accepts \( P_{i} \)'s \( \{g_{i,v}(y)\}_{\ell\in[L]} \). Due to the same reason in \( \text{Hyb}_{a} \), the output distribution of \( \text{Hyb}_{12} \) and \( \text{Hyb}_{13} \) are statistically close.

**Hyb_{13}**: In this hybrid, during the Completion Phase, when \( P_{w} \) is corrupted, for each honest \( P_{i} \in W \), instead of computing \( \tau_{i-w} \) with \( \{\tau_{i-w}^{(k)}\}_{k\in[m]} \), \( S \) computes \( \tau_{i-w} = g_{s,i} \cdot \mu_{i-w} + \sum_{k=1}^{m} r_{v}^{k} \cdot \nu_{i-w}^{(k)} \). Due to the same reason in \( \text{Hyb}_{12} \), the output distributions of \( \text{Hyb}_{12} \) and \( \text{Hyb}_{13} \) are statistically close.

**Hyb_{14}**: In this hybrid, during the Authentication Phase, if any honest party receives \( W \) from \( D \) and \( S \) checks \( W \) is correct, \( S \) reconstructs \( q_{1}(x), \ldots, q_{m}(x) \) and sends \( \{\text{Dealer}, \text{ACS}5, \{q_{1}(x), \ldots, q_{m}(x)\}\} \) to \( F_{\text{ ACS}5} \). This doesn't affect the output. Thus, \( \text{Hyb}_{13} \) and \( \text{Hyb}_{14} \) have the same output distribution.
Thus, \( \text{Hyb}_{15} \): In this hybrid, during the Completion Phase, for each honest \( P_v \), when \( P_v \) accepts \( 2t+1 \) different \( P'_t \)’s \( \{ f_{t,w}(\alpha_t) \}_{k \in [m], w \in [t]} \), instead of computing \( \{ g^t_{t,w}(\alpha_t) \}_{t \in [L], k \in [m'], w \in [t,0]} \) by himself, \( S \) lets \( \mathcal{F}_{\text{ACSS}} \) send output to \( P_v \). 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 \( \mathcal{F}_{\text{ACSS}} \) is not requested by \( D \), so it won’t affect the output distribution. If the authentication phase terminates, due to the same reason with \( \text{Hyb}_{6} \) when \( D \) is honest, the ACSS protocol must terminate and all the honest parties will 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 \( \mathcal{F}_{\text{ACSS}} \) is also determined by the row polynomials of those polynomials. Thus, \( \text{Hyb}_{14} \) and \( \text{Hyb}_{15} \) have the same output distribution.

Note that \( \text{Hyb}_{15} \) is the ideal-world scenario, \( \Pi_{\text{ACSS}} \) statistically-securely computes \( \mathcal{F}_{\text{ACSS}} \) when \( D \) is corrupted.

\[ \Box \]

### D.3 Analysis of the Communication Complexity

We make a recap of the parameters in our \( \Pi_{\text{ACSS}} \), \( D \) 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 \( \mathcal{F}_{\text{APICP}} \) 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 \( O(mL'n^2 \kappa) \) bits for all parties. Then each \( P_i \in \mathcal{P} \) broadcasts \( \text{OK}_i \) (each \( \text{OK}_i \) can be encoded with \( O(\log n) \) bits), the total communication is \( O(n^3 \log n) \) bits. Finally, \( D \) broadcasts set \( \mathcal{M} \), which requires \( O(n^3 \log n) \) bits. Here we omit the communication of each \( \mathcal{F}_{\text{APICP}} \), and we will compute it later. Therefore, we need communication of \( O(mL'n^2 \kappa + n^3 \log n) \) bits during the Sharing Phase. Since \( L' = L/n \), the communication cost is \( O(mL'n \kappa + n^3 \log n) \) bits.

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

During the Authentication Phase: For each \( P_i \in \mathcal{P} \):

1. For each \( P_i \), preparing random shares requires communication of \( O(Ln^3 \kappa + mn^4 \kappa + n^5 \kappa^2 + n^6) \) bits:
   
   (1) Each \( |\mu_{t,w}| \): \( O(Ln^3 \kappa + n^5 \kappa^2 + n^6) \) bits.
   
   (2) Each \( \{ \nu^{(k)}_{t,w} \}_{k \in [m]} \): \( O(mn^3 \kappa + n^5 \kappa^2 + n^6) \) bits.
   
   (3) Each \( \{ \nu^{(k)}_{w} \}_{w \in [t], k \in [m]} \): \( O(mn^4 \kappa + n^5 \kappa^2 + n^6) \) bits.
   
   (4) Each \( \{ \text{mask}^{(k)}_{t} \}_{t \in [L]} \): \( O(n^3 \kappa^2 + n^6) \) bits.

2. For each \( P_i \), computing \( P_i \)’s tags requires communication of \( O(mn \kappa + n^2 \kappa + n^3 \log n) \) bits:
   
   (1) Each \( P_j \in \mathcal{P} \) sends his shares of tags, resulting in a total communication of \( O(mn \kappa) \) bits.
   
   (2) \( P_i \) broadcasts a random element, which requires communication of \( O(n^2 \kappa) \) bits.
   
   (3) Executing \( \Pi_{\text{privrec}} \) requires communication of \( O(n \kappa) \) bits.
   
   (4) \( P_i \) broadcasts \( \text{Tag}_{\alpha_t} \), which requires \( O(n^2 \log n) \) bits.
   
   (5) \( D \) broadcasts \( \mathcal{W} \), which requires communication of \( O(n^3 \log n) \) bits.

The communication of realizing each \( \mathcal{F}_{\text{APICP}} \) is still omitted here, and we will compute them later. The total communication cost is \( O(Ln^5 \kappa + mn^6 \kappa + n^7 \kappa^2 + n^8) \) 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 \( O(mn \kappa + mL' \kappa + L' n \kappa) \)-bit communication:

1. \( P_v \) sends \( \{ g^{(k)}_{t,w}(\alpha_v) \}_{k \in [m], w \in [t]} \) to \( P_v \), which requires communication of \( O(mL' \kappa) \) bits.

2. Invoking \( \mathcal{F}_{\text{privrec}} \) requires communication of \( O(mn \kappa) \) bits.
3. \( P_e \) sends a random element to each \( P_i \), which requires communication of \( O(\kappa) \) bits.
4. \( P_i \) sends \( \tau_{\iota-v} \) and \( \{g_{\iota, i}(y)\}_{\iota \in [\ell]} \) to \( P_v \), which requires communication of \( O(L'n\kappa) \) bits.

Therefore, reconstructing row polynomials requires \( O(mn^3\kappa + mL'n^2\kappa + L'n^3\kappa) \) bits. During the Completion Phase, while reconstructing column polynomials, for each \( P_w \in \mathcal{P} \):
1. Each \( P_e \in \mathcal{P} \) sends \( \{f_{\iota, v}^{(k)}(\alpha_w)\}_{k \in [m], \iota \in [\ell]} \) to \( P_w \), resulting in a total communication of \( O(mL'n\kappa) \) bits.
2. \( P_e \) broadcasts a random element, which requires communication of \( O(n^2\kappa) \) bits.
3. Each \( P_i \in \mathcal{W} \) sends \( \tau_{\iota-w} \) and \( \{g_{\iota, i}(y)\}_{\iota \in [\ell]} \) to \( P_w \), resulting in a total communication of \( O(L'n^2\kappa) \) bits.

Therefore, reconstructing column polynomials requires \( O(mL'n^2\kappa + L'n^3\kappa) \) bits. Then, the total communication cost is \( O(mLn\kappa + mn^3\kappa + L'n^3\kappa) \) bits during the Completion Phase.

In the end, we consider the communication cost of \( \text{APICP} \). For each \( P_i \in \mathcal{M} \), according to Theorem 5, the communication of realizing each \( F_{\text{APICP}}(P_h, D) \) is \( O(mLk + m\ell T^2\kappa^2 + L T \kappa + nT^3\kappa^2) \) bits. Since we have defined \( m = m't + T' \) and \( T = n^3 + n \), the total communication cost is \( O(mLn\kappa + mn^3\kappa^2) \) bits.

Therefore, the protocol \( \Pi_{\text{ACSS}} \) requires communication of \( O(mLn\kappa + mn^3\kappa^2 + Ln^3\kappa) \) bits. Since \( mL = (m't + T') \cdot L'n = O(N) \), when we take \( m = n^4 \), the protocol \( \Pi_{\text{ACSS}} \) requires communication of \( O(Nn\kappa + n^{12}\kappa^2) \) bits.

### E The Asynchronous MPC Protocol

#### E.1 Functionality \( F_{\text{AMPC}} \)

The ideal functionality \( F_{\text{AMPC}} \) (see Fig. 35) from \( \text{CP23, Coh16} \) is as follows.

**Functionality \( F_{\text{AMPC}} \)**

\( F_{\text{AMPC}} \) 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 \( f : ((\{0, 1\}^* \cup \{\bot\})^{\mathbb{N}} \rightarrow (\{0, 1\}^* \cup \{\bot\})^\mathbb{N} \) for each party \( P_i \), initialize an input value \( x^{(i)} = \bot \) and output value \( y^{(i)} = \bot \).

- Upon receiving an input \( v \) from \( P_i \), if \( \text{CoreSet} \) has not been recorded yet or if \( P_i \in \text{CoreSet} \), set \( x^{(i)} = v \).
- Upon receiving a input \( \text{CoreSet} \) from \( \mathcal{S} \), verify that \( \text{CoreSet} \) is a subset of \( \mathcal{P} \) of size at least \( n - t \), else ignore the message. If \( \text{CoreSet} \) has not been recorded yet, then record \( \text{CoreSet} \) and for every \( P_i \notin \text{CoreSet} \), set \( x^{(i)} = 0 \).
- If the \( \text{CoreSet} \) has been recorded and the value \( x^{(i)} \) has been set to a value different from \( \bot \) for every \( P_i \in \text{CoreSet}, \) then compute \( (y^{(i)}_1, \ldots, y^{(i)}_{m}) = f(x^{(i)}_1, \ldots, x^{(i)}_{m}) \) and generate a request-based delayed output \( y^{(i)} \) for every \( P_i \in \mathcal{P} \).

**Figure 35:** Ideal functionality for asynchronous secure multiparty computation

#### E.2 Overview of the Construction of \( \Pi_{\text{AMPC}} \)

We recall the framework of constructing \( \Pi_{\text{AMPC}} \) in \( \text{CP17} \) 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:

- **Step1: Preparing the Beaver Triples.** Each party generates \( O(|C|) \) completely \( t \)-shared random multiplication triples \( ([a][b], [c][t]) \), where \( c = a \cdot b \). For this, each party invokes \( F_{\text{ACSS}} \) to share \( O(|C|) \) 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 \( F_{\text{ACSS}} \) 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 $x^{(i)}$ and invokes $F_{\text{ACSS}}$ 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 $\text{CoreSet}$ 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 $\text{CoreSet}$ 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 $F_{\text{privRec}}$ with his $t$-shares of the output gate for every $P_i \in P$ to help $P_i$ reconstruct his output $y^{(i)}$.

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

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