Earn While You Reveal: Private Set Intersection that Rewards Participants

***Abstract*—In Private Set Intersection protocols (PSIs), a non-empty result always reveals information about the private input sets of the parties. Moreover, in various variants of PSI, not all parties necessarily receive or are interested in the result. However, to date, the literature has assumed that those parties not receiving or not interested in the result still contribute their private input sets to the PSI free of charge, although doing so would cost them their privacy. In this work, we introduce a *multi-party* PSI, called "Anesidora". It *rewards* parties for contributing their private input sets to the protocol. Anesidora is designed with efficiency in mind, primarily relying on symmetric key primitives. Its complexities scale linearly with the number of parties and set cardinality. It maintains security even when the majority of parties are compromised by malicious colluding adversaries. Along the way, we introduce "Justitia", the first *fair multi-party PSI* and the new concept of *unforgeable polynomials*, which may hold independent significance. Anesidora has various applications, including its use in "Vertical Federate Learning" (VFL), where it can *incentivize* diverse parties to contribute their inputs to VFL and collaboratively develop global models.**

1. Introduction

Secure Multi-Party Computation (MPC) is a general-purpose cryptographic protocol that enables multiple mutually distrustful parties to collectively compute a certain functionality on their private inputs, safeguarding their inputs' privacy, beyond the result. Private Set Intersection (PSI) is a cub-class of MPC, that allows parties to compute the intersection of their private sets. It aims to efficiently attain the same security as MPC.

Two facts about PSIs exist: (i) a non-empty result always reveals something about the parties' private input sets (i.e., the set elements that are in the intersection), and (ii) various variants of PSIs do not output the result to all parties, even in those PSIs that do, not all of the parties are necessarily interested in it. Given these facts, a natural question arises: *How can we incentivize parties that do not receive the result or do not express interest in it to participate in a PSI?*

To date, the literature has not addressed the aforementioned question. It has assumed that all parties would engage in a PSI free of charge, resulting in them bearing the privacy burden (in addition to the costs associated with the PSI).[1]

In this work, we provide the first answer to the above question. We present a multi-party PSI, called "Anesidora", which enables the buyer who initiates the PSI computation (and has an interest in the result) to pay other parties proportionally based on the number of elements it learns about other parties' private inputs. Anesidora is efficient and predominantly *relies on symmetric key primitives*. Its computation and communication complexities scale linearly with the number of parties and set cardinality. Anesidora maintains security even in cases where most parties are compromised by malicious adversaries who may collude.

We construct Anesidora in a modular manner. Specifically, we introduce the formal concept of "PSI with Fair Compensation" ($\mathcal{PSI}^{\mathcal{FC}}$) and devise "Justitia" which realizes this concept. $\mathcal{PSI}^{\mathcal{FC}}$ guarantees that either all parties obtain the result, or in the event of an unfair protocol abortion (where only dishonest parties gain access to the result), honest parties will receive financial compensation. Justitia stands as *the inaugural* fair multi-party PSI.

Subsequently, we advance from the $\mathcal{PSI}^{\mathcal{FC}}$ to the "PSI with Fair Compensation and Reward" ($\mathcal{PSI}^{\mathcal{FCR}}$) notion and develop Anesidora to actualize $\mathcal{PSI}^{\mathcal{FCR}}$. The latter notion guarantees that honest parties (a) receive rewards irrespective of the honesty of all parties or if a subset aborts unfairly, and (b) are compensated in the case of an unfair termination. In pursuit of efficient PSIs, we introduce a novel primitive, called unforgeable polynomials, which may independently pique interest.

During the development of Justitia and Anesidora we had to tackle several challenges, including (i) minimizing overhead, (ii) safeguarding against collusion among malicious parties to prevent unfair payment, (iii) ensuring the transfer of the *intact* output from one subroutine as input to another, (iv) providing clients with rewards commensurate with the intersection size, and (v) maintaining the confidentiality of outgoing messages.

1. Combining a multi-party PSI with a payment mechanism that charges a fixed amount to the initiating buyer presents serious issues. Firstly, this approach compels the buyer to pay, even if malicious clients tamper with result accuracy, enabling them to gain knowledge without the buyer learning the correct result, as there exists no fair multi-party PSI in the literature. Secondly, it enforces a fixed fee irrespective of the intersection size. Overcharging dissuades buyers from participation while undercharging discourages other clients from engaging in the protocol. This highlights the need for a more equitable and flexible payment model in such scenarios.

1.1. Applications

Federated Learning (FL) is a machine learning framework where multiple parties collaboratively build machine learning models without revealing their sensitive input to their counterparts [1], [2]. Vertical Federated Learning (VFL) is an important variant of FL where the data is partitioned vertically. VFL schemes often rely on PSIs for entity alignment. The *importance of incentivizing parties in FL has already been recognized* in the literature. Efforts have been made to incentivize participants in FL, as evidenced by schemes detailed in [3], [4], [5], [6]. However, these schemes assume that the parties will adhere to the procedural instructions and that the FL process will be fully completed. We observed that it is feasible to use Anesidora to assess each party's contribution to VFL and reward them based on their data contribution. This ensures that participants receive compensation even if a malicious party prematurely aborts during the VFL execution.

Anesidora can also be utilized to (1) *enhance loan risk management*, by banks, like "WeBank" [7], that already rely on FL and PSI to gather customer data from partners, (2) *improve personalized advertising*, by marketing and advertising companies [8], [9], [10], or (3) *enhance malware detection and reduce false positives*, by antivirus companies [11]. In these cases, dataset contributors will receive rewards and be incentivized to collaborate through Anesidora.

Summary of Our Contributions. In this work, we: (1) introduce Anesidora, a *novel* multi-party PSI that incentivizes participants to contribute their set elements to the intersection, (2) introduce Justitia, which is the first fair multi-party PSI, (3) provide formal definitions and proofs for the above constructions within the simulation-based model, and (4) introduce unforgeable polynomials' concept.

2. Preliminaries

2.1. Notations and Assumptions

Table 2 in Appendix A summarizes the notations used in this paper. All arithmetic operations are defined over a field $\mathbb{F}_p$ of prime order. This paper employs a simulation-based secure computation model [12] to define and prove its protocols, addressing both active (malicious) and passive (semi-honest) adversaries. Appendix **??** presents a full version of the security model.

2.2. Smart Contracts

Cryptocurrencies, such as Bitcoin [13] and Ethereum [14], go beyond merely providing a decentralized currency; they also facilitate computations on transactions. These cryptocurrencies allow for a specific computational logic to be encoded in a computer program known as a *"smart contract"*. As of now, Ethereum stands as the foremost cryptocurrency framework that empowers users to define arbitrary smart contracts. Within this framework, contract code resides on the blockchain and is executed by all parties involved in maintaining the cryptocurrency. The correctness of program execution is ensured by the security of the underlying blockchain components. In this work, we use standard public Ethereum smart contracts.

2.3. Counter Collusion Smart Contracts

To enable a party, such as a client, to efficiently delegate computation to multiple potentially colluding third parties, like servers, Dong *et al.* [15] introduced two primary smart contracts: the "Prisoner's Contract" ($\mathcal{SC}_{\text{PC}}$) and the "Traitor's Contract" ($\mathcal{SC}_{\text{TC}}$). $\mathcal{SC}_{\text{PC}}$ is jointly signed by both the client and the servers and is designed to incentivize accurate computation. This contract mandates that each server must submit a deposit before the computation is delegated. Furthermore, it is equipped with an external auditor that can be invoked to identify any misbehaving server when it provides dissimilar results. If a server behaves honestly, it is eligible to withdraw its deposit. However, if the auditor detects a cheating server, a portion of its deposit is transferred to the client. If one server is honest while the other one cheats, the honest server receives a reward sourced from the deposit of the cheating server.

However, the dilemma, created by $\mathcal{SC}_{\text{PC}}$ between the two servers, can be resolved if they can establish an enforceable promise, such as through a "Colluder's Contract" ($\mathcal{SC}_{\text{CC}}$). In this contract, one party, referred to as the "ringleader", commits to paying a bribe to its counterpart if both parties engage in collusion and deliver an incorrect result to $\mathcal{SC}_{\text{PC}}$. To counter $\mathcal{SC}_{\text{CC}}$, Dong *et al.* proposed $\mathcal{SC}_{\text{TC}}$, which provides incentives for a colluding server to expose the other server and report the collusion without facing penalties from $\mathcal{SC}_{\text{PC}}$. In this work, we have made slight adjustments and employed these contracts. The relevant parameters for these contracts are outlined in Table 2. For a comprehensive description of the contracts, we refer readers to Appendix B.

2.4. Pseudorandom Function and Permutation

A pseudorandom function is a deterministic function that takes a key of length λ and an input; and outputs a value indistinguishable from that of a truly random function. In this paper, we use pseudorandom functions: $\texttt{PRF} : \{0,1\}^\lambda \times \{0,1\}^* \to \mathbb{F}_p$, where $\log_2(p) = \lambda$ is the security parameter. In practice, a pseudorandom function can be obtained from an efficient block cipher [16]. The definition of a pseudorandom permutation, $\texttt{PRP} : \{0,1\}^\lambda \times \{0,1\}^* \to \mathbb{F}_p$, is similar to that of a pseudorandom function, with a difference, it is required $\texttt{PRP}(k,.)$ to be indistinguishable from a uniform permutation, instead of a uniform function.

2.5. Commitment Scheme

A commitment scheme involves a *sender* and a *receiver*. It includes two phases, *commit* and *open*. In the *commit* phase, the sender commits to a message x as $\texttt{Com}(x,r) = \texttt{Com}_x$, that involves a secret value, r. In the *open* phase, the sender sends the opening $\ddot{x} := (x,r)$ to the receiver which verifies its correctness: $\texttt{Ver}(\texttt{Com}_x, \ddot{x}) \stackrel{?}{=} 1$ and accepts if the output is 1. A commitment scheme must satisfy: (a) *hiding*: it is infeasible for an adversary to learn any information about the committed message x, until the commitment com

is opened, and (b) *binding*: it is infeasible for a malicious sender to open a commitment com to different values than that was used in the commit phase. We refer readers to Appendix C for further details.

2.6. Hash Tables

A hash table is an array of bins, each containing a set of elements, and is paired with a hash function. To insert an element, we compute the element's hash and place the element into the bin corresponding to the computed hash value. In this paper, we ensure that the number of elements in each bin does not surpass a predefined capacity. By considering the maximum number of elements as c and the maximum size of a bin as d, we can calculate the number of bins, denoted as h, through an analysis of hash tables using the "balls into the bins" model [17]. Appendix D explains how the hash table parameters are configured. In this paper, we use hash tables to partition parties' sets into smaller sets to achieve efficiency.

2.7. Merkle Tree

A Merkle tree is a data structure that facilitates a concise commitment to a set of values or blocks, involving two parties: a prover and a verifier. The Merkle tree scheme includes three algorithms; namely, `MT.genTree`, `MT.prove`, and `MT.verify`. Briefly, the first algorithm constructs a Merkle tree on a set of blocks, the second generates a proof of a block's (or set of blocks') membership, and the third verifies the proof. Appendix E provides more details.

2.8. Polynomial Representation of Sets

In this representation, set elements $S = \{s_1, ..., s_d\}$ are defined over a field $\mathbb{F}_p$ and set S is represented as a polynomial: $\mathbf{p}(x) = \prod_{i=1}^{d}(x - s_i)$, where $\mathbf{p}(x) \in \mathbb{F}_p[X]$ and $\mathbb{F}_p[X]$ is a polynomial ring [18]. As shown in [19], [20], for sets $S^{(A)}$ and $S^{(B)}$ represented by polynomials $\mathbf{p}_A$ and $\mathbf{p}_B$, their product, $\mathbf{p}_A \cdot \mathbf{p}_B$, represents the set union, while their greatest common divisor, $gcd(\mathbf{p}_A, \mathbf{p}_B)$, represents the intersection.

For polynomials $\mathbf{p}_A$ and $\mathbf{p}_B$ of degree d, and random polynomials $\boldsymbol{\gamma}_A$ and $\boldsymbol{\gamma}_B$ of degree d, it is proven in [19], [20] that: $\boldsymbol{\theta} = \boldsymbol{\gamma}_A \cdot \mathbf{p}_A + \boldsymbol{\gamma}_B \cdot \mathbf{p}_B = \boldsymbol{\mu} \cdot gcd(\mathbf{p}_A, \mathbf{p}_B)$, where $\boldsymbol{\mu}$ is a uniformly random polynomial, and $\boldsymbol{\theta}$ contains only information about the elements in $S^{(A)} \cap S^{(B)}$. In this paper, the parties employ this technique to compute polynomial $\boldsymbol{\theta}$ that encodes the intersection. Given a polynomial $\boldsymbol{\theta}$ that encodes sets intersection, one can find the set elements in the intersection through one of the following two approaches:

1) *polynomial evaluation*: the party in possession of one of the original input sets, for example, $\mathbf{p}_A$, evaluates $\boldsymbol{\theta}$ at every element s_i of $\mathbf{p}_A$ and considers s_i in the intersection if $\mathbf{p}_A(s_i) = 0$.
2) *root extraction*: the party who does not have one of the original input sets, extracts the roots of $\boldsymbol{\theta}$, which contain the roots of (i) random polynomial $\boldsymbol{\mu}$ and (ii) the polynomial that represents the intersection, i.e., $gcd(\mathbf{p}_A, \mathbf{p}_B)$.

Within Approach 2, to distinguish errors (i.e., roots of $\boldsymbol{\mu}$) from the intersection, PSIs in [21], [20] use the *"hash-based padding technique"*. In this technique, every element u_i in the set universe $\mathcal{U}$, becomes $s_i = u_i || \mathtt{H}(u_i)$, where `H` is a cryptographic hash function with a sufficiently large output size. Given a field's arbitrary element, $s \in \mathbb{F}_p$ and `H`'s output size $|\mathtt{H}(.)|$, we can parse s into x_1 and x_2, such that $s = x_1 || x_2$ and $|x_2| = |\mathtt{H}(.)|$. In a PSI that uses polynomial representation and this padding technique, after we extract each root of $\boldsymbol{\theta}$, say s, we parse it into (x_1, x_2) and check $x_2 \stackrel{?}{=} \mathtt{H}(x_1)$. If the equation holds, then we consider s as an element of the intersection. In Justitia and Anesidora, the clients will use Approach 1: polynomial evaluation, while in Anesidora the auditor of the counter collusion contracts will use approach 2: root extraction.

2.9. Horner's Method

Horner's method [22] enables an efficient evaluation of polynomials at a specified point, e.g., x_0. In particular, when dealing with a polynomial in the following form: $\boldsymbol{\tau}(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + ... + a_d \cdot x^d$ and a given point: x_0, one can efficiently evaluate the polynomial at x_0 iteratively using the following method: $\boldsymbol{\tau}(x_0) = a_0 + x_0(a_1 + x_0(a_2 + ... + x_0(a_{d-1} + x_0 \cdot a_d)...)))$. Evaluating a polynomial of degree d naively requires d additions and $\frac{(d^2+d)}{2}$ multiplications. However, using Horner's method the evaluation requires only d additions and $2d$ multiplications. We use this method in this paper.

2.10. Oblivious Linear Function Evaluation

Oblivious Linear function Evaluation (`OLE`) is a two-party protocol that involves a sender and receiver. In `OLE`, the sender has two inputs $a, b \in \mathbb{F}_p$ and the receiver has a single input, $c \in \mathbb{F}_p$. The protocol allows the receiver to learn only $s = a \cdot c + b \in \mathbb{F}_p$, while the sender learns nothing. Ghosh *et al.* [23] proposed an efficient `OLE` that has $O(1)$ overhead and involves mainly symmetric key operations. Later, in [24] an enhanced `OLE`, called $\mathtt{OLE}^+$ was proposed. The latter ensures that the receiver cannot learn anything about the sender's inputs, even if it sets its input to 0. In this paper, we use $\mathtt{OLE}^+$. We refer readers to Appendix F, for its construction.

2.11. Coin-Tossing Protocol

A Coin-Tossing protocol, `CT`, allows two mutually distrustful parties, say A and B, to jointly generate a single random bit. Formally, `CT` computes the functionality $f_{\mathtt{CT}}(in_A, in_B) \rightarrow (out_A, out_B)$, which takes in_A and in_B as inputs of A and B respectively and outputs out_A to A and out_B to B, where $out_A = out_B$. A basic security requirement of a `CT` is that the resulting bit is (computationally) indistinguishable from a truly random bit. Two-party coin-tossing protocols can be generalised to *multi-party* coin-tossing ones to generate a *random string*. The overheads of multi-party coin-tossing protocols are often linear with the number of participants. In this paper, any secure multi-party

CT that generates a random string can be used. For the sake of simplicity, we allow a multi-party f_{CT} to take m inputs and output a single value, i.e., $f_{\text{CT}}(in_1, ..., in_m) \rightarrow out$. We refer readers to Appendix G for further details.

3. Overview of Multi-party PSI with Fair Compensation's Definition

Now, we present the concept of multi-party PSI with Fair Compensation ($\mathcal{PSI}^{\mathcal{FC}}$) which ensures that either all clients receive the result or honest parties are financially compensated in the event of an unfair protocol abortion, wherein only dishonest parties gain access to the result. In a $\mathcal{PSI}^{\mathcal{FC}}$, three types of parties are involved: (1) a set of clients $\{A_1, ..., A_m\}$ potentially *malicious* (i.e., active adversaries), where all but one may collude with each other, (2) a non-colluding dealer, D, potentially semi-honest, and (3) an auditor, Aud, potentially semi-honest, where all parties except Aud have input set. For simplicity, we assume that one can determine whether a given address belongs to Aud. The fundamental functionality computed by any multi-party PSI can be defined as $f^{\text{PSI}}(S_1, ..., S_m) \rightarrow (\underbrace{S_\cap, ..., S_\cap}_{m})$, where $S_\cap = S_1 \cap S_2, ..., \cap\, S_m$. We refer readers to Appendix H for formal definitions of $\mathcal{PSI}^{\mathcal{FC}}$.

4. Other Subroutines Used in Justitia

4.1. Verifiable Oblivious Polynomial Randomization (VOPR)

In VOPR, two types of party are involved, (i) a sender, potentially a passive adversary, and (ii) a receiver, potentially an active adversary. The protocol enables the receiver, with input polynomial $\boldsymbol{\beta}$ (of degree e'), and the sender, with input random polynomials $\boldsymbol{\psi}$ (of degree e) and $\boldsymbol{\alpha}$ (of degree $e + e'$), to compute: $\boldsymbol{\theta} = \boldsymbol{\psi} \cdot \boldsymbol{\beta} + \boldsymbol{\alpha}$, satisfying the following conditions: (a) the receiver learns only $\boldsymbol{\theta}$ and gains no knowledge of the sender's input even when setting $\boldsymbol{\beta} = 0$, (b) the sender gains no knowledge, and (c) protocol detects the receiver's misbehavior. Thus, the functionality that VOPR computes is defined as $f^{\text{VOPR}}((\boldsymbol{\psi}, \boldsymbol{\alpha}), \boldsymbol{\beta}) \rightarrow (\perp, \boldsymbol{\psi} \cdot \boldsymbol{\beta} + \boldsymbol{\alpha})$.

We will use VOPR in Justitia (in steps 6 and 7) for two main purposes: (a) to enable a party to re-randomize its counterpart's polynomial (representing its set) and (b) to impose a MAC-like structure on the randomized polynomial. This structure will allow a verifier to detect any modifications to VOPR's output.

Now, we will outline how we design VOPR without using any zero-knowledge proofs.[2] In the setup phase, both parties represent their input polynomials in the regular coefficient form. Therefore, the sender's polynomials are defined as $\boldsymbol{\psi} = \sum_{i=0}^{e} g_i \cdot x^i$ and $\boldsymbol{\alpha} = \sum_{j=0}^{e+e'} a_j \cdot x^j$ and the receiver's polynomial is defined as $\boldsymbol{\beta} = \sum_{i=0}^{e'} b_i \cdot x^i$, where $b_i \neq 0$. But, the sender computes each coefficient a_j (of polynomial $\boldsymbol{\alpha}$) as follows, $a_j = \sum_{t,k=0}^{t=e,\, k=e'} a_{t,k}$, where $t + k = j$ and each $a_{t,k}$ is a random value. For instance, if $e = 4$ and $e' = 3$, then $a_3 = a_{0,3} + a_{3,0} + a_{1,2} + a_{2,1}$. Shortly, we explain why polynomial $\boldsymbol{\alpha}$ is constructed this way.

In the computation phase, to compute polynomial $\boldsymbol{\theta}$, the two parties interactively multiply and add the related coefficients using OLE^+. Specifically, for every j (where $0 \leq j \leq e'$) the sender sends g_i and $a_{i,j}$ to an instance of OLE^+, while the receiver sends b_j to the same instance, which returns $c_{i,j} = g_i \cdot b_j + a_{i,j}$ to the receiver. This process is repeated for every i, where $0 \leq i \leq e$. Then, the receiver uses $c_{i,j}$ values to construct the resulting polynomial, $\boldsymbol{\theta} = \boldsymbol{\psi} \cdot \boldsymbol{\beta} + \boldsymbol{\alpha}$.

The sender imposes the above structure on (the coefficients of) $\boldsymbol{\alpha}$ during the setup to enable the parties to securely compute $\boldsymbol{\theta}$ via OLE^+. This structure serves two key purposes: (1) allowing the sender to blind each product $g_i \cdot b_j$ with random value $a_{i,j}$ which is a component of $\boldsymbol{\alpha}$'s coefficient and (2) enabling the receiver to construct a result polynomial in the form $\boldsymbol{\theta} = \boldsymbol{\psi} \cdot \boldsymbol{\beta} + \boldsymbol{\alpha}$.

To verify the result's correctness, the sender selects a random value, z, and transmits it to the receiver. Subsequently, the receiver computes $\boldsymbol{\theta}(z)$ and $\boldsymbol{\beta}(z)$ and sends these two values to the sender. The sender computes $\boldsymbol{\psi}(z)$ and $\boldsymbol{\alpha}(z)$ and proceeds to validate the equation $\boldsymbol{\theta}(z) = \boldsymbol{\psi}(z) \cdot \boldsymbol{\beta}(z) + \boldsymbol{\alpha}(z)$. If the check is successful, the sender accepts the result. Figure 1 describes VOPR in detail.

Note that VOPR requires the sender to insert non-zero coefficients, i.e., $b_i \neq 0$ for all $i, 0 \leq i \leq e'$. If the sender includes a zero-coefficient, then it will only obtain a random value (due to OLE^+), making it fail to pass VOPR's verification phase. However, this requirement does not affect the correctness of Justitia, as we will explain in Section 5.1 and Appendix O.3. Appendix I formally states the security of VOPR and proves it.

4.2. Zero-sum Pseudorandom Values Agreement Protocol (ZSPA)

ZSPA allows m parties (the majority of which is potentially malicious) to efficiently agree on (a set of vectors, where each i-th vector has) m pseudorandom values such that their sum equals zero. At a high level, the parties first sign a smart contract, register their addresses in it and run a coin-tossing protocol CT to agree on a key: k. Next, one of the parties generates $m - 1$ pseudorandom values $z_{i,j}$ (where $1 \leq j \leq m - 1$) using key k and PRF. It sets the last value as the additive inverse of the sum of the values generated, i.e., $z_{i,m} = -\sum_{j=1}^{m-1} z_{i,j}$ (similar to the XOR-based secret sharing [26]).

Then, it constructs a Merkel tree on top of the pseudorandom values and stores only the tree's root g and the key's hash value q in the smart contract. Then, each party (using the key) checks if the values (on the contract) have been computed correctly; if so, it sends a (signed) "approved"

2. Previously, Ghosh *et al.* [24] designed a protocol to meet similar security requirements that we laid out above. But, as shown in [25], it is susceptible to several serious attacks.

- *Input.*
  - *Public Parameters*: upper bound on input polynomials' degree: e and e'.
  - *Sender Input*: random polynomials: $\boldsymbol{\psi} = \sum_{i=0}^{e} g_i \cdot x^i$ and $\boldsymbol{\alpha} = \sum_{j=0}^{e+e'} a_j \cdot x^j$, where $g_i \xleftarrow{\$} \mathbb{F}_p$. Each a_j has the form: $a_j = \sum_{\substack{t,k=0 \\ k=e' \\ t=e}}^{} a_{t,k}$, such that $t+k=j$ and $a_{t,k} \xleftarrow{\$} \mathbb{F}_p$.
  - *Receiver Input*: polynomial $\boldsymbol{\beta} = \boldsymbol{\beta}_1 \cdot \boldsymbol{\beta}_2 = \sum_{i=0}^{e'} b_i \cdot x^i$, where $\boldsymbol{\beta}_1$ is a random polynomial of degree 1, $\boldsymbol{\beta}_2$ is an arbitrary polynomial of degree $e'-1$, and $b_i \neq 0$.
- *Output.* The receiver obtains $\boldsymbol{\theta} = \boldsymbol{\psi} \cdot \boldsymbol{\beta} + \boldsymbol{\alpha}$.

1) **Computation:**
   a) Sender and receiver together for every j, $0 \leq j \leq e'$, invoke $e+1$ instances of $\texttt{OLE}^+$. In particular, $\forall j, 0 \leq j \leq e'$: sender sends g_i and $a_{i,j}$ while the receiver sends b_j to $\texttt{OLE}^+$ that returns: $c_{i,j} = g_i \cdot b_j + a_{i,j}$ to the receiver $(\forall i, 0 \leq i \leq e)$.
   b) The receiver sums component-wise values $c_{i,j}$ that results:
$$\boldsymbol{\theta} = \boldsymbol{\psi} \cdot \boldsymbol{\beta} + \boldsymbol{\alpha} = \sum_{\substack{i,j=0 \\ j=e' \\ i=e}}^{} c_{i,j} \cdot x^{i+j}$$
2) **Verification:**
   a) Sender: picks a random non-zero value z and sends it to the receiver.
   b) Receiver: sends $\theta_z = \boldsymbol{\theta}(z)$ and $\beta_z = \boldsymbol{\beta}(z)$ to the sender.
   c) Sender: computes $\psi_z = \boldsymbol{\psi}(z)$ and $\alpha_z = \boldsymbol{\alpha}(z)$ and checks if equation $\theta_z = \psi_z \cdot \beta_z + \alpha_z$ holds. If the equation holds, it concludes that the computation was performed correctly. Otherwise, it aborts.

Figure 1: Verifiable Oblivious Polynomial Randomization.

message to the contract which only accepts messages from registered parties. Hence, the functionality that ZSPA computes is: $f^{\texttt{ZSPA}}(\underbrace{\perp, ..., \perp}_{m}) \rightarrow \underbrace{((k,g,q), ..., (k,g,q))}_{m}$, where g is the root of the Markle tree built on the pseudorandom values $z_{i,j}$, q is the hash value of the key used to generate the pseudorandom values, and $m \geq 2$. Figure 2 presents ZSPA in detail.

An extension of ZSPA will be used in (step 3 and 13a) Justitia to enable clients $\{A_1, ..., A_m\}$ to provably agree on sets of pseudorandom values. Every client uses the elements of these sets (as coefficients) to generate a distinct pseudorandom polynomial such that when all clients' pseudorandom polynomials are added it will result in 0, due to ZSPA's property. Each of these polynomials will be used by a client to blind the messages it sends to the smart contract, to protect the privacy of the plaintext message (from Aud, D, and the public). To compute the sum of the plaintext messages, one can easily sum the blinded messages, which removes the blinding polynomials.

Appendix J formally states the security of ZSPA and proves ZSPA's security.

- Parties. A set of clients $\{A_1, ..., A_m\}$.
- Input. m: the total number of participants, adr: an address of a deployed smart contract which maintains the clients' addresses and accepts messages only from them, and b: the total number of vectors. Let $b' = b - 1$.
- Output. k: a secret key that generates b vectors $[z_{0,1}, ..., z_{0,m}], ..., [z_{b',1}, ..., z_{b',m}]$, q: hash of the key, g: a Merkle tree's root, and a vector of (signed) messages.

1) **Coin-tossing.** $\texttt{CT}(in_1, ..., in_m) \rightarrow k$.
   All participants run a coin-tossing protocol to agree on PRF's key, k.
2) **Encoding.** $\texttt{Encode}(k, m) \rightarrow (g, q)$.
   One of the parties takes the following steps:
   a) for every i $(0 \leq i \leq b')$, generates m pseudorandom values. $\forall j, 1 \leq j \leq m-1$:
$$z_{i,j} = \texttt{PRF}(k, i||j), \quad z_{i,m} = -\sum_{j=1}^{m-1} z_{i,j}$$
   b) constructs a Merkel tree on top of all pseudorandom values, $\texttt{MT.genTree}(z_{0,1}, ..., z_{b',m}) \rightarrow g$.
   c) sends the tree's root: g, and the key's hash: $q = \texttt{H}(k)$ to adr.
3) **Verification.** $\texttt{Verify}(k, g, q, m, flag) \rightarrow (a, s)$. Each party A_t checks if, all $z_{i,j}$ values, the root g, and key's hash q have been correctly generated, by retaking step 2.
   - If the checks pass, then it sets $a = 1$ and
     - if $flag = A$, where $A \in \{A_1, ..., A_m\}$, then it sets s to a (signed) "approved" message, and sends s to adr.
     - if $flag = A$, where $A \notin \{A_1, ..., A_m\}$, then it sets $s = \perp$.
   - If the checks do not pass, it aborts by returning $a = 0$ and $s = \perp$.

Figure 2: Zero-sum Pseudorandom Values Agreement (ZSPA). Note, $flag$ lets an auditor locally run the verification without having to send a message to the smart contract.

4.3. ZSPA's Extension: ZSPA with an External Auditor (ZSPA-A)

Now, we present an extension of ZSPA, called ZSPA-A, which lets a (trusted) third-party auditor, Aud, help identify misbehaving clients in the ZSPA and generate a vector of random polynomials with a specific structure. Specifically, the functionality that ZSPA-A computes is defined as $f^{\texttt{ZSPA-A}}(\underbrace{\perp, ..., \perp}_{m}, b, \boldsymbol{\zeta}) \rightarrow \big(\underbrace{(k,g,q), ..., (k,g,q)}_{m}, \boldsymbol{\mu}^{(1)}, ..., \boldsymbol{\mu}^{(m)}\big)$, where g is the root of the Markle tree built on the pseudorandom values $z_{i,j}$, q is the hash value of the key used to generate the pseudorandom values, $m \geq 2$, b is constant value, $\boldsymbol{\zeta}$ is a random polynomial of degree 1, each $\boldsymbol{\mu}^{(j)}$ has the form $\boldsymbol{\mu}^{(j)} = \boldsymbol{\zeta} \cdot \boldsymbol{\xi}^{(j)} - \boldsymbol{\tau}^{(j)}$, $\boldsymbol{\xi}^{(j)}$ is a random polynomial of degree $b-2$, and $\boldsymbol{\tau}^{(j)} = \sum_{i=0}^{b-1} z_{i,j} \cdot x^i$.

Informally, ZSPA-A ensures that misbehaving parties are always detected, except with a negligible probability. Aud of this protocol will be invoked by Justitia when Justitia's smart contract detects that a combination of the messages sent by the clients is not well-formed. Later, in Justitia's proof,

we will show that even a *semi-honest* Aud who observes all messages that clients send to Justitia's smart contracts, cannot learn anything about their set elements. We present ZSPA-A in Figure 3.

- Parties. A set of clients $\{A_1, ..., A_m\}$ and an external auditor, Aud.
- Input. m: the total number of participants (excluding the auditor), $\boldsymbol{\zeta}$: a random polynomial of degree 1, b: the total number of vectors, and adr: a deployed smart contract's address. Let $b' = b - 1$.
- Output of each A_j. k: a secret key that generates b vectors $[z_{0,1}, ..., z_{0,m}], ..., [z_{b',1}, ..., z_{b',m}]$ of pseudorandom values, h: hash of the key, g: a Merkle tree's root, and a vector of (signed) messages.
- Output of Aud. L: a list of misbehaving parties' indices, and $\vec{\mu}$: a vector of random polynomials.

1) ZSPA **invocation.** $\texttt{ZSPA}(\perp, ..., \perp) \rightarrow \Big((k, g, q), ..., (k, g, q)\Big)$.
All parties in $\{A_1, ..., A_m\}$ call the same instance of ZSPA, which results in $(k, g, q), ..., (k, g, q)$.
2) **Auditor computation.** $\texttt{Audit}(m, \vec{k}, q, \boldsymbol{\zeta}, b, g) \rightarrow (L, \vec{\mu})$. Aud takes the below steps. Note, each $k_j \in \vec{k}$ is given by A_j. An honest party's input, k_j, equals k, where $1 \leq j \leq m$.
   a) runs the checks in the verification phase (i.e., Phase 3) of ZSPA for every j, i.e., $\texttt{Verify}(k_j, g, q, m, auditor) \rightarrow (a_j, s)$.
   b) appends j to L, if any checks fails, i.e., if $a_j = 0$. In this case, it skips the next two steps for the current j.
   c) For every i (where $0 \leq i \leq b'$), it recomputes m pseudorandom values: $\forall j, 1 \leq j \leq m-1 : z_{i,j} = \texttt{PRF}(k, i||j)$, $z_{i,m} = -\sum_{j=1}^{m-1} z_{i,j}$.
   d) generates polynomial $\boldsymbol{\mu}^{(j)}$ as: $\boldsymbol{\mu}^{(j)} = \boldsymbol{\zeta} \cdot \boldsymbol{\xi}^{(j)} - \boldsymbol{\tau}^{(j)}$, where $\boldsymbol{\xi}^{(j)}$ is a random polynomial of degree $b' - 1$ and $\boldsymbol{\tau}^{(j)} = \sum_{i=0}^{b'} z_{i,j} \cdot x^i$. By this step, a vector $\vec{\mu}$ containing at most m polynomials is generated.
   e) returns list L and $\vec{\mu}$.

Figure 3: ZSPA with an external auditor (ZSPA-A).

Theorem 1. *If* ZSPA *is secure,* H *is second-preimage resistant, and the correctness of* PRF*,* H*, and Merkle tree hold, then* ZSPA-A *securely computes* $f^{\texttt{ZSPA-A}}$ *in the presence of* $m - 1$ *malicious adversaries.*

Appendix K presents the proof of Theorem 1.

4.4. Unforgeable Polynomials

In this section, we introduce the new concept of unforgeable polynomials. Informally, an unforgeable polynomial is accompanied by a secret factor. To verify that an unforgeable polynomial has not been tampered with, a verifier can check whether the polynomial is divisible by the secret factor.

To transform an arbitrary polynomial $\boldsymbol{\pi}$ of degree d into an unforgeable polynomial $\boldsymbol{\theta}$, we can follow these steps: (i) select two secret random polynomials $(\boldsymbol{\zeta}, \boldsymbol{\gamma})$ and (ii) calculate $\boldsymbol{\theta} = \boldsymbol{\zeta} \cdot \boldsymbol{\pi} + \boldsymbol{\gamma} \bmod p$, where $deg(\boldsymbol{\zeta}) = 1$, and $deg(\boldsymbol{\gamma}) = d + 1$.

To determine whether $\boldsymbol{\theta}$ has been tampered with, a verifier (given $\boldsymbol{\theta}, \boldsymbol{\gamma}$, and $\boldsymbol{\zeta}$) can check if $\boldsymbol{\theta} - \boldsymbol{\gamma}$ is divisible by $\boldsymbol{\zeta}$. The security of an unforgeable polynomial asserts that an adversary (who is unaware of the two secret random polynomials) cannot tamper with an unforgeable polynomial without being detected, except with a negligible probability. Appendix L formally states it and proves its security.

An interesting feature of an unforgeable polynomial is that the verifier can perform the check without needing knowledge of the original polynomial $\boldsymbol{\pi}$. Another appealing aspect of the unforgeable polynomial is its support for *linear combination* and, consequently, *batch verification*. Specifically, to transform n arbitrary polynomials $[\boldsymbol{\pi}_1, ..., \boldsymbol{\pi}_n]$ into unforgeable polynomials, one can construct $\boldsymbol{\theta}_i = \boldsymbol{\zeta} \cdot \boldsymbol{\pi}_i + \boldsymbol{\gamma}_i \bmod p$, where $\forall i, 1 \leq i \leq n$.

To verify the integrity of all polynomials $[\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_n]$, a verifier can (i) compute their sum $\boldsymbol{\chi} = \sum_{i=1}^{n} \boldsymbol{\theta}_i$ and (ii) check if $\boldsymbol{\chi} - \sum_{i=1}^{n} \boldsymbol{\gamma}_i$ is divisible by $\boldsymbol{\zeta}$. The security of unforgeable polynomials states that an adversary (who is unaware of the two secret random polynomials for each $\boldsymbol{\theta}_i$) cannot tamper with any subset of the unforgeable polynomials without being detected, except with a negligible probability.

Appendix M formally states it and proves its security. In Justitia, we will employ unforgeable polynomials to enable a smart contract to efficiently verify the integrity of the polynomials sent by clients, ensuring that they are indeed the outputs of VOPR.

5. Concrete Construction of $\mathcal{PSI}^{\mathcal{FC}}$

5.1. An Overview of Justitia (JUS)

At a very high level, Justitia (JUS) operates as follows. Initially, each client encodes its set elements into a polynomial. All clients collectively sign a smart contract $\mathcal{SC}_{\texttt{JUS}}$ and make a predefined coin deposit into it. One of the clients, designated as the dealer, D, takes on the responsibility of randomizing the polynomials of the remaining clients and applying a specific structure to their respective polynomials. The clients also randomize D's polynomials. Next, all clients transmit their randomized polynomials to $\mathcal{SC}_{\texttt{JUS}}$.

$\mathcal{SC}_{\texttt{JUS}}$ consolidates all the polynomials and verifies whether the resultant polynomial adheres to the structure imposed by D. If it determines that the resultant polynomial lacks the prescribed structure, it triggers an auditor, Aud, to identify clients who may have acted improperly and impose penalties on them. However, if the resultant polynomial maintains the required structure, $\mathcal{SC}_{\texttt{JUS}}$ produces an encoded polynomial and reimburses the deposits made by the clients. In this case, all clients can utilize the encoded polynomial (provided by $\mathcal{SC}_{\texttt{JUS}}$) to find the intersection. Figure 7 in Appendix N outlines the interaction among the parties.

One of the novelties of JUS is its *lightweight verification* mechanism, which enables a smart contract $\mathcal{SC}_{\texttt{JUS}}$ to efficiently validate the accuracy of clients' messages while preserving the confidentiality of the clients' sets. To achieve this, D randomizes each client's polynomials and constructs

unforgeable polynomials on the randomized polynomials. If any client alters an unforgeable polynomial it receives and subsequently sends the modified polynomial to $\mathcal{SC}_{\texttt{JUS}}$, then $\mathcal{SC}_{\texttt{JUS}}$ can detect this manipulation by verifying if the sum of all clients' (unforgeable) polynomials is divisible by a specific polynomial of degree 1. This verification is considered lightweight for several reasons: (i) it does not use any public key cryptography (e.g., zero-knowledge proofs), which are computationally expensive, (ii) it only requires polynomial division, and (iii) it can perform batch verification by aggregating all clients' randomized polynomials and then verifying the result correctness.

Now, we will delve into a more detailed description of JUS. Initially, all clients sign and deploy $\mathcal{SC}_{\texttt{JUS}}$, each contributing a predetermined deposit amount to it. Then, they collectively execute a coin-tossing protocol CT to reach a consensus on a key, mk. This key will be used to generate a series of blinding polynomials, concealing the final result from the public. Following this, each client maps its set elements to a hash table and represents the contents of each bin (in the hash table) as a polynomial, $\boldsymbol{\pi}$.

Afterward, for each bin, the following steps are undertaken. All clients, excluding D, participate in ZSPA-A to reach an agreement on a set of pseudorandom blinding factors, ensuring that their sum equals zero. Next, D takes on the task of randomizing each client's polynomial $\boldsymbol{\pi}$ and constructing an unforgeable polynomial on it. To achieve this, D and every other client, C, participate in VOPR. This process results in C receiving a polynomial. Following this, D and each C invoke VOPR once more, this time to randomize D's polynomial, resulting in C obtaining another unforgeable polynomial. Note that the output of VOPR does not reveal any information about any client's original polynomial $\boldsymbol{\pi}$. This is because D has previously concealed this polynomial with another secret random polynomial during the execution of VOPR.

Each C adds the two polynomials together, applies blinding factors (using the output of ZSPA-A), and then forwards the result to $\mathcal{SC}_{\texttt{JUS}}$. Once all clients have transmitted their input polynomials to $\mathcal{SC}_{\texttt{JUS}}$, D sends a **switching polynomial** to $\mathcal{SC}_{\texttt{JUS}}$. This switching polynomial allows $\mathcal{SC}_{\texttt{JUS}}$ to obliviously alter the secret blinding polynomials previously employed by D during the execution of VOPR. This alteration ensures that each client's original polynomial $\boldsymbol{\pi}$ is blinded with a different blinding polynomial, which can be independently removed by the clients using mk.

Following this, D transmits a secret polynomial $\boldsymbol{\zeta}$ to $\mathcal{SC}_{\texttt{JUS}}$. This polynomial enables $\mathcal{SC}_{\texttt{JUS}}$ to validate the correctness of unforgeable polynomials. Subsequently, $\mathcal{SC}_{\texttt{JUS}}$ sums all clients' polynomials together and checks if $\boldsymbol{\zeta}$ can divide the sum. $\mathcal{SC}_{\texttt{JUS}}$ approves the clients' inputs if the polynomial can divide the sum; otherwise, it invokes Aud to identify misbehaving parties.

In case of misbehavior, all honest parties receive a refund of their deposits, while the deposits of the misbehaving parties are redistributed among the honest ones. If all clients behave honestly, then each client can independently ascertain the intersection. To achieve this, they use mk to remove the blinding polynomial from the sum (that the contract generated), evaluate the unblinded polynomial at each of their set elements, and consider an element to be part of the intersection if the evaluation yields zero.

5.2. Detailed Description of JUS

Now, we will elaborate on how JUS operates (for a description of the main notations used, see Table 2 in Appendix A).

1) All clients in $CL = \{A_1, ..., A_m, D\}$ sign a smart contract: $\mathcal{SC}_{\texttt{JUS}}$ and deploy it to a blockchain. Each client obtains the address of the deployed contract. Also, all clients participate in CT to agree on a secret master key, mk.
2) Each client in CL builds a hash table, HT, and inserts the set elements into it, i.e., $\forall i : \texttt{H}(s_i) = indx$, then $s_i \rightarrow \texttt{HT}_{indx}$. It pads every bin with random dummy elements to d elements (if needed). Then, for every bin, it constructs a polynomial whose roots are the bin's content: $\boldsymbol{\pi} = \prod_{i=1}^{d}(x - s'_i)$, where s'_i is either s_i or a random value.
3) Every client C in $CL \setminus D$, for every bin, agree on $b = 3d + 3$ vectors of pseudorandom blinding factors: $z_{i,j}$, such that the sum of each vector elements is zero, i.e., $\sum_{j=1}^{m} z_{i,j} = 0$, where $0 \leq i \leq b - 1$. To do that, they participate in step 1 of ZSPA-A. By the end of this step, for each bin, they agree on a secret key k (that will be used to generate the zero-sum values) as well as two values stored in $\mathcal{SC}_{\texttt{JUS}}$, i.e., q: the key's hash value and g: a Merkle tree's root. After time t_1, D ensures that all other clients have agreed on the vectors (i.e., all provided "approved" to the contract). If the check fails, it halts.
4) Each client in CL deposits $\ddot{y}+\ddot{ch}$ amount to $\mathcal{SC}_{\texttt{JUS}}$. After time t_2, $\mathcal{SC}_{\texttt{JUS}}$ checks if $(\ddot{y} + \ddot{ch}) \cdot (m + 1)$ amount has been deposited. If the check fails, it refunds the clients' deposit and halts.
5) D picks a random polynomial $\boldsymbol{\zeta} \xleftarrow{\$} \mathbb{F}_p[X]$ of degree 1, for each bin. It, for each C, allocates to each bin two random polynomials: $\boldsymbol{\omega}^{(D,C)}, \boldsymbol{\rho}^{(D,C)} \xleftarrow{\$} \mathbb{F}_p[X]$ of degree d, and two random polynomials: $\boldsymbol{\gamma}^{(D,C)}, \boldsymbol{\delta}^{(D,C)} \leftarrow \mathbb{F}_p[X]$ of degree $3d+1$. Also, each C, for each bin, picks two random polynomials: $\boldsymbol{\omega}^{(C,D)}, \boldsymbol{\rho}^{(C,D)} \xleftarrow{\$} \mathbb{F}_p[X]$ of degree d, and checks polynomials $\boldsymbol{\omega}^{(C,D)} \cdot \boldsymbol{\pi}^{(C)}$ and $\boldsymbol{\rho}^{(C,D)}$ do not contain zero coefficients. Appendix O.3, provides further discussion about this check.
6) D randomizes other clients' polynomials. To do so, for every bin, it invokes an instance of VOPR (presented in Fig. 1) with each client C; where D sends $\boldsymbol{\zeta} \cdot \boldsymbol{\omega}^{(D,C)}$ and $\boldsymbol{\gamma}^{(D,C)}$, while client C sends $\boldsymbol{\omega}^{(C,D)} \cdot \boldsymbol{\pi}^{(C)}$ to VOPR. Each client C, for every bin, receives a blinded polynomial of the following form:
$$\boldsymbol{\theta}_1^{(C)} = \boldsymbol{\zeta} \cdot \boldsymbol{\omega}^{(D,C)} \cdot \boldsymbol{\omega}^{(C,D)} \cdot \boldsymbol{\pi}^{(C)} + \boldsymbol{\gamma}^{(D,C)}$$
from VOPR. If any party aborts, all parties receive their deposit.

7) Each client C randomizes D's polynomial. To do that, each client C, for each bin, invokes an instance of VOPR with D, where each client C sends $\boldsymbol{\rho}^{(C,D)}$, while D sends $\boldsymbol{\zeta} \cdot \boldsymbol{\rho}^{(D,C)} \cdot \boldsymbol{\pi}^{(D)}$ and $\boldsymbol{\delta}^{(D,C)}$ to VOPR. Every client C, for each bin, receives a blinded polynomial of the following form: $\boldsymbol{\theta}_2^{(C)} = \boldsymbol{\zeta} \cdot \boldsymbol{\rho}^{(D,C)} \cdot \boldsymbol{\rho}^{(C,D)} \cdot \boldsymbol{\pi}^{(D)} + \boldsymbol{\delta}^{(D,C)}$ from VOPR. If a party aborts, all parties get their deposit back.
8) Each client C, for every bin, blinds the sum of polynomials $\boldsymbol{\theta}_1^{(C)}$ and $\boldsymbol{\theta}_2^{(C)}$ using the blinding factors: $z_{i,c}$, generated in step 3. Specifically, it computes the following blinded polynomial (for every bin): $\boldsymbol{\nu}^{(C)} = \boldsymbol{\theta}_1^{(C)} + \boldsymbol{\theta}_2^{(C)} + \boldsymbol{\tau}^{(C)}$, where $\boldsymbol{\tau}^{(C)} = \sum_{i=0}^{3d+2} z_{i,c} \cdot x^i$. Next, it sends all $\boldsymbol{\nu}^{(C)}$ to $\mathcal{SC}_{\text{JUS}}$. If any party aborts, the deposit will be refunded to all parties.
9) D ensures all clients sent their inputs to $\mathcal{SC}_{\text{JUS}}$. If the check fails, it halts and the deposit is refunded to all parties. Otherwise, it allocates a pseudorandom polynomial $\boldsymbol{\gamma}'$ of degree $3d$, to each bin. To do so, it uses mk to derive a key for each bin: $k_{indx} = \text{PRF}(mk, indx)$ and uses k_{indx} to generate $3d+1$ pseudorandom coefficients $g_{j,indx} = \text{PRF}(k_{indx}, j)$ where $0 \leq j \leq 3d$. Also, for each bin, it allocates a random polynomial $\boldsymbol{\omega}'^{(D)}$ of degree d.
10) D, for every bin, computes a switching polynomial of the form:
$$\boldsymbol{\nu}^{(D)} = \boldsymbol{\zeta} \cdot \boldsymbol{\omega}'^{(D)} \cdot \boldsymbol{\pi}^{(D)} - \sum_{C=A_1}^{A_m} (\boldsymbol{\gamma}^{(D,C)} + \boldsymbol{\delta}^{(D,C)}) + \boldsymbol{\zeta} \cdot \boldsymbol{\gamma}'$$
It sends to $\mathcal{SC}_{\text{JUS}}$ polynomials $\boldsymbol{\nu}^{(D)}$ and $\boldsymbol{\zeta}$, for each bin.
11) $\mathcal{SC}_{\text{JUS}}$ takes the following steps:
   a) for every bin, sums all related polynomials provided by all clients in $\bar{P}$:
$$\boldsymbol{\phi} = \boldsymbol{\nu}^{(D)} + \sum_{C=A_1}^{A_m} \boldsymbol{\nu}^{(C)}$$
$$= \boldsymbol{\zeta} \cdot \Big(\boldsymbol{\omega}'^{(D)} \cdot \boldsymbol{\pi}^{(D)} + \sum_{C=A_1}^{A_m} (\boldsymbol{\omega}^{(D,C)} \cdot \boldsymbol{\omega}^{(C,D)} \cdot \cdot \boldsymbol{\pi}^{(C)}) +$$
$$\boldsymbol{\pi}^{(D)} \cdot \sum_{C=A_1}^{A_m} (\boldsymbol{\rho}^{(D,C)} \cdot \boldsymbol{\rho}^{(C,D)}) + \boldsymbol{\gamma}' \Big)$$
   b) checks whether, for every bin, $\boldsymbol{\zeta}$ divides $\boldsymbol{\phi}$. If the check passes, it sets $Flag = True$. Otherwise, it sets $Flag = False$.
12) If $Flag = True$, then the following steps are taken:
   a) $\mathcal{SC}_{\text{JUS}}$ sends back each party's deposit, i.e., $\ddot{y} + \ddot{ch}$ amount.
   b) each client (given $\boldsymbol{\zeta}$ and mk) finds the elements in the intersection as follows.
      i) derives a bin's pseudorandom polynomial, $\boldsymbol{\gamma}'$, from mk.
      ii) removes the blinding polynomial from each bin's polynomial: $\boldsymbol{\phi}' = \boldsymbol{\phi} - \boldsymbol{\zeta} \cdot \boldsymbol{\gamma}'$.
      iii) evaluates each bin's unblinded polynomial at every element s_i belonging to that bin and considers the element in the intersection if the evaluation is zero: i.e., $\boldsymbol{\phi}'(s_i) = 0$.
13) If $Flag = False$, then the following steps are taken.
   a) Aud asks every client C to send to it the PRF's key (generated in step 3), for every bin. It inserts the keys to $\vec{k}$. It generates an empty list $\bar{L}$. Then, for every bin, Aud takes step 2 of ZSPA-A, i.e., invokes $\texttt{Audit}(m, \vec{k}, q, \boldsymbol{\zeta}, 3d+3, g) \rightarrow (L, \vec{\mu})$. Every time it invokes Audit, it appends the elements of returned L to $\bar{L}$. Aud for each bin sends $\vec{\mu}$ to $\mathcal{SC}_{\text{JUS}}$. It also sends to $\mathcal{SC}_{\text{JUS}}$ the list $\bar{L}$ of all misbehaving clients detected so far. The above check allows Aud to identify clients who misbehaved in ZSPA-A.
   b) to help identify further misbehaving clients, D takes the following steps, for each bin of client C whose ID is not in $\bar{L}$.
      i) computes polynomial $\boldsymbol{\chi}^{(D,C)}$ as follows.
$$\boldsymbol{\chi}^{(D,C)} = \boldsymbol{\zeta} \cdot \boldsymbol{\eta}^{(D,C)} - (\boldsymbol{\gamma}^{(D,C)} + \boldsymbol{\delta}^{(D,C)})$$
      where $\boldsymbol{\eta}^{(D,C)}$ is a fresh random polynomial of degree $3d+1$.
      ii) sends polynomial $\boldsymbol{\chi}^{(D,C)}$ to $\mathcal{SC}_{\text{JUS}}$.
      Note, if $\bar{L}$ contains all clients' IDs, then D does not need to take the above steps 13(b)i and 13(b)ii.
   c) $\mathcal{SC}_{\text{JUS}}$, takes the following steps to check if the client misbehaved, for each bin of client C whose ID is not in $\bar{L}$.
      i) computes polynomial $\boldsymbol{\iota}^{(C)}$ as follows:
      $\boldsymbol{\iota}^{(C)} = \boldsymbol{\chi}^{(D,C)} + \boldsymbol{\nu}^{(C)} + \boldsymbol{\mu}^{(C)} =$
      $= \boldsymbol{\zeta} \cdot (\boldsymbol{\eta}^{(D,C)} + \boldsymbol{\omega}^{(D,C)} \cdot \boldsymbol{\omega}^{(C,D)} \cdot \boldsymbol{\pi}^{(C)} + \boldsymbol{\rho}^{(D,C)} \cdot \boldsymbol{\rho}^{(C,D)} \cdot \boldsymbol{\pi}^{(D)} + \boldsymbol{\xi}^{(C)})$
      where $\boldsymbol{\mu}^{(C)} \in \vec{\mu}$ was sent to $\mathcal{SC}_{\text{JUS}}$ by Aud in step 13a.
      ii) checks if $\boldsymbol{\zeta}$ divides $\boldsymbol{\iota}^{(C)}$. If the check fails, it appends the client's ID to a list L'.
      If $\bar{L}$ contains all IDs, $\mathcal{SC}_{\text{JUS}}$ does not take the above two steps.
   d) $\mathcal{SC}_{\text{JUS}}$ refunds the honest parties' deposit. Also, it retrieves the total amount of $\ddot{ch}$ from the deposit of dishonest clients (i.e., those clients whose IDs are in $\bar{L}$ or L') and sends it to Aud. It also splits the remaining deposit of the misbehaving parties among the honest ones. Thus, each honest client receives $\ddot{y} + \ddot{ch} + \frac{m' \cdot (\ddot{y} + \ddot{ch}) - \ddot{ch}}{m - m'}$ amount in total, where m' is the total number of misbehaving parties.

We refer readers to Appendix O for a more in-depth discussion of $Justitia$ which encompasses (i) the inadequacy of strawman approaches such as using a server-aided PSI or charging the buyer a flat fee, (ii) instructions on how to eliminate the check in step 5, (iii) an outline of the primary challenges we addressed during $Justitia$ design, and (iv) choice of concrete (security) parameters. Appendix P presents a formal security theorem of JUS and its proof.

6. Definition of Multi-party PSI with Fair Compensation and Reward

Next, we enhance $\mathcal{PSI}^{\mathcal{FC}}$ to "multi-party PSI with Fair Compensation and Reward" ($\mathcal{PSI}^{\mathcal{FCR}}$), which not only provides the functionalities of $\mathcal{PSI}^{\mathcal{FC}}$, but also lets honest clients contributing their set receive a reward from the buyer who initiates the PSI computation.

In $\mathcal{PSI}^{\mathcal{FCR}}$, there are (1) a set of clients $\{A_1, ..., A_m\}$ within which a subset can potentially be active adversaries and engage in collusion, (2) a non-colluding dealer client, D, who may be semi-honest, and (3) an auditor Aud, who may also be semi-honest. Every client has an input set.

Moreover, there are two "extractor" clients: A_1 and A_2, where $(A_1, A_2) \in \{A_1, ..., A_m\}$. These extractor clients voluntarily undertake the task of extracting the (encoded) elements of the intersection and transmitting them to a public bulletin board, represented by a smart contract. In exchange for this service, they will receive payment. Alternatively, they could be chosen randomly. We assume that these two extractors act rationally when they intend to perform the paid task of extracting the intersection and reporting it to the smart contract. This rational behavior allows them to maximize their profit.[3] For simplicity, we designate client A_m as the buyer, implying that this party initiates the PSI computation and holds an interest in the outcome.

The formal definition of $\mathcal{PSI}^{\mathcal{FCR}}$ is based on the definition of $\mathcal{PSI}^{\mathcal{FC}}$. Nevertheless, in $\mathcal{PSI}^{\mathcal{FCR}}$, we ensure that honest non-buyer clients receive a *reward* for their participation in the protocol and for disclosing a portion of their inputs derived from the outcome. Appendix R elaborates on this approach and presents the formal definition of $\mathcal{PSI}^{\mathcal{FCR}}$.

7. Concrete Construction of $\mathcal{PSI}^{\mathcal{FCR}}$

7.1. Description of Anesidora (ANE)

To construct ANE, we mainly use JUS, deterministic encryption, "double-layered" commitments, the hash-based padding technique, and the counter-collusion smart contracts. Briefly, ANE works as follows. All clients execute step 1 of JUS to reach an agreement on a set of parameters and JUS's smart contract. They also deploy another smart contract, denoted as $\mathcal{SC}_{\text{ANE}}$, and agree on a secret key, mk'. Next, the buyer deposits a certain amount into $\mathcal{SC}_{\text{ANE}}$. This deposit will be distributed among honest clients as a reward.

The extractors and D deploy one of the counter-collusion smart contracts, namely $\mathcal{SC}_{\text{PC}}$. These three parties contribute a specific sum to $\mathcal{SC}_{\text{PC}}$. Each honest extractor will receive a portion of D's deposit for carrying out its task honestly, while each dishonest extractor will incur a penalty, causing a deduction from their deposit. Each client encrypts its set elements (under mk' using deterministic encryption) and represents the encrypted elements as a polynomial.

Following this, the extractors commit the encryption of their set elements and publicly disclose the outcome. Subsequently, all clients (including D) proceed with the remaining steps outlined in JUS using their input polynomials. This process results in the creation of a blinded polynomial (for each bin), the correctness of which is verified by $\mathcal{SC}_{\text{JUS}}$. There will be two cases:

1) If $\mathcal{SC}_{\text{JUS}}$ confirms the correctness of the result, all parties receive the deposits they initially placed in $\mathcal{SC}_{\text{JUS}}$. In this case, each extractor finds the set elements in the intersection. Additionally, each extractor provides proof to $\mathcal{SC}_{\text{ANE}}$ that the encryptions of the elements in the intersection match the commitments they previously published. If $\mathcal{SC}_{\text{ANE}}$ accepts both extractors' proofs, it compensates each client with a reward. This reward is funded from the buyer's deposit. The extractors receive refunds of their deposits and are also remunerated for their honest execution of the task. But, should $\mathcal{SC}_{\text{ANE}}$ decline to accept one of the extractors' proofs (or if one extractor betrays the other), $\mathcal{SC}_{\text{ANE}}$ then activates the auditor, specified in the counter-collusion contracts, to identify the misbehaving extractor. Subsequently, $\mathcal{SC}_{\text{ANE}}$ pays a reward to each honest client, with the reward sourced from the misbehaving extractor's deposit. Also, $\mathcal{SC}_{\text{ANE}}$ provides a refund of the buyer's deposit.
2) If $\mathcal{SC}_{\text{JUS}}$ does not approve the result's correctness and Aud identified misbehaving clients, honest clients will receive (a) their deposit back from JUS's contract, and (b) compensation and reward, funded by the misbehaving clients. Also, the buyer and extractors will have their deposits refunded by $\mathcal{SC}_{\text{ANE}}$.

Figure 8 in Appendix S depicts the parties' interaction.

7.2. Detailed Description of ANE

Next, we will provide a more detailed description of the protocol (Table 2 on page 15 summarizes the main notations used).

1) All clients in $CL = \{A_1, ..., A_m, D\}$ collectively initiate step 1 of JUS (in Section 5.1) to deploy JUS's contract $\mathcal{SC}_{\text{JUS}}$ and agree on a master key, mk.
2) All clients in CL deploy a new smart contract, $\mathcal{SC}_{\text{ANE}}$. The address of $\mathcal{SC}_{\text{ANE}}$ is given to all clients.
3) Buyer A_m, before time t_1, deposits $S_{min} \cdot \ddot{v}$ amount to $\mathcal{SC}_{\text{ANE}}$.
4) All clients after time $t_2 > t_1$ ensure that the buyer has deposited $S_{min} \cdot \ddot{v}$ amount on $\mathcal{SC}_{\text{ANE}}$. Otherwise, they halt.
5) D signs $\mathcal{SC}_{\text{PC}}$ with the extractors. $\mathcal{SC}_{\text{ANE}}$ transfers $S_{min} \cdot \ddot{r}$ amount (from the buyer's deposit) to $\mathcal{SC}_{\text{PC}}$ for each extractor. This is the maximum amount to be paid to an honest extractor for honestly declaring the elements of the intersection. Each extractor deposits $\ddot{d}' = \ddot{d} + S_{min} \cdot \ddot{f}$ amount in $\mathcal{SC}_{\text{PC}}$ at time t_3. At time t_4 all clients ensure that the extractors deposited a sufficient amount of coins; otherwise, they withdraw their deposit and halt.
6) D encrypts mk under the public key of the auditor (in $\mathcal{SC}_{\text{PC}}$); let ct_{mk} be the resulting ciphertext. It also generates a commitment of mk as follows: $z' = \texttt{PRF}(mk, 0)$, $com_{mk} = \texttt{Com}(mk, z')$. It stores ct_{mk} and com_{mk} in $\mathcal{SC}_{\text{ANE}}$.
7) All clients in CL engage in CT to agree on another key, mk'.

3. Thus, similar to any A_i in $\mathcal{PSI}^{\mathcal{FC}}$, these extractors might be corrupted by an active adversary and act maliciously during the PSI computation.

8) Each client in CL encrypts the elements of its set S : $\{s_1, ..., s_c\}$ as: $\forall i, 1 \leq i \leq c : e_i = \texttt{PRP}(mk', s_i)$. Then, it encodes its encrypted set element as $\bar{e}_i = e_i || \texttt{H}(e_i)$. Next, it constructs a hash table HT and inserts the encoded elements into the table. $\forall i : \texttt{H}(\bar{e}_i) = j$, then $\bar{e}_i \rightarrow \texttt{HT}_j$. It pads every bin with random dummy elements to d elements (if needed). Then, for every bin, it builds a polynomial whose roots are the bin's content: $\boldsymbol{\pi}^{(I)} = \prod_{i=1}^{d}(x - e'_i)$, where e'_i is either $\bar{e}_i$, or a dummy value.
9) Every extractor in $\{A_1, A_2\}$:
   a) for each j-th bin, commits to the bin's elements: $com_{i,j} = \texttt{Com}(e'_i, q_i)$, where q_i is a fresh randomness used for the commitment and e'_i is either $\bar{e}_i$, or a dummy value of the bin.
   b) constructs a Merkle tree on top of all committed values:
$$\texttt{MT.genTree}(com_{1,1}, ..., com_{d,h}) \rightarrow g$$
   c) stores the Merkle tree's root g in $\mathcal{SC}_{\texttt{ANE}}$.
10) All clients in CL run steps 3–11 of JUS, where each client now deposits (in the $\mathcal{SC}_{\texttt{JUS}}$) $\ddot{y}'$ amount where $\ddot{y}' > S_{min} \cdot \ddot{v} + \ddot{ch}$. Recall, at the end of step 11 of JUS for each j-th bin (i) a random polynomial $\boldsymbol{\zeta}$ is registered in $\mathcal{SC}_{\texttt{JUS}}$, (ii) a polynomial $\boldsymbol{\phi}$ (blinded by a random polynomial $\boldsymbol{\gamma}'$) is extracted by $\mathcal{SC}_{\texttt{JUS}}$, and (iii) $\mathcal{SC}_{\texttt{JUS}}$ checks this polynomial's correctness. If the latter check:
   - passes (i.e., $Flag = True$), then all parties run step 12 of JUS (with a minor difference, see Appendix V.5). In this case, each party receives $\ddot{y}'$ amount it deposited in $\mathcal{SC}_{\texttt{JUS}}$. They proceed to step 11 below.
   - fails (i.e., $Flag = False$), all parties run step 13 of JUS. Aud is paid $\ddot{ch}$ amount, and each honest party receives back its deposit, i.e., $\ddot{y}'$ amount. Also, from the misbehaving parties' deposit $\frac{m' \cdot \ddot{y}' - \ddot{ch}}{m - m'}$ amount is sent to each honest client, to reward and compensate the client $S_{min} \cdot \ddot{l}$ and $\frac{m' \cdot \ddot{y}' - \ddot{ch}}{m - m'} - S_{min} \cdot \ddot{l}$ amounts respectively, where m' is the total number of misbehaving parties. Also, $\mathcal{SC}_{\texttt{ANE}}$ returns to (a) the buyer its deposit (i.e., $S_{min} \cdot \ddot{v}$ amount initially paid to $\mathcal{SC}_{\texttt{ANE}}$), and (b) each extractor its deposit, i.e., $\ddot{d}'$ amount initially paid to $\mathcal{SC}_{\texttt{PC}}$. Then, the protocol halts.
11) Every extractor client:
   a) finds the intersection's elements as follows. It encodes each of its set elements to obtain $\bar{e}_i$, as explained in step 8. It determines to which bin the encrypted value belongs, i.e., $j = \texttt{H}(\bar{e}_i)$. It evaluates the resultant polynomial (for that bin) at the encrypted element and considers the element in the intersection if the evaluation is zero: $\boldsymbol{\phi}(\bar{e}_i) - \boldsymbol{\zeta}(\bar{e}_i) \cdot \boldsymbol{\gamma}'(\bar{e}_i) = 0$. If the extractor is a traitor, by this step it should have signed the traitor's contract $\mathcal{SC}_{\texttt{TC}}$ with D and provided related inputs to $\mathcal{SC}_{\texttt{TC}}$.
   b) proves that every element in the intersection is among the elements it has committed to. Specifically, for each element in the intersection, e.g., $\bar{e}_i$, it sends to $\mathcal{SC}_{\texttt{ANE}}$:
      - commitment $com_{i,j}$ (generated in step 9a, for $\bar{e}_i$) and its opening: $\hat{x}' := (\bar{e}_i, q_i)$.
      - proof h_i asserting $com_{i,j}$ is a leaf of Merkle tree with root g.
   c) sends the opening of com_{mk}, i.e., $\hat{x} := (mk, z')$, to $\mathcal{SC}_{\texttt{ANE}}$.
12) Contract $\mathcal{SC}_{\texttt{ANE}}$:
   a) verifies the opening of the commitment for mk, i.e., $\texttt{Ver}(com_{mk}, \hat{x}) = 1$. If the verification passes, it generates the index of the bin to which $\bar{e}_i$ belongs, i.e., $j = \texttt{H}(\bar{e}_i)$. It uses mk to derive the pseudorandom polynomial $\boldsymbol{\gamma}'$ for j-th bin.
   b) checks whether (i) the opening of commitment is valid, (ii) the Merkle tree proof is valid, and (iii) the encrypted element is the resulting polynomial's root. Specifically, it ensures that the following relation holds:
$$\big(\texttt{Ver}(com_{i,j}, \hat{x}') = 1\big) \wedge \big(\texttt{MT.verify}(h_i, g) = 1\big) \wedge$$
$$\big(\boldsymbol{\phi}(\bar{e}_i) - \boldsymbol{\zeta}(\bar{e}_i) \cdot \boldsymbol{\gamma}'(\bar{e}_i) = 0\big)$$
13) The parties are paid as follows.
   - if all proofs of both extractors are valid, both extractors provided identical elements of the intersections (for each bin), and there is no traitor, then $\mathcal{SC}_{\texttt{ANE}}$:
      a) takes $|S_\cap| \cdot m \cdot \ddot{l}$ amount from the buyer's deposit (in $\mathcal{SC}_{\texttt{ANE}}$) and distributes it among all clients, except the buyer.
      b) calls $\mathcal{SC}_{\texttt{PC}}$ which returns the extractors' deposit (i.e., $\ddot{d}'$ amount each) and pays each extractor $|S_\cap| \cdot \ddot{r}$ amount, for doing their job correctly.
      c) checks if $|S_\cap| < S_{min}$. If the check passes, then it returns $(S_{min} - |S_\cap|) \cdot \ddot{v}$ amount to the buyer.
   - if both extractors failed to deliver any result, then $\mathcal{SC}_{\texttt{ANE}}$:
      a) refunds the buyer, by sending $S_{min} \cdot \ddot{v}$ amount (deposited in $\mathcal{SC}_{\texttt{ANE}}$) back to the buyer.
      b) retrieves each extractor's deposit (i.e., $\ddot{d}$ amount) from the $\mathcal{SC}_{\texttt{PC}}$ and distributes it among the rest of the clients (except the buyer and extractors).
   - Otherwise (e.g., if some proofs are invalid, if an extractor's result is inconsistent with the other extractor's result, or there is a traitor), $\mathcal{SC}_{\texttt{ANE}}$ invokes (steps 8.c and 9 of) $\mathcal{SC}_{\texttt{PC}}$ and its auditor to identify the misbehaving extractor, with the help of ct_{mk} after decrypting it. $\mathcal{SC}_{\texttt{ANE}}$ asks $\mathcal{SC}_{\texttt{PC}}$ to pay the auditor the total amount of $\ddot{ch}$ taken from the deposit of the extractor(s) who provided incorrect result to $\mathcal{SC}_{\texttt{ANE}}$. Moreover,
      a) if <u>both extractors cheated</u>:
         i) if there <u>is no traitor</u>, then $\mathcal{SC}_{\texttt{ANE}}$ refunds the buyer, by sending $S_{min} \cdot \ddot{v}$ amount (deposited in $\mathcal{SC}_{\texttt{ANE}}$) back to the buyer. It also distributes $2 \cdot \ddot{d}' - \ddot{ch}$ amount (taken from the extractors' deposit in $\mathcal{SC}_{\texttt{PC}}$) among the rest of the clients (except the buyer and extractors).
         ii) if there <u>is a traitor</u>, then:

A) if the traitor delivered a correct result in $\mathcal{SC}_{\text{TC}}$, $\mathcal{SC}_{\text{ANE}}$ retrieves $\ddot{d}' - \ddot{d}$ amount from the other dishonest extractor's deposit (in $\mathcal{SC}_{\text{PC}}$) and distributes it among the rest of the clients (except the buyer and dishonest extractor). It asks $\mathcal{SC}_{\text{PC}}$ to send $|S_{\cap}| \cdot \ddot{r} + \ddot{d}' + \ddot{d} - c\ddot{h}$ amount to the traitor (via $\mathcal{SC}_{\text{TC}}$). $\mathcal{SC}_{\text{TC}}$ refunds the traitor's deposit, i.e., $c\ddot{h}$ amount. It refunds the buyer, by sending $S_{min} \cdot \ddot{v} - |S_{\cap}| \cdot \ddot{r}$ amount (deposited in $\mathcal{SC}_{\text{ANE}}$) to it.

B) if the traitor delivered an incorrect result in $\mathcal{SC}_{\text{TC}}$, $\mathcal{SC}_{\text{ANE}}$ pays the buyer and rest of the clients in the same way it does in step 13(a)i. $\mathcal{SC}_{\text{TC}}$ refunds the traitor, $c\ddot{h}$ amount.

b) if one of the extractors cheated:

i) if there is no traitor, $\mathcal{SC}_{\text{ANE}}$ calls $\mathcal{SC}_{\text{PC}}$ that (a) returns the honest extractor's deposit ($\ddot{d}'$ amount), (b) pays this extractor $|S_{\cap}| \cdot \ddot{r}$ amount, for doing its job honestly, and (c) pays this extractor $\ddot{d} - c\ddot{h}$ amount taken from the dishonest extractor's deposit. $\mathcal{SC}_{\text{ANE}}$ pays the buyer and other clients in the same way it does in step 13(a)iiA.

ii) if there is a traitor

A) if the traitor delivered a correct result in $\mathcal{SC}_{\text{TC}}$ (but cheated in $\mathcal{SC}_{\text{ANE}}$), $\mathcal{SC}_{\text{ANE}}$ calls $\mathcal{SC}_{\text{PC}}$ that (a) returns the other honest extractor's deposit ($\ddot{d}'$ amount), (b) pays the honest extractor $|S_{\cap}| \cdot \ddot{r}$ amount taken from the buyer's deposit, for doing its job honestly, (c) pays the honest extractor $\ddot{d} - c\ddot{h}$ amount taken from the traitor's deposit, (d) pays to the traitor $|S_{\cap}| \cdot \ddot{r}$ amount taken from the buyer's deposit (via the $\mathcal{SC}_{\text{TC}}$), and (e) refunds the traitor $\ddot{d}' - \ddot{d}$ amount taken from its own deposit. $\mathcal{SC}_{\text{TC}}$ refunds the traitor's deposit (i.e., $c\ddot{h}$ amount). $\mathcal{SC}_{\text{ANE}}$ takes $|S_{\cap}| \cdot m \cdot \ddot{l}$ amount from the buyer's deposit (in $\mathcal{SC}_{\text{ANE}}$) and distributes it among all clients, except the buyer. If $|S_{\cap}| < S_{min}$, $\mathcal{SC}_{\text{ANE}}$ returns $(S_{min} - |S_{\cap}|) \cdot \ddot{v}$ amount (deposited in $\mathcal{SC}_{\text{ANE}}$) to the buyer.

B) if the traitor delivered an incorrect result in $\mathcal{SC}_{\text{TC}}$ (and it cheated in $\mathcal{SC}_{\text{ANE}}$), then $\mathcal{SC}_{\text{ANE}}$ pays the honest extractor in the same way it does in step 13(b)iiA. $\mathcal{SC}_{\text{TC}}$ refunds the traitor's deposit, i.e., $c\ddot{h}$ amount. Also, $\mathcal{SC}_{\text{ANE}}$ pays the buyer and the rest of the clients in the same way it does in step 13(a)iiA.

In Appendix T, we elaborate on the primary challenges we had to confront during the protocol design, which encompassed rewarding clients proportionately to the intersection size and managing collusions among extractors. Also, we refer readers to Appendix V for a more discussion of Anesidora, where we explain why a naive approach is inadequate, and provide insights into the use of hash-based padding, encryption, and double-layered commitment.

We prove the security of ANE in Appendix U.

8. Related Work

Since their introduction in [18], various PSIs have been designed, categorized into *traditional* and *delegated* types. In traditional PSIs, data owners compute the result interactively using their local data. In this research domain, Raghuraman and Rindal [27] proposed two two-party PSIs, one secure against semi-honest/passive and the other against malicious/active adversaries. To date, these protocols are the fastest two-party PSIs. They rely on Oblivious Key-Value Stores (OKVS) data structure and Vector Oblivious Linear Evaluation (VOLE). These PSIs' computation cost is $O(c)$, where c is a set's cardinality. They also impose $O(c \log c^2 + \kappa)$ and $O(c \cdot \kappa)$ communication costs in the semi-honest and malicious models respectively, where κ is a security parameter. Also, researchers designed PSIs that enable multiple (i.e., more than two) parties to compute the intersection. The multi-party PSIs in [28], [29] are secure against passive adversaries while those in [30], [24], [31], [29], [32] were designed to remain secure against active ones. To date, the protocols in [29] and [32] are the most efficient multi-party PSIs designed to be secure against passive and active adversaries respectively. They maintain security even if the majority of parties are corrupt. The computation and communication complexities of the PSI in [29] are $O(c \cdot m^2 + c \cdot m)$ and $O(c \cdot m^2)$. The PSI in [32] has a parameter t that determines how many parties can collude with each other and must be set before the protocol's execution, where $t \in [2, m)$. Its computation and communication complexities are $O(c \cdot \kappa(m + t^2 - t(m+1)))$ and $O(c \cdot m \cdot \kappa)$ respectively.

Researchers proposed a "fair" two-party PSI in [33], ensuring that both parties receive the result or neither does if a malicious party aborts during the protocol's execution. It uses asymmetric-key primitives. The protocol's computation and communication complexities are $O(c^2)$ and $O(c)$ respectively. Since then, various fair two-party PSIs have been proposed, e.g., in [34], [35], [36]. Currently, the fair PSI presented in [36] exhibits superior complexity compared to previous fair PSIs. It uses ElGamal encryption and zero-knowledge proofs. Its overhead is $O(c)$. But, its overall cost is high, due to its reliance on asymmetric key primitives. To date, there exists no fair multi-party PSI. Justitia stands as the inaugural fair multi-party PSI.

Delegated PSIs leverage cloud computing for computation and/or storage. These protocols can be further categorized into those that facilitate *one-off* and *repeated* delegation of PSI computation. The most efficient one that supports one-off delegation is [37], designed for the two-party setting, with an overhead of $O(c)$.

The protocol in [21] is the first PSI that efficiently supports repeated delegation in the semi-honest model. It uses the polynomial representation of sets and hash tables. Its communication and computation complexities are $O(h \cdot d^2)$ and $O(h \cdot d)$ respectively, where h is the total number of bins in the hash table, d is a bin's capacity (often $d = 100$), and $h \cdot d$ is linear with c. Recently, a multi-party PSI that supports repeated delegation and efficient *updates* has been

proposed in [38]. It is in the semi-honest model. It imposes $O(h{\cdot}d^2{\cdot}m)$ and $O(h{\cdot}d{\cdot}m)$ computation and communication costs respectively, during the PSI computation.

9. Evaluation

In this section, we analyze the overhead associated with ANE and compare its costs and features with those of the two fastest and multiple-party PSIs in [38], [29], [32], [27]), as well as with the fair PSIs in [36], [33]. Table 1 summarizes the result of the comparison.

9.1. Computation Cost

9.1.1. Client's and Dealer's Costs. In step 1, the cost of each client (including D) is $O(m)$ and mainly involves an invocation of CT. In steps 2–5, their cost is negligible. In step 7, the clients' cost is $O(m)$, as they need to invoke an instance of CT. In step 8, each client invokes PRP and H linear with its set's size. It builds h polynomials, where each polynomial's construction involves d multiplications and additions. So, this step's complexity is $O(h \cdot d)$. As shown in [38], $O(h \cdot d) = O(|S|)$ and $d = 100$ for all set sizes, i.e., $h \cdot d \approx 4 \cdot c$, where c is the maximum set size.

In step 10, each client $A_1, ..., A_m$ (excluding D): (i) calls an instance of ZSPA-A that involves $O(h \cdot m)$ invocations of CT, $3h \cdot m(d+1)$ invocations of PRF, $3h \cdot m(d+1)$ addition, and $O(h \cdot m \cdot d)$ calls to H (in step 3 of subroutine JUS), (ii) invokes $2h$ instances of VOPR, where each VOPR invocation involves $2d(1+d)$ invocations of OLE^+, multiplications, and additions (in steps 6 and 7 of JUS), and (iii) performs $h(3d+2)$ addition (in step 8 of JUS).

If $Flag = True$, each client invokes $h(3d+1)$ instances of PRF, performs $h(3d+1)$ additions, and performs polynomial evaluations linear with $|S|$, where each evaluation involves $O(d)$ additions and multiplications. Step 6 involves only D whose cost in this step is constant, as it involves invoking public key encryption, PRF, and commitment once. Also, D: (a) invokes $2h \cdot m$ instances of VOPR (in steps 6 and 7 of JUS), (b) invokes $h(3d+1)$ instances of PRF (in step 9 of JUS), and (c) does $h(d^2+1)$ multiplications and $3h \cdot m \cdot d$ additions (in step 10 of JUS). If $Flag = False$, then D performs $O(h \cdot m \cdot d)$ multiplications and additions (in step 13 of JUS).

9.1.2. Auditor's Cost. When $Flag = False$, Aud invokes $3h \cdot m(d+1)$ instances of PRF, and $O(h \cdot m \cdot d)$ instances of H (in step 13 of JUS).

9.1.3. Extractor's Cost. In step 9, each extractor invokes the commitment scheme linearly with the number of its set cardinality $|S|$ and constructs a Merkle tree on top of the commitments. In step 11, each extractor invokes H linear with its set cardinality $|S|$. It also performs polynomial evaluations linear with $|S|$.

9.1.4. Smart Contracts' Cost. In step 10, the subroutine smart contract $\mathcal{SC}_{\text{JUS}}$ performs $h \cdot m(3d+1)$ additions and h polynomial divisions, where each division includes dividing a polynomial of degree $3d+1$ by a polynomial of degree 1 (in step 11 of JUS). In step 12a, $\mathcal{SC}_{\text{ANE}}$ invokes the commitment's verification algorithm Ver only once, calls at most $|S_\cap|$ instances of H, and invokes $|S_\cap|(3d+1)$ instances of PRF. In step 12b, $\mathcal{SC}_{\text{ANE}}$ invokes at most $|S_\cap|$ instances of Ver, and calls $O(|S_\cap| \cdot \log_2 |S|)$ instances of H. In the same step, it performs polynomial evaluation linear with $|S_\cap|$. Thus, its overall complexity is $O(|S_\cap|(d + \log_2 |S|))$.

9.2. Communication Cost

In steps 1 and 7, the clients' communication cost is dominated by the cost of CT which is $O(m)$. In steps 2–6, the clients' cost is negligible, as it involves sending a few transactions to the smart contracts. Step 9 involves only extractors whose cost is $O(h)$ as each of them only sends to $\mathcal{SC}_{\text{ANE}}$ a value for each bin. In step 10, the clients' cost is dominated by VOPR's cost; specifically, each pair of client and D invokes $O(d^2)$ instances of VOPR for each bin. Therefore, the cost of each client (excluding D) is $O(h{\cdot}d^2{\cdot}\bar{\xi})$ while the cost of D is $O(h \cdot d^2 \cdot \bar{\xi} \cdot m)$, where $\bar{\xi}$ is OLE's security parameter. Step 11 involves only the extractors, where each extractor's cost is dominated by the size of the Merkle tree's proof sent to $\mathcal{SC}_{\text{ANE}}$, i.e., $O(|S_\cap| \cdot \log_2 |S|)$, where $|S|$ is the extractor's set cardinality. In step 13, Aud sends h polynomials of degree $3d+1$ to $\mathcal{SC}_{\text{JUS}}$. Thus, its complexity is $O(h \cdot d)$. The other steps have negligible communication costs.

9.3. Comparison

We show that ANE provides multiple features, not found in other PSIs while maintaining a comparable overhead to efficient PSIs.

9.3.1. Computation Complexity. ANE's computation complexity is similar to that of PSI in [38], but is better than the multi-party PSI's complexity in [29] as the latter's complexity is quadratic with the number of parties, i.e., $O(|S| \cdot d \cdot m)$ versus $O(|S| \cdot m^2 + |S| \cdot m)$. ANE's complexity is better than the PSI's complexity in [32] that is quadratic with parameter t. Similar to the two-party PSIs in [36], [27], ANE's complexity is linear with $|S|$. The two-party PSI in [33] has a higher computation cost than ANE does, as its complexity is quadratic with sets' cardinality. Hence, the complexity of ANE is: (i) linear with the set size, similar to the above schemes except for the one in [33] and (ii) linear with the total number of parties, similar to the above multi-party schemes, excluding the one in [29].

9.3.2. Communication Complexity. ANE's communication complexity is slightly higher than that of the PSI in [38], by a factor of $d \cdot \bar{\xi}$. However, it is better than the PSI's complexity in [29] as the latter has a complexity quadratic with the number of parties. ANE's complexity is slightly higher than the one in [32], by a factor of d. Similar to the two-party PSIs in [36], [27], [33], ANE's complexity is linear with c. Hence, ANE's communication complexity is linear with the set cardinality and number of parties, similar to the above schemes except for the one in [29].

9.3.3. Features. ANE stands out as the sole scheme offering five vital features: fairness support, participant rewards, reliance on symmetric key primitives, multi-party capability,

TABLE 1: Comparison of the asymptotic complexities and features of state-of-the-art PSIs. In the table, t is a parameter that determines the maximum number of colluding parties and κ is a security parameter.

Schemes	Asymptotic Cost		Features				
	Computation	Communication	Fairness	Rewarding	Sym-key based	Multi-party	Active Adversary
[38]	$O(h \cdot d^2 \cdot m)$	$O(h \cdot d \cdot m)$	×	×	✓	✓	×
[36]	$O(\|S\|)$	$O(\|S\|)$	✓	×	×	×	✓
[33]	$O(\|S\|^2)$	$O(\|S\|)$	✓	×	×	×	✓
[29]	$O(\|S\| \cdot m^2 + \|S\| \cdot m)$	$O(\|S\| \cdot m^2)$	×	×	✓	✓	×
[32]	$O(\|S\| \cdot \kappa(m + t^2 - t(m+1)))$	$O(\|S\| \cdot m \cdot \kappa)$	×	×	✓	✓	✓
[27]	$O(\|S\|)$	$O(\|S\| \cdot \kappa)$	×	×	✓	×	✓
Ours: Anesidora (ANE)	$O(h \cdot d^2 \cdot m)$	$O(h \cdot d^2 \cdot \bar{\xi} \cdot m)$	✓	✓	✓	✓	✓

and security against active adversaries. In contrast, the next most comprehensive scheme, introduced in [32], offers only three of these features. The remaining protocols provide support for just two of the mentioned features. Our ANE and JUS are the only PSIs that use smart contracts (that require additional but standard blockchain-related assumptions). The other protocols do not utilize smart contracts.

9.4. Concrete Runtime Analysis

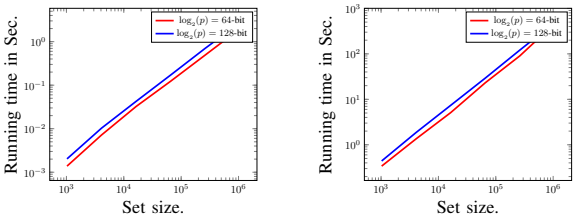
The primary operations involved in ANE are modular addition, multiplication, and invocations of (i) hash function H, (ii) VOPR (which itself involves invocations of OLE$^+$ and basic modular arithmetic), (iii) PRF, and (iv) polynomial divisions. The costs of H invocations and basic modular arithmetic are low. Thus, we will concentrate on the runtime of polynomial divisions, PRF, and OLE$^+$.

9.4.1. Implementation Environment. To evaluate the performance of polynomial division and PRF, we have developed prototype implementations written in C++. They can be found in [39], [40]. We use the NTL library [41] for polynomial divisions, the GMP library [42] for modular multiple precision arithmetic, and the CryptoPP library [43] for implementing PRF based on AES. All experiments were conducted on a MacBook Pro, equipped with a 2-GHz Quad-Core Intel processor and a 16-GB RAM. We run the experiments 100 times to estimate running times.

9.4.2. Choice of Parameters. Since the performance of polynomial division and PRF are influenced by the field size $\log_2(p)$ over which polynomials are defined and the output size (also referred to as $\log_2(p)$) respectively, we use different field sizes: 64 and 128 bits. In ANE, since increasing the total number of bins, h, will increase the number of polynomial divisions and it holds that $h \cdot d \approx 4 \cdot c$, we run the experiment on different set sizes c (and accordingly h), ranging from $2^{10} = 1024$ to $2^{20} = 1048576$. Also, the maximum number of client-side PRF invocations is $3657 \cdot h$ (with d set to 100 and m to 10). To assess the client-side costs related to PRF invocations, we conducted experiments for various values of h when c ranges from 2^{10} to 2^{20}.

9.4.3. Result. Increasing set size from 2^{10} to 2^{20} results in the following changes in the total running time of polynomial divisions: (i) from 0.001 to 1.96 seconds when the field size is 64 bits, and (ii) from 0.002 to 2.73 seconds when the field size is 128 bits. Figure 4a, summarizes the performance of polynomial divisions.

As we increase the number of PRF invocations from 146280 to 153385551, the running time (a) grows from 0.33 to 460 seconds when the output size is 64 bits and (b) increases from 0.44 to 486 seconds when the output size is 128 bits. Figure 4b, outlines the performance of PRF.



(a) Run-time of the division. (b) Run-time of PRF.

Figure 4: Performance of polynomial division and PRF.

The running time of OLE$^+$ is low too. For instance, Boyle *et al.* CCS'18 [44], proposed an efficient generalization of OLE called vector OLE, secure against malicious adversaries. Vector OLE allows the receiver to learn any linear combination of two vectors held by the sender. They estimated the running time of their scheme is about 26.3 milliseconds when the field size is 128 bits and the input vectors size is about 2^{20}. In ANE, a client invokes OLE$^+$ at most $404000 \cdot h$ times. Thus, it will invoke OLE$^+$ from 16160000 to 16944972000 times when c ranges from 2^{10} to 2^{20}, which will take the client from about 0.4 to 42 seconds. Hence, the total combined estimated cost of polynomial divisions, PRF invocations, and OLE$^+$ executions is between 0.84 and 913 seconds when c ranges from 2^{10} to 2^{20}.

Therefore, we estimate the total concrete costs of ANE to range between 1.68 and 1827 seconds. This projection includes an additional cost equivalent to the combined cost of polynomial divisions, PRF invocations, and OLE$^+$ executions, for other operations such as modular arithmetic and hash function invocations.

10. Conclusion

PSI holds significance due to its myriad applications. In this work, we introduced Justitia, the first multi-party fair PSI. Justitia guarantees that the outcome will be achieved collectively by all parties involved. In the event of an unfair protocol abortion, honest parties are assured of receiving financial compensation. Subsequently, we evolved our approach with Anesidora which stands as the first PSI that ensures honest parties contributing their private sets receive rewards proportionate to the number of elements they reveal.

References

[1] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, 2019.

[2] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *CoRR*, 2016.

[3] H. Yu, Z. Liu, Y. Liu, T. Chen, M. Cong, X. Weng, D. Niyato, and Q. Yang, "A sustainable incentive scheme for federated learning," *IEEE Intell. Syst.*, 2020.

[4] L. Lyu, J. Yu, K. Nandakumar, Y. Li, X. Ma, J. Jin, H. Yu, and K. S. Ng, "Towards fair and privacy-preserving federated deep models," *IEEE Trans. Parallel Distributed Syst.*, 2020.

[5] S. Fan, H. Zhang, Y. Zeng, and W. Cai, "Hybrid blockchain-based resource trading system for federated learning in edge computing," *IEEE Internet Things J.*, 2021.

[6] Y. Zhu, Z. Liu, P. Wang, and C. Du, "A dynamic incentive and reputation mechanism for energy-efficient federated learning in 6g," *Digit. Commun. Networks*, 2023.

[7] Y. Cheng, Y. Liu, T. Chen, and Q. Yang, "Federated learning for privacy-preserving AI," *Commun. ACM*, 2020.

[8] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung, "On deploying secure computing: Private intersection-sum-with-cardinality," in *EuroS&P*, 2020.

[9] L. Yizhen, "The practice of federated learning in tencent weishi advertising," 2022, https://cloud.tencent.com/developer/article/1872819.

[10] L. Luhua, "Huawei's exploration and application of federal advertising algorithms," 2022, https://zhuanlan.zhihu.com/p/558684266.

[11] S. Tamrakar, J. Liu, A. Paverd, J. Ekberg, B. Pinkas, and N. Asokan, "The circle game: Scalable private membership test using trusted hardware," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, 2017.

[12] O. Goldreich, *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Tech. Rep., 2019.

[14] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2014.

[15] C. Dong, Y. Wang, A. Aldweesh, P. McCorry, and A. van Moorsel, "Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing," in *CCS*. ACM, 2017.

[16] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. CRC Press, 2007.

[17] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking, "Balanced allocations: the heavily loaded case," in *STOC*, 2000, pp. 745–754.

[18] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *EUROCRYPT*, 2004.

[19] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," in *ACNS*, 2013.

[20] L. Kissner and D. X. Song, "Privacy-preserving set operations," in *CRYPTO*, 2005.

[21] A. Abadi, S. Terzis, R. Metere, and C. Dong, "Efficient delegated private set intersection on outsourced private datasets," *IEEE TDSC*, 2018.

[22] W. S. Dorn, "Generalizations of horner's rule for polynomial evaluation," *IBM Journal of Research and Development*, 1962.

[23] S. Ghosh, J. B. Nielsen, and T. Nilges, "Maliciously secure oblivious linear function evaluation with constant overhead," in *ASIACRYPT*, 2007.

[24] S. Ghosh and T. Nilges, "An algebraic approach to maliciously secure private set intersection," in *EUROCRYPT*, 2019.

[25] A. Abadi, S. J. Murdoch, and T. Zacharias, "Polynomial representation is tricky: Maliciously secure private set intersection revisited," *IACR Cryptol. ePrint Arch.*, 2021. [Online]. Available: https://eprint.iacr.org/2021/1009

[26] B. Schneier, *Applied cryptography - protocols, algorithms, and source code in C, 2nd Edition*. Wiley, 1996.

[27] S. Raghuraman and P. Rindal, "Blazing fast PSI from improved OKVS and subfield VOLE," in *CCS*, 2022.

[28] R. Inbar, E. Omri, and B. Pinkas, "Efficient scalable multiparty private set-intersection via garbled bloom filters," in *SCN*, 2018.

[29] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu, "Practical multi-party private set intersection from symmetric-key techniques," in *CCS*, 2017.

[30] A. Ben-Efraim, O. Nissenbaum, E. Omri, and A. Paskin-Cherniavsky, "Psimple: Practical multiparty maliciously-secure private set intersection," in *ASIA CCS*, 2022.

[31] E. Zhang, F. Liu, Q. Lai, G. Jin, and Y. Li, "Efficient multi-party private set intersection against malicious adversaries," in *CCSW*, 2019.

[32] O. Nevo, N. Trieu, and A. Yanai, "Simple, fast malicious multiparty private set intersection," in *CCS*, 2021.

[33] C. Dong, L. Chen, J. Camenisch, and G. Russello, "Fair private set intersection with a semi-trusted arbiter," in *DBSec*, 2013.

[34] S. K. Debnath and R. Dutta, "A fair and efficient mutual private set intersection protocol from a two-way oblivious pseudorandom function," in *ICISC*, 2014.

[35] ——, "Towards fair mutual private set intersection with linear complexity," *Secur. Commun. Networks*, 2016.

[36] ——, "New realizations of efficient and secure private set intersection protocols preserving fairness," in *ICISC*, 2016.

[37] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian, "Scaling private set intersection to billion-element sets," in *FC*, 2014.

[38] A. Abadi, C. Dong, S. J. Murdoch, and S. Terzis, "Multi-party updatable delegated private set intersection," in *FC*, 2022.

[39] Anonymous, "Source code for polynomial divisions," 2024, https://github.com/anonymous2012000/code-3.

[40] ——, "Source code for pseudorandom invocations," 2024, https://github.com/anonymous2012000/Code-4.

[41] V. Shoup, "NTL: A library for doing number theory," 1996.

[42] GNU Project, "The GNU multiple precision arithmetic library," 1991.

[43] W. Dai, "CryptoPP library multiple precision arithmetic library," 2015.

[44] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, "Compressing vector OLE," in *CCS*, 2018.

[45] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO*, 1991.

[46] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Authenticated hash tables," in *ACM CCS*, 2008, pp. 437–448.

[47] M. Blum, "Coin flipping by telephone - A protocol for solving impossible problems," in *COMPCON'82*. IEEE Computer Society, 1982.

[48] T. Moran, M. Naor, and G. Segev, "An optimally fair coin toss," in *TCC*, 2009.

[49] A. Beimel, E. Omri, and I. Orlov, "Protocols for multiparty coin toss with dishonest majority," in *CRYPTO*, 2010.

[50] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO*, 2017.

[51] A. Kiayias, H. Zhou, and V. Zikas, "Fair and robust multi-party computation using a global transaction ledger," in *EUROCRYPT*, 2016.

[52] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas, "Bitcoin as a transaction ledger: A composable treatment," in *CRYPTO*, 2017.

[53] C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros chronos: Permissionless clock synchronization via proof-of-stake," *IACR Cryptol. ePrint Arch.*, 2019.

[54] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Advances in Cryptology - EUROCRYPT*, 2015.

[55] A. Abadi, C. Dong, S. J. Murdoch, and S. Terzis, "Multi-party updatable delegated private set intersection –Full Version," 2022. [Online]. Available: https://fc22.ifca.ai/preproceedings/68.pdf

[56] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing," in *USENIX Security*, 2015.

[57] S. S. Arora, A. Beams, P. Chatzigiannis, S. Meiser, K. Patel, S. Raghuraman, P. Rindal, H. Shah, Y. Wang, Y. Wu, H. Yang, and M. Zamani, "Privacy-preserving financial anomaly detection via federated learning & multi-party computation," *CoRR*, 2023.

[58] D. C. Nguyen, Q. Pham, P. N. Pathirana, M. Ding, A. Seneviratne, Z. Lin, O. A. Dobre, and W. Hwang, "Federated learning for smart healthcare: A survey," *ACM Comput. Surv.*, 2023.

Appendix A. Notation Table

Table 2 summarizes the main notations used in the paper.

A.1. Security Model

We employ a standard simulation-based secure computation model [12] to define the proposed protocols. As our protocols encompass both static active and passive adversaries, we will restate formal definitions for each type.

A.1.1. Two-party Computation. A two-party protocol, Γ, is characterized by specifying a random process that maps pairs of inputs to pairs of outputs, one for each party. This process is referred to as a functionality $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where $f := (f_1, f_2)$. For every input pair (x, y), the output pair is a random variable $(f_1(x, y), f_2(x, y))$, such that the party with input x obtains $f_1(x, y)$ while the party with input y receives $f_2(x, y)$. If f is asymmetric and only one party (say the first one) receives the result, f is defined as $f := (f_1(x, y), \perp)$.

A.1.2. Security in the Presence of Passive Adversaries. In this setting, a protocol is secure if whatever can be computed by a party in the protocol can be computed using its input and output only.

Definition 1. Let f represent the deterministic functionality defined above. Protocol Γ securely computes f in the presence of a passive adversary if there exist polynomial time algorithms $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that: $\{\mathsf{Sim}_1(x, f_1(x, y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}_1^{\Gamma}(x, y)\}_{x,y}$ and $\{\mathsf{Sim}_2(y, f_2(x, y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}_2^{\Gamma}(x, y)\}_{x,y}$ where party i's view (during the execution of Γ) on input pair (x, y) is denoted by $\mathsf{View}_i^{\Gamma}(x, y)$ and equals $(w, r^i, m_1^i, ..., m_t^i)$, $w \in \{x, y\}$ is the input of i^{th} party, r_i is the outcome of this party's internal random coin tosses, and m_j^i represents the j^{th} message this party receives.

TABLE 2: Notation Table.

Setting	Symbol	Description
Generic	p	Large prime number
	$\mathbb{F}_p$	A finite field of prime order p
	CL	Set of all clients, $\{A_1, ..., A_m, D\}$
	D	Dealer client
	A_m	Buyer client
	m	Total number of clients (excluding D)
	$\mathtt{H}$	Hash function
	$\lvert S_\cap \rvert$	Intersection size
	S_{min}	Smallest set's size
	S_{max}	Largest set's size
	$\lvert$	Divisible
	$\setminus$	Set subtraction
	c	Set's cardinality
	h	Total number of bins in a hash table
	d	A bin's capacity
	λ	Security parameter
	$\mathtt{OLE}$	Oblivious Linear Evaluation
	$\mathtt{OLE}^+$	Advanced $\mathtt{OLE}$
	$\mathtt{Com}$	Commitment algorithm of commitment
	$\mathtt{Ver}$	Verification algorithm of commitment
	$\mathtt{MT.genTree}$	Tree construction algorithm of Merkle tree
	$\mathtt{MT.prove}$	Proof generation algorithm of Merkle tree
	$\mathtt{MT.verify}$	Verification algorithm of Merkle tree
	$\mathtt{CT}$	Coin tossing protocol
	$\mathtt{VOPR}$	Verifiable Oblivious Poly. Randomization
	$\mathtt{ZSPA}$	Zero-sum Pseudorandom Values Agreement
	$\mathtt{ZSPA\text{-}A}$	$\mathtt{ZSPA}$ with an External Auditor
	$\mathcal{PSI}^{\mathcal{FC}}$	Multi-party PSI with Fair Compensation
	$\mathcal{PSI}^{\mathcal{FCR}}$	Multi-party PSI with Fair Compensation and Reward
	$\mathtt{JUS}$	Protocol that realizes $\mathcal{PSI}^{\mathcal{FC}}$
	$\mathtt{ANE}$	Protocol that realizes $\mathcal{PSI}^{\mathcal{FCR}}$
	$\mathtt{PRF}$	Pseudorandom function
	$\mathtt{PRP}$	Pseudorandom permutation
	gcd	Greatest common divisor
	ϵ	Negligible function
Counter Collusion Contracts	$\mathcal{SC}_{\text{PC}}$	Prisoner's Contract
	$\mathcal{SC}_{\text{CC}}$	Colluder's Contract
	$\mathcal{SC}_{\text{TC}}$	Traitor's Contract
	$\ddot{c}$	Server's cost for computing a task
	$\ddot{ch}$	Auditor's cost for resolving disputes
	$\ddot{d}$	Deposit a server pays to get the job
	$\ddot{w}$	Amount a server receives for completing the task
	(pk, sk)	$\mathcal{SC}_{\mathtt{JUS}}$'s auditor's public-private key pair
Justitia (JUS)	$\mathcal{SC}_{\mathtt{JUS}}$	$\mathtt{JUS}$'s smart contract
	$\boldsymbol{\omega}, \boldsymbol{\omega'}, \boldsymbol{\rho}$	Random poly. of degree d
	$\boldsymbol{\gamma}, \boldsymbol{\delta}$	Random poly. of degree $d+1$
	$\boldsymbol{\nu}^{(C)}$	Blinded poly. sent by each C to $\mathcal{SC}_{\mathtt{JUS}}$
	$\boldsymbol{\phi}$	Blinded poly. encoding the intersection
	$\boldsymbol{\chi}$	Poly. sent to $\mathcal{SC}_{\mathtt{JUS}}$ to identify misbehaving parties
	$\bar{L}$	List of identified misbehaving parties
	$\ddot{y}$	A portion of a party's deposit into $\mathcal{SC}_{\mathtt{JUS}}$ transferred to honest clients if it misbehaves
	mk	Master key of $\mathtt{PRF}$
	Q^{Init}	Initiation predicate
	Q^{Del}	Delivery predicate
	$Q^{\text{UF-A}}$	UnFair-Abort predicate
	$Q^{\text{F-A}}$	Fair-Abort predicate
Anesidora (ANE)	$\mathcal{SC}_{\mathtt{ANE}}$	$\mathtt{ANE}$'s smart contract
	$\ddot{d}'$	Extractor's deposit
	$\ddot{y}'$	Each client's deposit into $\mathcal{SC}_{\mathtt{JUS}}$
	$\ddot{l}$	Reward a client earns for an intersection element
	$\ddot{r}$	Extractor's cost for extracting an intersection element
	$\ddot{f}$	Shorthand for $\ddot{l}(m-1)$
	$\ddot{v}$	Price a buyer pays for an intersection element $\ddot{v} = m \cdot \ddot{l} + 2\ddot{r}$
	mk'	Another master key of $\mathtt{PRF}$
	ct_{mk}	Encryption of mk under pk
	$Q_{\text{R}}^{\text{Del}}$	Delivery-with-Reward predicate
	$Q_{\text{R}}^{\text{UF-A}}$	UnFair-Abort-with-Reward predicate

A.1.3. Security in the Presence of Active Adversaries.. In this setting, correctness is required beyond the possibility that a corrupted party may learn more than it should. To capture the threats, a protocol's security is analyzed by comparing what an adversary can do in the real protocol

to what it can do in an ideal scenario. This is formalized by considering an ideal computation involving an incorruptible Trusted Third Party (TTP) to whom the parties send their inputs and receive the output of the ideal functionality.

Definition 2. Let f be the functionality defined above and Γ be a two-party protocol that computes f. Protocol Γ securely computes f in the presence of active adversaries if for every PPT adversary $\mathcal{A}$ for the real model, there exists a non-uniform PPT adversary (or simulator) Sim for the ideal model, such that for every $i \in \{0,1\}$, it holds that: $\{\mathsf{Ideal}^{f}_{\mathsf{Sim}(z),i}(x,y)\}_{x,y,z} \overset{c}{\equiv} \{\mathsf{Real}^{\Gamma}_{\mathcal{A}(z),i}(x,y)\}_{x,y,z}$,, where the ideal execution of f on inputs $(x,\ y)$ and z is denoted by $\mathsf{Ideal}^{f}_{\mathcal{A}(z),i}(x,\ y)$ and is defined as the output pair of the honest party and $\mathcal{A}$ from the ideal execution. The real execution of Γ is denoted by $\mathsf{Real}^{\Gamma}_{\mathcal{A}(z),i}(x,y)$, it is defined as the joint output of the parties engaging in the real execution of Γ, in the presence of $\mathcal{A}$.

Appendix B. Counter Collusion Contracts

In this section, we present descriptions of the Prisoner's Contract ($\mathcal{SC}_{\text{PC}}$), Colluder's Contract ($\mathcal{SC}_{\text{CC}}$), and Traitor's Contract ($\mathcal{SC}_{\text{TC}}$) originally introduced by Dong *et al.* [15]. As mentioned earlier, we have made slight adjustments to these contracts. We will mark these modifications in blue. To ensure a comprehensive understanding, below we reiterate the parameters utilized in these contracts.

- $\ddot{b}$: the bribe paid by the ringleader of the collusion to the other server in the collusion agreement, in $\mathcal{SC}_{\text{CC}}$.
- $\ddot{c}$: a server's cost for computing the task.
- $\ddot{ch}$: the fee paid to invoke an auditor for recomputing a task and resolving disputes.
- $\ddot{d}$: the deposit a server needs to pay to be eligible for getting the job.
- $\ddot{t}$: the deposit the colluding parties need to pay in the collusion agreement, in $\mathcal{SC}_{\text{CC}}$.
- $\ddot{w}$: the amount that a server receives for completing the task.
- $\ddot{w} \geq \ddot{c}$: the server would not accept underpaid jobs.
- $\ddot{ch} > 2\ddot{w}$: if it does not hold, then there would be no need to use the servers and the auditor would do the computation.
- (pk, sk): an asymmetric-key encryption's public-private key pair belonging to the auditor.

The following relations must be satisfied when establishing the contracts to ensure that the desirable equilibria are maintained:

$$(\ddot{d} > \ddot{c} + \ddot{ch}) \wedge (\ddot{b} < \ddot{c}) \wedge (\ddot{t} < \ddot{w} - \ddot{c} + 2\ddot{d} - \ddot{ch} - \ddot{b})$$

B.1. Prisoner's Contract ($\mathcal{SC}_{\text{PC}}$)

$\mathcal{SC}_{\text{PC}}$ has been designed for outsourcing a certain computation. It is signed by a client who delegates the computation and two other parties (or servers) who perform the computation. This contract aims to encourage accurate computation by implementing the following concept. It requires that each server submit a deposit before the delegation of computation. If a server adheres to honest behavior, it retains the ability to withdraw its deposit. If a server misbehaves (and is detected), its deposit is transferred to the client. When one of the servers behaves honestly while the other cheats, the honest server receives a reward. This reward is deducted from the deposit of the cheating server.

Hence, the goal of $\mathcal{SC}_{\text{PC}}$ is to create a Prisoner's dilemma between the two servers in the following sense. Although the servers may collude with each other (to cut costs and provide identical but incorrect computation results) which leads to a higher payoff than both behaving honestly, there is an even higher payoff if one of the servers manages to persuade the other server to collude and provide an incorrect result while itself provides a correct result. In this setting, each server knows that collusion is not stable as its counterpart will always try to deviate from the collusion to increase its payoff. If a server tries to convince its counterpart (without a credible and enforceable promise), then the latter party will consider it as a trap; consequently, collusion will not occur. Below, we restate the contract.

1) The contract is signed by three parties; namely, client D and two other parties E_1 and E_2. A third-party auditor will resolve any dispute between D and the servers. The address of another contract, called $\mathcal{SC}_{\text{ANE}}$, is hardcoded in this contract.
2) The servers agree to run computation f on input x, both of which have been provided by D.
3) The parties agree on three deadlines (T_1, T_2, T_3), where $T_{i+1} > T_i$.
4) D agrees to pay $\ddot{w}$ amount to each server for the correct and on-time computation. Therefore, D deposits $2 \cdot \ddot{w}$ amount in the contract. This deposit is transferred from $\mathcal{SC}_{\text{ANE}}$ to this contract.
5) Each server deposits $\ddot{d}' = \ddot{d} + \ddot{X}$ amount in the contract.
6) The servers must pay the deposit before T_1. If a server fails to meet the deadline, then the contract would refund the parties' deposit (if any) and terminate.
7) The servers must deliver the computation's result before T_2.
8) The following steps are taken when (i) both servers provided the computation's result or (ii) deadline T_2 elapsed.
   a) if both servers failed to deliver the computation's result, then the contract transfers their deposits to $\mathcal{SC}_{\text{ANE}}$.
   b) if both servers delivered the result, and the results are equal, then (after verifying the results) this contract must pay the agreed amount $\ddot{w}$ and refund the deposit of $\ddot{d}'$ amount to each server.
   c) otherwise, D raises a dispute with the auditor.
9) When a dispute is raised, the auditor (which is independent of Aud in JUS) re-generates the computation's result, by using `resComp(.)` described shortly in Appendix B.1.1. Let y_t, y_1, and y_2 be the result computed by the auditor, E_1, and E_2 respectively. The auditor uses the following role to identify the cheating party.

- if E_i failed to deliver the result (i.e., y_i is null), then it has cheated.
- if a result y_i has been delivered before the deadline and $y_i \neq y_t$, then E_i has cheated.

The auditor sends its verdict to $\mathcal{SC}_{\text{PC}}$.

10) Given the auditor's decision, the dispute is settled according to the following rules.
    - if none of the servers cheated, then this contract transfers to each server (i) $\ddot{w}$ amount for performing the computation and (ii) its deposit, i.e., $\ddot{d}'$ amount. The client also pays the auditor $\ddot{ch}$ amount.
    - if both servers cheated, then this contract (i) pays the auditor the total amount of $\ddot{ch}$, taken from the servers' deposit, and (ii) transfers to $\mathcal{SC}_{\text{ANE}}$ the rest of the deposit, i.e., $2 \cdot \ddot{d}' - \ddot{ch}$ amount.
    - if one of the servers cheated, then this contract (i) pays the auditor the total amount of $\ddot{ch}$, taken from the misbehaving server's deposit, (ii) transfers the honest server's deposit (i.e., $\ddot{d}'$ amount) back to this server, (iii) transfers to the honest server $\ddot{w} + \ddot{d} - \ddot{ch}$ amount (which covers its computation cost and the reward), and (iv) transfers to $\mathcal{SC}_{\text{ANE}}$ the rest of the misbehaving server's deposit, i.e., $\ddot{X}$ amount. The cheating server receives nothing.
11) After deadline T_3, if D has neither paid nor raised a dispute, then this contract pays $\ddot{w}$ to each server which delivered a result before deadline T_2 and refunds each server its deposit, i.e., $\ddot{d}'$ amount. Any deposit left after that will be transferred to $\mathcal{SC}_{\text{ANE}}$.

Now, let us elaborate on why we have implemented the aforementioned modifications to the original $\mathcal{SC}_{\text{PC}}$ introduced by Dong *et al.* [15]. In the original $\mathcal{SC}_{\text{PC}}$ (a) the client does not deposit any amount in this contract; instead, it directly sends its coins to a server (and auditor) according to the auditor's decision, (b) the computation correctness is determined only within this contract (with the involvement of the auditor if required), and (c) the auditor simply re-generates the computation's result given the computation's plaintext inputs.

Nevertheless, in ANE, (1) *all clients* need to deposit a certain amount in $\mathcal{SC}_{\text{ANE}}$ and only the contracts must transfer the parties' deposit, (2) $\mathcal{SC}_{\text{ANE}}$ also needs to verify a part of the computation's correctness without the involvement of the auditor and accordingly distribute the parties deposit based on the verification's outcome, and (3) the auditor must be able to re-generate the computation's result without being able to learn the computation's plaintext input, i.e., elements of the set.

Therefore, we have included the address of $\mathcal{SC}_{\text{ANE}}$ in $\mathcal{SC}_{\text{PC}}$ to enable the parties' deposit to be moved between the two contracts (if necessary). Moreover, we have allowed $\mathcal{SC}_{\text{PC}}$ to distribute the parties' deposit; thus, the requirements in points (1) and (2) are met. To satisfy the requirement in point (3) above, we have included a new algorithm, called resComp(.), in $\mathcal{SC}_{\text{PC}}$. Shortly, we will provide more details about this algorithm. Moreover, to make this contract compatible with ANE, we increased the amount of each server's deposit by $\ddot{X}$. Nevertheless, this adjustment does not change the logic behind $\mathcal{SC}_{\text{PC}}$'s design and its analysis.

B.1.1. Auditor's Result-Computation Algorithm. In this work, we use $\mathcal{SC}_{\text{PC}}$ to delegate the computation of intersection cardinality to two extractor clients, a.k.a. servers in the original $\mathcal{SC}_{\text{PC}}$. In this setting, the contract's auditor is invoked when an inconsistency is detected in step 13 of ANE. For the auditor to recompute the intersection cardinality, we have designed resComp(.) algorithm. The auditor uses this algorithm for every bin's index $indx$, where $1 \leq indx \leq h$ and h is the hash table's length. We present this algorithm in Figure 5. The auditor collects the inputs of this algorithm as follows: (a) reads random polynomial $\boldsymbol{\zeta}$, and blinded polynomial $\boldsymbol{\phi}$ from contract $\mathcal{SC}_{\text{JUS}}$, (b) reads the ciphertext of secret key mk from $\mathcal{SC}_{\text{ANE}}$, and (c) retrieves public parameters (des_{H}, h) from the hash table's public description.

Note that in the original $\mathcal{SC}_{\text{PC}}$, the auditor is assumed to be fully trusted. However, in this work, we have relaxed this assumption. We have designed ANE and resComp(.) in such a way that even a semi-honest auditor cannot learn anything about the actual elements of the sets, as they have been encrypted under a key unknown to the auditor.

$\text{resComp}(\boldsymbol{\zeta}, \boldsymbol{\phi}, sk, ct_{mk}, indx, des_{\text{H}}) \rightarrow R$

- *Input*. $\boldsymbol{\zeta}$: a random polynomial of degree 1, $\boldsymbol{\phi}$: a blinded polynomial of the form $\boldsymbol{\zeta} \cdot (\boldsymbol{\epsilon} + \boldsymbol{\gamma}')$ where $\boldsymbol{\epsilon}$ and $\boldsymbol{\gamma}'$ are arbitrary and pseudorandom polynomials respectively, $deg(\boldsymbol{\phi}) - 1 = deg(\boldsymbol{\gamma}')$, sk: the auditor's secret key, ct_{mk}: ciphertext of mk which is a key of PRF, $indx$: an input of PRF, and des_{H}: a description of hash function H.
- *Output*. R: a set of valid roots of unblinded $\boldsymbol{\phi}$.

1) decrypts the ciphertext ct_{mk} under key sk. Let mk be the result.
2) unblinds polynomial $\boldsymbol{\phi}$, as follows:
    a) re-generates pseudorandom polynomial $\boldsymbol{\gamma}'$ using key mk. Specifically, it uses mk to derive a key: $k = \text{PRF}(mk, indx)$. Then, it uses the derived key to generate $3d+1$ pseudorandom coefficients, i.e., $\forall j, 0 \leq j \leq deg(\boldsymbol{\phi}) - 1 : g_j = \text{PRF}(k, j)$. Next, it uses these coefficients to construct polynomial $\boldsymbol{\gamma}'$, i.e., $\boldsymbol{\gamma}' = \sum_{j=0}^{deg(\boldsymbol{\phi})-1} g_j \cdot x^j$.
    b) removes the blinding factor from $\boldsymbol{\phi}$. Specifically, it computes polynomial $\boldsymbol{\phi}'$ of the following form $\boldsymbol{\phi}' = \boldsymbol{\phi} - \boldsymbol{\zeta} \cdot \boldsymbol{\gamma}'$.
3) extracts roots of polynomial $\boldsymbol{\phi}'$.
4) finds valid roots, by (i) parsing each root $\bar{e}$ as (e_1, e_2) with the assistance of des_{H} and (ii) checking if $e_2 = \text{H}(e_1)$. It considers a root valid if this equation holds.
5) returns set R containing all valid roots.

Figure 5: Auditor's result computation, resComp(.), algorithm.

B.2. Colluder's Contract ($\mathcal{SC}_{\text{CC}}$)

Recall that $\mathcal{SC}_{\text{PC}}$ aimed at creating a dilemma between the two servers. However, this dilemma can be addressed if they can make an enforceable promise. This enforceable promise can be another smart contract, called Colluder's Contract ($\mathcal{SC}_{\text{CC}}$). This contract imposes additional rules that ultimately would affect the parties' payoffs and would make collusion the most profitable strategy for the colluding parties. In $\mathcal{SC}_{\text{CC}}$, the party who initiates the collusion would pay its counterpart a certain amount (or bribe) if both follow the collusion and provide an incorrect result of the computation to $\mathcal{SC}_{\text{PC}}$. Note, $\mathcal{SC}_{\text{CC}}$ requires both servers to send a fixed amount of deposit when signing the contract. The party who deviates from collusion will be punished by losing the deposit. Below, we reiterate the description of $\mathcal{SC}_{\text{CC}}$.

1) The contract is signed between the server who initiates the collusion, called ringleader (LDR), and the other server called follower (FLR).
2) The two agree on providing to $\mathcal{SC}_{\text{PC}}$ a different result res' than a correct computation of f on x would yield, i.e., $res' \neq f(x)$. Parameter res' is recorded in this contract.
3) LDR and FLR deposit $\ddot{t}+\ddot{b}$ and $\ddot{t}$ amounts in this contract respectively.
4) The above deposit must be paid before the result delivery deadline in $\mathcal{SC}_{\text{PC}}$, i.e., before deadline T_2. If this condition is not met, the parties' deposit in this contract is refunded and this contract is terminated.
5) When $\mathcal{SC}_{\text{PC}}$ is finalized (i.e., all the results have been provided), the following steps are taken.
   a) both follow the collusion: if both LDR and FLR provided res' to $\mathcal{SC}_{\text{PC}}$, then $\ddot{t}$ and $\ddot{t}+\ddot{b}$ amounts are delivered to LDR and FLR respectively. Therefore, FLR receives its deposit plus the bribe, which is $\ddot{b}$ amount.
   b) only FLR deviates from the collusion: if LDR and FLR provide res' and $res'' \neq res'$ to $\mathcal{SC}_{\text{PC}}$ respectively, then $2 \cdot \ddot{t} + \ddot{b}$ amount is transferred to LDR while nothing is sent to FLR.
   c) only LDR deviates from the collusion: if LDR and FLR provide $res'' \neq res'$ and res' to $\mathcal{SC}_{\text{PC}}$ respectively, then $2 \cdot \ddot{t} + \ddot{b}$ amount is sent to FLR while nothing is transferred to LDR.
   d) both deviate from the collusion: if LDR and FLR deviate and provide any result other than res' to $\mathcal{SC}_{\text{PC}}$, then $2 \cdot \ddot{t} + \ddot{b}$ amount is sent to LDR and $\ddot{t}$ amount is sent to FLR.

The amount of bribe a rational LDR is willing to pay is less than its computation cost (i.e., $\ddot{b} < \ddot{c}$). Otherwise, this collusion would not offer a higher payoff. We refer readers to [15] for further discussion.

B.3. Traitor's Contract ($\mathcal{SC}_{\text{TC}}$)

$\mathcal{SC}_{\text{TC}}$ incentivizes a colluding server (who has signed $\mathcal{SC}_{\text{CC}}$) to betray its counterpart and report the collusion without being penalized by $\mathcal{SC}_{\text{PC}}$. The Traitor's contract promises that the reporting server will not be punished by $\mathcal{SC}_{\text{PC}}$ which makes it safe for the reporting server to follow the collusion strategy (of $\mathcal{SC}_{\text{CC}}$), and get away from the punishment imposed by $\mathcal{SC}_{\text{PC}}$. Below, we restate the description of $\mathcal{SC}_{\text{TC}}$.

1) This contract is signed among D and the traitor server (TRA) who reports the collusion. This contract is signed only if the parties have already signed $\mathcal{SC}_{\text{PC}}$.
2) D signs this contract only with the first server who reports the collusion.
3) The traitor TRA must also provide to this contract the result of the computation, i.e., $f(x)$. The result provided in this contract could be different than the one provided in $\mathcal{SC}_{\text{PC}}$, e.g., when TRA has to follow $\mathcal{SC}_{\text{CC}}$ and provide an incorrect result to $\mathcal{SC}_{\text{PC}}$.
4) D needs to pay $\ddot{w} + \ddot{d'} + \ddot{d} - \ddot{ch}$ amount to this contract. This amount equals the maximum amount TRA could lose in $\mathcal{SC}_{\text{PC}}$ plus the reward. This deposit will be transferred via $\mathcal{SC}_{\text{ANE}}$ to this contract. TRA must deposit in this contract $\ddot{ch}$ amount to cover the cost of resolving a potential dispute.
5) This contract should be signed before the deadline T_2 for the delivery of the computation result in $\mathcal{SC}_{\text{PC}}$. If it is not signed on time, then this contract will be terminated and any deposit paid will be refunded.
6) It is required that the TRA provide the computation result to this contract before the above deadline T_2.
7) If this contract is fully signed, then during the execution of $\mathcal{SC}_{\text{PC}}$, D always raises a dispute, i.e., takes step 8c in $\mathcal{SC}_{\text{PC}}$.
8) After $\mathcal{SC}_{\text{PC}}$ is finalized (with the involvement of the auditor), the following steps are taken to pay the parties involved.
   a) if none of the servers cheated in $\mathcal{SC}_{\text{PC}}$ (according to the auditor), then $\ddot{w} + \ddot{d'} + \ddot{d} - \ddot{ch}$ amount is refunded to $\mathcal{SC}_{\text{ANE}}$ and TRA's deposit (i.e., $\ddot{ch}$ amount) is transferred to D. Nothing is paid to TRA.
   b) if in $\mathcal{SC}_{\text{PC}}$, the other server did not cheat and TRA cheated; however, TRA provided a correct result in this contract, then $\ddot{d'} + \ddot{d} - \ddot{ch}$ amount is transferred to $\mathcal{SC}_{\text{ANE}}$. Also, TRA gets its deposit back (i.e., $\ddot{ch}$ amount) plus $\ddot{w}$ amount for providing a correct result to this contract.
   c) if in $\mathcal{SC}_{\text{PC}}$, both servers cheated; however, TRA delivered a correct computation result to this contract, then TRA gets its deposit back (i.e., $\ddot{ch}$ amount), it also receives $\ddot{w} + \ddot{d'} + \ddot{d} - \ddot{ch}$ amount.
   d) otherwise, $\ddot{w} + \ddot{d'} + \ddot{d} - \ddot{ch}$ and $\ddot{ch}$ amounts are transferred to $\mathcal{SC}_{\text{ANE}}$ and TRA respectively.
9) If TRA provided a result to this contract, and deadline T_3 (in $\mathcal{SC}_{\text{PC}}$) has passed, then all deposits, if any left, will be transferred to TRA.

TRA must take the following three steps to report collusion: (i) it waits until $\mathcal{SC}_{\text{CC}}$ is signed by the other server, (ii) it reports the collusion to D before signing $\mathcal{SC}_{\text{CC}}$, and (iii) it signs $\mathcal{SC}_{\text{CC}}$ only after it signed $\mathcal{SC}_{\text{TC}}$ with D.

Appendix C. Commitment Scheme

A commitment scheme involves a *sender* and a *receiver*. It also involves two phases; namely, *commit* and *open*. In the commit phase, the sender commits to a message: x as $\texttt{Com}(x, r) = com$, that involves a secret value: $r \xleftarrow{\$} \{0,1\}^\lambda$. At the end of the commit phase, the commitment com is sent to the receiver. In the open phase, the sender sends the opening $\hat{x} := (x, r)$ to the receiver who verifies its correctness: $\texttt{Ver}(com, \hat{x}) \stackrel{?}{=} 1$ and accepts if the output is 1.

A commitment scheme must satisfy two properties: (a) *hiding*: it is infeasible for an adversary (i.e., the receiver) to learn any information about the committed message x, until the commitment com is opened, and (b) *binding*: it is infeasible for an adversary (i.e., the sender) to open a commitment com to different values $\hat{x}' := (x', r')$ than that was used in the commit phase, i.e., infeasible to find $\hat{x}'$, *s.t.* $\texttt{Ver}(com, \hat{x}) = \texttt{Ver}(com, \hat{x}') = 1$, where $\hat{x} \neq \hat{x}'$.

There exist efficient commitment schemes both in (a) the standard model, e.g., Pedersen scheme [45], and (b) the random oracle model using the well-known hash-based scheme such that committing is : $\texttt{H}(x||r) = com$ and $\texttt{Ver}(com, \hat{x})$ requires checking: $\texttt{H}(x||r) \stackrel{?}{=} com$, where $\texttt{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ is a collision-resistant hash function, i.e., the probability to find x and x' such that $\texttt{H}(x) = \texttt{H}(x')$ is negligible in the security parameter λ.

Appendix D. Hash Tables

We set the table's parameters appropriately to ensure the number of elements in each bin does not exceed a predefined capacity. Given the maximum number of elements c and the bin's maximum size d, we can determine the number of bins by analyzing hash tables under the balls into bins model [17].

Theorem 2. ***(Upper Tail in Chernoff Bounds)*** *Let X_i be a random variable defined as $X_i = \sum_{i=1}^{c} Y_i$, where $Pr[Y_i = 1] = p_i$, $Pr[Y_i = 0] = 1 - p_i$, and all Y_i are independent. Let the expectation be $\mu = \mathrm{E}[X_i] = \sum_{i=1}^{h} p_i$, then $Pr[X_i > d = (1+\sigma)\cdot\mu] < \left(\frac{e^\sigma}{(1+\sigma)^{(1+\sigma)}}\right)^\mu, \forall\sigma > 0$*

In this model, the expectation is $\mu = \frac{c}{h}$, where c is the number of balls and h is the number of bins. The above inequality provides the probability that bin i gets more than $(1+\sigma)\cdot\mu$ balls. Since there are h bins, the probability that at least one of them is overloaded is bounded by the union bound:

$$Pr[\exists i, X_i > d] \leq \sum_{i=1}^{h} Pr[X_i > d] = h \cdot \left(\frac{e^\sigma}{(1+\sigma)^{(1+\sigma)}}\right)^{\frac{c}{h}} \quad (1)$$

Thus, for a hash table of length $h = O(c)$, there is always an *almost constant* expected number of elements, d, mapped to the same bin with a high probability [46], e.g., $1 - 2^{-40}$.

Appendix O.5 explains how to establish concrete parameters for a hash table.

Appendix E. Merkle Tree

A Merkle tree is a data structure that supports a compact commitment of a set of values/blocks. As a result, it includes two parties, prover $\mathcal{P}$ and verifier $\mathcal{V}$. The Merkle tree scheme includes three algorithms ($\texttt{MT.genTree}, \texttt{MT.prove}$, $\texttt{MT.verify}$), defined as follows:

- The algorithm that constructs a Merkle tree, $\texttt{MT.genTree}$, is run by $\mathcal{V}$. It takes blocks, $u := u_1, ..., u_n$, as input. Then, it groups the blocks in pairs. Next, a collision-resistant hash function, $\texttt{H}(.)$, is used to hash each pair. After that, the hash values are grouped in pairs and each pair is further hashed, and this process is repeated until only a single hash value, called "root", remains. This yields a tree with the leaves corresponding to the input blocks and the root corresponding to the last remaining hash value. $\mathcal{V}$ sends the root to $\mathcal{P}$.
- The proving algorithm, $\texttt{MT.prove}$, is run by $\mathcal{P}$. It takes a block index, i, and a tree as inputs. It outputs a vector proof, of $\log_2(n)$ elements. The proof asserts the membership of i-th block in the tree and consists of all the sibling nodes on a path from the i-th block to the root of the Merkle tree (including i-th block). The proof is given to $\mathcal{V}$.
- The verification algorithm, $\texttt{MT.verify}$, is run by $\mathcal{V}$. It takes as an input i-th block, a proof, and the tree's root. It checks if the i-th block corresponds to the root. If the verification passes, it outputs 1; otherwise, it outputs 0.

The Merkle tree-based scheme has two properties: *correctness* and *security*. Informally, the correctness requires that if both parties run the algorithms correctly, then a proof is always accepted by $\mathcal{V}$. The security requires that a computationally bounded malicious $\mathcal{P}$ cannot convince $\mathcal{V}$ into accepting an incorrect proof, e.g., proof for a non-member block. The security relies on the assumption that it is computationally infeasible to find the hash function's collision. Usually, for the sake of simplicity, it is assumed that the number of blocks, n, is a power of 2. The height of the tree, constructed on m blocks, is $\log_2(n)$.

Appendix F. Enhanced OLE's Protocol

The PSIs proposed in [24] use an enhanced version of the $\texttt{OLE}$. The enhanced $\texttt{OLE}$ ensures that the receiver cannot learn anything about the sender's inputs, in the case where it sets its input to 0, i.e., $c = 0$. The enhanced $\texttt{OLE}$'s protocol (denoted by $\texttt{OLE}^+$) is presented in Figure 6. In this paper, $\texttt{OLE}^+$ is used as a subroutine in $\texttt{VOPR}$, in Section 4.1.

1) Receiver (input $c \in \mathbb{F}_p$): Pick a random value, $r \xleftarrow{\$} \mathbb{F}_p$, and send $(\texttt{inputS}, (c^{-1}, r))$ to the first $\mathcal{F}_{\texttt{OLE}}$.
2) Sender (input $a, b \in \mathbb{F}_p$): Pick a random value, $u \xleftarrow{\$} \mathbb{F}_p$, and send $(\texttt{inputR}, u)$ to the first $\mathcal{F}_{\texttt{OLE}}$, to learn $t = c^{-1} \cdot u + r$. Send $(\texttt{inputS}, (t + a, b - u))$ to the second $\mathcal{F}_{\texttt{OLE}}$.
3) Receiver: Send $(\texttt{inputR}, c)$ to the second $\mathcal{F}_{\texttt{OLE}}$ and obtain $k = (t + a) \cdot c + (b - u) = a \cdot c + b + r \cdot c$. Output $s = k - r \cdot c = a \cdot c + b$.

Figure 6: Enhanced Oblivious Linear function Evaluation ($\texttt{OLE}^+$) [24].

Appendix G. Coin-Tossing Protocol

A Coin-Tossing protocol, CT, allows two mutually distrustful parties, say A and B, to jointly generate a single random bit. Formally, CT computes the functionality $f_{\texttt{CT}}(in_A, in_B) \rightarrow (out_A, out_B)$, which takes in_A and in_B as inputs of A and B respectively and outputs out_A to A and out_B to B, where $out_A = out_B$. A basic security requirement of a CT is that the resulting bit is (computationally) indistinguishable from a truly random bit.

Blum proposed a simple CT in [47] that works as follows. Party A picks a random bit $in_A \xleftarrow{\$} \{0, 1\}$, commits to it and sends the commitment to B which sends its choice of random input, $in_B \xleftarrow{\$} \{0, 1\}$, to A. Then, A sends the opening of the commitment (including in_A) to B, which checks whether the commitment matches its opening. If so, each party computes the final random bit as $in_A \oplus in_B$.

There have also been *fair* coin-tossing protocols, e.g., in [48], that ensure either both parties learn the result or nobody does. These protocols can be generalized to *multi-party* coin-tossing protocols to generate a *random string* (rather than a single bit), e.g., see [49], [50]. The overall computation and communication complexities of (fair) multi-party coin-tossing protocols are often linear with the number of participants. In this paper, any secure multi-party CT that generates a random string can be used. For the sake of simplicity, we let a multi-party $f_{\texttt{CT}}$ take m inputs and output a single value, i.e., $f_{\texttt{CT}}(in_1, ..., in_m) \rightarrow out$.

Appendix H. Formal Definitions of $\mathcal{PSI}^{\mathcal{FC}}$

At a high level, the standard simulation-based paradigm ensures only privacy and output correctness [12]. To enhance the simulation-based model and accommodate additional security requirements (such as fairness, compensation, or transactions' correctness) researchers often: (i) define a set of predicates and (ii) use a wrapper that encapsulates and parameterizes the original paradigm with the defined predicates, e.g., see [51], [52], [53]. Hence, to formally define a $\mathcal{PSI}^{\mathcal{FC}}$, we parameterize f^{PSI} with four predicates, $Q := (Q^{\text{Init}}, Q^{\text{Del}}, Q^{\text{UF-A}}, Q^{\text{F-A}})$, which ensure that certain financial conditions are met. We borrow three of these predicates (i.e., $Q^{\text{Init}}, Q^{\text{Del}}, Q^{\text{UF-A}}$) from [51]. But, we will (i) introduce predicate $Q^{\text{F-A}}$ and (ii) provide more formal accurate definitions of these predicates.

- Q^{Init}: Initiation predicate. It defines the conditions under which a protocol realizing $\mathcal{PSI}^{\mathcal{FC}}$ should begin execution, i.e., when all set owners have sufficient deposit.
- Q^{Del}: Delivery predicate. It defines the circumstances in which parties receive their output, specifically when honest parties receive their deposit back.
- $Q^{\text{UF-A}}$: UnFair-Abort predicate. It specifies the conditions under which the simulator can force parties to abort if the adversary learns the output, i.e., when an honest party receives its deposit back along with a predefined amount of compensation.
- $Q^{\text{F-A}}$: Fair-Abort predicate. It specifies the conditions under which the simulator can compel parties to abort if the adversary receives no output, i.e., when honest parties receive their deposits.

By requiring any protocol that realizes $\mathcal{PSI}^{\mathcal{FC}}$, to implement a wrapped version of f^{PSI} that includes Q, we will ensure that an honest set owner only aborts unfairly if $Q^{\text{UF-A}}$ returns 1, it only aborts in a fair manner if $Q^{\text{F-A}}$ returns 1, and outputs a valid value if Q^{Del} returns 1.

Definition 3 (Q^{Init}: Initiation predicate)**.** Let $\mathcal{G}$ be a stable ledger, adr_{sc} be smart contract sc's address, Adr be a set of $m+1$ distinct addresses, and $\ddot{x}$ be a fixed amount of coins. Then, predicate $Q^{\text{Init}}(\mathcal{G}, adr_{sc}, m+1, Adr, \ddot{x})$ returns 1 if every address in Adr has at least $\ddot{x}$ coins in sc. Otherwise, it returns 0.

Definition 4 (Q^{Del}: Delivery predicate)**.** Let $pram := (\mathcal{G}, adr_{sc}, \ddot{x})$ be the parameters defined above, and $adr_i \in Adr$ be the address of an honest party. Then, predicate $Q^{\text{Del}}(pram, adr_i)$ returns 1 if adr_i has sent $\ddot{x}$ amount to sc and received $\ddot{x}$ amount from it; thus, its balance in sc is 0. Otherwise, it returns 0.

Definition 5 ($Q^{\text{UF-A}}$: UnFair-Abort predicate)**.** Let $pram := (\mathcal{G}, adr_{sc}, \ddot{x})$ be the parameters defined above, and $Adr' \subset Adr$ be a set containing honest parties' addresses, $m' = |Adr'|$, and $adr_i \in Adr'$. Let also G be a compensation function that takes as input three parameters $(\ddot{deps}, adr_i, m')$, where $\ddot{deps}$ is the number of coins that all $m+1$ parties deposit. It returns the amount of compensation each honest party must receive, i.e., $G(\ddot{deps}, ard_i, m') \rightarrow \ddot{x}_i$. Then, predicate $Q^{\text{UF-A}}$ is defined as $Q^{\text{UF-A}}(pram, G, \ddot{deps}, m', adr_i) \rightarrow (a, b)$, where $a = 1$ if adr_i is an honest party's address and adr_i has sent $\ddot{x}$ amount to sc and received $\ddot{x} + \ddot{x}_i$ from it, and $b = 1$ if adr_i is Aud's address and adr_i received $\ddot{x}_i$ from sc. Otherwise, $a = b = 0$.

Definition 6 ($Q^{\text{F-A}}$: Fair-Abort predicate)**.** Let $pram := (\mathcal{G}, adr_{sc}, \ddot{x})$ be the parameters defined above, and $Adr' \subset Adr$ be a set containing honest parties' addresses, $m' = |Adr'|$, $adr_i \in Adr'$, and adr_j be Aud's address. Let G be the compensation function, defined above, and let $G(deps, ard_j, m') \rightarrow \ddot{x}_j$ be the compensation that the auditor must receive. Then, predicate $Q^{\text{F-A}}(pram, G, \ddot{deps}, m', adr_i, adr_j)$ returns 1, if adr_i (s.t. $adr_i \neq adr_j$)

has sent $\ddot{x}$ amount to sc and received $\ddot{x}$ from it, and adr_j received $\ddot{x}_j$ from sc. Otherwise, it returns 0.

We observed that predicate $Q^{\text{F-A}}$ should have been defined in the generic framework in [51] too. Because the framework should have also captured the cases where an adversary may abort without learning any output after the onset of the protocol.

Definition 7 ($\mathcal{PSI}^{\mathcal{FC}}$)**.** Let f^{PSI} be the functionality defined above. Protocol Γ realizes f^{PSI} with Q-fairness in the presence of $m-1$ active-adversary clients or a passive dealer D or auditor Aud, if for every PPT adversary $\mathcal{A}$ for the real model, there is a PPT simulator Sim for the ideal model, such that for every $I \in \{A_1, ..., A_m, D, Aud\}$, it holds that: $\{\mathsf{Ideal}^{\mathcal{W}(f^{\text{PSI}},Q)}_{\mathsf{Sim}(z),I}(S_1, ..., S_{m+1})\}_{S_1,...,S_{m+1},z} \overset{c}{\equiv} \{\mathsf{Real}^{\Gamma}_{\mathcal{A}(z),I}(S_1, ..., S_{m+1})\}_{S_1,...,S_{m+1},z}$, where z is an auxiliary input given to $\mathcal{A}$ and $\mathcal{W}(f^{\text{PSI}}, Q)$ is a functionality that wraps f^{PSI} with predicates $Q := (Q^{\text{Init}}, Q^{\text{Del}}, Q^{\text{UF-A}}, Q^{\text{F-A}})$.

Appendix I.
Security Theorem and Proof of VOPR

Theorem 3. *Let f^{VOPR} be the functionality defined above. If the enhanced* OLE *(i.e.,* OLE^+*) is secure against malicious adversaries, then the* VOPR*, presented in Figure 1, securely computes f^{VOPR} in the presence of (i) semi-honest sender and honest receiver or (ii) malicious receiver and honest sender.*

Proof. Before proving Theorem 3, we present Lemma 1 and Theorem 4 that will be used in the proof of Theorem 3. Informally, Lemma 1 states that the evaluation of a random polynomial at a fixed value results in a uniformly random value.

Lemma 1. *Let x_i be an element of a finite field $\mathbb{F}_p$, picked uniformly at random and $\boldsymbol{\mu}(x)$ be a random polynomial of constant degree d (where $d = const(p)$) and defined over $\mathbb{F}_p[X]$. Then, the evaluation of $\boldsymbol{\mu}(x)$ at x_i is distributed uniformly at random over the elements of the field, i.e., $Pr[\boldsymbol{\mu}(x_i) = y] = \frac{1}{p}$, where $y \in \mathbb{F}_p$.*

Proof. Let $\boldsymbol{\mu}(x) = a_0 + \sum_{j=1}^{d} a_j x^j$, where the coefficients are distributed uniformly at random over the field.

Then, for any choice of x and random coefficients $a_1, ..., a_d$, it holds that: $Pr[\boldsymbol{\mu}(x) = y] = Pr[\sum_{i=0}^{d} a_j \cdot x^j = y] = Pr[a_0 = y - \sum_{j=1}^{d} a_j \cdot x^j] = \frac{1}{p}$, $\forall y \in \mathbb{F}_p$, as a_0 has been picked uniformly at random from $\mathbb{F}_p$. □

Informally, Theorem 4 states that the product of two arbitrary polynomials (in coefficient form) is a polynomial whose roots are the union of the two original polynomials. Below, we formally state it. The theorem has been taken from [25].

Theorem 4. *Let $\mathbf{p}$ and $\mathbf{q}$ be two arbitrary non-constant polynomials of degree d and d' respectively, such that $\mathbf{p}, \mathbf{q} \in \mathbb{F}_p[X]$ and they are in coefficient form. Then, the product of the two polynomials is a polynomial whose roots include precisely the two polynomials' roots.*

We refer readers to Appendix W for the proof of Theorem 4. Next, we prove the main theorem, i.e., Theorem 3, by considering the case where each party is corrupt, in turn.

Case 1: Corrupt sender. In the real execution, the sender's view is defined as follows:

$$\mathsf{View}^{\text{VOPR}}_S\Big((\boldsymbol{\psi}, \boldsymbol{\alpha}), \boldsymbol{\beta}\Big) = \{\boldsymbol{\psi}, \boldsymbol{\alpha}, r_S, \boldsymbol{\beta}(z), \boldsymbol{\theta}(z), \mathsf{View}^{\text{OLE}^+}_S, \perp\}$$

where r_S is the outcome of internal random coins of the sender and $\mathsf{View}^{\text{OLE}^+}_S$ refers to the sender's real-model view during the execution of OLE^+. The simulator $\mathsf{Sim}^{\text{VOPR}}_S$, which receives $\boldsymbol{\psi}$ and $\boldsymbol{\alpha}$, works as follows.

1) generates an empty view. It appends to the view polynomials $(\boldsymbol{\psi}, \boldsymbol{\alpha})$ and coins r'_S chosen uniformly at random.
2) computes polynomial $\boldsymbol{\beta} = \boldsymbol{\beta}_1 \cdot \boldsymbol{\beta}_2$, where $\boldsymbol{\beta}_1$ is a random polynomial of degree 1 and $\boldsymbol{\beta}_2$ is an arbitrary polynomial of degree $e' - 1$. Next, it constructs polynomial $\boldsymbol{\theta}$ as follows: $\boldsymbol{\theta} = \boldsymbol{\psi} \cdot \boldsymbol{\beta} + \boldsymbol{\alpha}$.
3) chooses value $z \xleftarrow{\$} \mathbb{F}_p$. Then, it evaluates polynomials $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ at point z. This results in values β_z and θ_z respectively. It appends these two values to the view.
4) extracts the sender-side simulation of OLE^+ from OLE^+'s simulator. Let $\mathsf{Sim}^{\text{OLE}^+}_S$ be this simulation. Note, the latter simulation is guaranteed to exist, as OLE^+ has been proven secure (in [24]). It appends $\mathsf{Sim}^{\text{OLE}^+}_S$ and $\perp$ to its view.

Now, we are prepared to demonstrate that the two views are computationally indistinguishable. The sender's inputs are identical in both models, so they have identical distributions. Since the real-model semi-honest adversary samples its randomness according to the protocol's description, the random coins in both models have identical distributions. Next, we explain why values $\boldsymbol{\beta}(z)$ in the real model and β_z in the ideal model are (computationally) indistinguishable. In the real model, $\boldsymbol{\beta}(z)$ is the evaluation of polynomial $\boldsymbol{\beta} = \boldsymbol{\beta}_1 \cdot \boldsymbol{\beta}_2$ at random point z, where $\boldsymbol{\beta}_1$ is a random polynomial. We know that $\boldsymbol{\beta}(z) = \boldsymbol{\beta}_1(z) \cdot \boldsymbol{\beta}_2(z)$, for any (non-zero) z.

Moreover, by Lemma 1, we know that $\boldsymbol{\beta}_1(z)$ is a uniformly random value. Therefore, $\boldsymbol{\beta}(z) = \boldsymbol{\beta}_1(z) \cdot \boldsymbol{\beta}_2(z)$ is a uniformly random value as well. In the ideal world, polynomial $\boldsymbol{\beta}$ has the same structure as $\boldsymbol{\beta}$ has (i.e., $\boldsymbol{\beta} = \boldsymbol{\beta}_1 \cdot \boldsymbol{\beta}_2$, where $\boldsymbol{\beta}_1$ is a random polynomial). That means β_z is a uniformly random value too. Thus, $\boldsymbol{\beta}(z)$ and β_z are computationally indistinguishable. Next, we turn our attention to values $\boldsymbol{\theta}(z)$ in the real model and θ_z in the ideal model. We know that $\boldsymbol{\theta}(z)$ is a function of $\boldsymbol{\beta}_1(z)$, as polynomial $\boldsymbol{\theta}$ has been defined as $\boldsymbol{\theta} = \boldsymbol{\psi} \cdot (\boldsymbol{\beta}_1 \cdot \boldsymbol{\beta}_2) + \boldsymbol{\alpha}$. Similarly, θ_z is a function of β_z. As we have already discussed, $\boldsymbol{\beta}(z)$ and β_z are computationally indistinguishable, so are their functions $\boldsymbol{\theta}(z)$ and θ_z. Moreover, as OLE^+ has been proven secure,

$\mathsf{View}_S^{\mathtt{OLE}^+}$ and $\mathsf{Sim}_S^{\mathtt{OLE}^+}$ are computationally indistinguishable. It is also clear that $\perp$ is identical in both models. We deduce that the two views are computationally indistinguishable.

Case 2: Corrupt receiver. Let $\mathsf{Sim}_R^{\mathtt{VOPR}}$ be the simulator, in this case, which uses a subroutine adversary, $\mathcal{A}_R$. $\mathsf{Sim}_R^{\mathtt{VOPR}}$ works as follows.

1) simulates $\mathtt{OLE}^+$ and receives $\mathcal{A}_R$'s input coefficients b_j for all j, $0 \leq j \leq e'$, as we are in $f_{\mathtt{OLE}^+}$-hybrid model.
2) reconstructs polynomial $\boldsymbol{\beta}$, given the above coefficients.
3) simulates the honest sender's inputs as follows. It picks two random polynomials: $\boldsymbol{\psi} = \sum_{i=0}^{e} g_i \cdot x^i$ and $\boldsymbol{\alpha} = \sum_{j=0}^{e+e'} a_j \cdot x^j$, such that $g_i \xleftarrow{\$} \mathbb{F}_p$ and every a_j has the form: $a_j = \sum_{t,k=0}^{t=e,k=e'} a_{t,k}$, where $t+k=j$ and $a_{t,k} \xleftarrow{\$} \mathbb{F}_p$.
4) sends to $\mathtt{OLE}^+$'s functionality values g_i and $a_{i,j}$ and receives $c_{i,j}$ from this functionality (for all i,j).
5) sends all $c_{i,j}$ to TTP and receives polynomial $\boldsymbol{\theta}$.
6) picks a random value z from $\mathbb{F}_p$. Then, it computes $\psi_z = \boldsymbol{\psi}(z)$ and $\alpha_z = \boldsymbol{\alpha}(z)$.
7) sends z and all $c_{i,j}$ to $\mathcal{A}_R$ which sends back θ_z and β_z to the simulator.
8) sends ψ_z and α_z to $\mathcal{A}_R$.
9) checks if the following relation holds:

$$\beta_z = \boldsymbol{\beta}(z) \quad \wedge \quad \theta_z = \boldsymbol{\theta}(z) \quad \wedge \quad \boldsymbol{\theta}(z) = \psi_z \cdot \beta_z + \alpha_z \tag{2}$$

If Relation 2 does not hold, it aborts (i.e., sends abort signal Λ to the sender) and still proceeds to the next step.
10) outputs whatever $\mathcal{A}_R$ outputs.

We initially direct our attention to the adversary's output. Both values of z in the real and ideal models have been selected uniformly at random from $\mathbb{F}_p$. Therefore, they have identical distributions. In the real model, values ψ_z and α_z are the result of the evaluation of two random polynomials at (random) point z. In the ideal model, values ψ_z and α_z are also the result of the evaluation of two random polynomials (i.e., $\boldsymbol{\psi}$ and $\boldsymbol{\alpha}$) at point z. By Lemma 1, we know that the evaluation of a random polynomial at an arbitrary value yields a uniformly random value in $\mathbb{F}_p$. Therefore, the distribution of pair (ψ_z, α_z) in the real model is identical to that of pair (ψ_z, α_z) in the ideal model.

Moreover, the final result (i.e., values $c_{i,j}$) in the real model shares the same distribution as the final result (i.e., values $c_{i,j}$) in the ideal model, since they are the outputs of the ideal calls to $f_{\mathtt{OLE}^+}$, within the context of the $f_{\mathtt{OLE}^+}$-hybrid model.

Next, we turn our attention to the sender's output. We will demonstrate that the output distributions of the honest sender in both the ideal and real models exhibit statistical closeness. Our focus will be on the probability that it aborts in each model, as it does not receive any other output. In the ideal model, $\mathsf{Sim}_R^{\mathtt{VOPR}}$ is given the honestly generated result polynomial $\boldsymbol{\theta}$ (computed by TTP) and the adversary's input polynomial $\boldsymbol{\beta}$. $\mathsf{Sim}_R^{\mathtt{VOPR}}$ aborts with a probability of 1 if Relation 2 (on page 22) does not hold. However, in the real model, the honest sender (in addition to its inputs) is given only β_z and θ_z and is not given polynomials $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$; it wants to check if the following equation holds, $\theta_z = \psi_z \cdot \beta_z + \alpha_z$.

Note that polynomial $\boldsymbol{\theta} = \boldsymbol{\psi} \cdot \boldsymbol{\beta} + \boldsymbol{\alpha}$ (resulted from $c_{i,j}$) is well-structured, as it satisfies the following three conditions (regardless of the adversary's input $\boldsymbol{\beta}$ to $\mathtt{OLE}^+$):

1) $deg(\boldsymbol{\theta}) = Max\Big(deg(\boldsymbol{\beta}) + deg(\boldsymbol{\psi}), deg(\boldsymbol{\alpha})\Big)$, as $\mathbb{F}_p[X]$ is an integral domain and $(\boldsymbol{\psi}, \boldsymbol{\alpha})$ are random polynomials.
2) the roots of the product polynomial $\boldsymbol{\nu} = \boldsymbol{\psi} \cdot \boldsymbol{\beta}$ contains exactly both polynomials' roots, by Theorem 4, on page 21.
3) the roots of $\boldsymbol{\nu} + \boldsymbol{\alpha}$ is the intersection of the roots of $\boldsymbol{\nu}$ and $\boldsymbol{\alpha}$, as demonstrated in [20].

Furthermore, polynomial $\boldsymbol{\theta}$ reveals no information (about $\boldsymbol{\psi}$ and $\boldsymbol{\alpha}$ except their degrees) to the adversary and the pair (ψ_z, α_z) is given to the adversary after it sends the pair (θ_z, β_z) to the sender. There are exactly four cases where pair (θ_z, β_z) can be constructed by the real-model adversary. Below, we state each case and the probability that the adversary is detected in that case during the verification, i.e., $\theta_z \stackrel{?}{=} \psi_z \cdot \beta_z + \alpha_z$.

1) $\theta_z = \boldsymbol{\theta}(z) \wedge \beta_z = \boldsymbol{\beta}(z)$. This is a trivial non-interesting case, as the adversary has behaved honestly, so it can always pass the verification.
2) $\theta_z \neq \boldsymbol{\theta}(z) \wedge \beta_z = \boldsymbol{\beta}(z)$. In this case, the adversary is detected with a probability of 1.
3) $\theta_z = \boldsymbol{\theta}(z) \wedge \beta_z \neq \boldsymbol{\beta}(z)$. In this case, the adversary is also detected with a probability of 1.
4) $\theta_z \neq \boldsymbol{\theta}(z) \wedge \beta_z \neq \boldsymbol{\beta}(z)$. In this case, the adversary is detected with an overwhelming probability, i.e., $1 - \frac{1}{2^{2\lambda}}$.

As demonstrated above, in the real model, the minimum probability of the honest sender aborting in the event of adversarial behavior is $1 - \frac{1}{2^{2\lambda}}$. Hence, the output distributions of the honest sender in both the ideal and real models are statistically close, i.e., 1 vs $1 - \frac{1}{2^{2\lambda}}$.

In conclusion, we establish that the distribution of the joint outputs from the honest sender and the adversary in both the real and ideal models is computationally indistinguishable. □

Appendix J.
Security Theorem and Proof of ZSPA

Theorem 5. *Let $f^{\mathtt{ZSPA}}$ be the functionality defined above. If* CT *is secure against a malicious adversary and the correctness of* PRF*,* H*, and Merkle tree hold, then* ZSPA*, in Figure 2, securely computes $f^{\mathtt{ZSPA}}$ in the presence of $m-1$ malicious adversaries.*

Proof. For the sake of simplicity, we will assume the sender, which generates the result, directly transmits the result to the

remaining parties (i.e., receivers) rather than sending it to a smart contract. We first consider the case in which the sender is corrupt.

Case 1: Corrupt sender. Let $\mathsf{Sim}_S^{\texttt{ZSPA}}$ be the simulator using a subroutine adversary, $\mathcal{A}_S$. $\mathsf{Sim}_S^{\texttt{ZSPA}}$ operates as follows.

1) simulates CT and receives the output value k from $f_{\texttt{CT}}$, as we are in $f_{\texttt{CT}}$-hybrid model.
2) sends k to TTP and receives back from it m pairs, where each pair is of the form (g, q).
3) sends k to $\mathcal{A}_S$ and receives back from it m pairs where each pair is of the form (g', q').
4) checks whether the following equations hold (for each pair): $g = g' \ \wedge \ q = q'$. If the two equations do not hold, then it aborts (i.e., sends abort signal Λ to the receiver) and proceeds to the next step.
5) outputs whatever $\mathcal{A}_S$ outputs.

Initially, we focus on the adversary's output. In the real model, the only messages that the adversary receives are those messages it receives as the result of the ideal call to $f_{\texttt{CT}}$. These messages have an identical distribution to the distribution of the messages in the ideal model, as the CT is secure.

Now, we move on to the receiver's output. We will demonstrate that the output distributions of the honest receiver in the ideal and real models are computationally indistinguishable. In the real model, each element of pair (g, p) is the output of a deterministic function on the output of $f_{\texttt{CT}}$. We know the output of $f_{\texttt{CT}}$ in the real and ideal models have an identical distribution, and so do the evaluations of deterministic functions (i.e., Merkle tree, H, and PRF) on them, as long as these three functions' correctness hold. Therefore, each pair (g, q) in the real model has an identical distribution to pair (g, q) in the ideal model. For the same reasons, the honest receiver in the real model aborts with the same probability as $\mathsf{Sim}_S^{\texttt{ZSPA}}$ does in the ideal model.

We conclude that the distributions of the joint outputs of the adversary and honest receiver in the real and ideal models are computationally indistinguishable.

Case 2: Corrupt receiver. Let $\mathsf{Sim}_R^{\texttt{ZSPA}}$ be the simulator that uses subroutine adversary $\mathcal{A}_R$. $\mathsf{Sim}_R^{\texttt{ZSPA}}$ operates as follows.

1) simulates CT and receives the output value k from $f_{\texttt{CT}}$.
2) sends k to TTP and receives back m pairs of the form (g, q) from TTP.
3) sends (k, g, q) to $\mathcal{A}_R$ and outputs whatever $\mathcal{A}_R$ outputs.

In the real model, the adversary receives two sets of messages, the first set includes the transcripts (including k) it receives when it makes an ideal call to $f_{\texttt{CT}}$ and the second set includes pair (g, q). As we already discussed above (because we are in the $f_{\texttt{CT}}$-hybrid model) the distributions of the messages it receives from $f_{\texttt{CT}}$ in the real and ideal models are identical.

Moreover, the distribution of $f_{\texttt{CT}}$'s output (i.e., $\bar{k}$ and k) in both models is identical; therefore, the honest sender's output distribution in both models is identical. As we already discussed, the evaluations of deterministic functions (i.e., Merkle tree, H, and PRF) on $f_{\texttt{CT}}$'s outputs have an identical distribution. Therefore, each pair (g, q) in the real model has an identical distribution to the pair (g, q) in the ideal model.

Hence, the distribution of the joint outputs of the adversary and honest receiver in the real and ideal models is indistinguishable. □

In addition to the security guarantee stated by Theorem 5 (which ensures computation correctness against malicious sender or receiver) ZSPA also provides (a) privacy against the public, and (b) non-refutability. In informal terms, privacy in this context implies that, with the contract's state (i.e., g and q), an external party cannot glean any information about any of the pseudorandom values, z_j. On the other hand, non-refutability signifies that if a party sends "approved", they cannot later deny knowledge of the values whose representations are stored in the contract.

Theorem 6. *If* H *is preimage resistance,* PRF *is secure, the signature scheme used in the smart contract is secure (i.e., existentially unforgeable under chosen message attacks), and the blockchain is secure (i.e., offers persistence and liveness properties [54]) then* ZSPA *offers (i) privacy against the public and (ii) non-refutability.*

Proof. We will initially focus on privacy. Since key k for PRF has been uniformly and randomly selected, and H exhibits pre-image resistance, the probability that the adversary can find k (given g), is negligible with respect to the security parameter, i.e., $\epsilon(\lambda)$. Furthermore, because PRF is secure (i.e., its outputs are indistinguishable from random values) and H is pre-image resistance, given the Merkle tree's root g, the probability that the adversary can find a leaf node, which is the output of PRF, is $\epsilon(\lambda)$ too.

Now we move on the non-refutability. Due to the persistency property of the blockchain, when a transaction or message reaches a depth of more than v in the blockchain of one honest player (where v is a security parameter), it will be included in the blockchain of every honest player with overwhelming probability, and it will be given a permanent position in the blockchain, making it highly unlikely to be modified.

Furthermore, thanks to the liveness property, all transactions originating from honest parties will eventually reach a depth of more than v blocks in an honest player's blockchain. Consequently, the adversary cannot carry out a selective denial-of-service attack against honest account holders. Moreover, due to the security of the digital signature (i.e., existentially unforgeable under chosen message attacks), one cannot deny having sent the messages to the blockchain and smart contract. □

Appendix K. Proof of ZSPA-A

Proof. First, we consider the case where a sender, who (may collude with $m - 2$ senders and) generates pairs (g, q), is corrupt.

Case 1: Corrupt sender. Let $\mathsf{Sim}_S^{\texttt{ZSPA-A}}$ be the simulator using a subroutine adversary, $\mathcal{A}_S$. Below, we explain how $\mathsf{Sim}_S^{\texttt{ZSPA-A}}$ works.

1) simulates CT and receives the output value k from $f_{\texttt{CT}}$.
2) sends k to TTP and receives back from it m pairs, where each pair is of the form (g, q).
3) sends k to $\mathcal{A}_S$ and receives back from it m pairs where each pair is of the form (g', q').
4) constructs an empty vector L. $\mathsf{Sim}_S^{\texttt{ZSPA-A}}$ checks whether the following equations hold for each j-th pair: $g = g' \ \wedge \ q = q'$. If these two equations do not hold, it sends an abort message Λ to other receiver clients, appends the index of the pair (i.e., j) to L, and proceeds to the next step for the valid pairs. In the case where there are no valid pairs, it moves on to step 9.
5) picks a random polynomial $\boldsymbol{\zeta}$ of degree 1. Moreover, for every $j \notin L$, $\mathsf{Sim}_S^{\texttt{ZSPA-A}}$ picks a random polynomial $\boldsymbol{\xi}^{(j)}$ of degree $b' - 1$, where $1 \leq j \leq m$.
6) computes m pseudorandom values for every i, j', where $0 \leq i \leq b'$ and $j' \notin L$ as follows. $\forall j', 1 \leq j' \leq m-1 : z_{i,j} = \texttt{PRF}(k, i||j')$ and $z_{i,m} = -\sum_{j'=1}^{m-1} z_{i,j}$
7) generates polynomial $\boldsymbol{\mu}^{(j)}$, for every $j \notin L$, as follows: $\boldsymbol{\mu}^{(j)} = \boldsymbol{\zeta} \cdot \boldsymbol{\xi}^{(j)} - \boldsymbol{\tau}^{(j)}$, where $\boldsymbol{\tau}^{(j)} = \sum_{i=0}^{b'} z_{i,j} \cdot x^i$.
8) sends the above $\boldsymbol{\zeta}$, $\boldsymbol{\xi}^{(j)}$, and $\boldsymbol{\mu}^{(j)}$ to all parties (i.e., $\mathcal{A}_S$ and the receivers), for every $j \notin L$.
9) outputs whatever $\mathcal{A}_S$ outputs.

Now, we focus on the adversary's output. In the real model, the messages that the adversary receives include those messages it receives as the result of the ideal call to $f_{\texttt{CT}}$ and $(\boldsymbol{\zeta}, \boldsymbol{\xi}^{(j)}, \boldsymbol{\mu}^{(j)})$, where $j \notin L$ and $1 \leq j \leq m$. Those messages yielded from the ideal calls have an identical distribution to the distribution of the messages in the ideal model, as CT is secure. The distribution of each $\boldsymbol{\mu}^{(j)}$ depends on the distribution of its components; namely, $\boldsymbol{\zeta}, \boldsymbol{\xi}^{(j)}$, and $\boldsymbol{\tau}^{j}$. As we are in the $f_{\texttt{CT}}$-hybrid model, the distributions of $\boldsymbol{\tau}^{(j)}$ in the real model and $\boldsymbol{\tau}^{(j)}$ in the ideal model are identical, as they were derived from the output of $f_{\texttt{CT}}$. Furthermore, in the real model, each polynomial $\boldsymbol{\zeta}$ and $\boldsymbol{\xi}^{(j)}$ has been picked uniformly at random and they are independent of the clients' and the adversary's inputs. The same arguments hold for $(\boldsymbol{\zeta}, \boldsymbol{\xi}^{(j)}, \boldsymbol{\mu}^{(j)})$ in the ideal model. Therefore, $(\boldsymbol{\zeta}, \boldsymbol{\xi}^{(j)}, \boldsymbol{\mu}^{(j)})$ in the real model and $(\boldsymbol{\zeta}, \boldsymbol{\xi}^{(j)}, \boldsymbol{\mu}^{(j)})$ in the ideal model have identical distributions.

Next, we turn our attention to the receiver's output. We will show that the output distributions of an honest receiver and the auditor in the ideal and real models are computationally indistinguishable. In the real model, each element of the pair (g, p) is the output of a deterministic function on the output of $f_{\texttt{CT}}$. We know the outputs of $f_{\texttt{CT}}$ in the real and ideal models have an identical distribution, and so do the evaluations of deterministic functions (namely Merkle tree, H, and PRF) on them. Therefore, each pair (g, q) in the real model has an identical distribution to the pair (g, q) in the ideal model. For the same reasons, the honest receiver in the real model aborts with the same probability as $\mathsf{Sim}_S^{\texttt{ZSPA-A}}$ does in the ideal model.

The same argument holds for the arbiter's output, as it performs the same checks that an honest receiver does. Thus, the distribution of the joint outputs of the adversary, honest receiver, and honest in the real and ideal models is computationally indistinguishable.

Case 2: Corrupt receiver. Let $\mathsf{Sim}_R^{\texttt{ZSPA-A}}$ be the simulator that uses subroutine adversary $\mathcal{A}_R$. Below, we explain how $\mathsf{Sim}_R^{\texttt{ZSPA-A}}$ works.

1) simulates ZSPA and receives the m output pairs of the form (k, g, q) from $f^{\texttt{ZSPA}}$.
2) sends (k, g, q) to $\mathcal{A}_R$ and receives m keys, k'_j, where $1 \leq j \leq m$.
3) generates an empty vector L. Next, for every j, $\mathsf{Sim}_R^{\texttt{ZSPA-A}}$ computes q'_j as $\texttt{H}(k'_j) = q_j$. It generates g_j as follows.
   a) for every i (where $0 \leq i \leq b'$), generates m pseudorandom values as below. $\forall j, 1 \leq j' \leq m-1 : z_{i,j} = \texttt{PRF}(k'_j, i||j'), \quad z_{i,m} = -\sum_{j=1}^{m-1} z_{i,j}$
   b) constructs a Merkle tree on top of all pseudorandom values, $\texttt{MT.genTree}(z_{0,1}, ..., z_{b',m}) \rightarrow g'_j$.
4) checks if the following equations hold for each j-th pair: $(k = k'_j) \ \wedge \ (g = g'_j) \ \wedge \ (q = q'_j)$.
5) If these equations do not hold for j-th value, it appends j to L and proceeds to the next step for the valid value. In the case where there is no valid value, it moves on to step 9.
6) picks a random polynomial $\boldsymbol{\zeta}$ of degree 1. Also, for every $j \notin L$, it picks a random polynomial $\boldsymbol{\xi}^{(j)}$ of degree $b' - 1$, where $1 \leq j \leq m$.
7) generates polynomial $\boldsymbol{\mu}^{(j)}$, for every $j \notin L$, as follows: $\boldsymbol{\mu}^{(j)} = \boldsymbol{\zeta} \cdot \boldsymbol{\xi}^{(j)} - \boldsymbol{\tau}^{(j)}$, where $\boldsymbol{\tau}^{(j)} = \sum_{i=0}^{b'} z_{i,j} \cdot x^i$, and values $z_{i,j}$ were generated in step 3a.
8) sends the above $\boldsymbol{\zeta}$, $\boldsymbol{\xi}^{(j)}$, and $\boldsymbol{\mu}^{(j)}$ to $\mathcal{A}_R$, for every $j \notin L$ and $1 \leq j \leq m$.
9) outputs whatever $\mathcal{A}_R$ outputs.

In the real model, the adversary receives two sets of messages, the first set includes the transcripts (including k, g, q) it receives when it makes an ideal call to $f^{\texttt{ZSPA}}$ and the second set includes pairs $(\boldsymbol{\zeta}, \boldsymbol{\xi}^{(j)}, \boldsymbol{\mu}^{(j)})$, for every $j \notin L$ and $1 \leq j \leq m$. Since we are in the $f^{\texttt{ZSPA}}$-hybrid model and (based on our assumption) there is at least one honest party participated in ZSPA (i.e., there are at most $m-1$ malicious participants of ZSPA), the distribution of the messages it receives from $f^{\texttt{ZSPA}}$ in the real and ideal models is identical. Furthermore, as we discussed in Case 1, $(\boldsymbol{\zeta}, \boldsymbol{\xi}^{(j)}, \boldsymbol{\mu}^{(j)})$ in the real model and $(\boldsymbol{\zeta}, \boldsymbol{\xi}^{(j)}, \boldsymbol{\mu}^{(j)})$ in the ideal model have identical distribution. The honest sender's output distribution in both models is identical, as the distribution of $f_{\texttt{CT}}$'s output (i.e., k) in both models is identical.

Now we show that the probability that the auditor aborts in the ideal and real models is statistically close. In the ideal model, $\mathsf{Sim}_R^{\texttt{ZSPA-A}}$ is given the ideal functionality's output that includes key k. Therefore, it can check whether the key that $\mathcal{A}_R$ sends to it equals k, i.e., $k \stackrel{?}{=} k'_j$. Thus, it aborts with the

probability 1. However, in the real model, an honest auditor is not given the output of CT (say key k) and it can only check whether the key is consistent with the hash value q and the Merkle tree's root g stored on the blockchain.

This implied that the adversary can distinguish between the two models if, in the real model, it sends a key $\ddot{k}$, such that $\ddot{k} \neq k$, and still passes the checks. More precisely, it sends the invalid key $\ddot{k}$ that can generate valid pair (g, q), as follows: $\mathtt{H}(\ddot{k}) = q$ and $\mathtt{MT.genTree}(z'_{0,1}, ..., z'_{b',m}) \rightarrow g$, where each $z'_{i,j}$ is derived from $\ddot{k}$ using the same technique described in step 3 above.

This means that the adversary breaks the second pre-image resistance property of H. However, H is indeed the second-pre-image resistance and the probability that the adversary succeeds in finding the second pre-image is negligible in the security parameter, i.e., $\epsilon(\lambda)$ where λ is the hash function's security parameter. Therefore, in the real model, the auditor aborts if an invalid key is provided with a probability $1 - \epsilon(\lambda)$ which is statically close to the probability that $\mathsf{Sim}_R^{\text{ZSPA-A}}$ aborts in the same situation in the ideal model, i.e., $1 - \epsilon(\lambda)$ vs 1. Hence, the distribution of the joint outputs of the adversary, honest sender, and honest auditor in the real and ideal models is indistinguishable. □

Appendix L.
Definition and proof of Unforgeable Polynomial

Theorem 7 (Unforgeable Polynomial)**.** *Let polynomials $\boldsymbol{\zeta}$ and $\boldsymbol{\gamma}$ be secret uniformly random polynomials (i.e., $\boldsymbol{\zeta}, \boldsymbol{\gamma} \xleftarrow{\$} \mathbb{F}_p[x]$), polynomial $\boldsymbol{\pi}$ be an arbitrary polynomial, $deg(\boldsymbol{\zeta}) = 1$, $deg(\boldsymbol{\gamma}) = d + 1$, $deg(\boldsymbol{\pi}) = d$, and p be a λ-bit prime number. Also, let polynomial $\boldsymbol{\theta}$ be defined as $\boldsymbol{\theta} = \boldsymbol{\zeta} \cdot \boldsymbol{\pi} + \boldsymbol{\gamma} \bmod p$. Given $(\boldsymbol{\theta}, \boldsymbol{\pi})$, the probability that an adversary (which does not know $\boldsymbol{\zeta}$ and $\boldsymbol{\gamma}$) can forge $\boldsymbol{\theta}$ to an arbitrary polynomial $\boldsymbol{\delta}$ such that $\boldsymbol{\delta} \neq \boldsymbol{\theta}$, $deg(\boldsymbol{\delta}) = poly(\lambda)$, and $\boldsymbol{\zeta}$ divides $\boldsymbol{\delta} - \boldsymbol{\gamma}$ is negligible in the security parameter λ, i.e., $Pr[\boldsymbol{\zeta} \mid (\boldsymbol{\delta} - \boldsymbol{\gamma})] \leq \epsilon(\lambda)$.*

We proceed to prove Theorem 7.

Proof. Let $\boldsymbol{\tau} = \boldsymbol{\delta} - \boldsymbol{\gamma}$ and $\boldsymbol{\zeta} = a \cdot x + b$. Since $\boldsymbol{\gamma}$ is a random polynomial of degree $d + 1$ and unknown to the adversary when given $(\boldsymbol{\theta}, \boldsymbol{\pi})$, the adversary cannot gain any information about the factor $\boldsymbol{\zeta}$. From its perspective, every polynomial of degree 1 in $\mathbb{F}_p[X]$ is equally likely to be $\boldsymbol{\zeta}$.

Furthermore, the polynomial $\boldsymbol{\tau}$ has at most $Max\big(deg(\boldsymbol{\delta}), d + 1\big)$ irreducible non-constant factors. For $\boldsymbol{\zeta}$ to divide $\boldsymbol{\tau}$, one of the factors of $\boldsymbol{\tau}$ must be equal to $\boldsymbol{\zeta}$. We also know that $\boldsymbol{\zeta}$ has been picked uniformly at random (i.e., $a, b \xleftarrow{\$} \mathbb{F}_p$). Thus, the probability that $\boldsymbol{\zeta}$ divides $\boldsymbol{\tau}$ is negligible with respect to the security parameter, λ. Specifically,

$$Pr[\boldsymbol{\zeta} \mid (\boldsymbol{\delta} - \boldsymbol{\gamma})] \leq \frac{Max\big(deg(\boldsymbol{\delta}), d + 1\big)}{2^{\lambda}} = \epsilon(\lambda)$$

□

Appendix M.
Definition and Proof of Unforgeable Polynomials' Linear Combination

Theorem 8 (Unforgeable Polynomials' Linear Combination)**.** *Let polynomial $\boldsymbol{\zeta}$ be a secret polynomial picked uniformly at random; also, let $\vec{\boldsymbol{\gamma}} = [\boldsymbol{\gamma}_1, ..., \boldsymbol{\gamma}_n]$ be a vector of secret uniformly random polynomials (i.e., $\boldsymbol{\zeta}, \boldsymbol{\gamma}_i \xleftarrow{\$} \mathbb{F}_p[x]$), $\vec{\boldsymbol{\pi}} = [\boldsymbol{\pi}_1, ..., \boldsymbol{\pi}_n]$ be a vector of arbitrary polynomials, $deg(\boldsymbol{\zeta}) = 1, deg(\boldsymbol{\gamma}_i) = d + 1$, $deg(\boldsymbol{\pi}_i) = d$, p be a λ-bit prime number, and $1 \leq i \leq n$. Moreover, let polynomial $\boldsymbol{\theta}_i$ be defined as $\boldsymbol{\theta}_i = \boldsymbol{\zeta} \cdot \boldsymbol{\pi}_i + \boldsymbol{\gamma}_i \bmod p$, and $\vec{\boldsymbol{\theta}} = [\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_n]$. Given $(\vec{\boldsymbol{\theta}}, \vec{\boldsymbol{\pi}})$, the probability that an adversary (which does not know $\boldsymbol{\zeta}$ and $\vec{\boldsymbol{\gamma}}$) can forge t polynomials, without loss of generality, let us say $\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_t \in \vec{\boldsymbol{\theta}}$, to arbitrary polynomials $\boldsymbol{\delta}_1, ..., \boldsymbol{\delta}_t$ such that $\sum_{j=1}^{t} \boldsymbol{\delta}_j \neq \sum_{j=1}^{t} \boldsymbol{\theta}_j$, $deg(\boldsymbol{\delta}_j) = const(\lambda)$, and $\boldsymbol{\zeta}$ divides $(\sum_{j=1}^{t} \boldsymbol{\delta}_j + \sum_{j=t+1}^{n} \boldsymbol{\theta}_j - \sum_{j=1}^{n} \boldsymbol{\gamma}_j)$ is negligible in the security parameter λ, i.e.,*

$$Pr[\boldsymbol{\zeta} \mid (\sum_{j=1}^{t} \boldsymbol{\delta}_j + \sum_{j=t+1}^{n} \boldsymbol{\theta}_j - \sum_{j=1}^{n} \boldsymbol{\gamma}_j)] \leq \epsilon(\lambda)$$

Next, we prove Theorem 8.

Proof. This proof serves as a generalization of that for Theorem 7. Let $\boldsymbol{\tau}_j = \boldsymbol{\delta}_j - \boldsymbol{\gamma}_j$ and $\boldsymbol{\zeta} = a \cdot x + b$. Since every $\boldsymbol{\gamma}_j$ is a random polynomial of degree $d + 1$ and unknown to the adversary, given $(\vec{\boldsymbol{\theta}}, \vec{\boldsymbol{\pi}})$, the adversary cannot learn anything about the factor $\boldsymbol{\zeta}$. Each polynomial $\boldsymbol{\tau}_j$ has at most $Max\big(deg(\boldsymbol{\delta}_j), d+1\big)$ irreducible non-constant factors. We know that $\boldsymbol{\zeta}$ has been picked uniformly at random (i.e., $a, b \xleftarrow{\$} \mathbb{F}_p$). Therefore, the probability that $\boldsymbol{\zeta}$ divides $\sum_{j=1}^{t} \boldsymbol{\delta}_j + \sum_{j=t+1}^{n} \boldsymbol{\theta}_j - \sum_{j=1}^{n} \boldsymbol{\gamma}_j$ equals the probability that $\boldsymbol{\zeta}$ equals to one of the factors of every $\boldsymbol{\tau}_j$, that is negligible in the security parameter. Concretely,

$$Pr[\boldsymbol{\zeta} \mid (\sum_{j=1}^{t} \boldsymbol{\delta}_j + \sum_{j=t+1}^{n} \boldsymbol{\theta}_j - \sum_{j=1}^{n} \boldsymbol{\gamma}_j)] \leq$$

$$\frac{\prod_{j=1}^{t} Max\big(deg(\boldsymbol{\delta}_j), d+1\big)}{2^{\lambda t}} = \epsilon(\lambda)$$

□

Appendix N.
Justitia's Workflow

Figure 7 outlines the interaction between parties in Justitia.

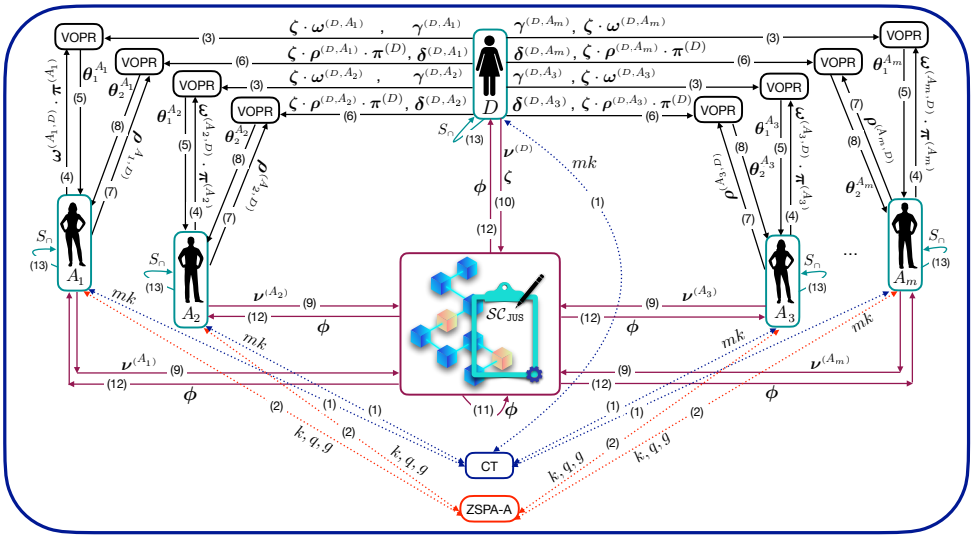


Figure 7: Outline of the interactions between parties in Justitia.

Appendix O. Further Discussion on Justitia

O.1. Input and Output Privacy

Intuitively, all parties locally blind/encrypt all (set element related) messages before they send them the smart contract $\mathcal{SC}_{\text{JUS}}$. Moreover, the resultant polynomial that $\mathcal{SC}_{\text{JUS}}$ computes is in a blinded form and it reveals only the intersection of the sets after it is unblinded (as discussed in Section 2.8, on page 3).

Hence, (i) no party (including the protocol's participants) can learn other parties input set elements and (ii) the non-participants of the protocol cannot learn anything about the protocol's output, not even the intersection cardinality.

O.2. Strawman Approaches

O.2.1. Relying on a Server-aide PSI. One may be tempted to replace $Justitia$ with a scheme in which all clients send their encrypted sets to a server (potentially semi-honest and plays Aud's role) which computes the result in a privacy-preserving manner. We highlight that the main difference is that in this (hypothetical) scheme the server is *always involved*; whereas, in our protocol, Aud remains offline as long as the clients behave honestly and it is invoked only when the contract detects misbehaviors.

O.2.2. Charging the Buyer a Flat Fee. One might wish to incorporate a straightforward payment mechanism into an existing multi-party PSI such that a buyer is always required to pay a fixed amount, which could depend on factors such as the total number of clients and the minimum size of the sets. However, this approach presents challenges, because:

1) it compels the buyer to make payments even if certain malicious clients misbehave during the protocol execution and compromise the accuracy of the results. This situation could enable malicious clients to gain access to the result at the buyer's expense without allowing the buyer to obtain the correct result. Unfortunately, there is no fair multi-party PSI in the existing literature that guarantees either all parties learn the correct result or none of them do.
2) it compels the buyer to make payments regardless of the precise size of the intersection. If the buyer ends up paying more than the amount it would have paid for the exact cardinality of the intersection, it might become discouraged from participating in the protocol altogether. Conversely, if the buyer has to pay less than what it would have paid for the size of the intersection, it may discourage other clients from participating in the protocol.

O.3. Error Probability

Recall that in JUS, during step 5, each client C needs to ensure that polynomials $\boldsymbol{\omega}^{(C,D)} \cdot \boldsymbol{\pi}^{(C)}$ and $\boldsymbol{\rho}^{(C,D)}$ do not contain any zero coefficient. This check ensures that client C (the receiver in VOPR) does not insert any zero values to VOPR, especially within OLE^{+}, which is a subroutine of VOPR. If a zero value is inserted in VOPR, an honest receiver will only learn a random value, and more importantly they cannot successfully pass VOPR's verification phase. However, this check can be omitted from 5, if we permit JUS to produce an error with a very low probability.[4] In the remainder of this section, we demonstrate that this probability is negligible with regard to the security parameter.

4. By error we mean even if all parties are honest, VOPR halts when an honest party inserts 0 to it.

Initially, we focus on the product $\boldsymbol{\omega}^{(C,D)} \cdot \boldsymbol{\pi}^{(C)}$. We know that $\boldsymbol{\pi}^{(C)}$ takes the form of $\prod_{i=1}^{d}(x - s'_i)$, where s'_i is either a set element s_i or a random value. Thus, all of its coefficients are non-zero. Additionally, the probability that at least one of the coefficients of d-degree random polynomial $\boldsymbol{\omega}^{(C,D)}$ equals 0 is at most $\frac{d+1}{p}$. Below, we formally state this.

Theorem 9. *Let $\boldsymbol{\delta} = \sum_{t=0}^{d} u_t \cdot x^t$, where $u_t \xleftarrow{\$} \mathbb{F}_p$, for all $t, 0 \leq t \leq d$. Then, the probability that at least one of the coefficients equals 0 is at most $\frac{d+1}{p}$, i.e.,*

$$Pr[\exists u_t, u_t = 0] \leq \frac{d+1}{p}$$

Proof. The proof is straightforward. Since $\boldsymbol{\delta}$'s coefficients are picked uniformly at random from $\mathbb{F}_p$, the probability that u_t equals 0 is $\frac{1}{p}$. Since $\boldsymbol{\delta}$ is of degree d, due to the union bound, the probability that at least one of u_ts equals 0 is at most $\frac{d+1}{p}$. □

Next, we demonstrate that the probability of the polynomial $\boldsymbol{\omega}^{(C,D)} \cdot \boldsymbol{\pi}^{(C)}$ having at least one zero coefficient is negligible with respect to the security parameter. We assume polynomial $\boldsymbol{\omega}^{(C,D)}$ has no zero coefficient.

Theorem 10. *Let $\boldsymbol{\alpha} = \sum_{j=0}^{m} a_i \cdot x^i$ and $\boldsymbol{\beta} = \sum_{j=0}^{n} b_j \cdot x^j$, where $a_i \xleftarrow{\$} \mathbb{F}_p$ and $a_i, b_j \neq 0$, for all $i, j, 0 \leq i \leq m$ and $0 \leq j \leq n$. Also, let $\boldsymbol{\gamma} = \boldsymbol{\alpha} \cdot \boldsymbol{\beta} = \sum_{j=0}^{m+n} c_j \cdot x^j$. Then, the probability that at lest one of the coefficients of polynomial $\boldsymbol{\gamma}$ equals 0 is at most $\frac{m+n+1}{p}$, i.e.,*

$$Pr[\exists c_j, c_j = 0] \leq \frac{m+n+1}{p}$$

Proof. Each coefficient c_k of $\boldsymbol{\gamma}$ can be defined as $c_k = \sum_{\substack{j=0 \\ i=0}}^{\substack{i=m \\ j=n}} a_i \cdot b_j$, where $i + j = k$. We can rewrite c_k as $c_k = a_w \cdot b_z + \sum_{\substack{j=0, j\neq z \\ i=0, i\neq w}}^{\substack{i=m \\ j=n}} a_i \cdot b_j$, where $w + z = i + j = k$. We consider two cases for each c_k:

- Case 1: $\sum_{\substack{j=0, j\neq z \\ i=0, i\neq w}}^{\substack{i=m \\ j=n}} a_i \cdot b_j = 0$. This is a trivial case because with the probability of 1 it holds that $c_k = a_w \cdot b_z \neq 0$. Because by definition $a_w, b_z \neq 0$ and $\mathbb{F}_p$ is an integral domain.
- Case 2: $q = \sum_{\substack{j=0, j\neq z \\ i=0, i\neq w}}^{\substack{i=m \\ j=n}} a_i \cdot b_j \neq 0$. In this case, for event $c_k = a_w \cdot b_z + q = 0$ to occur, $a_w \cdot b_z$ must equal the additive inverse of q. Since a_w has been uniformly selected at random, the probability of such an event occurring is $\frac{1}{p}$.

The above analysis is for a single c_k. Thus, as a result of applying the union bound, the probability that at least one of the coefficients c_k equals 0 is at most $\sum_{j=0}^{m+n} \frac{1}{p} = \frac{m+n+1}{p}$. □

Next we turn out attention to $\boldsymbol{\rho}^{(C,D)}$. Due to Theorem 9, the probability that at least one of the coefficients of $\boldsymbol{\rho}^{(C,D)}$ equals 0 is at most $\frac{d+1}{p}$. Hence, due to Theorems 9, 10, and union bound, the probability that at least one of the coefficients in $\boldsymbol{\omega}^{(C,D)} \cdot \boldsymbol{\pi}^{(C)}$ and $\boldsymbol{\rho}^{(C,D)}$ equals 0 is at most $\frac{3d+2}{p}$, which is negligible in p.

O.4. Main Challenges that Justitia Overcomes

To design an efficient scheme that realizes $\mathcal{PSI}^{\mathcal{FC}}$, we had to address several key challenges. Below, we outline these challenges.

O.4.1. Keeping Overall Complexities Low. In general, in multi-party PSIs, each client must exchange messages with the other clients and potentially engage in secure computations with them, as seen in [28], [29]. This can lead to communication and computational costs that grow quadratically with the number of clients.

To tackle this challenge, we employed two strategies: (a) allowing one of the clients to act as a dealer, interacting with the remaining clients[5], and (b) implementing a smart contract that serves as a bulletin board for receiving most messages and conducting lightweight computations on the clients' messages. The combination of these approaches ensures that the overall communication and computation remain linear in relation to the number of clients (and the cardinality of sets).

O.4.2. Randomising Input Polynomials.. In multi-party PSIs that utilize the polynomial representation, it is crucial for a client's input polynomial to undergo randomization by another client [25]. To achieve this securely and efficiently, we required the dealer and each client to jointly participate in an instance of VOPR, a protocol we developed in Section 4.

O.4.3. Preserving the Privacy of Outgoing Messages. While the utilization of public smart contracts, such as Ethereum, helps maintain overall complexity low, it introduces another challenge. Specifically, if clients fail to safeguard the privacy of the messages they transmit to the smart contracts, then both other clients (e.g., the dealer) and individuals who are not participants in PSI (i.e., the public) can gain access to the clients' set elements and/or the intersection.

To ensure the efficient protection of each client's messages sent to the contracts from the dealer, we necessitate that the clients, excluding the dealer, participate in ZSPA-A. This protocol allows each client to create a pseudorandom polynomial, which they can employ to obscure their messages. To safeguard the privacy of the intersection from

5. This approach has similarity with the non-secure PSIs in [24].

the public, we require that all clients to run a coin-tossing protocol to reach a consensus on a blinding polynomial. This blinding polynomial will be used to obscure the final result that encodes the intersection on the smart contract.

O.4.4. Ensuring the Correctness of Subroutine Protocols' Outputs. Typically, any MPC protocol designed to withstand active adversaries incorporates a verification mechanism to detect any tampering with message integrity during the protocol's execution. This applies to the subroutine protocols we utilize, namely VOPR and ZSPA-A.

However, relying solely on this type of check is not always adequate. There are situations where the output of one MPC serves as input to another MPC, and it becomes essential to guarantee that the unaltered output of the first MPC is securely passed to the second one. This holds for our PSI's subroutines as well.

To address this challenge, we employ unforgeable polynomials. Specifically, the output of VOPR is an unforgeable polynomial that encodes the actual output. If the adversary tampers with the VOPR's output and later uses it, then a verifier can detect this tampering.

We obtain the same integrity guarantee for the output of ZSPA-A without any additional effort. This is because (i) VOPR is called before ZSPA-A, and (ii) if clients use the unaltered outputs of ZSPA-A, then the final result (i.e., the sum of all clients' messages) will not contain any output of ZSPA-A, as they will cancel each other out. Thus, by verifying the correctness of the final result, one can ensure the correctness of the outputs of VOPR and ZSPA-A, in a single step.

O.4.5. Confidentiality of polynomial ζ. The dealer is the sole entity responsible for selecting and possessing knowledge of the secret polynomial ζ. This polynomial remains confidential until all parties submit their messages to the smart contract. Subsequently, in step 10 on page 8, the dealer transmits ζ to the smart contract. After this juncture, ζ is no longer a secret. However, malicious parties gain no advantage from knowing ζ, as they have already submitted their inputs to the smart contract.

O.5. Concrete Parameters

O.5.1. Hash Table Parameters.. As detailed in Appendix D, a hash table is characterized by several key parameters:

- h: the number of bins.
- d: the maximum size (or capacity) of each bin.
- c: the maximum number of elements that are mapped to the hash table.
- pr: the probability that the number of elements mapped to a bin does not exceed a predefined capacity.

It is important to note that in the context of PSI, the parameter c represents the maximum of the sizes of all sets involved.

The literature, as evidenced in sources such as [55], [29], [56]) has extensively examined the specific parameters of a hash table, even within the context of PSI. For instance, as illustrated in Section 6 and Appendix J.1 of [55], when c falls within the range of $[2^{10},\ 2^{20}]$ and pr is set to 2^{-40}, then d is 100. Furthermore, we have $h \approx \frac{4 \cdot c}{d}$. To provide a concrete value, when $c = 2^{20}$ and $d = 100$, we would have $h = 41943$.

O.5.2. Field Size.. In this paper, all arithmetic operations are defined over a finite field $\mathbb{F}_p$, where $\log_2(p) = \lambda$ represents the security parameter. The outputs of $\mathtt{PRF}(.)$ and $\mathtt{PRP}(.)$ are also of size λ. Concrete value of λ can be set based on the maximum bit size of set elements. For instance, one can choose $\lambda = 60$, when the maximum bit size of set elements is slightly less than 60 or $\lambda = 128$ if it is slightly less than 128.

Appendix P. Security Theorem and Proof of JUS

In this section, we initially present a formal security theorem of JUS. Then, we provide the proof for this theorem, which establishes the security of JUS.

Theorem 11. *Let polynomials $\boldsymbol{\zeta}$, $\boldsymbol{\omega}$, and $\boldsymbol{\gamma}$ be three secret uniformly random polynomials. If $\boldsymbol{\theta} = \boldsymbol{\zeta} \cdot \boldsymbol{\omega} \cdot \boldsymbol{\pi} + \boldsymbol{\gamma} \bmod p$ is an unforgeable polynomial (w.r.t. Theorem 7),* ZSPA-A, VOPR, PRF, *and smart contracts are secure, then* JUS *securely realizes f^{PSI} with Q-fairness (w.r.t. Definition 7) in the presence of $m-1$ active-adversary clients (i.e., A_js) or a passive dealer, auditor, or public.*

Proof. We prove Theorem 11 by considering the case where each party is corrupt, at a time.

Case 1: Corrupt $m-1$ clients in $\{A_1, ..., A_m\}$. Let P' be a set of at most $m-1$ corrupt clients, where $P' \subset \{A_1, ..., A_m\}$. Let set $\hat{P}$ be defined as follows: $\hat{P} = \{A_1, ..., A_m\} \backslash P'$. Moreover, let $\mathsf{Sim}_A^{\mathtt{JUS}}$ be the simulator, which uses a subroutine adversary, $\mathcal{A}$. Below, we explain how $\mathsf{Sim}_A^{\mathtt{JUS}}$ (which receives the input sets of honest dealer D and honest client(s) in $\hat{P}$) works.

1) constructs and deploys a smart contract. It sends the contract's address to $\mathcal{A}$.
2) simulates CT and receives the output value, mk, from its functionality, $f_{\mathtt{CT}}$.
3) simulates ZSPA-A for each bin and receives the output value, (k, g, q), from its functionality, $f^{\mathtt{ZSPA\text{-}A}}$.
4) deposits in the contract the total amount of $(\ddot{y} + \ddot{ch}) \cdot (m - |P'| + 1)$ coins on behalf of D and honest client(s) in $\hat{P}$. It sends to $\mathcal{A}$ the amount deposited in the contract.
5) checks if $\mathcal{A}$ has deposited $(\ddot{y} + \ddot{ch}) \cdot |P'|$ amount. If the check fails, it instructs the ledger to refund the coins that every party deposited and sends message $abort_1$ to TTP (and accordingly to all parties); it outputs whatever $\mathcal{A}$ outputs and then halts.
6) selects a random polynomial $\boldsymbol{\zeta}$ of degree 1, for each bin. Also, $\mathsf{Sim}_A^{\mathtt{JUS}}$, for each client $C \in \{A_1, ..., A_m\}$ allocates to each bin two random polynomials: $(\boldsymbol{\omega}^{(D,C)}, \boldsymbol{\rho}^{(D,C)})$ of

degree d and two random polynomials: ($\boldsymbol{\gamma}^{(D,C)}$, $\boldsymbol{\delta}^{(D,C)}$) of degree $3d+1$. Moreover, $\mathsf{Sim}_A^{\text{JUS}}$ for every honest client $C' \in \hat{P}$, for each bin, picks two random polynomials: ($\boldsymbol{\omega}^{(C',D)}$, $\boldsymbol{\rho}^{(C',D)}$) of degree d.

7) simulates VOPR using inputs $\boldsymbol{\zeta} \cdot \boldsymbol{\omega}^{(D,C)}$ and $\boldsymbol{\gamma}^{(D,C)}$ for each bin. Accordingly, it receives the inputs of clients $C'' \in P'$, i.e., $\boldsymbol{\omega}^{(C'',D)} \cdot \boldsymbol{\pi}^{(C'')}$, from its functionality f^{VOPR}, for each bin.
8) extracts the roots of polynomial $\boldsymbol{\omega}^{(C'',D)} \cdot \boldsymbol{\pi}^{(C'')}$ for each bin and appends those roots that are in the sets universe to a new set $S^{(C'')}$.
9) simulates VOPR again using inputs $\boldsymbol{\zeta} \cdot \boldsymbol{\rho}^{(D,C)} \cdot \boldsymbol{\pi}^{(D)}$ and $\boldsymbol{\delta}^{(D,C)}$, for each bin.
10) sends to TTP the input sets of all parties; namely, (i) client D's input set: $S^{(D)}$, (ii) honest clients' input sets: $S^{(C')}$ for all C' in $\hat{P}$, and (iii) $\mathcal{A}$'s input sets: $S^{(C'')}$, for all C'' in P'. For each bin, it receives the intersection set, $S_{\cap}$, from TTP.
11) represents the intersection set for each bin as a polynomial, $\boldsymbol{\pi}$, as follows: $\boldsymbol{\pi} = \prod_{i=1}^{|S_{\cap}|}(x - s_i)$, where $s_i \in S_{\cap}$.
12) constructs polynomials $\boldsymbol{\theta}_1^{(C')} = \boldsymbol{\zeta} \cdot \boldsymbol{\omega}^{(D,C')} \cdot \boldsymbol{\omega}^{(C',D)} \cdot \boldsymbol{\pi} + \boldsymbol{\gamma}^{(D,C')}$, $\boldsymbol{\theta}_2^{(C')} = \boldsymbol{\zeta} \cdot \boldsymbol{\rho}^{(D,C')} \cdot \boldsymbol{\rho}^{(C',D)} \cdot \boldsymbol{\pi} + \boldsymbol{\delta}^{(D,C')}$, and $\boldsymbol{\nu}^{(C')} = \boldsymbol{\theta}_1^{(C')} + \boldsymbol{\theta}_2^{(C')} + \boldsymbol{\tau}^{(C')}$, for each bin and each honest client $C' \in \hat{P}$, where $\boldsymbol{\tau}^{(C)} = \sum_{i=0}^{3d+2} z_{i,c} \cdot x^i$ and each $z_{i,c}$ is derived from k.
13) sends to $\mathcal{A}$ polynomial $\boldsymbol{\nu}^{(C')}$ for each bin and each client C'.
14) receives $\boldsymbol{\nu}^{(C'')}$ from $\mathcal{A}$, for each bin and each client $C'' \in P'$. It ensures that the output for every C'' has been provided. Otherwise, it halts.
15) if there is any abort within steps 7–14, then it sends $abort_2$ to TTP and instructs the ledger to refund the coins that every party deposited. It outputs whatever $\mathcal{A}$ outputs and then halts.
16) constructs polynomial $\boldsymbol{\nu}^{(D)} = \boldsymbol{\zeta} \cdot \boldsymbol{\omega'}^{(D)} \cdot \boldsymbol{\pi} - \sum_{C=A_1}^{A_m}(\boldsymbol{\gamma}^{(D,C)} + \boldsymbol{\delta}^{(D,C)}) + \boldsymbol{\zeta} \cdot \boldsymbol{\gamma}'$ for each bin, where $\boldsymbol{\omega'}^{(D)}$ is a fresh random polynomial of degree d and $\boldsymbol{\gamma}'$ is a pseudorandom polynomial derived from mk.
17) sends to $\mathcal{A}$ polynomials $\boldsymbol{\nu}^{(D)}$ and $\boldsymbol{\zeta}$ for each bin.
18) given each $\boldsymbol{\nu}^{(C'')}$, computes polynomial $\boldsymbol{\phi}'$ as follows $\boldsymbol{\phi}' = \sum_{\forall C'' \in P'} \boldsymbol{\nu}^{(C'')} - \sum_{\forall C'' \in P'} (\boldsymbol{\gamma}^{(D,C'')} + \boldsymbol{\delta}^{(D,C'')})$, for every bin. Then, $\mathsf{Sim}_A^{\text{JUS}}$ checks whether $\boldsymbol{\zeta}$ divides $\boldsymbol{\phi}'$, for every bin. If the check passes, it sets $Flag = True$. Otherwise, it sets $Flag = False$.
19) if $Flag = True$:
    a) instructs the ledger to send back each party's deposit, i.e., $\ddot{y} + \ddot{ch}$ amount. It sends a message $deliver$ to TTP.
    b) outputs whatever $\mathcal{A}$ outputs and then halts.
20) if $Flag = False$:
    a) receives $|P'|$ keys of the PRF from $\mathcal{A}$, i.e., $\vec{k}' = [k'_1, ..., k'_{|P'|}]$, for every bin.
    b) checks whether the following equation holds: $k'_j = k$, for every $k'_j \in \vec{k}'$. Note that k is the output of $f^{\text{ZSPA-A}}$ generated in step 3. It constructs an empty list L' and appends to it the indices (e.g., j) of the keys that do not pass the above check.
    c) simulates ZSPA-A and receives from $f^{\text{ZSPA-A}}$ the output that contains a vector of random polynomials, $\vec{\mu}'$, for each valid key.
    d) sends to $\mathcal{A}$, the list L' and vector $\vec{\mu}'$, for every bin.
    e) for each bin of client C whose index (or ID) is not in L' computes polynomial $\boldsymbol{\chi}^{(D,C)}$ as follows: $\boldsymbol{\chi}^{(D,C)} = \boldsymbol{\zeta} \cdot \boldsymbol{\eta}^{(D,C)} - (\boldsymbol{\gamma}^{(D,C)} + \boldsymbol{\delta}^{(D,C)})$, where $\boldsymbol{\eta}^{(D,C)}$ is a fresh random polynomial of degree $3d+1$. Note, C includes both honest and corrupt clients, except those clients whose index is in L'. $\mathsf{Sim}_A^{\text{JUS}}$ sends every polynomial $\boldsymbol{\chi}^{(D,C)}$ to $\mathcal{A}$.
    f) given each $\boldsymbol{\nu}^{(C'')}$ (by $\mathcal{A}$ in step 14), computes polynomial $\boldsymbol{\phi}'^{(C'')}$ as follows: $\boldsymbol{\phi}'^{(C'')} = \boldsymbol{\nu}^{(C'')} - \boldsymbol{\gamma}^{(D,C'')} - \boldsymbol{\delta}^{(D,C'')}$, for every bin. Then, $\mathsf{Sim}_A^{\text{JUS}}$ checks whether $\boldsymbol{\zeta}$ divides $\boldsymbol{\phi}'^{(C'')}$, for every bin. It appends the index of those clients that did not pass the above check to a new list, L''. Note that $L' \cap L'' = \perp$.
    g) if L' or L'' is not empty, then it instructs the ledger: (i) to refund the coins of those parties whose index is not in L' and L'', (ii) to retrieve $\ddot{ch}$ amount from the adversary (i.e., one of the parties whose index is in one of the lists) and send the $\ddot{ch}$ amount to the auditor, and (iii) to compensate each honest party (whose index is not in the two lists) $\frac{m' \cdot (\ddot{y} + \ddot{ch}) - \ddot{ch}}{m - m'}$ amount, where $m' = |L'| + |L''|$. Then, it sends message $abort_3$ to TTP.
    h) outputs whatever $\mathcal{A}$ outputs and halts.

Next, we demonstrate that the real and ideal models are computationally indistinguishable, with our initial focus on the adversary's output. In the real and ideal models, the adversary sees the transcripts of ideal calls to f_{CT} as well as this functionality outputs, i.e., mk.

Due to the security of CT (as we are in the f_{CT}-hybrid world), the transcripts of f_{CT} in both models have identical distribution, which also applies to the random output of f_{CT}, namely mk. The same principle holds for the transcripts and outputs (k, g, q) of $f^{\text{ZSPA-A}}$ that the adversary observes in the two models. Also, the deposit amount is identical in both models. Thus, in the case where $abort_1$ is disseminated at this point; the adversary's output distribution in both models is identical.

The adversary also observes (the transcripts and) outputs of ideal calls to f^{VOPR} in both models, i.e., output $(\boldsymbol{\theta}_1^{(C'')} = \boldsymbol{\zeta} \cdot \boldsymbol{\omega}^{(D,C'')} \cdot \boldsymbol{\omega}^{(C'',D)} \cdot \boldsymbol{\pi}^{(C'')} + \boldsymbol{\gamma}^{(D,C'')}, \boldsymbol{\theta}_2^{(C'')} = \boldsymbol{\zeta} \cdot \boldsymbol{\rho}^{(D,C'')} \cdot \boldsymbol{\rho}^{(C'',D)} \cdot \boldsymbol{\pi}^{(D)} + \boldsymbol{\delta}^{(D,C'')})$ for each corrupted client C''. However, due to the security of VOPR, the $\mathcal{A}$'s view, regarding VOPR, in both models have identical distribution. In the real model, the adversary observes the polynomial $\boldsymbol{\nu}^{(C)}$ that each honest client C stores in the smart contract. Nevertheless, this is a blinded polynomial comprising of two uniformly random blinding polynomials (i.e., $\boldsymbol{\gamma}^{(D,C)}$ and $\boldsymbol{\delta}^{(D,C)}$) unknown to the adversary. In the ideal model, $\mathcal{A}$ is given polynomial $\boldsymbol{\nu}^{(C')}$ for each honest client C'. This polynomial has also been blinded via two uniformly

random polynomials (i.e., $\boldsymbol{\gamma}^{(D,C')}$ and $\boldsymbol{\delta}^{(D,C')}$) unknown to $\mathcal{A}$. Thus, $\boldsymbol{\nu}^{(C)}$ in the real model and $\boldsymbol{\nu}^{(C)}$ in the ideal model have identical distributions. As a result, in the case where $abort_2$ is disseminated at this point; the adversary's output distribution in both models is identical.

Furthermore, in the real world, the adversary observes polynomials $\boldsymbol{\zeta}$ and $\boldsymbol{\nu}^{(D)}$ that D stores in the smart contract. Nevertheless, $\boldsymbol{\zeta}$ is a uniformly random polynomial, also polynomial $\boldsymbol{\nu}^{(D)}$ has been blinded; its blinding factors are the additive inverse of the sum of the random polynomials $\boldsymbol{\gamma}^{(D,C)}$ and $\boldsymbol{\delta}^{(D,C)}$ unknown to the adversary, for every client $C \in \{A_1, ..., A_m\}$ and D. In the ideal model, $\mathcal{A}$ is given $\boldsymbol{\zeta}$ and $\boldsymbol{\nu}^{(D)}$, where the former is a random polynomial and the latter is a blinded polynomial that has been blinded with the additive inverse of the sum of random polynomials $\boldsymbol{\gamma}^{(D,C)}$ and $\boldsymbol{\delta}^{(D,C)}$ unknown to it, for all client C. Therefore, $(\boldsymbol{\zeta}, \boldsymbol{\nu}^{(D)})$ in the real model and $(\boldsymbol{\zeta}, \boldsymbol{\nu}^{(D)})$ in the ideal model component-wise have identical distribution.

Additionally, the sum of less than $m+1$ blinded polynomials $\boldsymbol{\nu}^{(A_1)}, ..., \boldsymbol{\nu}^{(A_m)}, \boldsymbol{\nu}^{(D)}$ in the real model has identical distribution to the sum of less than $m+1$ blinded polynomials $\boldsymbol{\nu}^{(A_1)}, ..., \boldsymbol{\nu}^{(A_m)}, \boldsymbol{\nu}^{(D)}$ in the ideal model, as this combination would still be blinded by a set of random blinding polynomials unknown to the adversary. Now we discuss why the two polynomials $\frac{\boldsymbol{\phi}}{\boldsymbol{\zeta}} - \boldsymbol{\gamma}'$ in the real model and $\frac{\boldsymbol{\phi}}{\boldsymbol{\zeta}} - \boldsymbol{\gamma}'$ in the ideal model are indistinguishable. Note that, in this proof, we divide and then subtract polynomials $\boldsymbol{\phi}$ because the adversary already knows (and must know) polynomials $(\boldsymbol{\zeta}, \boldsymbol{\gamma}')$. In the real model, polynomial $\frac{\boldsymbol{\phi}}{\boldsymbol{\zeta}} - \boldsymbol{\gamma}'$ takes the following form:

$$\begin{aligned}\frac{\boldsymbol{\phi}}{\boldsymbol{\zeta}} - \boldsymbol{\gamma}' &= \boldsymbol{\omega}'^{(D)} \cdot \boldsymbol{\pi}^{(D)} + \sum_{C=A_1}^{A_m} (\boldsymbol{\omega}^{(D,C)} \cdot \boldsymbol{\omega}^{(C,D)} \cdot \boldsymbol{\pi}^{(C)}) + \\ &+ \boldsymbol{\pi}^{(D)} \cdot \sum_{C=A_1}^{A_m} (\boldsymbol{\rho}^{(D,C)} \cdot \boldsymbol{\rho}^{(C,D)}) \\ &= \boldsymbol{\mu} \cdot gcd(\boldsymbol{\pi}^{(D)}, \boldsymbol{\pi}^{(A_1)}, ..., \boldsymbol{\pi}^{(A_m)})\end{aligned} \tag{3}$$

In Equation 3, every element of $[\boldsymbol{\omega}'^{(D)}, ..., \boldsymbol{\omega}^{(D,C)}, \boldsymbol{\rho}^{(D,C)}]$ is a uniformly random polynomial for every client $C \in \{A_1, ..., A_m\}$ (including corrupt ones) and client D. This is because it has been chosen by (in this case honest) client D. Thus, as shown in Section 2.8, $\frac{\boldsymbol{\phi}}{\boldsymbol{\zeta}} - \boldsymbol{\gamma}'$ takes the form $\boldsymbol{\mu} \cdot gcd(\boldsymbol{\pi}^{(D)}, \boldsymbol{\pi}^{(A_1)}, ..., \boldsymbol{\pi}^{(A_m)})$, where $\boldsymbol{\mu}$ is a uniformly random polynomial and $gcd(\boldsymbol{\pi}^{(D)}, \boldsymbol{\pi}^{(A_1)}, ..., \boldsymbol{\pi}^{(A_m)})$ represents the intersection of the input sets.

In the ideal model, $\mathcal{A}$ can construct polynomial $\boldsymbol{\phi}$ using its (well-formed) inputs $\boldsymbol{\nu}^{(C'')}$ and polynomials $\boldsymbol{\nu}^{(C')}$ that the simulator has transmitted to it, for all $C' \in \hat{P}$ and all $C'' \in P'$. Thus, in the ideal model, polynomial $\frac{\boldsymbol{\phi}}{\boldsymbol{\zeta}} - \boldsymbol{\gamma}'$ has the following form:

$$\begin{aligned}\frac{\boldsymbol{\phi}}{\boldsymbol{\zeta}} - \boldsymbol{\gamma}' = \boldsymbol{\pi} \cdot \Big(\sum_{\forall C' \in \hat{P}} (\boldsymbol{\omega}^{(D,C')} \cdot \boldsymbol{\omega}^{(C',D)}) + \\ \sum_{\forall C' \in \hat{P}} (\boldsymbol{\rho}^{(D,C')} \cdot \boldsymbol{\rho}^{(C',D)}) \Big) + \Big(\sum_{\forall C'' \in P'} (\boldsymbol{\omega}^{(D,C'')} \cdot \\ \boldsymbol{\omega}^{(C'',D)} \cdot \boldsymbol{\pi}^{(C'')}) + \boldsymbol{\pi}^{(D)} \cdot \sum_{\forall C'' \in P'} (\boldsymbol{\rho}^{(D,C'')} \cdot \\ \cdot \boldsymbol{\rho}^{(C'',D)}) \Big) = \boldsymbol{\mu} \cdot gcd(\boldsymbol{\pi}, \boldsymbol{\pi}^{(D)}, \boldsymbol{\pi}^{(C'')})\end{aligned} \tag{4}$$

In Equation 4, every element of the vector $[\boldsymbol{\omega}^{(D,C')}, \boldsymbol{\omega}^{(D,C'')}, \boldsymbol{\rho}^{(D,C')}, \boldsymbol{\rho}^{(D,C'')}]$ is a uniformly random polynomial for all $C' \in \hat{P}$ and all $C'' \in P'$, because they have been selected by $\mathsf{Sim}_A^{\text{JUS}}$. Therefore, $\frac{\boldsymbol{\phi}}{\boldsymbol{\zeta}} - \boldsymbol{\gamma}'$ takes the form $\boldsymbol{\mu} \cdot gcd(\boldsymbol{\pi}, \boldsymbol{\pi}^{(D)}, \boldsymbol{\pi}^{(C'')})$, such that $\boldsymbol{\mu}$ is a uniformly random polynomial and $gcd(\boldsymbol{\pi}, \boldsymbol{\pi}^{(D)}, \boldsymbol{\pi}^{(C'')})$ represents the intersection of the input sets. We know that $gcd(\boldsymbol{\pi}^{(D)}, \boldsymbol{\pi}^{(A_1)}, ..., \boldsymbol{\pi}^{(A_m)}) = gcd(\boldsymbol{\pi}, \boldsymbol{\pi}^{(D)}, \boldsymbol{\pi}^{(C'')})$, because $\boldsymbol{\pi}$ includes the intersection of all clients' sets. Moreover, $\boldsymbol{\mu}$ has identical distribution in the two models because they are uniformly random polynomials. Thus, $\frac{\boldsymbol{\phi}}{\boldsymbol{\zeta}} - \boldsymbol{\gamma}'$ in the real model and $\frac{\boldsymbol{\phi}}{\boldsymbol{\zeta}} - \boldsymbol{\gamma}'$ in the ideal model are indistinguishable.

Now we focus on the case where $Flag = False$. In the real model, the adversary observes the output of $\texttt{Audit}(.)$ which is a list of indices L and a vector of random polynomials $\vec{\mu}$ picked by an honest auditor. In the ideal model, $\mathcal{A}$ is given a list L' of indices and a vector of random polynomials $\vec{\mu}'$ picked by the simulator. Thus, the pair $(L, \vec{\mu})$ in the real model has identical distribution to the pair $(L', \vec{\mu}')$ in the ideal model.

Moreover, in the real model, the adversary observes each polynomial $\boldsymbol{\chi}^{(D,C)} = \boldsymbol{\zeta} \cdot \boldsymbol{\eta}^{(D,C)} - (\boldsymbol{\gamma}^{(D,C)} + \boldsymbol{\delta}^{(D,C)})$ that D stores in the contract, for each bin and each client C whose index is not in L. This is a blinded polynomial with blinding factor $\boldsymbol{\eta}^{(D,C)}$ which itself is a uniformly random polynomial picked by D. In the ideal model, $\mathcal{A}$ is given a polynomial of the form $\boldsymbol{\chi}^{(D,C)} = \boldsymbol{\zeta} \cdot \boldsymbol{\eta}^{(D,C)} - (\boldsymbol{\gamma}^{(D,C)} + \boldsymbol{\delta}^{(D,C)})$, for each bin and each client C whose index is not in L'. This is also a blinded polynomial whose blinding factor is $\boldsymbol{\eta}^{(D,C)}$ which itself is a random polynomial picked by the simulator. Therefore, $\boldsymbol{\chi}^{(D,C)}$ in the real model has identical distribution to $\boldsymbol{\chi}^{(D,C)}$ in the ideal model.

In the real model, the adversary observes polynomial $\boldsymbol{\iota}^{(C)} = \boldsymbol{\zeta} \cdot (\boldsymbol{\eta}^{(D,C)} + \boldsymbol{\omega}^{(D,C)} \cdot \boldsymbol{\omega}^{(C,D)} \cdot \boldsymbol{\pi}^{(C)} + \boldsymbol{\rho}^{(D,C)} \cdot \boldsymbol{\rho}^{(C,D)} \cdot \boldsymbol{\pi}^{(D)} + \boldsymbol{\xi}^{(C)})$ which is a blinded polynomial whose blinding factor is the sum of the above random polynomials, i.e., $\boldsymbol{\eta}^{(D,C)} + \boldsymbol{\xi}^{(C)}$. In the ideal model, $\mathcal{A}$ already has polynomials $\boldsymbol{\chi}^{(D,C)}, \boldsymbol{\nu}^{(C)}$, and $\boldsymbol{\mu}^{(C)}$, where $\boldsymbol{\mu}^{(C)} \in \vec{\mu}'$. This enables $\mathcal{A}$ to compute $\boldsymbol{\iota}^{(C)} = \boldsymbol{\chi}^{(D,C)} + \boldsymbol{\nu}^{(C)} + \boldsymbol{\mu}^{(C)} = \boldsymbol{\zeta} \cdot (\boldsymbol{\eta}^{(D,C)} + \boldsymbol{\omega}^{(D,C')} \cdot \boldsymbol{\omega}^{(C',D)} \cdot \boldsymbol{\pi} + \boldsymbol{\rho}^{(D,C')} \cdot \boldsymbol{\rho}^{(C',D)} \cdot \boldsymbol{\pi} + \boldsymbol{\xi}^{(C)})$, where $\boldsymbol{\xi}^{(C)}$ is a random blinding polynomial used in $\boldsymbol{\mu}^{(C)}$. Nevertheless, $\boldsymbol{\iota}^{(C)}$ itself is a blinded polynomial whose blinding factor is the sum of random polynomials, i.e., $\boldsymbol{\eta}^{(D,C)} + \boldsymbol{\xi}^{(C)}$. Hence, the distribution of polynomial $\boldsymbol{\iota}^{(C)}$ in the real model and $\boldsymbol{\iota}^{(C)}$ in the ideal model are identical. Moreover, the integer $\ddot{y} +$

$\ddot{ch} + \frac{m' \cdot (\ddot{y} + \ddot{ch}) - \ddot{ch}}{m - m'}$ has identical distribution in both models.

Next, we demonstrate that the honest party aborts with the same probability in the real and ideal models. Due to the security of CT, an honest party (during the execution of CT) aborts with the same probability in both models. In this scenario, the adversary gains no information about the parties' input sets and the intersection of these sets because the parties have not yet transmitted any encoded input sets. The same principle applies to the probability of honest parties aborting during the execution of ZSPA-A. In this case, an aborting adversary also learns nothing about the parties' input set and the sets' intersection.

Since the deposit of all parties is made public, an honest party can check and identify whether all parties have deposited a sufficient amount with the same probability in both models. If parties halt due to insufficient deposits, no one gains knowledge about the parties' input sets or the intersection of the sets because the inputs (representation) have not been transmitted at this stage.

Due to the security of VOPR, honest parties abort with the same probability in both models. In the case of an abort during VOPR execution, the adversary would learn nothing (i) about its counter party' input set due to the security of VOPR, and (ii) about the rest of the honest parties' input sets and the intersection as the other parties' input sets are still blinded by random blinding factors known only to D. In the real model, D can check if all parties provided their encoded inputs, by reading from the smart contract.

The simulator can also perform the same verification to ensure that $\mathcal{A}$ has supplied the encoded inputs of all corrupt parties. Hence, in both models, an honest party would detect the violation of this requirement, which is providing all encoded inputs, with the same probability. Even in this scenario, if an adversary aborts by not providing proof of its encoded inputs, it gains no knowledge about the input sets of honest parties or the intersection, for the previously explained reasons.

In the real model, the smart contract sums every client C's polynomial $\boldsymbol{\nu}^{(C)}$ with each other and with D's polynomial $\boldsymbol{\nu}^{(D)}$, which removes the blinding factors that D initially inserted (during the execution of VOPR). After that, it checks whether the result is divisible by $\boldsymbol{\zeta}$. Due to the following reasons the smart contract can detect if a set of outputs of VOPR has been tampered with, with a probability at least $1 - \epsilon(\lambda)$:

- Theorem 8, i.e., unforgeable polynomials' linear combination.
- The fact that the smart contract is given the random polynomial $\boldsymbol{\zeta}$ in plaintext.
- No party (excluding honest client D) knew anything about $\boldsymbol{\zeta}$ before they send their input to the contract.
- The security of the smart contract, i.e., the adversary cannot influence the correctness of the above verification performed by the contract.

In the ideal model, $\mathsf{Sim}_A^{\text{JUS}}$ also can remove the blinding factors and it knows the random polynomial $\boldsymbol{\zeta}$, unlike the adversary who does not know $\boldsymbol{\zeta}$ when it sends the outputs of VOPR to $\mathsf{Sim}_A^{\text{JUS}}$. So, $\mathsf{Sim}_A^{\text{JUS}}$ can detect when $\mathcal{A}$ modifies a set of the outputs of VOPR that were sent to $\mathsf{Sim}_A^{\text{JUS}}$ with a probability at least $1 - \epsilon(\lambda)$, due to Theorem 8. Hence, the smart contract in the real model and the simulator in the ideal model would abort with a similar probability.

Moreover, due to the security of ZSPA-A, the probability that an invalid key $k_i \in \overrightarrow{k}$ is added to the list L in the real world is similar to the probability that $\mathsf{Sim}_A^{\text{JUS}}$ detects an invalid key $k_i' \in \overrightarrow{k'}$ in the ideal world.

In the real model, when $Flag = False$, the smart contract can identify each ill-structured output of VOPR (i.e., $\boldsymbol{\nu}^{(C)}$) with a probability of at least $1 - \epsilon(\lambda)$ by checking whether $\boldsymbol{\zeta}$ divides $\boldsymbol{\iota}^{(C)}$, for the following reasons: (a) Theorem 7 (i.e., unforgeable polynomial), (b) the fact that the smart contract is given $\boldsymbol{\zeta}$ in plaintext, (c) no party (excluding honest client D) knew anything about $\boldsymbol{\zeta}$ before they send their input to the contract, and (d) the security of the contract.

In the ideal model, when $Flag = False$, given each $\boldsymbol{\nu}^{(C'')}$, $\mathsf{Sim}_A^{\text{JUS}}$ can remove its blinding factors from $\boldsymbol{\nu}^{(C'')}$ which results in $\boldsymbol{\phi}'^{(C'')}$ and then check if $\boldsymbol{\zeta}$ divides $\boldsymbol{\phi}'^{(C'')}$. The simulator can also detect an ill-structured $\boldsymbol{\nu}^{(C'')}$ with a probability of at least $1 - \epsilon(\lambda)$, due to Theorem 7, the fact that the simulator is given $\boldsymbol{\zeta}$ in plaintext, and the adversary is not given any knowledge about $\boldsymbol{\zeta}$ before it sends to the simulator the outputs of VOPR. Hence, the smart contract in the real model and $\mathsf{Sim}_A^{\text{JUS}}$ in the ideal model would detect an ill-structured input of an adversary with the same probability.

Now, we analyze the output of the predicates $(Q^{\text{Init}}, Q^{\text{Del}}, Q^{\text{UF-A}}, Q^{\text{F-A}})$ in the real and ideal models. In the real model, all clients proceed to prepare their input set only if the predefined amount of coins has been deposited by the parties. Otherwise, they will be refunded and the protocol halts. In the ideal model also the simulator proceeds to prepare its inputs only if a sufficient amount of deposit has been put in the contract. Otherwise, it would send message $abort_1$ to TTP. Thus, in both models, the parties proceed to prepare their inputs only if $Q^{\text{Init}}(.) \rightarrow 1$.

In the real model, if there is an abort after the parties ensure that there is enough deposit and before client D provides its encoded input to the contract, then all parties would be able to retrieve their deposit in full. In this case, the aborting adversary would not be able to learn anything about honest parties' input sets, because the parties' input sets are still blinded by random blinding polynomials known only to client D.

In the ideal model, if there is any abort during steps 7–14, then the simulator sends $abort_2$ to TTP and instructs the ledger to refund the coins that every party deposited.

Also, in the case of an abort (within the above two points of time), the auditor is not involved. Thus, in both models, if an abort occurs within the specified time frames, we would have $Q^{\text{F-A}}(.) \rightarrow 1$. In the real model, if $Flag = True$, then all parties would be able to learn the intersection, and the smart contract refunds all parties, i.e., sends each party $\ddot{y} + \ddot{ch}$ amount which is the amount each party initially deposited.

In the ideal model, when $Flag = True$, then $\mathsf{Sim}_A^{\text{JUS}}$ can extract the intersection (by summing the output of VOPR provided by all parties and removing the blinding polynomials) and sends back each party's deposit, i.e., $\ddot{y}+\ddot{ch}$ amount.

Hence, in both models in the case of $Flag = True$, when all of the parties receive the result, we would have $Q^{\text{Del}}(.) \rightarrow 1$. In the real model, when $Flag = False$, only the adversary which might corrupt m' clients would be able to learn the result; in this case, the contract sends (i) $\ddot{ch}$ amount to the auditor, and (ii) $\frac{m' \cdot (\ddot{y}+\ddot{ch})-\ddot{ch}}{m-m'}$ amount as a compensation, to each honest party, in addition to each party's deposit $\ddot{y} + \ddot{ch}$.

In the ideal model, when $Flag = False$, $\mathsf{Sim}_A^{\text{JUS}}$ sends $abort_3$ to TTP and instructs the ledger to distribute the same amount among the auditor (e.g., with address adr_j) and every honest party (e.g., with address adr_i) as the contract does in the real model. Thus, in both models when $Flag = False$, we would have $Q^{\text{UF-A}}(., ., ., ., adr_i) \rightarrow (a = 1, .)$ and $Q^{\text{UF-A}}(., ., ., ., adr_j) \rightarrow (., b = 1)$.

We conclude that the distributions of the joint outputs of the honest client $C \in \hat{P}$, client D, Aud, and the adversary in the real and ideal models are computationally indistinguishable.

Case 2: Corrupt dealer D. In the real execution, the dealer's view is defined as follows:

$$\mathsf{View}_D^{\text{JUS}}\Big(S^{(D)}, (S^{(1)}, ..., S^{(m)})\Big) =$$

$$\{S^{(D)}, adr_{sc}, m \cdot (\ddot{y} + \ddot{ch}), r_D, \mathsf{View}_D^{\text{CT}},$$

$$k, g, q, \mathsf{View}_D^{\text{VOPR}}, \boldsymbol{\nu}^{(A_1)}, ..., \boldsymbol{\nu}^{(A_m)}, S_{\cap}\}$$

where $\mathsf{View}_D^{\text{CT}}$ and $\mathsf{View}_D^{\text{VOPR}}$ refer to the dealer's real-model view during the execution of CT and VOPR respectively. Moreover, r_D is the outcome of internal random coins of client D, and adr_{sc} is the address of contract $\mathcal{SC}_{\text{JUS}}$. The simulator $\mathsf{Sim}_D^{\text{JUS}}$, which receives all parties' input sets, works as follows.

1) receives from the subroutine adversary polynomials: $\boldsymbol{\zeta}, (\boldsymbol{\gamma}^{(A_1)}, \boldsymbol{\delta}^{(A_1)}), ..., (\boldsymbol{\gamma}^{(A_m)}, \boldsymbol{\delta}^{(A_m)}), (\boldsymbol{\omega}'^{(A_1)}, \boldsymbol{\rho}'^{(A_1)}), ..., (\boldsymbol{\omega}'^{(A_m)}, \boldsymbol{\rho}'^{(A_m)})$,
where $deg(\boldsymbol{\gamma}^{(C)}) = deg(\boldsymbol{\delta}^{(C)}) = 3d + 1, deg(\boldsymbol{\omega}'^{(C)}) = deg(\boldsymbol{\rho}'^{(C)}) = d$, and $deg(\boldsymbol{\zeta}) = 1$, and $C \in \{A_1, ..., A_m\}$.
2) generates an empty view. It appends to the view, the input set $S^{(D)}$. It constructs a smart contract and deploys it. Let adr_{sc} be the contract's address. It appends adr_{sc} to the view.
3) appends to the view integer $m \cdot (\ddot{y} + \ddot{ch})$ and coins r'_D chosen uniformly at random.
4) extracts the simulation of CT from CT's simulator for client D. Let $\mathsf{Sim}_D^{\text{CT}}$ be the simulation. It appends $\mathsf{Sim}_D^{\text{CT}}$ to the view.
5) picks a random key, k', and derives pseudorandom values $z'_{i,j}$ from the key (the same way is done in Figure 2). It constructs a Merkle tree on top of all values $z'_{i,j}$. Let g' be the root of the resulting tree. It appends k', g', and $q' = \mathtt{H}(k')$ to the view.
6) invokes VOPR's functionality twice and extracts the simulation of VOPR from VOPR's simulator for client D. Let $\mathsf{Sim}_D^{\text{VOPR}}$ be the simulation. It appends $\mathsf{Sim}_D^{\text{VOPR}}$ to the view.
7) given the parties' input sets, computes a polynomial $\boldsymbol{\pi}$ that represents the intersection of the sets.
8) picks m random polynomials $\boldsymbol{\tau}^{(A_1)}, ..., \boldsymbol{\tau}^{(A_m)}$ of degree $3d + 1$ such that their sum is 0.
9) picks m pairs of random polynomials $(\boldsymbol{\omega}^{(A_1)}, \boldsymbol{\rho}^{(A_1)}), ..., (\boldsymbol{\omega}^{(A_m)}, \boldsymbol{\rho}^{(A_m)})$, where each polynomial is of degree d. Then, $\mathsf{Sim}_D^{\text{JUS}}$ for each client $C \in \{A_1, ..., A_m\}$ computes polynomial:
$$\boldsymbol{\nu}^{(C)} = \boldsymbol{\zeta} \cdot \boldsymbol{\pi} \cdot (\boldsymbol{\omega}^{(C)} \cdot \boldsymbol{\omega}'^{(C)} + \boldsymbol{\rho}^{(C)} \cdot \boldsymbol{\rho}'^{(C)}) + \boldsymbol{\delta}^{(C)} + \boldsymbol{\gamma}^{(C)} + \boldsymbol{\tau}^{(C)}$$
10) appends $\boldsymbol{\nu}^{(A_1)}, ..., \boldsymbol{\nu}^{(A_m)}$ and the intersection of the sets $S'_{\cap}$ to the view.

Next, we will demonstrate that the two views are computationally indistinguishable. D's input $S^{(D)}$ is identical in both models. Therefore, they have identical distributions. Furthermore, the contract's address has the same distribution in both views, the same applies to the integer $m \cdot (\ddot{y} + \ddot{ch})$. Since the real-model semi-honest adversary samples its randomness according to the protocol's description, the random coins in both models (i.e., r_D and r'_D) have identical distributions.

Also, due to the security of CT, $\mathsf{View}_D^{\text{CT}}$ and $\mathsf{Sim}_D^{\text{CT}}$ have identical distributions. Keys k and k' have identical distributions too, because both have been chosen uniformly at random from the same domain.

In the real model, each element of the pair (g, p) is the output of a deterministic function on a random value k. We know that k in the real model has identical distribution to k' in the ideal model, so do the evaluations of deterministic functions (i.e., Merkle tree, H, and PRF) on them. Therefore, each pair (g, q) in the real model component-wise has an identical distribution to each pair (g', q') in the ideal model. Furthermore, due to the security of the VOPR, $\mathsf{View}_D^{\text{VOPR}}$ and $\mathsf{Sim}_D^{\text{VOPR}}$ have identical distributions.

In the real model, each $\boldsymbol{\nu}^{(C)}$ has been blinded by a pseudorandom polynomial (i.e., derived from PRF's output) unknown to client D. In the ideal model, however, each $\boldsymbol{\nu}^{(C)}$ has been blinded by a random polynomial unknown to client D. Due to the security of PRF, its outputs are computationally indistinguishable from truly random values. Therefore, $\boldsymbol{\nu}^{(C)}$ in the real model and $\boldsymbol{\nu}^{(C)}$ in the ideal model are computationally indistinguishable.

Now we focus on the sum of all $\boldsymbol{\nu}^{(C)}$ in the real model and the sum of all $\boldsymbol{\nu}^{(C)}$ in the ideal model, as adding them together would remove the above blinding polynomials that are unknown to client D. Specifically, in the real model, after client D sums all $\boldsymbol{\nu}^{(C)}$ and removes the blinding factors and $\boldsymbol{\zeta}$ that it initially imposed, it would get a polynomial of the form $\hat{\boldsymbol{\phi}} = \frac{\sum_{C=A_1}^{A_m} \boldsymbol{\nu}^{(C)} - \sum_{C=A_1}^{A_m} (\boldsymbol{\gamma}^{(D,C)} + \boldsymbol{\delta}^{(D,C)})}{\boldsymbol{\zeta}} = \sum_{C=A_1}^{A_m} (\boldsymbol{\omega}^{(D,C)} \cdot \boldsymbol{\omega}^{(C,D)} \cdot \boldsymbol{\pi}^{(C)}) + \boldsymbol{\pi}^{(D)} \cdot \sum_{C=A_1}^{A_m} (\boldsymbol{\rho}^{(D,C)} \cdot \boldsymbol{\rho}^{(C,D)})$,

where $\boldsymbol{\omega}^{(C,D)}$ and $\boldsymbol{\rho}^{(C,D)}$ are random polynomials unknown to client D.

In the ideal model, after summing all $\boldsymbol{\nu}^{(C)}$ and removing the random polynomials that it already knows, it would get a polynomial of the following form: $\hat{\phi}' = \frac{\sum_{C=A_1}^{A_m} \boldsymbol{\nu}^{(C)} - \sum_{C=A_1}^{A_m} (\boldsymbol{\gamma}^{(C)} + \boldsymbol{\delta}^{(C)})}{\boldsymbol{\zeta}} = \boldsymbol{\pi} \cdot \sum_{C=A_1}^{A_m} (\boldsymbol{\omega}'^{(C)} \cdot \boldsymbol{\omega}^{(C)}) + (\boldsymbol{\rho}'^{(C)} \cdot \boldsymbol{\rho}^{(C)})$.

As shown in Section 2.8, polynomial $\hat{\phi}$ has the form $\boldsymbol{\mu} \cdot gcd(\boldsymbol{\pi}^{(D)}, \boldsymbol{\pi}^{(A_1)}, ..., \boldsymbol{\pi}^{(A_m)})$, where $\boldsymbol{\mu}$ is a uniformly random polynomial and $gcd(\boldsymbol{\pi}^{(D)}, \boldsymbol{\pi}^{(A_1)}, ..., \boldsymbol{\pi}^{(A_m)})$ represents the intersection of the input sets. Moreover, it is evident that $\hat{\phi}'$ has the form $\boldsymbol{\mu} \cdot \boldsymbol{\pi}$, where $\boldsymbol{\mu}$ is a random polynomial and $\boldsymbol{\pi}$ represents the intersection. We know that both $gcd(\boldsymbol{\pi}^{(D)}, \boldsymbol{\pi}^{(A_1)}, ..., \boldsymbol{\pi}^{(A_m)})$ and $\boldsymbol{\pi}$ represent the same intersection. Furthermore, $\boldsymbol{\mu}$ in the real model and $\boldsymbol{\mu}$ in the ideal model have identical distributions as they are uniformly random polynomials. Thus, two polynomials $\hat{\phi}$ and $\hat{\phi}'$ are indistinguishable. Also, the output $S_{\cap}$ is identical in both views. We conclude that the two views are computationally indistinguishable.

Case 3: Corrupt auditor. In this scenario, we can leverage the proof we have already presented for Case 1 (i.e., $m-1$ client A_js are corrupt), to construct a simulator that produces a view computationally distinguishable from that of the semi-honest auditor in the real model.

The reason is that, in the worst-case scenario where $m-1$ malicious client A_js reveal their input sets and randomness to the auditor, the auditor's view would be similar to the view of these corrupt clients, which we have shown to be indistinguishable. The only extra messages the auditor generates, that a corrupt client A_j would not see in plaintext, are random blinding polynomials $(\boldsymbol{\xi}^{(A_1)}, ..., \boldsymbol{\xi}^{(A_m)})$ generated during the execution of `Audit`(.) of `ZSPA-A`. However, these polynomials have been chosen uniformly at random and independent of the parties' input sets. Thus, if the smart contract detects misbehavior and invokes the auditor, even if $m-1$ corrupt client A_j reveals their input sets, then the auditor cannot learn anything about honest parties' input sets.

Case 4: Corrupt public. In the real model, the view of the public (i.e., non-participants of the protocol) is defined as below:

$$\mathsf{View}_{Pub}^{\text{JUS}}\Big(\perp, S^{(D)}, (S^{(A_1)}, ..., S^{(A_m)})\Big) =$$

$$\{\perp, adr_{sc}, (m+1) \cdot (\ddot{y} + \ddot{ch}), k, g, q, \boldsymbol{\nu}^{(A_1)}, ..., \boldsymbol{\nu}^{(A_m)}, \boldsymbol{\nu}^{(D)}\}$$

Now, we describe how the simulator $\mathsf{Sim}_{Pub}^{\text{JUS}}$ operates.

1) generates an empty view and appends to it an empty symbol, $\perp$. It constructs and deploys a smart contract. It appends the contract's address, adr_{sc} and integer $(m+1) \cdot (\ddot{y} + \ddot{ch})$ to the view.
2) picks a random key, k', and derives pseudorandom values $z'_{i,j}$ from the key, in the same way, done in Figure 2. It constructs a Merkle tree on top of the $z'_{i,j}$ values. Let g' be the root of the resulting tree. It appends k', g', and $q' = \mathtt{H}(k')$ to the view.
3) for each client $C \in \{A_1, ..., A_m\}$ and client D generates a random polynomial of degree $3d+1$ (for each bin), i.e., $\boldsymbol{\nu}^{(A_1)}, ..., \boldsymbol{\nu}^{(A_m)}, \boldsymbol{\nu}^{(D)}$.

Next, we will illustrate that the two views are computationally indistinguishable. In both views, $\perp$ is identical. Also, the contract's addresses (i.e., adr_{sc}) have the same distribution in both views and so has the integer $(m+1) \cdot (\ddot{y} + \ddot{ch})$. Keys k and k' have identical distributions as well because both of them have been picked uniformly at random from the same domain. In the real model, each element of pair (g, p) is the output of a deterministic function on the random key k. We know that k in the real model has an identical distribution to k' in the ideal model, and so do the evaluations of deterministic functions on them. Hence, each pair (g, q) in the real model component-wise has an identical distribution to each pair (g', q') in the ideal model.

In the real model, each polynomial $\boldsymbol{\nu}^{(C)}$ is a blinded polynomial comprising of two uniformly random blinding polynomials (i.e., $\boldsymbol{\gamma}^{(D,C)}$ and $\boldsymbol{\delta}^{(D,C)}$) unknown to the adversary. In the ideal model, each polynomial $\boldsymbol{\nu}^{(C)}$ is a random polynomial; thus, polynomials $\boldsymbol{\nu}^{(A_1)}, ..., \boldsymbol{\nu}^{(A_m)}$ in the real model have identical distribution to polynomials $\boldsymbol{\nu}^{(A_1)}, ..., \boldsymbol{\nu}^{(A_m)}$ in the ideal model. Similarly, polynomial $\boldsymbol{\nu}^{(D)}$ has been blinded in the real model; its blinding factors are the additive inverse of the sum of the random polynomials $\boldsymbol{\gamma}^{(D,C)}$ and $\boldsymbol{\delta}^{(D,C)}$ unknown to the adversary. In the ideal model, polynomial $\boldsymbol{\nu}^{(D)}$ is a uniformly random polynomial; thus, $\boldsymbol{\nu}^{(D)}$ in the real model and $\boldsymbol{\nu}^{(D)}$ in the ideal model have identical distributions.

Moreover, in the real model even though the sum $\boldsymbol{\phi}$ of polynomials $\boldsymbol{\nu}^{(A_1)}, ..., \boldsymbol{\nu}^{(A_m)}, \boldsymbol{\nu}^{(D)}$ would remove some of the blinding random polynomials, it is still a blinded polynomial with a pseudorandom blinding factor $\boldsymbol{\gamma}'$ (derived from the output of `PRF`), unknown to the adversary. In the ideal model, the sum of polynomials $\boldsymbol{\nu}^{(A_1)}, ..., \boldsymbol{\nu}^{(A_m)}$, $\boldsymbol{\nu}^{(D)}$ is also a random polynomial. Thus, the sum of the above polynomials in the real model is computationally indistinguishable from the sum of those polynomials in the ideal model.

We conclude that the two views are computationally indistinguishable. □

Appendix Q. Formal Definitions of Predicates of $\mathcal{PSI}^{\mathcal{FCR}}$

Below, we present the formal definition of predicates $Q_{\mathrm{R}}^{\text{Del}}$ and $Q_{\mathrm{R}}^{\text{UF-A}}$.

Definition 8 ($Q_{\mathrm{R}}^{\text{Del}}$: Delivery-with-Reward predicate)**.** Let $\mathcal{G}$ be a stable ledger, adr_{sc} be smart contract sc's address, $adr_i \in Adr$ be the address of an honest party, $\ddot{x}$ be a fixed amount of coins, and $pram := (\mathcal{G}, adr_{sc}, \ddot{x})$. Let R be a reward function that takes as input the computation result: res, a party's address: adr_i, a reward a party should receive for each unit of revealed information: $\ddot{l}$, and input size:

$inSize$. Then R is defined as follows. If adr_i belongs to a non-buyer, then it returns the total amount that adr_i should be rewarded and if adr_i belongs to a buyer client, then it returns the reward's leftover that the buyer can collect, i.e., $R(res, adr_i, \ddot{l}, inSize) \rightarrow r\ddot{e}w_i$. Then, the delivery with reward predicate $Q_{\text{R}}^{\text{Del}}(pram, adr_i, res, \ddot{l}, inSize)$ returns 1 if adr_i has sent $\ddot{x}$ amount to sc and received at least $\ddot{x}+r\ddot{e}w_i$ amount from it. Else, it returns 0.

Definition 9 ($Q_{\text{R}}^{\text{UF-A}}$: UnFair-Abort-with-Reward predicate)**.** Let
$pram := (\mathcal{G}, adr_{sc}, \ddot{x})$ be the parameters defined above, and $Adr' \subset Adr$ be a set containing honest parties' addresses, $m' = |Adr'|$, and $adr_i \in Adr'$. Let also G be a compensation function that takes as input three parameters $(\ddot{deps}, adr_i, m')$, where $\ddot{deps}$ is the amount of coins that all $m+1$ parties deposit, adr_i is an honest party's address, and $m' = |Adr'|$; it returns the amount of compensation each honest party must receive, i.e., $G(\ddot{deps}, adr_i, m') \rightarrow \ddot{x}_i$. Let R be the reward function defined above, i.e., $R(res, adr_i, \ddot{l}, inSize) \rightarrow r\ddot{e}w_i$, and let $p\hat{r}am := (res, \ddot{l}, inSize)$. Then, predicate $Q_{\text{R}}^{\text{UF-A}}$ is defined as $Q_{\text{R}}^{\text{UF-A}}(pram, p\hat{r}am, G, R, \ddot{deps}, m', adr_i) \rightarrow (a, b)$, where $a = 1$ if adr_i is an honest party's address which has sent $\ddot{x}$ amount to sc and received $\ddot{x} + \ddot{x}_i + r\ddot{e}w_i$ from it, and $b = 1$ if adr_i is an auditor's address which received $\ddot{x}_i$ from sc. Otherwise, $a = b = 0$.

Appendix R.
Formal Definition of $\mathcal{PSI}^{\mathcal{FCR}}$

The formal definition of $\mathcal{PSI}^{\mathcal{FCR}}$ is developed based on the definition of $\mathcal{PSI}^{\mathcal{FC}}$. Nevertheless, in $\mathcal{PSI}^{\mathcal{FCR}}$, we ensure that honest non-buyer clients receive a *reward* for their participation in the protocol and for disclosing a portion of their inputs derived from the outcome. We achieve this through two main upgrades: (i) we enhance Q^{Del} to $Q_{\text{R}}^{\text{Del}}$ to ensure that when honest clients receive the result, an honest non-buyer client receives its deposit back along with a reward, while a buyer client receives its deposit back minus the paid reward and (ii) we upgrade $Q^{\text{UF-A}}$ to $Q_{\text{R}}^{\text{UF-A}}$ to ensure that when an adversary aborts in the protocol unfairly, an honest party receives its deposit back along with a predefined amount of compensation and a reward. Predicates Q^{Init} and $Q^{\text{F-A}}$ remain unchanged. We denote these four predicates as $\bar{Q} := (Q^{\text{Init}}, Q_{\text{R}}^{\text{Del}}, Q_{\text{R}}^{\text{UF-A}}, Q^{\text{F-A}})$.

For formal definitions of $Q_{\text{R}}^{\text{Del}}$ and $Q_{\text{R}}^{\text{UF-A}}$, please refer to Appendix Q. Next, we present the formal definition of $\mathcal{PSI}^{\mathcal{FCR}}$.

Definition 10 ($\mathcal{PSI}^{\mathcal{FCR}}$)**.** Let f^{PSI} be the multi-party PSI functionality defined in Section 3. Protocol Γ realizes f^{PSI} with $\bar{Q}$-fairness-and-reward in the presence of $m-3$ active-adversary clients and two rational clients $A_i s$ or a passive dealer D or auditor Aud, if for every non-uniform PPT adversary $\mathcal{A}$ for the real model, there is a non-uniform PPT simulator Sim for the ideal model, such that for every $I \in \{A_1, ..., A_m, D, Aud\}$, it holds: $\{\mathsf{Ideal}^{\mathcal{W}(f^{\text{PSI}}, \bar{Q})}_{\mathsf{Sim}(z), I}(S_1, \ ..., S_{m+1})\}_{S_1,...,S_{m+1},z} \overset{c}{\equiv} \{\mathsf{Real}^{\Gamma}_{\mathcal{A}(z), I}(S_1, ..., S_{m+1})\}_{S_1,...,S_{m+1},z}$, where z is an auxiliary input given to $\mathcal{A}$ and $\mathcal{W}(f^{\text{PSI}}, \bar{Q})$ is a functionality that wraps f^{PSI} with predicates $\bar{Q} := (Q^{\text{Init}}, Q_{\text{R}}^{\text{Del}}, Q_{\text{R}}^{\text{UF-A}}, Q^{\text{F-A}})$.

Appendix S.
Anesidora's Workflow

Figure 8 outlines the interaction between parties in Anesidora.

Appendix T.
Main Challenges that Anesidora Overcomes

T.1. Rewarding Clients Proportionate to the Intersection Size

In PSIs, the main private information about the clients that is revealed to a result recipient is the private set elements that the clients have in common. Thus, honest clients must receive a reward proportionate to the intersection cardinality, from a buyer. To receive the reward, the clients need to reach a consensus on the intersection cardinality. The naive way to do that is to let every client find the intersection and declare it to the smart contract. Under the assumption that the majority of clients are honest, then the smart contract can reward the honest result recipient (from the buyer's deposit). But, the honest majority assumption is strong in the context of multi-party PSI.

Moreover, this approach requires all clients to extract the intersection, which would increase the overall costs. Some clients may not even be interested in or available to do so. This task could also be conducted by a single entity, such as the dealer; but this approach would introduce a single point of failure and all clients have to depend on this entity. To address these challenges, we allow any two clients to become extractors. Each of them finds and sends to the contract the (encrypted) elements in the intersection. The contract compensates them if it determines their honesty. This approach allows us to avoid (i) relying on the assumption of an honest majority, (ii) requiring all clients to find the intersection, and (iii) depending on a single trusted or semi-honest party to perform the task.

T.2. Dealing with Extractors' Collusion

Introducing two extractors brings about another challenge: the possibility of collusion between them (as well as with the buyer) to present a consistent but erroneous result. This kind of behavior cannot be discerned by a verifier unless the verifier consistently undertakes the delegated task themselves, which would defeat the purpose of delegation. To efficiently confront this challenge, we implement counter-collusion smart contracts (as detailed in Section 2.3) that create distrust among the extractors and provide incentives for them to behave honestly.

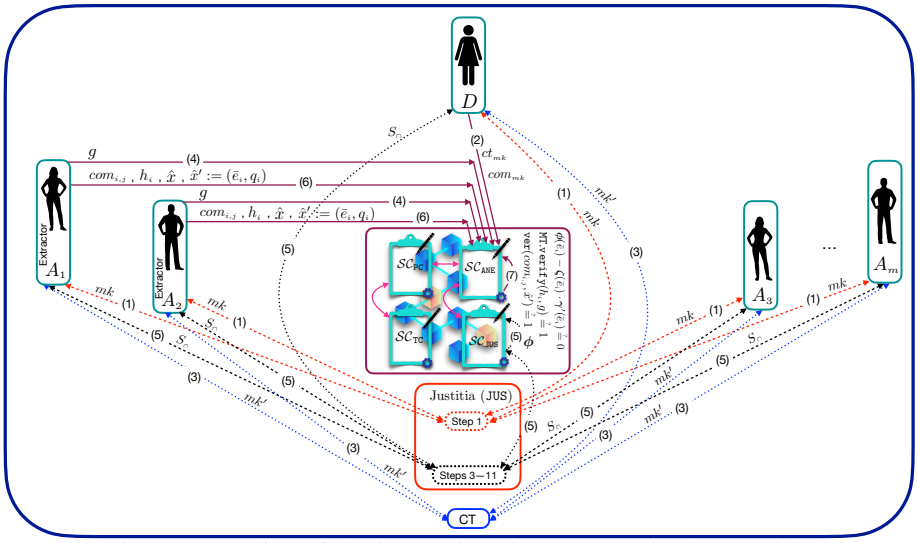


Figure 8: Outline of the interactions between parties in Anesidora.

Appendix U. Proof of ANE

In this section, we present a theorem formally stating the security of ANE and then prove this theorem.

Theorem 12. *If* PRP, PRF, *the commitment scheme, smart contracts, the Merkle tree scheme,* JUS *and the counter-collusion contracts are secure and the public key encryption is semantically secure, then* ANE *realizes* f^{PSI} *with* $\bar{Q}$*-fairness-and-reward (w.r.t. Definition 10) in the presence of* $m-3$ *active-adversary clients* A_j*s and two rational clients* A_i*s or a passive dealer* D *or auditor* Aud*, or public which sees the intersection cardinality.*

Proof. We prove the theorem by considering the case where each party is corrupt, at a time.

Case 1: Corrupt extractors $\{A_1, A_2\}$ **and** $m-3$ **clients in** $\{A_3, ..., A_m\}$. Let set G include extractors $\{A_1, A_2\}$ and a set of at most $m-3$ corrupt clients in $\{A_3, ..., A_m\}$. Let set $\hat{G}$ be a set of honest clients in $\{A_3, ..., A_m\}$. Also, let $\mathsf{Sim}_A^{\text{ANE}}$ be the simulator. We allow the simulator to interact with (i) active adversary $\mathcal{A}'$ that may corrupt $m-3$ clients in $\{A_3, ..., A_m\}$, and (ii) two rational adversaries $\mathcal{A}'' := (\mathcal{A}_1, \mathcal{A}_2)$ that corrupt extractors (A_1, A_2) component-wise. In the simulation, before the point where the extractors are invoked to provide proofs and results, the simulator directly deals with active adversary $\mathcal{A}'$. However, when the extractors are involved (to generate proofs and extract the result) we require the simulator to interact with each rational adversary $\mathcal{A}_1$ and $\mathcal{A}_2$. We allow these two subroutine adversaries $(\mathcal{A}', \mathcal{A}'')$ to internally interact with each other. Now, we explain how $\mathsf{Sim}_A^{\text{ANE}}$, which receives the input sets of honest dealer D and honest client(s) in $\hat{G}$, works.

1) constructs and deploys two smart contracts (for JUS and ANE). It sends the contracts' addresses to $\mathcal{A}'$. It also simulates CT and receives the output value, mk, from its functionality, f_{CT}.
2) deposits $S_{min} \cdot \ddot{v}$ amount to $\mathcal{SC}_{\text{ANE}}$ if buyer A_m is honest, i.e., $A_m \in \hat{G}$. Otherwise, $\mathsf{Sim}_A^{\text{ANE}}$ checks if $\mathcal{A}'$ has deposited $S_{min} \cdot \ddot{v}$ amount in $\mathcal{SC}_{\text{ANE}}$. If the check fails, it instructs the ledger to refund the coins that every party deposited and sends message $abort_1$ to TTP (and accordingly to all parties); it outputs whatever $\mathcal{A}'$ outputs and then halts.
3) constructs and deploys a (Prisoner's) contract and transfers $\ddot{w} = S_{min} \cdot \ddot{r}$ amount for each extractor. $\mathsf{Sim}_A^{\text{ANE}}$ ensures that each extractor deposited $\ddot{d}' = \ddot{d} + S_{min} \cdot \ddot{f}$ coins in this contract; otherwise, it instructs the ledger to refund the coins that every party deposited and sends message $abort_1$ to TTP; it outputs whatever $\mathcal{A}'$ outputs and then halts.
4) encrypts mk under the public key of the dispute resolver; let ct_{mk} be the resulting ciphertext. It also generates a commitment of mk as follows: $com_{mk} = \mathtt{Com}(mk, \mathtt{PRF}(mk, 0))$. It stores ct_{mk} and com_{mk} in $\mathcal{SC}_{\text{ANE}}$.
5) simulates CT again and receives the output value mk' from f_{CT}.
6) receives from $\mathcal{A}'$ a Merkle tree's root g' for each extractor.
7) simulates the steps of 3–11 in JUS. For completeness, we include the steps that the simulator takes in this proof. Specifically, $\mathsf{Sim}_A^{\text{ANE}}$:
   a) simulates ZSPA-A for each bin and receives the output value (k, g, q) from $f^{\text{ZSPA-A}}$.
   b) deposits in the contract the amount of $\ddot{y}' = S_{min} \cdot \ddot{v} + \ddot{ch}$ for client D and each honest client in $\hat{G}$. It sends to $\mathcal{A}'$ the amount deposited in the contract.
   c) checks if $\mathcal{A}'$ has deposited $\ddot{y}' \cdot |G|$ amount (in addition to $\ddot{d}'$ amount deposited in step 3 above). If the check fails, it instructs the ledger to refund the coins

that every party deposited and sends message $abort_1$ to TTP (and accordingly to all parties); it outputs whatever $\mathcal{A}'$ outputs and then halts.

d) picks a random polynomial $\boldsymbol{\zeta}$ of degree 1, for each bin. $\mathsf{Sim}_A^{\texttt{ANE}}$, for each client $C \in \{A_1, ..., A_m\}$ allocates to each bin two degree d random polynomials: ($\boldsymbol{\omega}^{(D,C)}$, $\boldsymbol{\rho}^{(D,C)}$), and two degree $3d+1$ random polynomials: ($\boldsymbol{\gamma}^{(D,C)}$, $\boldsymbol{\delta}^{(D,C)}$). Also, $\mathsf{Sim}_A^{\texttt{ANE}}$ for each honest client $C' \in \hat{G}$, for each bin, picks two degree d random polynomials: ($\boldsymbol{\omega}^{(C',D)}$, $\boldsymbol{\rho}^{(C',D)}$).

e) simulates VOPR using inputs $\boldsymbol{\zeta} \cdot \boldsymbol{\omega}^{(D,C)}$ and $\boldsymbol{\gamma}^{(D,C)}$ for each bin. It receives the inputs of clients $C'' \in G$, i.e., $\boldsymbol{\omega}^{(C'',D)} \cdot \boldsymbol{\pi}^{(C'')}$, from its functionality $f^{\texttt{VOPR}}$, for each bin.

f) extracts the roots of polynomial $\boldsymbol{\omega}^{(C'',D)} \cdot \boldsymbol{\pi}^{(C'')}$ for each bin and appends those roots that are in the sets universe to a new set $S^{(C'')}$.

g) simulates again VOPR using inputs $\boldsymbol{\zeta} \cdot \boldsymbol{\rho}^{(D,C)} \cdot \boldsymbol{\pi}^{(D)}$ and $\boldsymbol{\delta}^{(D,C)}$, for each bin.

h) sends to TTP the input sets of all parties; specifically, (i) client D's input set: $S^{(D)}$, (ii) honest clients' input sets: $S^{(C')}$ for all C' in $\hat{G}$, and (iii) $\mathcal{A}'$'s input sets: $S^{(C'')}$, for all C'' in G. For each bin, it receives the intersection set, $S_\cap$, from TTP.

i) represents the intersection set for each bin as a polynomial, $\boldsymbol{\pi}$, as follows. First, it encrypts each element s_i of $S_\cap$ as $e_i = \texttt{PRP}(mk', s_i)$. Second, it encodes each encrypted element as $\bar{e}_i = e_i || \texttt{H}(e_i)$. Third, it constructs $\boldsymbol{\pi}$ as $\boldsymbol{\pi} = \prod_{i=1}^{|S_\cap|}(x - s_i) \cdot \prod_{j=1}^{d-|S_\cap|}(x - u_j)$, where u_j is a dummy value.

j) constructs polynomials $\boldsymbol{\theta}_1^{(C')} = \boldsymbol{\zeta} \cdot \boldsymbol{\omega}^{(D,C')} \cdot \boldsymbol{\omega}^{(C',D)} \cdot \boldsymbol{\pi} + \boldsymbol{\gamma}^{(D,C')}$, $\boldsymbol{\theta}_2^{(C')} = \boldsymbol{\zeta} \cdot \boldsymbol{\rho}^{(D,C')} \cdot \boldsymbol{\rho}^{(C',D)} \cdot \boldsymbol{\pi} + \boldsymbol{\delta}^{(D,C')}$, and $\boldsymbol{\nu}^{(C')} = \boldsymbol{\theta}_1^{(C')} + \boldsymbol{\theta}_2^{(C')} + \boldsymbol{\tau}^{(C')}$, for each bin and each honest client $C' \in \hat{G}$, such that $\boldsymbol{\tau}^{(C)} = \sum_{i=0}^{3d+2} z_{i,c} \cdot x^i$ and each value $z_{i,c}$ is derived from key k generated in step 7a. It sends to $\mathcal{A}'$ polynomial $\boldsymbol{\nu}^{(C')}$ for each bin and each honest client $C' \in \hat{G}$.

k) receives $\boldsymbol{\nu}^{(C'')}$ from $\mathcal{A}'$, for each bin and each corrupt client $C'' \in G$. It checks whether the output for every C'' has been provided. Otherwise, it halts.

l) if there is any abort within steps 7e–7k, then it sends $abort_2$ to TTP and instructs the ledger to refund the coins that every party deposited. It outputs whatever $\mathcal{A}'$ outputs and then halts.

m) constructs polynomial $\boldsymbol{\nu}^{(D)} = \boldsymbol{\zeta} \cdot \boldsymbol{\omega'}^{(D)} \cdot \boldsymbol{\pi} - \sum_{C=A_1}^{A_m}(\boldsymbol{\gamma}^{(D,C)} + \boldsymbol{\delta}^{(D,C)}) + \boldsymbol{\zeta} \cdot \boldsymbol{\gamma}'$ for each bin on behalf of client D, where $\boldsymbol{\omega'}^{(D)}$ is a fresh random polynomial of degree d and $\boldsymbol{\gamma}'$ is a pseudorandom polynomial derived from mk.

n) sends to $\mathcal{A}'$ polynomials $\boldsymbol{\nu}^{(D)}$ and $\boldsymbol{\zeta}$ for each bin.

o) computes polynomial $\boldsymbol{\phi}'$ as $\boldsymbol{\phi}' = \sum_{\forall C'' \in G} \boldsymbol{\nu}^{(C'')} - \sum_{\forall C'' \in G}(\boldsymbol{\gamma}^{(D,C'')} + \boldsymbol{\delta}^{(D,C'')})$, for every bin. Next, it checks if $\boldsymbol{\zeta}$ divides $\boldsymbol{\phi}'$, for every bin. If the check passes, it sets $Flag = True$. Otherwise, it sets $Flag = False$.

p) if $Flag = True$, then instructs the ledger to send back each party's deposit, i.e., $\ddot{y}'$ amount. It sends a message $deliver$ to TTP. It proceeds to step 8 below.

q) if $Flag = False$:

i) receives $|G|$ keys of the PRF from $\mathcal{A}'$, i.e., $\vec{k}' = [k'_1, ..., k'_{|G|}]$, for every bin.

ii) checks if $k'_j = k$, for every $k'_j \in \vec{k}'$. Recall, k was generated in step 3. It constructs an empty list L' and appends to it the indices (e.g., j) of the keys that do not pass the above check.

iii) receives from $f^{\texttt{ZSPA-A}}$ the output containing a vector of random polynomials, $\vec{\mu}'$, for each valid key.

iv) sends to $\mathcal{A}'$, L' and $\vec{\mu}'$, for every bin.

v) for each bin of client C whose index is not in L' computes polynomial $\boldsymbol{\chi}^{(D,C)}$ as $\boldsymbol{\chi}^{(D,C)} = \boldsymbol{\zeta} \cdot \boldsymbol{\eta}^{(D,C)} - (\boldsymbol{\gamma}^{(D,C)} + \boldsymbol{\delta}^{(D,C)})$, where $\boldsymbol{\eta}^{(D,C)}$ is a fresh random polynomial of degree $3d+1$. Note that C includes both honest and corrupt clients, except those clients whose index is in L'. $\mathsf{Sim}_A^{\texttt{ANE}}$ sends every polynomial $\boldsymbol{\chi}^{(D,C)}$ to $\mathcal{A}'$.

vi) given each $\boldsymbol{\nu}^{(C'')}$ (by $\mathcal{A}'$ in step 7k), computes polynomial $\boldsymbol{\phi}'^{(C'')}$ as follows: $\boldsymbol{\phi}'^{(C'')} = \boldsymbol{\nu}^{(C'')} - \boldsymbol{\gamma}^{(D,C'')} - \boldsymbol{\delta}^{(D,C'')}$, for every bin. $\mathsf{Sim}_A^{\texttt{ANE}}$ checks if $\boldsymbol{\zeta}$ divides $\boldsymbol{\phi}'^{(C'')}$, for every bin. It appends the index of those clients that did not pass the above check to a new list, L''.

vii) if L' or L'' is not empty, then instructs the ledger: (a) to refund $\ddot{y}'$ amount to each client whose index is not in L' and L'', (b) to retrieve $\ddot{ch}$ amount from the adversary (i.e., one of the parties whose index is in one of the lists) and send the $\ddot{ch}$ amount to Aud, and (c) to reward and compensate each honest party (whose index is not in the two lists) $\frac{m' \cdot \ddot{y}' - \ddot{ch}}{m - m'}$ amount, where $m' = |L'| + |L''|$. Then, it sends message $abort_3$ to TTP.

viii) outputs whatever $\mathcal{A}'$ outputs and halts.

8) for each $I \in \{1, 2\}$, receives from $\mathcal{A}_I$ (1) a set $E^{(I)}$ of encoded encrypted elements, e.g., $\bar{e}_i$, in the intersection, (2) each $\bar{e}_i$'s commitment $com_{i,j}$, (3) each $com_{i,j}$'s opening $\hat{x}'$, (4) a proof h_i that each $com_{i,j}$ is a leaf node of a Merkle tree with root g' (given to simulator in step 6 above), and (5) the opening $\hat{x}$ of commitment com_{mk}.

9) encrypts each element s_i of $S_\cap$ as $e_i = \texttt{PRP}(mk', s_i)$. Then, it encodes each encrypted element as $\bar{e}_i = e_i || \texttt{H}(e_i)$. Let set S' include all encoded encrypted elements in the intersection.

10) sings a $\mathcal{SC}_{\text{TC}}$ with $\mathcal{A}_I$, if $\mathcal{A}_I$ decides to be a traitor extractor. In this case, $\mathcal{A}_I$, provides the intersection to $\mathcal{SC}_{\text{TC}}$. $\mathsf{Sim}_A^{\texttt{ANE}}$ checks this intersection's validity. Shortly (in step 16b), we will explain how $\mathsf{Sim}_A^{\texttt{ANE}}$ acts based on the outcome of this check.

11) checks if each set $E^{(I)}$ equals set S'.

12) checks if $com_{i,j}$ matches the opening $\hat{x}'$ and the opening corresponds to a unique element in S'.
13) verifies each commitment's proof, h_i. Specifically, given the proof and root g, it ensures the commitment $com_{i,j}$ is a leaf node of a Merkle tree with a root node g'. It also checks whether the opening $\hat{x}$ matches com_{mk}.
14) if all the checks in steps 11–13 pass, then instructs the ledger (i) to take $|S_\cap| \cdot m \cdot \ddot{l}$ amount from the buyer's deposit and distributes it among all clients, except the buyer, (ii) to return the extractors deposit (i.e., $\ddot{d}'$ amount each) and pay each extractor $|S_\cap| \cdot \ddot{r}$ amount, and (iii) to return $(S_{min} - |S_\cap|) \cdot \ddot{v}$ amount to the buyer.
15) if neither extractor sends the extractor's set intersection $(E^{(\mathcal{A}_1)}, E^{(\mathcal{A}_2)})$ in step 8, then instructs the ledger (i) to refund the buyer, by sending $S_{min} \cdot \ddot{v}$ amount back to the buyer and (ii) to retrieve each extractor's deposit (i.e., $\ddot{d}'$ amount) from $\mathcal{SC}_{\text{PC}}$ and distribute it among the rest of the clients (except the buyer and extractors).
16) if the checks in step 14 fail or in step 15 both $\mathcal{A}_1$ and $\mathcal{A}_2$ send the extractors' set intersection but they are inconsistent with each other, then it tags the extractor whose proof or set intersection was invalid as a misbehaving extractor. $\mathsf{Sim}_A^{\text{ANE}}$ instructs the ledger to pay the auditor (of $\mathcal{SC}_{\text{PC}}$) the total amount of $\ddot{ch}$ coins taken from the misbehaving extractor(s) deposit. Furthermore, $\mathsf{Sim}_A^{\text{ANE}}$ takes the following steps.
    a) if both extractors cheated and there is no traitor, then instructs the ledger (i) to refund the buyer $S_{min} \cdot \ddot{v}$ amount, and (ii) to take $2\ddot{d}' - \ddot{ch}$ amount from the misbehaving extractors' deposit and distribute it to the rest of the clients except the buyer and extractors.
    b) if both extractors cheated, there is a traitor, and the traitor delivered a correct result (in step 10), then instructs the ledger (i) to take $\ddot{d}' - \ddot{d}$ amount from the other misbehaving extractor's deposit and distribute it among the rest of the clients (except the buyer and dishonest extractor), (ii) to distribute $|S_\cap| \cdot \ddot{r} + \ddot{d}' + \ddot{d} - \ddot{ch}$ amount to the traitor, (iii) to refund the traitor $\ddot{ch}$ amount, and (iv) to refund the buyer $S_{min} \cdot \ddot{v} - |S_\cap| \cdot \ddot{r}$ amount.
    c) if both extractors cheated, there is a traitor, and the traitor delivered an incorrect result (in step 10), then instructs the ledger to distribute coins the same way it does in step 16a.
    d) if one of the extractors cheated and there is no traitor, then instructs the ledger (i) to return the honest extractor's deposit (i.e., $\ddot{d}'$ amount), (ii) to pay the honest extractor $|S_\cap| \cdot \ddot{r}$ amount, (iii) to pay this extractor $\ddot{d} - \ddot{ch}$ amount taken from the dishonest extractor's deposit, and (iv) to pay the buyer and the rest of the clients the same way it does in step 16b.
    e) if one of the extractors cheated, there is a traitor, and the traitor delivered a correct result (in step 10), then instructs the ledger (i) to return the other honest extractor's deposit (i.e., $\ddot{d}'$ amount), (ii) to pay the honest extractor $|S_\cap| \cdot \ddot{r}$ amount, taken from the buyer's deposit, (iii) to pay the honest extractor $\ddot{d} - \ddot{ch}$ amount, taken from the traitor's deposit, (iv) to pay to the traitor $|S_\cap| \cdot \ddot{r}$ amount, taken from the buyer's deposit, (v) to refund the traitor $\ddot{d}' - \ddot{d}$ amount, (vi) to refund the traitor $\ddot{ch}$ amount, (vii) to take $|S_\cap| \cdot m \cdot \ddot{l}$ amount from the buyer's deposit and distribute it among all clients, except the buyer, and (viii) to return $(S_{min} - |S_\cap|) \cdot \ddot{v}$ amount back to the buyer.
    f) if one of the extractors cheated, there is a traitor, and the traitor delivered an incorrect result (in step 10), then instructs the ledger (i) to pay the honest extractor the same way it does in step 13(b)iiA, (ii) to refund the traitor $\ddot{ch}$ amount, and (iii) to pay the buyer and the rest of the clients in the same way it does in step 16b.
    g) outputs whatever $\mathcal{A}$ outputs and then halts.

Next, we demonstrate that the real and ideal models are computationally indistinguishable. Initially, we focus on the adversary's output. The addresses of the smart contracts have identical distribution in both models. In the real and ideal models, the adversary sees the transcripts of ideal calls to f_{CT} as well as the functionality outputs (mk, mk'). Due to the security of CT (as we are in the f_{CT}-hybrid world), the transcripts of f_{CT} in both models have identical distribution, so have the random outputs of f_{CT}, i.e., (mk, mk'). Furthermore, the deposit amounts $S_{min} \cdot \ddot{v}$ and $\ddot{w}$ have identical distributions in both models.

Due to the semantical security of the public key encryption, the ciphertext ct_{mk} in the real model is computationally indistinguishable from the ciphertext ct_{mk} in the ideal model. Due to the hiding property of the commitment scheme, commitment com_{mk} in the real model is computationally indistinguishable from commitment com_{mk} in the ideal model.

Furthermore, due to the security of JUS, all transcripts and outputs produced in the ideal model, in step 7 on page 35, have identical distribution to the corresponding transcripts and outputs produced in JUS in the real model. The address of $\mathcal{SC}_{\text{TC}}$ has the same distribution in both models. The amounts each party receives in the real and ideal models are the same, except when both extractors produce an identical and incorrect result (i.e., intersection) in the real model, as we will shortly discuss, this would not occur under the assumption that the extractors are rational and due to the security of the counter collusion smart contracts.

Now, we demonstrate that an honest party aborts with the same probability in the real and ideal models. As before, for the sake of completeness, we include the JUS in the following discussion as well. Due to the security of CT, an honest party, during CT invocation, aborts with the same probability in both models; in this case, the adversary learns nothing about the parties' input set and the sets' intersection as the parties have not sent out any encoded input set yet. In both models, an honest party can read the smart contract and check if sufficient amounts of coins have been deposited.

Thus, it would halt with the same probability in both models. If the parties halt because of insufficient amounts of deposit, no one could learn about (i) the parties' input set and (ii) the sets' intersection because the inputs (representation) have not been dispatched at this point. Due to the security of ZSPA-A, an honest party during ZSPA-A execution aborts with the same probability in both models. In this case, an aborting adversary also learns nothing about the parties' input set and the sets' intersection.

Due to the security of VOPR, honest parties abort with the same probability in both models. In the case where a party aborts during the execution of VOPR, the adversary would learn nothing (i) about its counter party's input set, and (ii) about the rest of the honest parties' input sets and the intersection as the other parties' input sets remain blinded by random blinding factors known only to client D.

In the real model, client D can check if all parties provided their encoded inputs via reading the state of the smart contract. The simulator can perform the same check to ensure $\mathcal{A}'$ has provided the encoded inputs of all corrupt parties. Hence, in both models, an honest party with the same probability detects if not all encoded inputs have been provided. In this case, if an adversary aborts and does not provide its encoded inputs (i.e., polynomials $\boldsymbol{\nu}^{(C'')}$), then it learns nothing about the honest parties' input sets and the intersection, for the same reason explained above.

In the real model, the contract sums every client C's polynomial $\boldsymbol{\nu}^{(C)}$ with each other and with client D's polynomial $\boldsymbol{\nu}^{(D)}$, that ultimately removes the blinding factors that D initially inserted (during the VOPR execution), and then checks if the result is divisible by $\boldsymbol{\zeta}$. Due to the following reasons the contract can detect if a set of outputs of VOPR were tampered with, with a probability at least $1-\epsilon(\lambda)$: (a) Theorem 8, (b) the fact that the smart contract is given the random polynomial $\boldsymbol{\zeta}$ in plaintext, (c) no party (except honest D) knew polynomial $\boldsymbol{\zeta}$ before they send their input to the contract, and (d) the security of the contract, i.e., the adversary cannot influence the correctness of the smart contract's verifications.

In the ideal model, in step 7o on page 36, $\mathsf{Sim}_A^{\text{ANE}}$ can remove the blinding factors and it knows the random polynomial $\boldsymbol{\zeta}$. Therefore, $\mathsf{Sim}_A^{\text{ANE}}$ can detect when $\mathcal{A}'$ tampers with a set of the outputs of VOPR (sent to $\mathsf{Sim}_A^{\text{ANE}}$) with a probability at least $1-\epsilon(\lambda)$, due to Theorem 8. This means that the smart contract in the real model and the simulator in the ideal model would abort with a similar probability.

Due to the security of ZSPA-A, the probability that in the real model an invalid $k_i \in \overrightarrow{k}$ is appended to L is similar to the probability that $\mathsf{Sim}_A^{\text{ANE}}$ detects an invalid $k'_i \in \overrightarrow{k'}$ in the ideal model. In the real model, when $Flag = False$, the smart contract can identify each ill-structured output of VOPR (i.e., $\boldsymbol{\nu}^{(C)}$) with a probability of at least $1-\epsilon(\lambda)$ by checking whether $\boldsymbol{\zeta}$ divides $\boldsymbol{\nu}^{(C)}$, due to (a) Theorem 7 (i.e., unforgeable polynomial), (b) the fact that the smart contract is given $\boldsymbol{\zeta}$ in plaintext, (c) no party (except honest client D) knew anything about $\boldsymbol{\zeta}$ before they send their input to the contract, and (d) the security of the contract.

In the ideal model, when $Flag = False$, given each $\boldsymbol{\nu}^{(C'')}$, $\mathsf{Sim}_A^{\text{ANE}}$ can remove its blinding factors from $\boldsymbol{\nu}^{(C'')}$ which results in $\boldsymbol{\phi}'^{(C'')}$ and then can check if $\boldsymbol{\zeta}$ divides $\boldsymbol{\phi}'^{(C'')}$, in step 7(q)vi on page 36. $\mathsf{Sim}_A^{\text{ANE}}$ can detect an ill-structured $\boldsymbol{\nu}^{(C'')}$ with a probability of at least $1-\epsilon(\lambda)$, due to Theorem 7, the fact that the simulator is given $\boldsymbol{\zeta}$ in plaintext, and the adversary is not given any knowledge about $\boldsymbol{\zeta}$ before it sends to the simulator the outputs of VOPR. Therefore, the smart contract in the real model and $\mathsf{Sim}_A^{\text{ANE}}$ in the ideal model can detect an ill-structured input of an adversary with the same probability.

The smart contract in the real model and $\mathsf{Sim}_A^{\text{ANE}}$ in the ideal model can detect and abort with the same probability if the adversary provides an invalid opening to each commitment $com_{i,j}$ and com_{mk}, due to the binding property of the commitment scheme. Also, the smart contract in the real model and $\mathsf{Sim}_A^{\text{ANE}}$ in the ideal model, can abort with the same probability if a Merkle tree proof is invalid, due to the security of the Merkle tree, i.e., due to the collision resistance of Merkle tree's hash function.

Note that in the ideal model, $\mathsf{Sim}_A^{\text{ANE}}$ can detect and abort with a probability of 1, if $\mathcal{A}_I$ does not send to the simulator all encoded encrypted elements of the intersection, i.e., when $E^{(I)} \neq S'$. Because the simulator already knows all elements in the intersection (and the encryption key). Thus, it can detect with a probability of 1 if both the intersection sets that the extractors provide are identical but incorrect.

In the real world, if the extractors collude with each other and provide identical but incorrect intersections, then an honest client (or the smart contract) cannot detect it. Thus, the adversary can distinguish the two models, based on the probability of aborting. However, under the assumption that the smart contracts of Dong *et al.* [15] are secure (i.e., are counter-collusion), and the extractors are rational, such an event (i.e., providing identical but incorrect result without one extractor betraying the other) would not occur in either model, as the real model and $(\mathcal{A}_1, \mathcal{A}_2)$ rational adversaries follow the strategy that leads to a higher payoff. Specifically, as shown in [15], providing incorrect but identical results is not the preferred strategy of the extractors; instead, the betrayal of one extractor by the other is the most profitable strategy in the case of (enforceable) collusion between the two extractors. This also implies that the amounts that the extractors would receive in both models are identical.

Now, we analyze the output of the predicates $\bar{Q} := (Q^{\text{Init}}, Q_{\text{R}}^{\text{Del}}, Q_{\text{R}}^{\text{UF-A}}, Q^{\text{F-A}})$ in the real and ideal models. In the real model, all clients proceed to prepare their input set only if the predefined amount of coins has been deposited by the parties; otherwise (if in steps 2,4,5 of ANE and step 4 of JUS there is not enough deposit), they will be refunded and the protocol halts. In the ideal model, the simulator proceeds to prepare its inputs only if enough deposit has been placed in the contract. Otherwise, it would send message $abort_1$ to TTP, during steps 2–7c. Thus, in both models, the parties proceed to prepare their inputs only if $Q^{\text{Init}}(.) \rightarrow 1$.

In the real model, if there is an abort after the parties ensure there is enough deposit and before client D provides its encoded input to the contract, then all parties can retrieve

their deposit in full. In this case, the aborting adversary cannot learn anything about honest parties' input sets, as the parties' input sets have been blinded by random blinding polynomials known only to client D. In the ideal model, if there is any abort during steps 7e–7k, then the simulator sends $abort_2$ to TTP and instructs the ledger to refund every party's deposit. In the case of an abort, within the above two steps, the auditor is not involved and paid. Therefore, in both models, in the case of an abort within the above steps, we would have $Q^{\text{F-A}}(.) \rightarrow 1$.

In the real model, if $Flag = True$, then all parties can locally extract the intersection, regardless of the extractors' behavior. In this case, each honest party receives $\ddot{y}'$ amount that it initially deposited in $\mathcal{SC}_{\texttt{JUS}}$. Moreover, each honest party receives *at least* $|S_{\cap}| \cdot \ddot{l}$ amount as a reward, for contributing to the result. In this case, the honest buyer always collects the leftover of its deposit. Specifically, if both extractors act honestly, and the intersection cardinality is smaller than $|S_{min}|$, then the buyer collects its deposit leftover, after paying all honest parties. If any extractor misbehaves, then the honest buyer fully recovers its deposit (and the misbehaving extractor pays the rest).

Even if an extractor misbehaves and subsequently acts as a traitor to rectify its prior misconduct, the buyer recovers the remaining deposit if the intersection's cardinality is less than $|S_{min}|$. In the ideal model, when $Flag = True$, then $\mathsf{Sim}_A^{\texttt{ANE}}$ can extract the intersection by summing the output of VOPR provided by all parties and removing the blinding polynomials. In this case, it sends back each party's deposit placed in $\mathcal{SC}_{\texttt{JUS}}$, i.e., $\ddot{y}'$ amount. Also, in this case, each honest party receives at least $|S_{\cap}| \cdot \ddot{l}$ amount as a reward and the honest buyer always collects the leftover of its deposit. Thus, in both models in the case of $Flag = True$, we would have $Q_{\text{R}}^{\text{Del}}(.) \rightarrow 1$.

In the real model, when $Flag = False$, only the adversary can learn the result. In this case, the contract sends (i) $\ddot{ch}$ amount to Aud, and (ii) $\frac{m' \cdot \ddot{y}' - c\ddot{h}}{m - m'}$ amount, as compensation and reward, to each honest party, in addition to each party's initial deposit. In the ideal model, when $Flag = False$, $\mathsf{Sim}_A^{\texttt{ANE}}$ sends $abort_3$ to TTP and instructs the ledger to distribute the same amount the contract distributes among the auditor (e.g., with address adr_j) and every honest party (e.g., with address adr_i) in the real model. Thus, in both models when $Flag = False$, we would have $Q_{\text{R}}^{\text{UF-A}}(.,.,.,.,adr_i) \rightarrow (a = 1, .)$ and $Q_{\text{R}}^{\text{UF-A}}(.,.,.,.,adr_j) \rightarrow (., b = 1)$.

We conclude that the distribution of the joint outputs of the honest client $C \in \hat{G}$, client D, Aud, and the adversary in the real and ideal models are computationally indistinguishable.

Case 2: Corrupt dealer D. In the real execution, the dealer's view is defined as follows:

$$\mathsf{View}_D^{\texttt{ANE}}\Big(S^{(D)}, (S^{(1)}, ..., S^{(m)})\Big) =$$

$$\{S^{(D)}, adr_{sc}, S_{\text{min}} \cdot \ddot{v}, 2 \cdot \ddot{d}', r_D, \mathsf{View}_D^{\texttt{JUS}}, \mathsf{View}_D^{\texttt{CT}}, (com_{1,j},$$

$$\bar{e}_1, q_1, h_1), \ldots, (com_{sz,j'}, \bar{e}_{sz}, q_{sz}, h_{sz}), g, \hat{x} := (mk, z'), S_{\cap}\}$$

where $\mathsf{View}_D^{\texttt{CT}}$ and $\mathsf{View}_D^{\texttt{VOPR}}$ refer to D's real-model view during the execution of CT and VOPR respectively. Furthermore, r_D is the outcome of internal random coins of D, adr_{sc} is the address of contract, $\mathcal{SC}_{\texttt{ANE}}$, $(j, ..., j') \in \{1, ..., h\}$, $z' = \texttt{PRF}(\bar{mk}, 0)$, $sz = |S_{\cap}|$, and h_i is a Merkle tree proof asserting that $com_{i,j}$ is a leaf node of a Merkle tree with root node g. The simulator $\mathsf{Sim}_D^{\texttt{ANE}}$, which receives all parties' input sets, operates as follows.

1) generates an empty view. It appends to the view, the input set $S^{(D)}$. It constructs and deploys a smart contract. It appends the contract's address, adr_{sc}, to the view.
2) appends to the view integer $S_{min} \cdot \ddot{v}$, and $2 \cdot \ddot{d}'$. Also, it appends uniformly random coins r'_D to the view.
3) extracts the simulation of JUS from JUS's simulator for client D. Let $\mathsf{Sim}_D^{\texttt{JUS}}$ be the simulation, that also includes a random key mk. It appends $\mathsf{Sim}_D^{\texttt{JUS}}$ to the view.
4) extracts the simulation of CT from CT's simulator, yielding the simulation $\mathsf{Sim}_D^{\texttt{CT}}$ that includes its output mk'. It appends $\mathsf{Sim}_D^{\texttt{CT}}$ to the view.
5) encrypts each element $s_{i,j}$ in the intersection set $S_{\cap}$ as $e_{i,j} = \texttt{PRP}(mk', s_{i,j})$ and then encodes the result as $\bar{e}_{i,j} = e_{i,j}||\texttt{H}(e_{i,j})$. It commits to each encoded value as $com_{i,j} = \texttt{Com}(\bar{e}_{i,j}, q_{i,j})$, where j is the index of the bin to which $\bar{e}_{i,j}$ belongs and $q_{i,j}$ is a random value.
6) constructs $(com'_{1,1}, ..., com'_{d,h})$ where each $com'_{i,j}$ is a value picked uniformly at random from the commitment scheme output range. For every j-th bin, it sets each $com'_{i',j}$ to unique $com_{i,j}$ if value $com_{i,j}$ for j-th bin has been generated in step 5. Otherwise, the original values of $com'_{i',j}$ remain unchanged.
7) constructs a Merkle tree on top of the values $(com'_{1,1}, ..., com'_{d,h})$ generated in step 6. Let g be the resulting tree's root.
8) for each element $s_{i,j}$ in the intersection, it constructs $(com'_{i',j}, \bar{e}_{i,j}, q_{i,j}, h_{i,j})$, where $com'_{i',j}$ is the commitment of $\bar{e}_{i,j}$, $com'_{i',j} \in com$, $(\bar{e}_{i,j}\ q_{i,j})$ is the commitment's opening generated in step 5, and $h_{i,j}$ is a Merkle tree proof asserting that $com'_{i',j}$ is a leaf of a Merkle tree with root g. It appends all $(com'_{i',j}, \bar{e}_{i,j}, q_{i,j}, h_{i,j})$ along with g to the view.
9) generates a commitment to mk as $com_{mk} = \texttt{Com}(mk, z')$ where $z' = \texttt{PRF}(mk, 0)$. It appends (mk, z') and $S_{\cap}$ to the view.

Now, let us delve into the reasons why the two views are computationally indistinguishable. D's input $S^{(D)}$ is identical in both models, so they have identical distributions. The contract's address has the same distribution in both models. The same holds for the integers $S_{min} \cdot \ddot{v}$ and $2 \cdot \ddot{d}'$. Moreover, because the real-model semi-honest adversary samples its randomness according to the protocol's description, the random coins in both models (i.e., r_D and r'_D) have identical distribution. Due to the security of JUS, $\mathsf{View}_D^{\texttt{JUS}}$ and $\mathsf{Sim}_D^{\texttt{JUS}}$ have identical distribution, so do $\mathsf{View}_D^{\texttt{CT}}$ and $\mathsf{Sim}_D^{\texttt{CT}}$ due to the security of CT. The intersection elements in both models have identical distributions and the encryption

scheme is schematically secure. Therefore, each intersection element's representation (i.e., $\bar{e}_i$ in the real model and $\bar{e}_{i,j}$ in the deal model) are computationally indistinguishable. Each q_i in the real model and $q_{i,j}$ in the ideal model have identical distributions as they have been picked uniformly at random. Each commitment $com_{i,j}$ in the real model is computationally indistinguishable from commitment $com'_{i',j}$ in the ideal model.

In the real model, each Merkle tree proof h_i contains two leaf nodes (along with intermediary nodes that are the hash values of a subset of leaf nodes) that are themselves the commitment values. Furthermore, for each h_i, only one of the leaf node's openings (that contains an element in the intersection) is seen by D. The same holds in the ideal model, with the difference that for each Merkle tree proof $h_{i,j}$ the leaf node whose opening is not provided is a random value, instead of an actual commitment.

However, due to the hiding property of the commitment scheme, in the real and ideal models, these two leaf nodes (whose openings are not provided) and accordingly the two proofs are computationally indistinguishable. In both models, values mk and z' are random values, so they have identical distributions. Furthermore, the intersection $S_\cap$ is identical in both models.

Thus, we conclude that the two views are computationally indistinguishable.

Case 3: Corrupt auditor. In this case, the real-model view of the auditor is defined as follows:

$$\mathsf{View}^{\mathtt{ANE}}_{Aud}\Big(\perp, S^{(D)}, (S^{(1)}, ..., S^{(m)})\Big) =$$

$$\{\mathsf{View}^{\mathtt{JUS}}_{Aud}, adr_{sc}, S_{min} \cdot \ddot{v}, 2 \cdot \ddot{d}', (com_{1,j}, \bar{e}_1, q_1, h_1), \ldots,$$
$$(com_{sz,j'}, \bar{e}_{sz}, q_{sz}, h_{sz}), g, \hat{x} := (mk, z')\}$$

Due to the security of JUS, Aud's view $\mathsf{View}^{\mathtt{JUS}}_{Aud}$ during the execution of JUS can be easily simulated. As we have shown in Case 2, for the remaining transcript, its real-model view can be simulated too. There is a difference between Case 3 and Case 2; namely, in the former case, Aud does not have the PRP's key mk' used to encrypt each set element. However, due to the security of PRP, it cannot distinguish each encrypted encoded element from a uniformly random element and cannot distinguish $\mathtt{PRP}(mk', .)$ from a uniform permutation. Therefore, each value $\bar{e}_j$ in the real model has identical distribution to each value $\bar{e}_{i,j}$ (as defined in Case 2) in the ideal model, as both are the outputs of PRP.

We conclude that the two views are computationally indistinguishable.

Case 4: Corrupt public. In the real model, the view of the public (i.e., non-participants of the protocol) is defined as below:

$$\mathsf{View}^{\mathtt{ANE}}_{Pub}\Big(\perp, S^{(D)}, (S^{(A_1)}, ..., S^{(A_m)})\Big) =$$

$$\{\mathsf{View}^{\mathtt{JUS}}_{Pub}, adr_{sc}, S_{min} \cdot \ddot{v}, 2 \cdot \ddot{d}', (com_{1,j}, \bar{e}_1, q_1, h_1), \ldots,$$
$$(com_{sz,j'}, \bar{e}_{sz}, q_{sz}, h_{sz}), g, \hat{x} := (mk, z')\}$$

Due to the security of JUS, the public's view $\mathsf{View}^{\mathtt{JUS}}_{Pub}$ during JUS's execution can be simulated in the same manner as described in Case 4 on page 28. The rest of the public's view overlaps with Aud's view in Case 3, excluding $\mathsf{View}^{\mathtt{JUS}}_{Aud}$. Therefore, we can use the argument provided in Case 3 to show that the rest of the public's view can be simulated. We conclude that the public's real and ideal views are computationally indistinguishable. □

Appendix V. Further Discussion on Anesidora

V.1. Application to Vertical Federated Learning

Federated Learning (FL) is a machine learning framework where multiple parties collaboratively build machine learning models without revealing their sensitive input data to their counterparts [1], [2]. FL can be categorized into three classes, based on how data is partitioned.

Vertical Federated Learning (VFL) is one of the primary classes that has found applications in dealing with crime [57], developing financial risk models [7], or healthcare [58]. VFL refers to the FL setting where datasets distributed among different parties share the same samples while holding different features.

For different parties to identify the common samples (or rows), often VFL schemes use PSIs, before they start developing a global model [1]. Nevertheless, similar to the MPC schemes, VFL schemes assume that parties contribute their private inputs free of charge. The importance of developing an incentivization mechanism for (V)FL has been recognized in the literature [3], [4], [5], [6].

This mechanism gains particular relevance in scenarios where participants may simultaneously function as business rivals. In such cases, each participant may harbor concerns that sharing their private data for training FL models could inadvertently benefit their competitors, particularly the party interested in and receiving the final result.

Moreover, parties engaging in the FL procedure must contend with a substantial overhead that may act as a deterrent to their participation.

Anesidora emerges as a potential solution to the aforementioned research question, offering the capacity to supplant the current PSIs employed in VFL schemes. In this framework, Anesidora assumes a pivotal role in motivating diverse parties to share their private inputs, thereby nurturing collaboration in the creation of global models. This ensures that participants receive fair compensation for their contributions. The fundamental concept is to establish a system where collaborative machine learning efforts are not only secure through privacy-preserving measures but are also actively incentivized, fostering increased involvement and participation.

V.2. Strawman Approach

There is a simpler but more expensive approach to finding the intersection without the participation of the extractors. In this approach, the smart contract identifies the (encoded) elements of the intersection and distributes the parties' deposits based on the number of elements it discovers. While this method is simpler, as it eliminates the need for both (i) the involvement of the extractors and (ii) the three counter-collusion contracts, it comes at a higher cost.

This increased cost arises because the contract itself must factorize the unblinded resulting polynomial and locate the roots, incurring an $O(d^2)$ computational cost for each bin, where d is the size of each bin. In contrast, our proposed approach offloads such computations off-chain, resulting in a lower monetary computation cost.

V.3. Encrypting Set Elements

In ANE, each party initially encrypts its set elements (using deterministic encryption) to preserve the privacy of the elements from non-participants of the protocol, e.g., the public. This allows extractors to prove to the (public) smart contract $\mathcal{SC}_{\texttt{ANE}}$ that they have already committed to the encryption of the elements in the intersection without revealing the actual plaintext elements.

V.4. The Use of the Hash-based Padding

The reason each client uses the hash-based padding to encode each encrypted element e_i as $\bar{e}_i = e_i||\texttt{H}(e_i)$ is to allow the auditor in the counter collusion contracts to find the error-free intersection, without having to access to one of the original (encrypted) sets.

V.5. Extracting the Elements in the Intersection

Compared to JUS, there is a minor difference in finding the result in ANE. Specifically, because in ANE each set element s_i is encoded as (i) $e_i = \texttt{PRP}(mk', s_i)$ and then (ii) $\bar{e}_i = e_i||\texttt{H}(e_i)$ by a client, then when the client wants to find the intersection it needs to first regenerate $\bar{e}_i$ as above and then treat it as a set element to check if $\phi'(\bar{e}_i) = 0$, in step 12(b)iii of JUS.

V.6. The Use of Double-layered Commitments

In ANE, each extractor employs double-layered commitments, which involve initially committing to the encryption of each element and subsequently constructing a Merkle tree on top of all these commitments. This approach is adopted for efficiency and privacy reasons.

Constructing a Merkle tree on top of the commitments enables the extractor to store just a single value in $\mathcal{SC}_{\texttt{ANE}}$ resulting in significantly lower storage costs compared to the alternative scenario where all commitments would need to be stored in $\mathcal{SC}_{\texttt{ANE}}$. Additionally, committing to the encryption of the elements allows the extractor to conceal from other clients the encryption of those elements that do not belong to the intersection. It is important to note that simply encrypting each element would not suffice to protect one client's elements from the other clients, as they all possess knowledge of the decryption key.

V.7. Inserting Garbage Inputs

To potentially increase their reward, malicious clients might be tempted to insert "garbage" elements into their sets, hoping that these spurious elements will appear in the result, thus yielding a greater reward. However, their efforts would be in vain as long as a semi-honest client (such as dealer D) includes genuine set elements. In this scenario, according to the set intersection definition, those garbage elements will not be part of the intersection.

V.8. Inserting All Elements of Set Universe

To maximize its reward, a client may attempt to encompass all elements of the set universe in its input, where the size of the set universe is often much larger than the size of a set. However, in our proposed PSIs, there exists a publicly known upper bound on the number of elements that a set can contain. This upper limit is determined by the hash table parameters (d: a bin's capacity and h: total number of bins) that specify the maximum number of elements (i.e., $h \cdot d$) that can be mapped to the hash table. Thus, by appropriately configuring the hash table parameters, we can effectively address the aforementioned challenge.

V.9. Using Generic Reward Function

In ANE, for the sake of simplicity, we allowed each party to receive a fixed reward, i.e., $\ddot{l}$, for every element it contributes to the intersection. It is possible to make the process more flexible/generic. For instance, we could define a Reward Function RF that takes $\ddot{l}$, an (encoded) set element e_i in the intersection, its distribution/value val_{e_i}, and output a reward rew_{e_i} that each party should receive for contributing that element to the intersection, i.e., $RF(\ddot{l}, e_i, val_{e_i}) \rightarrow rew_{e_i}$.

Appendix W. Proof of Theorem 4

Below, we restate the proof of Theorem 4, taken from [25].

Proof. Let $P = \{p_1, ..., p_t\}$ and $Q = \{q_1, ..., q_{t'}\}$ be the roots of polynomials $\mathbf{p}$ and $\mathbf{q}$ respectively. By the Polynomial Remainder Theorem, polynomials $\mathbf{p}$ and $\mathbf{q}$ can be written as $\mathbf{p}(x) = \mathbf{g}(x) \cdot \prod_{i=1}^{t}(x - p_i)$ and $\mathbf{q}(x) = \mathbf{g}'(x) \cdot \prod_{i=1}^{t'}(x - q_i)$ respectively, where $\mathbf{g}(x)$ has degree $d - t$ and $\mathbf{g}'(x)$ has

degree $d' - t'$. Let the product of the two polynomials be $\mathbf{r}(x) = \mathbf{p}(x) \cdot \mathbf{q}(x)$. For every $p_i \in P$, it holds that $\mathbf{r}(p_i) = 0$. Because (a) there exists no non-constant polynomial in $\mathbb{F}_p[X]$ that has a multiplicative inverse (so it could cancel out factor $(x - p_i)$ of $\mathbf{p}(x)$) and (b) p_i is a root of $\mathbf{p}(x)$. The same argument can be used to show for every $q_i \in Q$, it holds that $\mathbf{r}(q_i) = 0$. Thus, $\mathbf{r}(x)$ preserves roots of both $\mathbf{p}$ and $\mathbf{q}$. Moreover, $\mathbf{r}$ does not have any other roots (than P and Q). In particular, if $\mathbf{r}(\alpha) = 0$, then $\mathbf{p}(\alpha) \cdot \mathbf{q}(\alpha) = 0$. Since there is no non-trivial divisors of zero in $\mathbb{F}_p[X]$ (as it is an integral domain), it must hold that either $\mathbf{p}(\alpha) = 0$ or $\mathbf{q}(\alpha) = 0$. Hence, $\alpha \in P$ or $\alpha \in Q$. $\square$