Proof of Compliance for Anonymous, Unlinkable Messages

Mingxun Zhou Elaine Shi Giulia Fanti

Carnegie Mellon University

Abstract

Anonymous systems are susceptible to malicious activity. For instance, in anonymous payment systems, users may engage in illicit practices like money laundering. Similarly, anonymous federated learning systems decouple user updates to a central machine learning model from the user's identity; malicious users can manipulate their updates to poison the model. Today, compliance with system-generated rules in such systems can be guaranteed at the level of a single message by utilizing Zero-Knowledge Proofs (ZKP). However, it remains unclear how to prove compliance for rules that are defined over a *collection* of a user's messages, without compromising the unlinkability of the messages.

To address this challenge, we propose an efficient protocol called Shuffle-ZKP, which enables users within an unlinkable messaging system to collectively prove their compliance. Our protocol leverages a distributed and private set equality check protocol along with generic Non-Interactive Zero-Knowledge (NIZK) proof systems. We also provide an additional attributing protocol to identify misbehaving users. We theoretically analyze the protocol's correctness and privacy properties; we then implement and test it across multiple use cases. Our empirical results show that in use cases involving thousands of users, each user is able to generate a compliance proof within 0.2-10.6 seconds, depending on the use case, while the additional communication overhead remains under 3KB. Furthermore, the protocol is computationally efficient on the server side; the verification algorithm requires a few seconds to handle thousands of users in all of our use cases.

1 Introduction

Unlinkability of transactions is an essential property of many systems that protect user privacy; here, *unlinkability* means that different transactions originating from the same user cannot be linked to each other.[1] In the financial domain, examples of unlinkable payment systems include anonymous cryptocurrencies (e.g., Zcash [HBH+16]) and certain central-bank digital currency (CBDC) designs [AČE+20, WKDC22, KKS22]. Another example arises in systems that collect and process user data, such as federated learning systems; here, *shuffling* is a process by which users' messages are de-identified by intermediaries. Sender metadata is stripped from each user's data, then raw data is sent to a central server, thus unlinking data from its sender. Shuffling can be a powerful tool in distributed private data analytics, including as an enabler for differentially private (DP) pipelines and as a building block for other cryptographic protocols like Private Information Retrieval and Multi-Party Computation [IKOS06, ZWL+19, AIVG22, BHNS20, fuc22, LCC+21, GDD+21, CJMP21].

While unlinkability has many important benefits, it can also introduce opportunities for abuse. For example, users can submit invalid or illegal transactions or data in an effort to steal money, circumvent regulations [BM19], or poison machine learning models [SKL17]. Hence, an important

[1] There exist stronger forms of unlinkability, but this notion of source unlinkability is the focus of this work. It is formally modeled by a shuffling functionality, defined in §2.2.

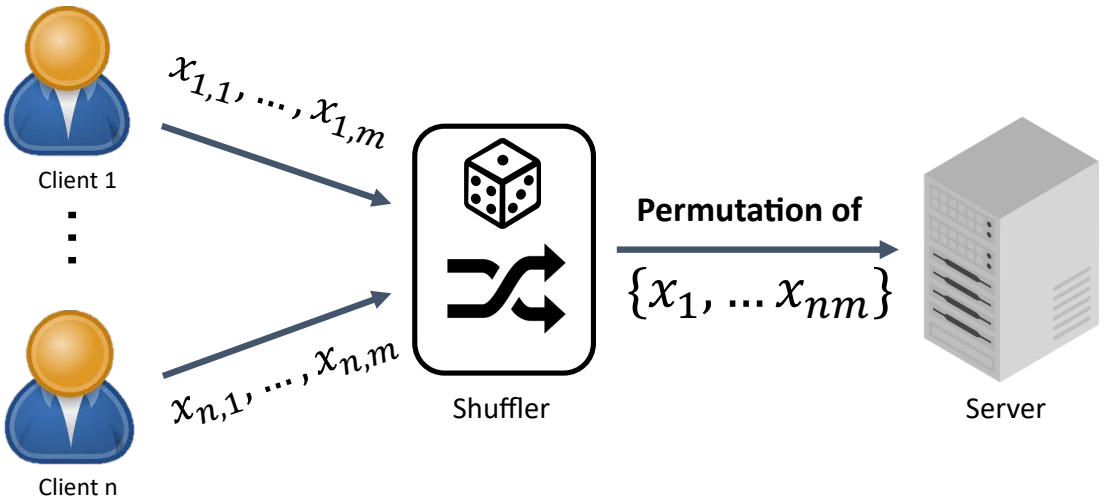

Figure 1: The shuffle model. Clients sends their messages to the shuffler, which randomly permutes the messages and sends them messages to the server.

goal is to **design tools to verify that *all* of a user's (unlinkable) transactions satisfy some desired properties.** For example:

- In an anonymous payment system, clients can make encrypted payments that are unlinkable. The service provider (e.g., validators or auditors) may wish to verify that no single user sends more than \$10,000 to a single recipient within some time frame to comply with anti money laundering (AML) regulations [oT23], without compromising transaction unlinkability.
- In Federated Learning (FL), the central aggregation server may wish to verify that clients' model updates are bounded in norm to prevent them from backdooring or poisoning the model [FCJG20, XHCL19]. In practice, client updates may be split into many shares and anonymized by a shuffler for privacy reasons [GDD+21, BBGN20]. Hence, the bounded-norm condition should hold for all the shares of a client's update *in aggregate*, without compromising the source unlinkability.

More precisely, we consider an abstract model of these examples as follows. Each client $i \in [n]$ has a vector of m messages denoted $\mathbf{x}_i := (x_{i,1}, x_{i,2}, \ldots, x_{i,m})$. Now, every client $i \in [n]$ sends its messages $\mathbf{x}_i$ to the shuffler $\mathcal{S}$ (Figure 1). The shuffler applies a random permutation to the union of all messages, producing an order-insensitive multiset $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n) := \{x_{i,j}\}_{i\in[n],j\in[m]}$. This multiset $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is then forwarded to the untrusted server, who stores the data and/or uses it to perform downstream tasks, such as statistical analysis or machine learning. Here, each user's message $\boldsymbol{x}_i$ are anonymous and unlinkable in the sense that they cannot be linked by the service provider to the message's creator (e.g., linking $x_{i,j}$ to i), or to other transactions by the same creator (e.g., $x_{i,j}$ and $x_{i,k}$ cannot be linked to each other when $j \neq k$). For example, in Zcash, users' transactions are rendered unlinkable through a specific encryption process; several central bank digital currency (CBDC) proposals involve similar methods of processing unlinkable transactions [WKDC22, KKS22].

To prevent abuse, the service provider may require a compliance rule for clients' data (e.g., the sum of financial transaction to a given destination is $\leq \$10,000$), which can be formally represented as a relation $\mathcal{R}$ on the collection of the client's data: $\mathcal{R}(\mathbf{x}_i) = 1$ iff the client's data comply with the rule. However, this is challenging in unlinkable systems.

How can the server verify that all messages from a user are compliant, without breaking unlinkability?

An inefficient strawman solution. This problem can be solved with a large instance of secure multiparty computation (MPC), where each client's anonymous messages are the secret inputs and all clients collaboratively compute a generic NIZK proof. This solution is known as *collaborative ZK*, proposed by Ozdemir and Boneh [OB22] and also by Dayama et al. [DPP+22]. However, this approach is impractical in terms of both computation and communication, as the number of parties and the amount of data can be very large. For example, the witness in this case (i.e., knowledge of the partition of S) is distributed across all n participants, which gives communication (and computation) costs that scale at least as $\Omega(n^2)$ and high concrete costs.

This Paper. We propose the Shuffle-ZKP protocol for privacy-preserving compliance checking in the shuffle model. The protocol has two parts: (1) The server detects if *any* client is non-compliant; (2) If the server detects non-compliance, a second attribution protocol determines which client(s) misbehaved. The main technical challenge is that the clients should provide a *distributed* zero-knowledge argument that the partition of the transaction set S is *collectively* known by the clients; however, clients cannot directly communicate, so each client only sees their local data.

(1) Non-compliance detection: Our key insight for non-compliance detection involves designing a *distributed* and *private* version of a polynomial-based set equality check [CBBZ23, BM14] to ensure consistency between the clients' messages and the server's view on the shuffled sets. Each client masks their polynomial evaluation with masking messages and sends them to the shuffler. The central server receives the union set of masking messages. To break anonymity, the server must find a partition over the (shuffled) set of clients' messages and masking messages, such that each subset in the partition corresponds to a different client's messages and masking messages. By choosing the number of masking messages judiciously, we can prove statistical unlinkability guarantees. Due to an elegant analysis in Balle et al. [BBGN20], each client only needs $O\left(1+\frac{\lambda}{\log n}\right)$ masking messages. Here, λ is the security parameter.

(2) Non-compliance attribution: Our attribution sub-protocol ensures that if a compliance violation was detected in Phase 1, at least one misbehaving client will be identified and no honest client will be blamed, except with negligible probability. Since all clients mask their evaluations with masking messages, the server cannot directly identify a bad client if the joint polynomial evaluation comparison fails. To catch the corrupted clients, we introduce an extra round of an attribution protocol. Each client will privately download membership proofs for their messages and show them to the server with zero-knowledge proofs. We also include a privacy-preserving cross-checking step to identify those corrupted clients who illegally reuse membership proofs.

Our contributions in this work are threefold:

1. We formulate the compliance checking problem for multi-message unlinkable systems, and propose a two-round protocol, Shuffle-ZKP, in the honest-but-curious adversary model with $O\left(1+\frac{\lambda}{\log n}\right)$ per-client communication complexity.[2] We provide an extension that tolerates a malicious server.

2. We provide an additional non-compliance attribution protocol to catch non-compliant clients. The protocol combines efficient membership proofs and a privacy-preserving repeated message detection mechanism.

[2]The common reference string (CRS) is not included here.

3. We implement our protocol on three use cases: a shuffle-DP summation protocol, a secure vector summation protocol and and an AML regulation protocol on a simple anonymous transaction system. In experiments with 1000 users, each client has less than 3 KB communication cost. The per-client computation time ranges between 0.2-10.6s on different tasks and the amortized server time are only a few milliseconds. The implementation can be found at https://github.com/shufflezkp/shuffle-zkp-open.

2 Problem Definition and Use Cases

We first introduce the problem definition, and show multiple use cases for this definition. We consider a setup with an untrusted server, n clients, and a trusted shuffler $\mathcal{S}$, which enables clients to send anonymous messages to the server. Each client $i \in [n]$ has a vector of m messages denoted $\mathbf{x}_i := (x_{i,1}, x_{i,2}, \ldots, x_{i,m})$. Every client $i \in [n]$ sends its messages $\mathbf{x}_i$ to the shuffler $\mathcal{S}$. The shuffler applies a random permutation to the union of all messages, producing an order-insensitive multiset $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n) := \{x_{i,j}\}_{i\in[n],j\in[m]}$. The multiset $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is forwarded to the untrusted server. [3]

Moreover, there is a compliance check over each client's data and the server wants to verify that the data submitted by every client passes the compliance check. We model the compliance check as an NP relation $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$ where $\mathbf{w}_i$ is a private witness known to client i.[4] Then, the server and the clients engage in an interactive protocol in which the clients prove they have a valid input vector $\mathbf{x}_i$ and a valid witness w_i to the relation $\mathcal{R}_i$. If such an interactive protocol satisfies the following properties, it is called a $\mathsf{Shuffle\text{-}ZKP}$ protocol:

- *Completeness:* If all the clients hold a valid $\mathbf{x}_i$ and a w_i that $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$, then the protocol should output 1 except with negligible probability.
- *Soundness:* Even if all clients are malicious, if any client does not hold a valid $\mathbf{x}_i$ or a w_i, then the protocol aborts or outputs 0, except with negligible probability.
- *t-zero-knowledge:* Assuming the server and t out of n clients are corrupted by an adversary, the adversary still learns nothing from the protocol, except that all the honest clients hold valid inputs and witnesses.

If possible, we also aim to achieve:

- *Misbehavior Identifiability:* If the protocol aborts or outputs 0, the server can report at least one misbehaved client, and no honest client will be reported, except with negligible probability.

We formalize these properties in Section 2.2.

2.1 Example Use Cases

Input Validation for Private Data Analytics. Shuffling is an important primitive for private data analytics [IKOS06, BEM+17, CSU+19, EFM+19, FMT22, BBGN20, GGK+21, Wan23, ZSCM23]. We consider summation [IKOS06, BBGN20] and histogram computation [GGK+21], which are used in private telemetry, federated analytics/learning, and anonymous voting.

[3] The server may have side information about the messages. For example, the server may know the order of anonymized transactions due to the ordering in a public ledger. The privacy requirement is that the server cannot learn any additional information at the end of the protocol.

[4] Note that the compliance check $\mathcal{R}_i$ can, in general, be different for each client (e.g., client-specific spending limits).

- *Summation*: Each client holds a secret value x_i and the server wants to learn the sum $\sum_{i\in[n]} x_i$. Ishai et al. [IKOS06] proposed that each client i can additively split its value to a vector of $m = \Omega(\lambda \log n)$ values $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,m})$ such that $x_{i,1} + \cdots + x_{i,m} = x_i$. All the values are randomly permuted by the shuffler and the server sums the shuffled values. The server wants to verify that each client's input value is within a predefined range, L. We define the NP relation $\mathcal{R}(\mathbf{x}_i) = 1$ only if $\sum_{j\in[m]} x_{i,j} \in L$.
- *Histogram*: Each client holds a vector of different item indices $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,m})$ from some domain D and the server wants to compute the frequency for each item in D. Each client sends all the indices to the shuffler and the server computes the frequencies from the shuffled set. The server wants to ensure that no client sends two identical indices to the shuffler, which can bias the outcome (e.g., voting twice for a candidate). We define the local relation to be $\mathcal{R}_i(\mathbf{x}_i) = 1$ iff $x_{i,1}, \ldots, x_{i,m}$ are unique.

Anti-Money Laundering (AML) Compliance in Private Payment Systems. Many private payment system designs, including CBDCs [eur23, KKS22, WKDC22, TBA+22] and private consortium blockchains [CZJ+17], aim to protect user privacy without sacrificing compliance. Suppose each time a user makes a transaction, some metadata (e.g. the hash) is shared with a regulator, which does not learn the sender/receiver ID or the amount. The regulator requires each user (or bank) to prove that they satisfy AML requirements. We define the plaintext transactions of a user to be the witness $w_i = (w_{i,1}, \ldots, w_{i,m})$, whereas the message vector $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,m})$ contains the hashes of the transactions. Define the relation: $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$ only if all the hashes are computed correctly and all the transactions pass the AML compliance rule. This use case can be extended to private consortium blockchains (e.g., [CZJ+17]) where each participating bank provides proofs on behalf of multiple clients.

2.2 Formal Problem Syntax

We now formally define the syntax of a *distributed zero-knowledge proof system for shuffled data*, henceforth denoted Shuffle-ZKP. Recall that each client i's input is $\mathbf{x}_i$ and w_i. The server's input is $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, which is defined as the output of a secure shuffler applied to $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

Secure Shuffler We use the notation $\mathcal{F}_{\text{shuffle}}$ to denote an ideal functionality of a secure shuffler. Assume there are n clients and each client is expected to submit m messages,

- Input: $\mathcal{F}_{\text{shuffle}}$ collects m messages from each client;
- Output: $\mathcal{F}_{\text{shuffle}}$ outputs an order-insensitive set including all messages it received.

Shuffle-ZKP Syntax The parties engage in an interactive protocol, at the end of which the server outputs a bit indicating `accept` or `reject`. We may also allow a setup algorithm.

Throughout the paper, we use λ to denote the security parameter. We assume that the length of each $\mathbf{x}_i$ and w_i are upper bounded by $\mathsf{poly}(\lambda)$ for some fixed polynomial $\mathsf{poly}(\cdot)$, and moreover, the size of the circuit that checks $\mathcal{R}_i(\mathbf{x}_i, w_i)$ is also upper bounded by $\mathsf{poly}(\lambda)$ for some fixed polynomial $\mathsf{poly}(\cdot)$. We also assume that the number of clients n is upper bounded some fixed polynomial in λ.

Definition 2.1 (Shuffle-ZKP)**.** A Shuffle-ZKP protocol for the NP relations $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ involves the a randomized algorithm Setup and an interactive protocol Π between the server and the n clients.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \mathcal{R}_1, \ldots, \mathcal{R}_n)$: outputs the public parameters pp.
- $\{0,1\} \leftarrow \Pi(\mathsf{pp}, \mathbf{x}_1, \ldots, \mathbf{x}_n, w_1, \ldots, w_n)$: the server's input is the $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and pp, and each client i's input is $(\mathbf{x}_i, w_i)$ and pp. At the end of the protocol, the server outputs either 0 or 1.

Security. Henceforth, given $\mathbf{x} := (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, and $\mathbf{w} := (w_1, \ldots, w_n)$, and NP relations $\boldsymbol{\mathcal{R}} := (\mathcal{R}_1, \ldots, \mathcal{R}_n)$, we use the short-hand $\boldsymbol{\mathcal{R}}(\mathbf{x}, \mathbf{w}) = 1$ to mean that for every $i \in [n]$, $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$.

- *Completeness*: Completeness requires that for any $\lambda, n \in \mathbb{N}$, for any $(\mathbf{x}, \mathbf{w})$ such that $\boldsymbol{\mathcal{R}}(\mathbf{x}, \mathbf{w}) = 1$,
$$\Pr\left[\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}}),\ \Pi(\mathsf{pp}, \mathbf{x}, \mathbf{w}) = 1\right] = 1$$
- *Soundness*: For any non-uniform probabilistic polynomial-time (PPT) algorithms $\boldsymbol{\mathcal{P}} = (\mathcal{P}_1, \ldots, \mathcal{P}_n)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $\mathbf{x}$ for which there does not exist a valid $\mathbf{w}$ such that $\boldsymbol{\mathcal{R}}(\mathbf{x}, \mathbf{w}) = 1$,
$$\Pr\left[\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}}),\ \Pi^{\boldsymbol{\mathcal{P}}}(\mathsf{pp}, \mathbf{x}) = 1\right] \leq \mathsf{negl}(\lambda)$$
$\Pi^{\boldsymbol{\mathcal{P}}}(\mathsf{pp}, \mathbf{x})$ denotes an execution of Π in which each client $i \in [n]$ runs the algorithm $\mathcal{P}_i$ that is not necessarily the honest algorithm, and moreover, the honest server's input is $\mathsf{Multiset}(\mathbf{x})$.
- *t-honest-verifier-zero-knowledge*: We say that a $\mathsf{Shuffle\text{-}ZKP}$ scheme for the NP relations $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ satisfies t-honest-verifier-zero-knowledge, iff there exists a PPT simulator Sim such that for any non-uniform PPT adversary $\mathcal{A}$ controlling the server and a coalition $\mathcal{C} \subseteq [n]$ of at most t clients, the outputs of the following two experiments are computationally indistinguishable where $\mathcal{H} := [n] \backslash \mathcal{C}$:
  1. *Real*:
     - $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}})$;
     - $\mathcal{A}(1^\lambda, n, \mathsf{pp})$ outputs $\{\mathbf{x}_i, w_i\}_{i \in [n]}$ as well as some state st, where it is required that $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$ for $i \in [n]$;
     - run the honest protocol Π where the clients' inputs are $\{\mathbf{x}_i, w_i\}_{i \in [n]}$ and pp, and the server's input is $\mathsf{Multiset}(\{\mathbf{x}_i\}_{i \in [n]})$ and pp;
     - output st and the views of $\mathcal{C}$ and the server in Π.
  2. *Ideal*:
     - $(\mathsf{pp}, \tau) \leftarrow \mathsf{Sim}(1^\lambda, n, \boldsymbol{\mathcal{R}})$;
     - $\mathcal{A}(1^\lambda, n, \mathsf{pp})$ outputs $\{\mathbf{x}_i, w_i\}_{i \in [n]}$ and st where it is required that $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$ for $i \in [n]$;
     - run the simulator $\mathsf{view} \leftarrow \mathsf{Sim}(\tau, \mathsf{Multiset}(\{\mathbf{x}_i\}_{i \in \mathcal{H}}), \{\mathbf{x}_i, w_i\}_{i \in \mathcal{C}})$ which is provided with the *multiset* of honest clients' input items $\mathsf{Multiset}(\{\mathbf{x}_i\}_{i \in \mathcal{H}})$, and the inputs and witness of the corrupt clients $\{\mathbf{x}_i, w_i\}_{i \in \mathcal{C}}$);
     - output st and view.
- *t-zero-knowledge*: We say that a $\mathsf{Shuffle\text{-}ZKP}$ scheme for the NP relations $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ satisfies t-zero-knowledge, iff there exists a PPT simulator Sim such that for any non-uniform PPT adversary $\mathcal{A}$ controlling the server and a set of at most t clients denoted $\mathcal{C} \subset [n]$, the adversary's views in the following two experiments are computationally indistinguishable where $\mathcal{H} := [n] \backslash \mathcal{C}$:

1. *Real*:
   - $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}})$;
   - $\mathcal{A}(1^\lambda, n, \mathsf{pp})$ outputs $\{\mathbf{x}_i, w_i\}_{i\in\mathcal{H}}$ where it is required that $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$ for $i \in \mathcal{H}$;
   - run the protocol Π where $\mathcal{A}$ controls $\mathcal{C}$ and the server, and honest clients inputs are $\{\mathbf{x}_i, w_i\}_{i\in\mathcal{H}}$;
2. *Ideal*:
   - $(\mathsf{pp}, \tau) \leftarrow \mathsf{Sim}(1^\lambda, n, \boldsymbol{\mathcal{R}})$;
   - $\mathcal{A}(1^\lambda, n, \mathsf{pp})$ outputs $\{\mathbf{x}_i, w_i\}_{i\in\mathcal{H}}$ where it is required that $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$ for $i \in \mathcal{H}$;
   - $\mathcal{A}$ now interacts with the simulator $\mathsf{Sim}(\tau, \{\mathbf{x}_i\}_{i\in\mathcal{H}})$ which is provided only the *multiset* of the honest clients' input items $\mathsf{Multiset}(\{\mathbf{x}_i\}_{i\in\mathcal{H}})$ but not the witnesses.

Extension: Shuffle-ZKP with misbehavior identifiability. We also define the misbehavior identifiability of Shuffle-ZKP protocol as following. In this case, we augment the protocol's output. First, if the protocol Π aborts, we let the default output as 0. Also, whenever the protocol Π outputs 0, it also outputs a list of indices $\mathcal{I} \subseteq [n]$, indicating the misbehaved clients identified by the server.

We say the protocol satisfies misbehavior identifiability if the following statement holds. Suppose any non-uniform PPT adversary $\mathcal{A}$ controls a set of t clients, denoted $\mathcal{C} \subset [n]$. Denote the honest client set as $\mathcal{H} = [n]/\mathcal{C}$. Assume each honest client $i \in \mathcal{H}$ has a valid input $(\mathbf{x}_i, \mathbf{w}_i)$ such that $\mathcal{R}_i(\mathbf{x}_i, \mathbf{w}_i) = 1$ and denote their inputs as $\mathbf{x}_\mathcal{H}$. $\Pi^{\boldsymbol{\mathcal{A}}}(\mathsf{pp}, \mathbf{x}_\mathcal{H})$ denotes an execution of Π in which $\mathcal{A}$ controls the clients in $\mathcal{C}$ and the honest clients have input $\mathbf{x}_\mathcal{H}$. Then,

$$\Pr\left[\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}}), \Pi^{\boldsymbol{\mathcal{A}}}(\mathsf{pp}, \mathbf{x}_\mathcal{H}) = (0, \mathcal{I}) : |\mathcal{I}| \geq 1 \wedge \mathcal{I} \subseteq \mathcal{C}\right] \geq 1 - \mathsf{negl}(\lambda).$$

3 Overview

We first provide an intuitive overview of the Shuffle-ZKP protocol. The protocol has two phases: (1) **Non-compliance detection**, and (2) **non-compliance attribution**. In Phase 1, clients submit proofs that allow the server to determine if *any* of them has violated the compliance condition. If a violation is detected In Phase 1, Phase 2 is run, in which the server can identify which client was responsible for the violation. The protocol is formally specified in §5 (non-compliance detection) and §6 (non-compliance attribution).

A straightforward idea to detecting and attributing non-compliance would be for every client to directly submit a NIZK proof, which proves that the client holds valid local inputs/messages $\mathbf{x}_i$ and a valid witness w_i to the local relation $\mathcal{R}_i$. However, if we choose the local messages $\mathbf{x}_i$ as *public* witnesses, the unlinkability of those messages will be broken. On the other hand, if we make them *private* witnesses, the client can cheat by replacing the actual messages with arbitrary messages when producing the proof. This violates *consistency* of the messages: that is, we want clients to use the same messages to produce a proof as the ones they send to the shuffler. The challenge remains how to solve the consistency issue efficiently.

3.1 Phase 1: Non-Compliance Detection

Straw Man: Cryptographic accumulators. One natural attempt is to use cryptographic accumulators (e.g. Merkle Trees or RSA accumulators). The server would accumulate all the messages it receives from the shuffler and share the accumulated result to the clients. Then, clients

would need to prove that the messages used in their proving process have valid membership proofs to the accumulator. This solution has two problems: 1) fetching the membership proofs privately incurs significant additional cost; 2) clients could reuse membership proofs and thus still violate consistency.

Proposed approach. Instead, we use polynomial-based set equality checks [CBBZ23, BM14] (also known as a permutation check). Suppose we want to verify the equality of two sets $S_1, S_2 \subseteq \mathbb{F}$. We can construct two polynomials $f(r) = \prod_{x \in S_1}(x - r)$, $g(r) = \prod_{x \in S_2}(x - r)$. When the two sets are equal, the two polynomials are simply equal. When the two sets are not equal, by the Schwartz-Zippel lemma [Sch80], when we pick a random evaluation point r from $\mathbb{F}$, the two polynomials will have the same evaluation only with probability $\frac{\max(|S_1|,|S_2|)}{|\mathbb{F}|}$. As long as the field size is exponentially large, comparing the polynomial evaluations is a reliable way to test set equality.

We are now checking the equality between the message set collected by the server and the set of messages that those clients use to generate their proofs. The server can construct the polynomial itself from the set of messages received from the shuffler. Each client can construct their local polynomials from their local sets of messages and the product of the clients' polynomials will be exactly the polynomial the server constructs due to the commutative property of the field's multiplication. Thus, the clients can first commit to their local $\mathbf{x}_i$, then given a random point, each client provides the evaluation of $\prod_{j \in [m]}(x_{i,j} - r)$ and proves the correctness of the evaluation alongside with the actual compliance check using generic NIZK.

Reduce leakage with dummy messages. Vanilla polynomial checks in our setup leak information. Consider a message set $2, 3, 5, 7$ with the first client contributing $2, 3$ and the second client contributing $5, 7$. If the evaluation point is $r = 0$, the first client's evaluation is $(2 - 0)(3 - 0) = 6$. The server can deduce that 2 and 3 are from the first client, as it is the only combination resulting in an evaluation value of 6. Our idea is to make this "guessing" process exponentially more difficult: each client multiplies d random dummy field elements to its evaluations, and send the d dummies to the shuffler. The server only sees the multiset of nd dummies and is able to compute the product of all clients' dummies, so it can still compute the right product of the clients' polynomial evaluations. If the server wants to break the unlinkability of any client's messages, it needs to find the right set of d dummies out of nd total dummies. Intuitively, this should be hard if d is large enough—there are $\binom{nd}{d}$ possible combinations (more than n^d). Formally, the security of this protocol relies on a key lemma developed by previous split-and-mix shuffle-aggregation protocols [IKOS06, BBGN20] based on random graph theory, which shows that $d = \Omega(1 + \frac{\lambda}{\log n})$ is sufficient to achieve statistical security.

3.2 Phase 2: Non-Compliance Attribution

The server can directly identify clients who submit invalid proofs. However, if the polynomial check fails (consistency issue), attribution is non-trivial because all the clients' evaluations are masked.

We focus on the case where the set of the messages used by the clients during the proof generation process (denoted as S'_x) is not consistent with the message set received by the server during initial message collection (denoted as S_x). If all clients follow the protocol, $S'_x = S_x$. We assume each client submits exactly m messages (otherwise they will be identified by the shuffler), and all messages submitted by the clients are unique (this assumption is not needed in our full protocol design). When $|S_x| = |S'_x|$ and $S'_x \neq S_x$, there are two possibilities:

1. **Case 1: A client submitted a proof over a message that was not in the original set.** I.e., there exists message $x \in S'_x$ s.t. $x \notin S_x$. E.g., $S'_x = \{0, 1, 2, 5\}$ and $S_x = \{0, 1, 2, 3\}$.

2. **Case 2: Some client(s) use a message multiple times in their proofs.** I.e., there are (incorrectly) repeated elements in S'_x. For example, $S'_x = \{0, 1, 2, 2\}$ and $S_x = \{0, 1, 2, 3\}$.

Case 1: We can require clients to prove their messages belong to S_x. For example, the clients can use a Private Information Retrieval (PIR) protocol [CGKS95, CG97] to fetch a Merkle proof for each of its messages. Then, the client provides an additional proof to the server, first proving that the messages used in this extra proof are consistent with the messages used in the detection protocol by checking against the commitment, and also proving the memberships of all those messages. We deliberately avoid membership proofs in the detection protocol to keep overhead low. However, we tolerate these overheads in the attribution protocol, since we assume it is run infrequently.
Case 2: Catching repeated messages is more subtle because the adversary may corrupt multiple clients and only share messages across different clients. For example, client 1 uses messages $\{1, 2, 3\}$ and client 2 uses messages $\{3, 4, 5\}$ to generate their proofs. This requires a cross-checking identification protocol without compromising the unlinkability of the original messages.

Our protocol works as follows. We first augment the original protocol: when collecting the messages $\{x_{i,j}\}_{i\in[n],j\in[m]}$, each client i randomly samples a λ-bit serial number $\mathsf{sx}_{i,j}$ and attaches the commitment of the serial number, denoted as $\widetilde{\mathsf{sx}}_{i,j}$, to each of its message as the form of $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$. Then, in the attribution protocol, we require each client to *directly open all the serial numbers to the server*, and also prove that for each serial number $\mathsf{sx}_{i,j}$, its corresponding message $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ is indeed included in the message set observed by the server. To do that, the client fetches the digest of S_x and a membership proof (a Merkle proof) $\mathsf{mkx}_{i,j}$ for the message $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ upfront.

From the server's perspective, if it sees two repeated serial numbers, it can report those clients who submit those serial numbers because they are using repeated messages to generate proofs. Also, the adversary cannot force an honest client to be reported, because the probability of the adversary successfully guessing the random serial numbers sampled by the honest clients is negligibly small. For the privacy, as long as the commitment scheme is computationally hiding, the server still cannot link the serial numbers submitted by honest clients to any specific messages, so the zero-knowledge property of the protocol is still preserved.

4 Primitives

We use the following primitives in our construction.

Non-Interactive Commitment A non-interactive commitment algorithm $\mathsf{commit}(1^\lambda, x; r)$ takes in a security parameter 1^λ, a message $x \in \{0,1\}^{\ell(\lambda)}$, and a random string $r \in \{0,1\}^\lambda$, and outputs a committed value C. Henceforth let the message length $\ell(\lambda)$ be a polynomial function in λ. We require that a non-interactive commitment scheme satisfy the following properties:

- *Computationally hiding.* For any $x, y \in \{0,1\}^{\ell(\lambda)}$, it must be that $\mathsf{commit}(1^\lambda, x)$ and $\mathsf{commit}(1^\lambda, y)$ are computationally indistinguishable. We write $\mathsf{commit}(1^\lambda, x)$ to denote the randomized algorithm that first samples $r \xleftarrow{\$} \{0,1\}^\lambda$ and then calls $\mathsf{commit}(1^\lambda, x; r)$.

- *Computationally binding.* For any non-uniform P.P.T. $\mathcal{A}$,

$$\Pr[\mathcal{A}(1^\lambda) \to (x_1, r_1, x_2, r_2) : \\ \mathsf{commit}(1^\lambda, x_1; r_1) = \mathsf{commit}(1^\lambda, x_2; r_2)] \leq \mathsf{negl}(\lambda).$$

Non-Interactive Zero-Knowledge Argument A non-interactive argument system for a familiy of NP relations $\{\mathcal{R}_\lambda\}_\lambda$ indexed by λ consists of the following (possibly randomized) algorithms:

- $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$: samples and outputs a common reference string denoted crs.
- $\pi \leftarrow \mathsf{P}(\mathsf{crs}, x, w)$: takes in the common reference string crs, a statement x and a witness w such that $\mathcal{R}_\lambda(x, w) = 1$, outputs a proof π.
- 0 or $1 \leftarrow \mathsf{V}(\mathsf{crs}, x, \pi)$: takes in the common reference string crs, a statement x, and a purported proof π, outputs either 0 or 1 indicating "reject" or "accept".

We require the following properties:

1. *Completeness.* For any λ, for any (x, w) such that $\mathcal{R}_\lambda(x, w) = 1$, it holds that

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), \pi \leftarrow \mathsf{P}(\mathsf{crs}, x, w) : V(\mathsf{crs}, x, \pi) = 1\right] = 1$$

2. *Soundness.* For any non-uniform probabilistic polynomial-time (PPT) prover P^*, there exists a negligible function $\mathsf{negl}(\cdot)$, such that

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), (x, \pi) \leftarrow P^*(\mathsf{crs}) : V(\mathsf{crs}, x, \pi) = 1 \text{ but } x \notin \mathcal{R}_\lambda\right] \leq \mathsf{negl}(\lambda)$$

   In the above, we use $x \notin \mathcal{R}_\lambda$ to mean that there does not exist a w such that $\mathcal{R}_\lambda(x, w) = 1$.

3. *Knowledge soundness.* For any non-uniform deterministic algorithm $\mathcal{A}$, there exist a non-uniform polynomial-time extractor $\mathcal{X}_\mathcal{A}$ and a negligible function $\mathsf{negl}(\cdot)$ such that for any auxiliary string z,

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), ((x, \pi); w) \leftarrow (\mathcal{A}||\mathcal{X}_\mathcal{A})(\mathsf{crs}, z) : \mathsf{V}(\mathsf{crs}, x, \pi) = 1 \wedge \mathcal{R}_\lambda(x, w) = 0\right] \leq \mathsf{negl}(\lambda)$$

4. *Zero-knowledge.* Intuitively, a non-interactive argument system is computationally zero-knowledge if one can simulate the proof of a true statement without knowing the witness. Formally, a non-interactive argument system satisfies adaptive multi-theorem computational zero-knowledge, iff there exists a PPT simulator (S_1, S_2), such that for any non-uniform PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) : \mathcal{A}^{P(\mathsf{crs}, \cdot, \cdot)}(1^\lambda, \mathsf{crs}) = 1\right] \overset{\mathsf{negl}(\lambda)}{\approx} \Pr\left[(\mathsf{crs}, \tau) \leftarrow S_1(1^\lambda) : \mathcal{A}^{\overline{S}(\tau, \cdot, \cdot)}(1^\lambda, \mathsf{crs}) = 1\right]$$

   where τ is a trapdoor, and $\overline{S}(\tau, x, w)$ is the following oracle: upon receiving (τ, x, w), it checks whether $\mathcal{R}_\lambda(x, w) = 1$. If so, output $S_2(\tau, x)$, which simulates a proof without knowing the witness; otherwise, output $\perp$. Moreover, the notation $\overset{\mathsf{negl}(\lambda)}{\approx}$ means that the left-hand side and the right-hand side differ by at most $\mathsf{negl}(\lambda)$.

5 Phase 1: Non-Compliance Detection

We now introduce our Shuffle-ZKP non-compliance attribution protocol, which is designed for efficiency. If any client fails the compliance check, the protocol raises an alarm that *at least one* of the clients is non-compliant. This triggers the heavier non-compliance attribution protocol in Section 6.

5.1 Honest-Verifier Shuffle-ZKP Protocol

We start with a Shuffle-ZKP protocol assuming the server is semi-honest (honest-verifier). The pseudocode is presented in Figure 3 and an illustration is in Figure 2.

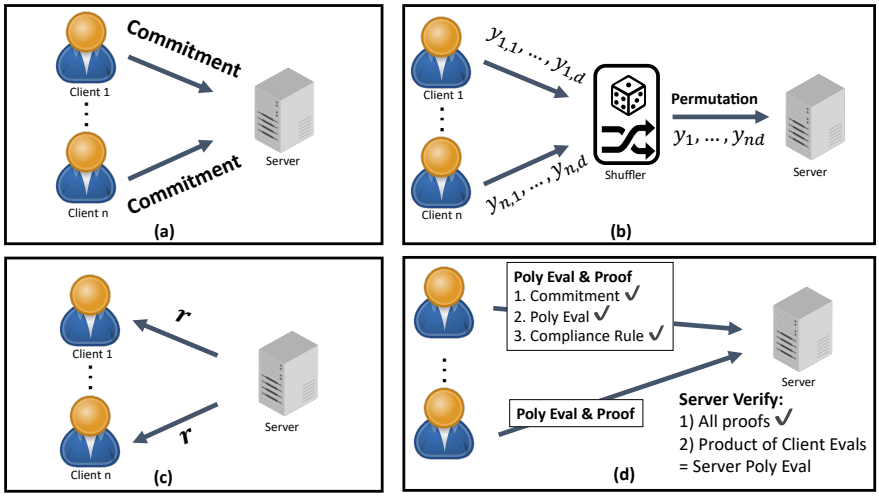

Figure 2: Demostration of our protocol. (a) Step 1: the clients send commitments to the server; (b) Step 2: each client i sends the dummies $y_{i,1}, \ldots, y_{i,d}$ to the shuffler; (c) Step 3: the server broadcasts a challenge r; (d) the client i sends a field element z_i and a proof to the server, proving that correctness of the commitment and the polynomial evaluation $z_i = \prod_{j\in[m]}(x_{i,j} - r)\prod_{j\in[d]} y_{i,j}$, and also the compliance to the rule. The server verifies all the proofs and the correctness of the polynomial evaluation, $\prod_{j\in[nm]}(x_j - r)\prod_{j\in[nd]} y_j = z_1 z_2 \cdots z_n$.

Context. We assume each client i already submits its anonymous messages $x_{i,1}, \ldots, x_{i,m}$ to the shuffler, and the server receives the shuffled message set $\{x_1, \ldots, x_{nm}\}$. Henceforth, let NIZK_i be a NIZK scheme for the relation $\widetilde{R}_i$ (described below). We describe a Shuffle-ZKP protocol in the $\mathcal{F}_{\text{shuffle}}$-hybrid model. We shall assume that each client i's input $\mathbf{x}_i \in \mathbb{F}^m$ is a vector of length m and each element in the vector is from some finite field $\mathbb{F}$[5].

Setup. The protocol starts with a setup phase where the clients the server jointly generate the public parameters. Each client i and the server serves as a pair of prover and verifier in the underlying NIZK protocol, so they share a common reference string crs_i.

[5]In practice, each individual message may not be a field element. In this case, we can use a hash function to map the original message to a field element and during the proof generation, the client additionally proves the correctness of the hashing. Given the collision-resistant property of the hash function, all properties of the protocol should hold.

Stage 1: Client preparation. As mentioned before, we use a polynomial-based set equality check to ensure consistency. To further ensure privacy, each client now samples some dummy elements to mask their local polynomial evaluation. We let client i sample d dummies from $\{\mathbb{F}/\{0\}\}$, denoted as $y_{i,1}, \ldots, y_{i,d}$. The client compute the product of the dummies as $\rho_i = \prod_{j\in[d]} y_{i,j}$. The client now sends a commitment, $\mathsf{com}_i = \mathsf{commit}(\mathbf{x}_i, \rho_i; \gamma_i)$ to the server where γ_i denotes a freshly sample random string. The client also sends the dummies to the shuffler.

Stage 2: Server broadcasting a challenge. The server broadcasts a random field r to all clients.

Stage 3: Client proof generation. Upon receiving the challenge r, the client can compute the masked polynomial evaluation $z_i = \prod_{j\in[m]}(x_{i,j} - r)\rho_i$. Now each client is ready to prove its compliance, which we model as an NP relation $\mathcal{R}_i(\mathbf{x}_i, w_i)$. To include the consistency check, we now define an augmented NP relation $\widetilde{R}_i$ as following.

- Public statement (z_i, com_i, r): the client's masked polynomial evaluation z_i, the commitment com_i including the client's messages and the mask, and also the evaluation point r;
- Private witness $(\mathbf{x}_i, \gamma_i, w_i, \rho_i)$: the client's messages $\mathbf{x}_i$, the client's witness w_i to the compliance check, the randomness γ_i used in the commitment and the product of the dummies, ρ_i.
- $\widetilde{R}_i((z_i, \mathsf{com}_i, r), (\mathbf{x}_i, \gamma_i, w_i, \rho_i)) = 1$ iff
  - $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$;
  - $\mathsf{commit}((\mathbf{x}_i, \rho_i); \gamma_i) = \mathsf{com}_i$; and
  - $\prod_{j\in[m]}(x_{i,j} - r) \cdot \rho_i = z_i$.

The client generates the proof π_i by invoking the prover algorithm of the underlying NIZK protocol, then sends the proof and the masked evaluation z_i to the server.

Stage 4: Server verification. The server first verifies all the proofs. If any proof fails to be verified, the server can identify the bad client. If all the proofs can be verified, the server also compare the polynomial evaluations to ensure consistency. Upon receiving messages $\{x_i\}_{i\in[nm]}$ and dummies $\{y_i\}_{i\in[nd]}$ from the shuffler, the server computes $\prod_{i\in[nm]}(x_i - r)$ as its evaluation and computes the product of the dummies $\prod_{i\in[nd]} y_i$. Finally, the server compares the product of the clients' masked evaluations to its evalutaion by checking if $\prod_{i\in[n]} z_i = \prod_{i\in[nm]}(x_i - r) \cdot \prod_{i\in[nd]} y_i$.

Parameter choices. With n clients and at most t corrupted clients, to ensure privacy, we set the dummy number as

$$d = \left\lceil \frac{2\lambda + \log_2 |\mathbb{F}|}{\log_2(n-t) - \log_2 e} + 2 \right\rceil.$$

Here, λ is the statistical security parameter (and there is a separate computational security parameter associated with the NIZK construction), and we assume there more than 19 honest clients[6]. The reason behind the formula is that we will use the following key lemma from Balle et al. [BBGN20] to prove the privacy. We will consider the multiplicative group $(\mathbb{F} \setminus \{0\}, \cdot)$ as the abelian group $\mathbb{G}$ when we use the lemma.

[6]If there are no more than 19 honest clients, we can use $d = \lceil 1.5 \log_2(|\mathbb{F}|) + \log_2 n + \lambda \rceil$ as specified in Ishai et al. [IKOS06]

$\underline{\mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}})}$**:** For $i \in [n]$, run $\mathsf{crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Gen}(1^\lambda)$. Set the public parameters $\mathsf{pp} := \{\mathsf{crs}_i\}_{i\in[n]}$.

$\Pi(\mathsf{pp}, \mathbf{x}_1, \ldots, \mathbf{x}_n, w_1, \ldots, w_n)$

- **Each client** i: /* *client* i*'s input:* $\mathbf{x}_i, w_i$, pp */
  - Sample $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,d}) \xleftarrow{\$} \{\mathbb{F}/\{0\}\}^d$ and let $\rho_i := \prod_{j\in[d]} y_{i,j}$.
  - Send $(y_{i,1}, \ldots, y_{i,d})$ to $\mathcal{F}_{\text{shuffle}}$, which sends $\mathsf{Multiset}(\{y_{i,j}\}_{i\in[n],j\in[d]})$ to the server.
  - Send $\mathsf{com}_i = \mathsf{commit}((\mathbf{x}_i, \rho_i); \gamma_i)$ directly to the server where γ_i denotes sampled random coins.
- **Server**: /* *server's input:* $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, pp*/
  - Sample a random challenge $r \xleftarrow{\$} \mathbb{F}$ and send r to all clients.
- **Each client** i:
  - Parse $\mathbf{x}_i := (x_{i,1}, \ldots, x_{i,m})$, and compute $z_i := \prod_{j\in[m]}(x_{i,j} - r) \cdot \rho_i$.
  - Parse $\mathsf{pp} := (\mathsf{crs}_1, \ldots, \mathsf{crs}_n)$, and call $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{P}(\mathsf{crs}_i, (z_i, \mathsf{com}_i, r), (\mathbf{x}_i, \gamma_i, w_i, \rho_i))$.
  - Send (z_i, π_i) to the server.
- **Server**: Output 1 if the following checks pass, else output 0:
  - For $i \in [n]$, $\mathsf{NIZK}_i.\mathsf{V}(\mathsf{crs}_i, (z_i, \mathsf{com}_i, r), \pi_i) = 1$.
  - $\prod_{i\in[n]} z_i = \prod_{i\in[n],j\in[m]}(x_{i,j} - r) \cdot \prod_{i\in[n],j\in[d]} y_{i,j}$. The server can easily compute both sides of the equation knowing $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and $\mathsf{Multiset}(\mathbf{y}_1, \ldots, \mathbf{y}_n)$.

Figure 3: Our main stage protocol.

Lemma 5.1 (Key lemma from [BBGN20] (simplified))**.** *Suppose $n \geq 19$. Given an abelian group $(\mathbb{G}, \cdot)$ and two vectors $(\mu_1, \ldots, \mu_n) \in \mathbb{G}^n$ and $(\mu'_1, \ldots, \mu'_n) \in \mathbb{G}^n$ such that $\prod_{i\in[n]} \mu_i = \prod_{i\in[n]} \mu'_i$. For $i \in [n]$, let $(y_{i,1}, \ldots, y_{i,d})$ and $(y'_{i,1}, \ldots, y'_{i,d})$ be random linear shares of μ_i and μ'_i, respectively. Let $Y = \mathsf{Multiset}(y_{1,1}, \ldots, y_{n,d})$ and $Y' = \mathsf{Multiset}(y'_{1,1}, \ldots, y'_{n,d})$. Then, as long as $d \geq \left\lceil \frac{2\lambda + \log_2(|\mathbb{G}|)}{\log_2(n) - \log_2 e} + 1 \right\rceil$ and $d \geq 4$, the statistical distance between Y and Y' is negligibly small.*

In our practical implementation, we will use a prime field $\mathbb{F}_p$ where p is a 254-bit prime. Suppose we want to achieve a statistical security failure probability of 2^{-80}. Then, with 100 honest clients, we can choose $d = 82$, with 1000 honest clients, we can choose $d = 51$, and when there are 10,000 honest clients, we can choose $d = 37$.

Security. We formally prove the following security theorem. We will defer the proofs to Appendix A.1.

Theorem 5.2 (Shuffle-ZKP construction with honest-verifier zero-knowledge)**.** *Suppose that $|\mathbb{F}|$ is superpolynomial in λ, $n - t \geq 19$ and $d \geq \left\lceil \frac{2\lambda + \log_2(|\mathbb{F}| - 1)}{\log_2(n-t) - \log_2 e} + 2 \right\rceil$. Further, suppose that the underlying* NIZK *satisfies completeness, soundness, and zero-knowledge, and the commitment scheme* comm

is computationally binding and computationally hiding. Then, the above Shuffle-ZKP *construction satisfies completeness, soundness, and t-honest-verifier-zero-knowledge.*

Asymptotic Costs. We show that Shuffle-ZKP is efficient in terms of asymptotic costs.

Theorem 5.3. *Suppose* $\log_2(|\mathbb{F}|) = O(\lambda)$ *and* $d = O(\lambda)$. *Assume that the underlying* NIZK *has* $\widetilde{O}_\lambda(c)$[7] *proving time,* $\widetilde{O}_\lambda(1)$ *verifying time and* $\widetilde{O}_\lambda(1)$ *proof size given a size-c relation, and the commitment scheme* comm *has* $\widetilde{O}_\lambda(m)$ *computation time and* $\widetilde{O}_\lambda(1)$ *commitment size. Then, the above* Shuffle-ZKP *construction that checks the predicate* $\mathcal{R}$ *on each client has* $\widetilde{O}_\lambda(|\mathcal{R}| + m)$ *per client time,* $\widetilde{O}_\lambda(1)$ *per client communication, and* $\widetilde{O}_\lambda(nm)$ *total server time.*

Proof. We discuss the communication cost first. Each client needs to send the dummies $(y_{i,1}, \ldots, y_{i,d})$ to the shuffler, and since $d = O(\lambda)$, sending $(y_{i,1}, \ldots, y_{i,d})$ requires only $O_\lambda(1)$ bits. Further, each client needs to send com_i, z_i and π_i, i.e., the commitment, the masked polynomial evaluation and the proof to the server and receives a challenge from the server. The commitment com_i is $\widetilde{O}_\lambda(1)$ bits long. Sending z_i and receiving the challenge take only $O_\lambda(1)$ communication, and sending π_i requires only $\widetilde{O}_\lambda(1)$ overhead. Therefore, each client's communication overhead is only $\widetilde{O}_\lambda(1)$, not counting the overhead of transmitting the $\mathbf{x}_i$ to the shuffler. The server's communication cost is the sum of the clients' costs since the communication pattern is client-server only.

We now analyze the client and server's computation costs. Each client needs to generate a proof for the relation $\widetilde{R}_i$, which $\widetilde{\mathcal{R}}_i$ checks the original compliance rule predicate $\mathcal{R}_i$ and checks correct computation of a commitment and a product calculation. Therefore, $|\widetilde{\mathcal{R}}_i| = |\mathcal{R}_i| + O_\lambda(m)$ where $|\mathcal{R}|$ denotes the size of the constraint representation of checking the NP relation $\mathcal{R}$, and recall that m is the length of client i's vector $\mathbf{x}_i$. The server's verification time is $\widetilde{O}_\lambda(n)$ since verifying each client's proof takes $\widetilde{O}_\lambda(1)$ time. The server also needs to compute polynomial evaluation and the product of the dummies, which takes $\widetilde{O}_\lambda(nm)$ time. □

Practical Instantiation. We use MiMC hash [AGR+16] (as a random oracle) to instantiate the commitment scheme. We provide two instantiations of our protocol with two NIZK schemes, Groth16 [Gro16] and Plonk [GWC19]. Groth16 has better concrete performances but requires a per-circuit setup. Plonk has a universal trusted setup. Practitioners can choose one instantiation based on the actual scenarios. We use the gnark library [BPH+22] for the implementation. In Section 7, we evaluate our protocol in realistic scenarios and report the concrete performance. Notably, even for a predicate of size around one million, the per client run time is around 10 seconds and the per-client communication is around 2-3 KB.

5.2 Achieving Zero-Knowledge Against a Malicious Verifier

The construction in Section 5.1 provides only honest-verifier zero-knowledge. A malicious server can intentionally pick a challenge r that equals one of the messages. Then, the server learns the source of that particular message, because the client who submitted it will have a polynomial evaluation of 0. We observe that if the random challenge r does not equal to any one messages, the protocol is zero-knowledge (we still need r to have high entropy to ensure soundness).

To additionally achieve zero-knowledge against a malicious verifier, one naïve way is the use a generalization of the generic transformation described by Damgård [Dam] to our setting. Another naïve approach is to use a coin toss protocol among the server and the clients to jointly sample the random challenge. These naive approaches would add high overhead to the protocol. We propose a practical and nearly overhead-free upgrade to the protocol.

[7]The notation $\widetilde{O}(\cdot)$ hides the polylogarithmic terms.

A practical upgrade. The upgrade relies on a non-programmable random oracle[8], denoted $\mathsf{RO} : \{0,1\}^* \to \{0,1\}^\lambda$. We let the server commit to the challenge upfront by hashing it with a random oracle. The clients check if the opening agrees with the commitment when the challenge is opened. Further, the challenge is sampled from a different half of the field than the messages.

- Assume we use a prime-order field $\mathbb{F}_p$. We will encode the clients' messages $x_{i,j}$ using the first half of the field $\{0, \ldots, \lfloor p/2 \rfloor\}$.
- At the beginning of the protocol, the server picks a random challenge r from $\{\lfloor p/2 \rfloor + 1, \ldots, p-1\}$, and it computes $\widetilde{r} := \mathsf{RO}(r)$. The server sends $\widetilde{r}$ to all clients over a broadcast channel.
- When the server announces the challenge r, all clients check that $\mathsf{RO}(r) = \widetilde{r}$ and that $r \geq \lfloor p/2 \rfloor + 1$, and they abort if the check fails.

The size of the field $\mathbb{F}$ used here is exponentially large in the security parameter λ, and therefore the challenge r has high entropy. In this case, $\widetilde{r} := \mathsf{RO}(r)$ can be viewed as a secure commitment to the challenge r.

We prove that this upgraded protocol satisfies soundness, and t-zero-knowledge in Appendix A.2.

6 Phase 2: Non-compliance attribution

If Phase 1 (non compliance-detection) fails, it means that at least one of the clients has misbehaved. The server can directly identify those who submits invalid proofs during the detection protocol. In our non-compliance attribution protocol (Phase 2), we identify clients that fail the consistency check. We devise a full protocol that satisfies completeness, soundness, and t-zero-knowledge as before, and also *misbehavior Identifiability*. That is, except with negligible probability, if the protocol aborts or the verification fails, then at least one corrupt client will be identified. Further, except with negligible probability, no honest client will be identified assuming the server is honest.

6.1 Attribution Protocol

In Figure 4, we describe a full Shuffle-ZKP protocol with an extra round of attribution protocol. An intuitive overview is included in Section 3.2. The full protocol also takes account inconsistencies between the dummy sets.

Message Collection Phase. We augment the collection phase: each client i samples serial numbers $\mathsf{sx}_{i,1}, \ldots, \mathsf{sx}_{i,m}$, and $\mathsf{sy}_{i,1}, \ldots, \mathsf{sy}_{i,d}$ for their m messages and d dummies, repectively. The client generates the commitments of all the serial numbers as $\widetilde{\mathsf{sx}}_{i,1}, \ldots, \widetilde{\mathsf{sx}}_{i,m}$ and $\widetilde{\mathsf{sy}}_{i,1}, \ldots, \widetilde{\mathsf{sy}}_{i,d}$. The client will then bind the message $x_{i,j}$ and $\widetilde{\mathsf{sx}}_{i,j}$ together and send $\{(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})\}_{j \in [m]}$ to the shuffler.

Main Stage. Same as before, except that each client now binds the dummy $y_{i,j}$ and $\widetilde{\mathsf{sy}}_{i,j}$ together and sends $(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})$ to the shuffler. The shuffler identifies who fails to send desired number of messages. The server identifies who submits invalid proofs.

[8] We need to modify the formal definition of shuffle-ZKP to account for the random oracle. See Appendix A.2.

- **Message Collection Phase – Each Client** i: /* *client* i*'s input:* $\mathbf{x}_i$ */*
  - Sample $(\mathsf{sx}_{i,1}, \ldots, \mathsf{sx}_{i,m}) \xleftarrow{\$} \{0,1\}^{\lambda \cdot m}$, $(\mathsf{sy}_{i,1}, \ldots, \mathsf{sy}_{i,d}) \xleftarrow{\$} \{0,1\}^{\lambda \cdot d}$.
  - Let $\widetilde{\mathsf{sx}}_{i,j} = \mathsf{commit}(\mathsf{sx}_{i,j}; \beta_{i,j})$ and $\widetilde{\mathsf{sy}}_{i,j} = \mathsf{commit}(\mathsf{sy}_{i,j}; \beta'_{i,j})$ where $\beta_{i,j}, \beta'_{i,j}$ is sampled randomly.
  - Initially, instead of sending $\mathbf{x}_i := \{x_{i,j}\}_{j \in [m]}$ to $\mathcal{F}_{\text{shuffle}}$, send $\{(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})\}_{j \in [m]}$ to $\mathcal{F}_{\text{shuffle}}$.
- **Main Stage**:
  - Execute the protocol in Section 5.2. For each client i, instead of sending $\mathbf{y}_i := \{y_{i,j}\}_{j \in [d]}$ to $\mathcal{F}_{\text{shuffle}}$, send $\{(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})\}_{j \in [d]}$ to $\mathcal{F}_{\text{shuffle}}$. If $\mathcal{F}_{\text{shuffle}}$ aborts[a], report the the corrupt players identified by $\mathcal{F}_{\text{shuffle}}$.
  - If the server detects an invalid proof or fails to hear from any client i, report client i.
  - If the server outputs 0 in the main stage (i.e., verification fails), proceed to the attribution protocol.
- **Attribution Protocol**: // $\widetilde{S}_x := \mathsf{Multiset}\left((x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})_{i \in [n], j \in [m]}\right)$, $\widetilde{S}_y := \mathsf{Multiset}\left((y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})_{i \in [n], j \in [d]}\right)$

  Each client i**:**
  - Fetch the digests $\mathsf{digest}(\widetilde{S}_x)$ and $\mathsf{digest}(\widetilde{S}_y)$ of the sets $\widetilde{S}_x$ and $\widetilde{S}_y$ from the server.
  - For $j \in [m]$, privately retrieve a membership proof $\mathsf{mkx}_{i,j}$ for $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ from the server; for each $j \in [d]$, privately retrieve a membership proof $\mathsf{mky}_{i,j}$ for $(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})$ from the server;
  - Call the NIZK prover to generate a proof π'_i:
    - Public statement: $\mathsf{com}_i, \mathsf{digest}(\widetilde{S}_x), \mathsf{digest}(\widetilde{S}_y), \mathsf{sx}_{i,1}, \ldots, \mathsf{sx}_{i,m}, \mathsf{sy}_{i,1}, \ldots, \mathsf{sy}_{i,m}$;
    - Private witness: $\gamma_i, \rho_i, (x_{i,j}, \widetilde{\mathsf{sx}}_{i,j}, \beta_{i,j})_{j \in [m]}, (y_{i,j}, \widetilde{\mathsf{sy}}_{i,j}, \beta'_{i,j})_{j \in [d]}$;
    - Relation:
      1. $\mathsf{com}_i = \mathsf{commit}((\mathbf{x}_i, \rho_i); \gamma_i)$;
      2. $\rho_i = \prod_{j \in [d]} y_{i,j}$;
      3. For $j \in [m]$, $\mathsf{mkx}_{i,j}$ is a valid membership proof for $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ w.r.t. $\mathsf{digest}(\widetilde{S}_x)$;
      4. For $j \in [m]$, $\widetilde{\mathsf{sx}}_{i,j} = \mathsf{commit}(\mathsf{sx}_{i,j}; \beta_{i,j})$;
      5. For $j \in [d]$, $\mathsf{mky}_{i,j}$ is a valid membership proof for $(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})$ w.r.t. $\mathsf{digest}(\widetilde{S}_y)$;
      6. For $j \in [d]$, $\widetilde{\mathsf{sy}}_{i,j} = \mathsf{commit}(\mathsf{sy}_{i,j}; \beta'_{i,j})$;
  - Send $\mathsf{sx}_{i,1}, \ldots, \mathsf{sx}_{i,m}, \mathsf{sy}_{i,1}, \ldots, \mathsf{sy}_{i,d}$, and the proof π'_i to the server.

  Server:
  - For $i \in [n]$, verify the proof π'_i. If the proof is invalid, report client i.
  - If a serial-number collision of the form $\mathsf{sx}_{i,j} = \mathsf{sx}_{i',j'}$ or $\mathsf{sy}_{i,j} = \mathsf{sy}_{i',j'}$ is found where $(i,j) \neq (i',j')$, then report clients i and i' (including the case when $i' = i$).

[a] $\mathcal{F}_{\text{shuffle}}$ aborts if any client who submits not exactly m message tuples or d dummy tuples.

Figure 4: **Full protocol with misbehavior identifiability.** The variables com_i, γ_i, and ρ_i are inherited from the main stage of the protocol described earlier.

Attribution protocol: client side algorithm. As mentioned before, client i needs to show that for each $j \in [m]$, the message tuple $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ belongs to the multiset $\widetilde{S}_x := \mathsf{Multiset}\left((x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})_{i\in[n],j\in[m]}\right)$. The client fetches the digest $\mathsf{digest}(\widetilde{S}_x)$ and privately fetch a membership proof $\mathsf{mkx}_{i,j}$ for the tuple w.r.t. the succinct digest from the server. The client similarly fetches the membership proofs for all d dummy tuples, $\{(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})\}_{j\in[d]}$. Now, the client i generates a proof showing that:

- The messages and the dummies used in this proof are consistent with the previous commitments;
- For $j \in [m]$, $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j}) \in \widetilde{S}_x$;
- For $j \in [d]$, $(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j}) \in \widetilde{S}_y$.

Also, the client needs to reveal the serial numbers $\{\mathsf{sx}_{i,j}\}_{j\in[m]}$ and $\{\mathsf{sy}_{i,j}\}_{j\in[d]}$ to argue that it does share any messages. It includes the following statement in the proof:

- For $j \in [m]$, $\widetilde{\mathsf{sx}}_{i,j}$ is a valid commitment of $\mathsf{sx}_{i,j}$;
- For $j \in [d]$, $\widetilde{\mathsf{sy}}_{i,j}$ is a valid commitment of $\mathsf{sy}_{i,j}$.

Attribution protocol: server side algorithm. The server side algorithm is straightforward:

- Identify who submits an invalid proof;
- Identify who shares the same serial numbers.

6.2 Theoretical Analysis

The protocol's soundness is straightforward, and follows from the soundness of the main phase which we proved in Appendix A.2. Completeness is easily verified. Below, we show the asymptotic bounds and prove zero knowledge, no false implication, and attribution completeness.

Asymptotic Bounds. Except the private queries of the membership proofs, the attribution protocol only incur a very light overhead. The clients attach a λ-bit string to each of their submissions. In the additional round, the client submits an additional proof and $2m$ serial numbers to the server. The size of the relation each client needs to prove is of size $\widetilde{O}_\lambda(m)$, so the additional computation time per client is also $\widetilde{O}_\lambda(m)$ with a quasilinear proving algorithm.

For the private queries, each client can use a batch PIR scheme [ACLS18, MR23] to fetch m membership proofs at once. The server computation will be $\widetilde{O}_\lambda(nm)$ and the communication cost will be $\widetilde{O}_\lambda(m)$.

Theorem 6.1 (Full construction)**.** *Suppose that the underlying* NIZK *has $\widetilde{O}_\lambda(c)$ proving time, $\widetilde{O}_\lambda(1)$ verifying time and $\widetilde{O}_\lambda(1)$ proof size given a size-c relation, the commitment scheme* comm *has $\widetilde{O}_\lambda(m)$ computation time and $\widetilde{O}_\lambda(1)$ commitment size. Also, given a database of size $\widetilde{O}_\lambda(nm)$, fetching m membership proofs takes $\widetilde{O}_\lambda(nm)$ server time and $\widetilde{O}_\lambda(m)$ communication cost. Then, the full construction to check the predicate $\mathcal{R}$ on every client has $\widetilde{O}_\lambda(|\mathcal{R}| + m)$ per client time, $\widetilde{O}_\lambda(n^2 m)$ total server time and $\widetilde{O}_\lambda(m)$ per client communication.*

Privacy. We prove that our full protocol with attribution satisfies zero-knowledge even against a malicious verifier. Due to the introduction of the committed serial numbers, our full protocol has slightly different syntax than the earlier main-stage protocol. Specifically, consider the main stage and the attribution protocols—we can view each client i's input as not only $\mathbf{x}_i, w_i$ but also $\{\mathsf{sx}_{i,j}, \beta_{i,j}\}_{j\in[m]}$. The server's input is $\widetilde{S} := \mathsf{Multiset}\left((x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})_{i\in[n],j\in[m]}\right)$. We will imagine that the serial numbers $\mathsf{sy}_{i,j}$ for the dummy pool and their commitments are sampled during the main stage, so they are included in the input.

We can still prove the t-zero-knowledge property of the protocol with the new syntax.

Theorem 6.2. *Given a security parameter λ, assume $|\mathbb{F}|$ is superpolynomial in λ, $n - t \geq 19$, $d \geq \left\lceil \frac{2\lambda+\log_2(|\mathbb{F}|-1)}{\log_2(n-t)-\log_2 e} + 2 \right\rceil$. Moreover, suppose that the underlying NIZK satisfies zero-knowledge, the commitment scheme is computationally hiding, the private retrieval of the merkle proofs are achieved using a secure PIR scheme, and $\mathsf{RO}(\cdot)$ is a non-programmable random oracle. Then, our full protocol with blame attribution satisfies t-zero-knowledge.*

The proof is deferred to Appendix B.

Misbehavior Identifiability. Here, we assume the adversary only controls corrupted clients and the server is honest – if the server is malicious, it can arbitrarily report any client and there is no guarantee whatsoever. We now show that the protocol satisfies misbehavior identifiability.

Theorem 6.3. *Assume that*

1. *The server is honest and the adversary corrupts a subset of clients $\mathcal{C} \subseteq [n]$. Honest entities communicate through secure channels;*
2. *The shuffle protocol $\mathcal{F}_{\text{shuffle}}$ satisfies identifiable abort – the protocol identifies the clients that do not submit the expected number of messages;*
3. *The NIZK scheme is sound;*
4. *The commitment scheme is computationally hiding and binding.*

Then, the attribution protocol satisfies misbehavior identifiability, such that if the verification fails,

1. *(No false implication) Except with negligible probability, the server will not report any honest client;*
2. *(Attribution completeness) Except with negligible probability, the server reports at least one client in $\mathcal{C}$.*

We defer the proof to Appendix B. Note that the attribution protocol may not identify all corrupted clients. In the worst case, the adversary can rearrange the messages of the corrupted clients even before the detection protocol begins, so that most of the corrupted clients can just follow the protocol honestly and only a small subset of the corrupted clients will be identified.

7 Empirical Evaluation

Implementation. We use gnark [BPH+22], an open-sourced NIZK library to implement our protocol. We implement the non-compliance detection protocol assuming a semi-honest adversary. We do not implement the attribution protocol since our asymptotic analysis Theorem 6.1 shows the

cost of the PIR parts dominates the overheads, while the remaining parts have the same asymptotic costs as the detection protocol. Designing a good batch PIR scheme is itself an active research area and is beyond the scope of this paper. We use a prime modulo field $\mathbb{F}_r$ with r being a 254-bit prime associated with the BN254 elliptic curve. We use a security parameter of $\lambda = 80$. We instantiate our protocol with two proof schemes, Groth16 [Gro16] and Plonk [GWC19]. Groth16 requires a per-circuit setup and Plonk has a universal setup. We use the MiMC hash [AGR+16] for the commitment scheme with a 254-bit random salt. The experiments are run on single server with a 2.4GHz Intel Xeon E5-2680 CPU and a 256 GB RAM. To avoid excessive repetition, we only generate real proofs for 10 clients and generate dummy proofs for the others. We compute the amortized cost accordingly.

Settings. We implement three use cases. In all three use cases, there are 1000 clients with no more than 500 corrupted clients. We list the concrete settings here.

Shuffle-DP summation protocol with contribution constraint checking. We implement the shuffle-DP summation protocol from Balle et al. [BBGN20].

- Each client has an integer secret value v_i in $[0, 1000]$. The client adds a noise y_i to v_i, where y_i is distributed as the difference between two independent random variables sampled from $\textbf{Polya}(1/n, e^{-\varepsilon/1000})$ as specified in [BBGN20]. This ensures the final sum value satisfies ε-shuffle-DP with $\varepsilon = 1.0$. Each client will split its noisy value $v_i + y_i$ into 60 additive shares (ensuring 80-bit statistical security) and the server sums up all the shares.
- *Constraint:* The contribution of each client to the final sum, i.e., $v_i + y_i$, is less than 1500.[9]

Notice that for this implementation, we do not verify if the client's noise is generated correctly. We propose a more involved protocol that can further verify the noise generation process Appendix D.

Secure vector summation protocol with L_2 norm constraint. We extend the secure summation protocol from Balle et al. [BBGN20] to sum vectors with an L_2 norm constraint, which applies to update aggregation in federated learning.

- Each client has a 50-dimension vector $\vec{x}_i$. The client splits the corresponding value in each dimension of its vector to 60 additive shares and the server sums all the shuffled shares. A client's shares across different dimensions are unlinkable.
- *Constraint:* The L_2 norm of each client's vector, i.e., $\|\vec{x}_i\|_2$, is no more than 1000.

An anti-money laundering protocol for a simplified privacy-preserving transaction system.

- Each client has 200 transactions of the form $(\mathsf{src}, \mathsf{dst}, \mathsf{amt})$, denoting the source account number, the destination account number and and the amount, respectively. For each transaction, the client samples a random salt slt and computes the transaction's MiMC [AGR+16] hash $h = \mathsf{Hash}(\mathsf{src}||\mathsf{dst}||\mathsf{amt}||\mathsf{slt})$. The server only sees all the shuffled hashes.

[9] We choose 1500 because the probability that an honest client's noise being more than 500 is small enough. This is only for demonstration and can be adjusted according to other scenarios.

- *Constraint:* 1) The hash values are computed correctly. 2) The total amount sent to any individual destination is no more than \$10,000.

This setting is a simplified caricature of real-world transaction systems for demonstration purposes.

7.1 Shuffle-ZKP Protocol's Communication Cost

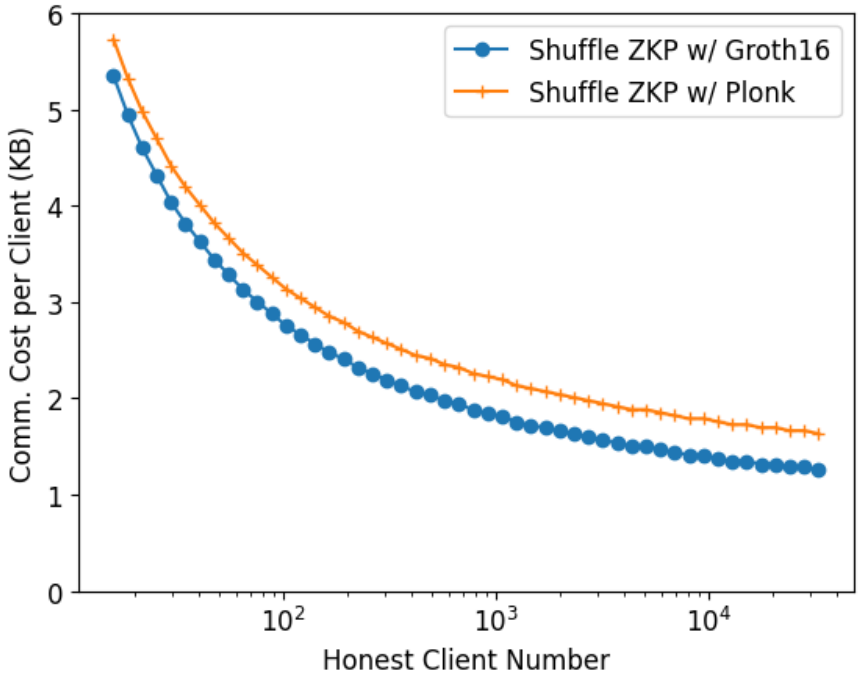

Figure 5: Empirical communication overhead per client in our protocol.

We define the per-client communication cost as the total bits each client communicates with the shuffler and the server after the setup phase. The per-client communication cost is dominated by the dummies used to mask the polynomial evaluations, independent of the concrete use cases. Each client needs to send $d = O(1 + \frac{\lambda}{n-t})$ dummies to the shuffler. Given more honest clients, the dummies sent to the shuffler per client are less. With our parameter setting, each client will not send more than 60 dummies to the server. The client will also send one commitment, one field element denoting the masked local polynomial evaluation and one NIZK proof to the server. We plot the communication overhead for our protocol in Fig 5 given different honest client numbers and two different proof systems. Given a wide range of client numbers, the communication cost is no more than 6KB.

7.2 Shuffle-ZKP Protocol's Computation and Memory Costs

We measure the computation and the communication cost for three use cases and show the results in Table 1. For each task, we repeat the experiment five times and report the average results. The amortized server time shows the server's total computation time amotized by the number of clients. Since the clients also need to store the proving key to generate the proof, the proving key sizes capture the clients' storage costs. Moreover, we also measure the the number of constraints needed

Use Case	# Constraints	Proving Key Size	Client Time	Amortized Server Time	Communication Per Client
			Based on Groth16		
Shuffle-DP Sum	2.4×10^4	4.2MB	0.2s	2.5ms	2.2KB
Vector Sum	9.9×10^5	163.3MB	8.5s	3.7ms	2.2KB
Simple AML	8.2×10^5	132.4MB	2.6s	3.5ms	2.2KB
			Based on Plonk		
Shuffle-DP Sum	3.6×10^4	22.5MB	0.4s	2.7ms	2.6KB
Vector Sum	1.3×10^6	721.4MB	10.6s	6.0ms	2.6KB
Simple AML	1.4×10^6	721.4MB	10.4s	2.8ms	2.6KB

Table 1: The computation and memory costs of Shuffle-ZKP protocol. "# Constraints" stands for number of constraints in the corresponding constraint system after compiling the program with gnark [BPH+22]. "Client Time" denotes the average of the clients' computation time. "Amortized Server Time" denotes the amortized computation time that the server spends on each individual client.

Components	Shuffle-DP	Vector Sum	Simple AML
Total	2.4×10^4	9.9×10^5	8.2×10^5
Compliance Check	4.0×10^3	2.0×10^3	7.6×10^5
Commitment	2.0×10^4	9.9×10^5	6.7×10^4
Polynomial Check	61	3001	201

Table 2: Breakdown of the constraints required by each component during the proof generation process for each task.

for each task after we compile the circuit to the R1CS constraint system. The constraint number can be viewed as a proxy for the complexity of each task.

Client Time. As shown in Table 1, the client time depends on the concrete use cases, and the Groth16-based instantiation is faster than the Plonk-based instantiation in general. We observe that the proof generation time is the dominant factor of the clients' computation time, and the more constraints it takes to describe the compliance rule, the more time it takes for the client to generate the proof. The constraint number for the Shuffle-DP Sum use case is between 2.4×10^4 to 3.6×10^4, and the average client time is 0.2s to 0.4s. The other two use cases have around one million constraints and the client time is significantly longer. The Groth16-based protocol takes 8.5s and 2.6s per client for the Vector Sum and the AML tasks, while the Plonk-based protocol takes 10.4-10.6s for those two tasks.

Server Time. The server computation is highly efficient – the amortized computation time (the total server time divided by the number of clients) is no more than 11ms for all use cases. This is because verifying a proof in Groth16 or Plonk only takes $\widetilde{O}_\lambda(1)$ time, and the server only needs to evaluate the polynomial evaluation besides the proof verification, which takes $O_\lambda(nm)$ time in total.

Storage Cost. The proving key sizes reflect the storage cost of the clients and also scale with the constraint number. The Groth16 system has smaller proving key sizes that are only 4.2MB for the shuffle-DP summation and no more than 200MB for the vector summation and the simple AML tasks. The plonk system has larger proving key sizes that are 22.5MB for the shuffle-DP summation and around 700MB for the remaining tasks.

8 Additional Related Work

Distributed ZK Proofs. Aside from Ozdemir and Boneh [OB22] and Dayama et al. [DPP+22], there are also several works study generic ZK proofs under different distributed settings. DIZK [WZC+18], deVirgo [XZC+22] and Pianist [LXZ+23] mainly consider how to accelerate proof generation using multiple distributed workers. Notably, Pianist also provides a robustness guarantee that those workers who output malformed proofs will be caught. However, these approaches do not provide privacy guarantee against corrupted workers. EOS [CLMZ23] and zkSaaS [GGJ+23] show how a client with weak computation power can delegate the proof generation to a group of stronger servers. EOS works under the setting where the client can stay online during the whole generation process, while ZKSaas allows the client to go offline. Both of the constructions protect the privacy of the witness against partially corrupted servers with different corruption thresholds, respectively. Their constructions do not fit in our setting where multiple clients wish to generate a joint proof.

Verifiability in Aggregagtion protocol Bell et al. [BGL+23] proposed ACORN, an input validation scheme for the pairwise-mask-based secure summation protocol (also known as secure aggregation). ACORN requires each client to submit a Zero-Knowledge proof to prove the validity of their inputs to the protocol (e.g. within a certain range). Notice that consistency is also one of the main challenge in their setting. The protocol requires each client participates in then participate in a collaborative version of Schnorr proof to ensure the summation of the masks used by the clients is consistent with the aggregated mask value observed by the server. In our setting, the consistency challenge is different: we care about the consistency of the set equality (and thus applying to different applications) while ACORN cares about the consistency of the summation of masks. There are also notable verifiable aggregation schemes [CGB17, BBCG+19, AGJ+22, DPRS23] in the multi-server setting, which assumes the existence of mulitple non-colluding servers. Each client will generate a secret-shared version of their inputs and send each shared piece to one of the serves. All servers (act as distributed verifiers) and the client (act as the prover) jointly interacts in a proving protocol ensure the inputs are valid. Instead, our work focuses on the single-server and muliti-client setting.

Algorithmic Solutions to Regulate Privacy-preserving Financial Systems. Recently, there has been a significant effort from academic researchers on seeking algorithmic solutions to regulate privacy-preserving financial systems. Most of them focus on building new systems from scratch with regulation capabilities. Our solution can be considered as an extra auditing or compliance checking tool to existing systems.

Some previous works target traditional settings such as inter-bank transaction systems. ZKLedger [NVV18] provides auditing capabilities for an auditor to look at an commitment-based public ledger and ask a participant (usually a bank) to generate a proof that shows the compliance and the consistency to the public ledger. ZKLedger solves the consistency issue by requiring each transaction in the ledger to record the change of the deposits for all participants, thus a participant has to use all transactions during the proofs. As an optimization, MiniLedger [CB21] uses crytpographic accumulators to reduce the cost of computing/storing the public ledger. Nonetheless, each transaction in ZKLedger and MiniLedger requires an full coordination between all the pariticipants, and thus only applicable when the number of participants are limited.

Some proposals aim to build regulation-friendly Central Bank Digital Currencies. For example, PRCash [WKČČ19] and UTT [TBA+22] are anonymous digital currency designs that enforce an spending amount on users' anonymous transactions given each time period. The privacy in these systems are achieved by distributing the trust to multiple regulating agencies or validators.

Account-based systems, such as Platypus [WKDC22] and PEReDi [KKS22], introduces per-account history information accumulators and the user needs to prove their compliance against the history accumulator, thus enabling compliance checking for the whole account history.

For the fully decentralized setting, Garman, Green and Miers [GGM16] extends the Decentralized Anonymous Payment (DAP) abstraction [SCG+14] for fully decentralized cryptocurrencies. There proposal is a bitcoin-like (also known as UTXO-based) transaction system that adds policy-related counter to each coin and includes a policy checking step when each unspent coin is being consuming. Zhaolu, Wan and Wang [ZWW22] introduces a DAP system with filters and supervisors, who identify suspicious transactions and initiate inspection operations, respectively. Badertscher, Sedaghat and Waldner [BSW23] presents Unlinkable Policy-compliant Signatures for DAP system, in which the re-randomizable public keys can ensure anonymity and a compliance proof based on the sender/recipient's attributes is verified during each transaction.

Cryptographic solutions are introduced to other types of new financial systems for regulation, such as accountable private cryptocurrency exchange [LQT23] and privacy-enhanced coin-mixing pool [BF23].

Acknowledgment

This work is in part supported by a grant from ONR, a gift from Cisco, NSF awards under grant numbers CIF-1705007, 2128519 and 2044679, and a Packard Fellowship.

References

[AČE+20] Sarah Allen, Srdjan Čapkun, Ittay Eyal, Giulia Fanti, Bryan Alexander Ford, James Grimmelmann, Ari Juels, Kari Kostiainen, Sarah Meiklejohn, Andrew Miller, et al. Design choices for central bank digital currency. 2020.

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *S&P*, 2018.

[AGJ+22] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In *SCN*. Springer, 2022.

[AGR+16] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. Cryptology ePrint Archive, Paper 2016/492, 2016. https://eprint.iacr.org/2016/492.

[AIVG22] Kinan Dak Albab, Rawane Issa, Mayank Varia, and Kalman Graffi. Batched differentially private information retrieval. In *USENIX Security*, pages 3327–3344, 2022.

[BBCG+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In *CRYPTO*. Springer, 2019.

[BBGN20] Borja Balle, James Bell, Adria Gascón, and Kobbi Nissim. Private summation in the multi-message shuffle model. In *CCS*, 2020.

[BC22] Ari Biswas and Graham Cormode. Verifiable differential privacy for when the curious become dishonest. *arXiv preprint*, 2022.

[BEM+17] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *SOSP*, 2017.

[BF23] Josh Beal and Ben Fisch. Derecho: Privacy pools with proof-carrying disclosures. *Cryptology ePrint Archive*, 2023.

[BGL+23] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. Acorn: Input validation for secure aggregation. In *USENIX Security*, 2023.

[BHNS20] Amos Beimel, Iftach Haitner, Kobbi Nissim, and Uri Stemmer. On the round complexity of the shuffle model. In *TCC*. Springer, 2020.

[Blu83] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 15:23–27, 1983.

[BM14] Anudhyan Boral and Michael Mitzenmacher. Multi-party set reconciliation using characteristic polynomials. In *52nd Annual Allerton Conference on Communication, Control, and Computing*. IEEE, 2014.

[BM19] Raffaella Barone and Donato Masciandaro. Cryptocurrency or usury? crime and alternative money laundering techniques. *European Journal of Law and Economics*, 47:233–254, 2019.

[BPH+22] Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. Consensys/gnark: v0.7.0, 2022.

[BSW23] Christian Badertscher, Mahdi Sedaghat, and Hendrik Waldner. Fine-grained accountable privacy via unlinkable policy-compliant signatures. *Cryptology ePrint Archive*, 2023.

[CB21] Panagiotis Chatzigiannis and Foteini Baldimtsi. Miniledger: compact-sized anonymous and auditable distributed payments. In *European Symposium on Research in Computer Security*. Springer, 2021.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *Eurocrypt*. Springer, 2023.

[CG97] Benny Chor and Niv Gilboa. Computationally private information retrieval. In *STOC*, 1997.

[CGB17] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI*, 2017.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, 1995.

[CJMP21] Albert Cheu, Matthew Joseph, Jieming Mao, and Binghui Peng. Shuffle private stochastic convex optimization. *arXiv preprint arXiv:2106.09805*, 2021.

[CLMZ23] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. Eos: Efficient private delegation of zksnark provers. In *USENIX Security*, 2023.

[CSU+19] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In *Eurocrypt*. Springer, 2019.

[CZJ+17] Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed E. Kosba, Ari Juels, and Elaine Shi. Solidus: Confidential distributed ledger transactions via PVORM. 2017.

[Dam] Ivan Damgård. On Σ protocols. https://www.cs.au.dk/~ivan/Sigma.pdf.

[DPP+22] Pankaj Dayama, Arpita Patra, Protik Paul, Nitin Singh, and Dhinakaran Vinayagamurthy. How to prove any np statement jointly? efficient distributed-prover zero-knowledge protocols. *PETS*, 2022.

[DPRS23] Hannah Davis, Christopher Patton, Mike Rosulek, and Phillipp Schoppmann. Verifiable distributed aggregation functions. *Cryptology ePrint Archive*, 2023.

[EFM+19] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *SODA*. SIAM, 2019.

[eur23] Proposal for a regulation of the european parliament and of the council on the establishment of the digital euro. https://finance.ec.europa.eu/system/files/2023-06/230628-proposal-digital-euro-regulation_en.pdf, 2023.

[FCJG20] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to {Byzantine-Robust} federated learning. In *USENIX Security*, 2020.

[FMT22] Vitaly Feldman, Audra McMillan, and Kunal Talwar. Hiding among the clones: A simple and nearly optimal analysis of privacy amplification by shuffling. IEEE, 2022.

[fuc22] Cobalt: Telemetry with built-in privacy. https://fuchsia.googlesource.com/fuchsia/+/refs/heads/main/src/cobalt/, 2022.

[GDD+21] Antonious Girgis, Deepesh Data, Suhas Diggavi, Peter Kairouz, and Ananda Theertha Suresh. Shuffled model of differential privacy in federated learning. In *AISTATS*, 2021.

[GGJ+23] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zksaas: Zero-knowledge snarks as a service. *Cryptology ePrint Archive*, 2023.

[GGK+21] Badih Ghazi, Noah Golowich, Ravi Kumar, Rasmus Pagh, and Ameya Velingker. On the power of multiple anonymous messages: Frequency estimation and selection in the shuffle model of differential privacy. In *Eurocrypt*. Springer, 2021.

[GGM16] Christina Garman, Matthew Green, and Ian Miers. Accountable privacy for decentralized anonymous payments. In *FC*, 2016.

[GJJZ22] Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Yinuo Zhang. Succinct zero knowledge for floating point computations. In *CCS*, 2022.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. Springer, 2016.

[GWC19] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

[HBH+16] Daira Hopwood, Sean Bowe, Taylor Hornby, Nathan Wilcox, et al. Zcash protocol specification. 2016.

[IKOS06] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography from anonymity. In *FOCS*. IEEE, 2006.

[KKS22] Aggelos Kiayias, Markulf Kohlweiss, and Amirreza Sarencheh. Peredi: Privacy-enhanced, regulated and distributed central bank digital currencies. In *CCS*, 2022.

[LCC+21] Ruixuan Liu, Yang Cao, Hong Chen, Ruoyang Guo, and Masatoshi Yoshikawa. Flame: Differentially private federated learning in the shuffle model. In *AAAI*, 2021.

[LQT23] Ya-Nan Li, Tian Qiu, and Qiang Tang. Pisces: Private and compliable cryptocurrency exchange. *arXiv preprint*, 2023.

[LXZ+23] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. *Cryptology ePrint Archive*, 2023.

[MR23] Muhammad Haris Mughees and Ling Ren. Vectorized batch private information retrieval. In *S&P*. IEEE, 2023.

[NVV18] Neha Narula, Willy Vasquez, and Madars Virza. zkledger: Privacy-preserving auditing for distributed ledgers. In *NSDI*, 2018.

[OB22] Alex Ozdemir and Dan Boneh. Experimenting with collaborative {zk-SNARKs}:{Zero-Knowledge} proofs for distributed secrets. 2022.

[oT23] United State Department of Treasury. Money laudering prevention: A money services business guide. https://www.fincen.gov/sites/default/files/shared/prevention_guide.pdf, 2023.

[SCG+14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *S&P*. IEEE, 2014.

[Sch80] Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 1980.

[SKL17] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. *NeurIPS*, 2017.

[TBA+22] Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. Utt: Decentralized ecash with accountable privacy. *Cryptology ePrint Archive*, 2022.

[Wan23] Shaowei Wang. Privacy amplification via shuffling: Unified, simplified, and tightened. *arXiv preprint arXiv:2304.05007*, 2023.

[WKČČ19] Karl Wüst, Kari Kostiainen, Vedran Čapkun, and Srdjan Čapkun. Prcash: Fast, private and regulated transactions for digital currencies. In *FC*. Springer, 2019.

[WKDC22] Karl Wüst, Kari Kostiainen, Noah Delius, and Srdjan Capkun. Platypus: a central bank digital currency with unlinkable transactions and privacy-preserving regulation. 2022.

[WZC+18] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. {DIZK}: A distributed zero knowledge proof system. In *USENIX Security*, 2018.

[XHCL19] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *ICLR*, 2019.

[XZC+22] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. In *CCS*, 2022.

[ZSCM23] Mingxun Zhou, Elaine Shi, T-H Hubert Chan, and Shir Maimon. A theory of composition for differential obliviousness. In *Eurocrypt*. Springer, 2023.

[ZWL+19] Zhilin Zhang, Ke Wang, Weipeng Lin, Ada Wai-Chee Fu, and Raymond Chi-Wing Wong. Practical access pattern privacy by combining pir and oblivious shuffle. In *CIKM*, 2019.

[ZWW22] Tianyu Zhaolu, Zhiguo Wan, and Huaqun Wang. Division of regulatory power: Collaborative regulation for privacy-preserving blockchains. *Cryptology ePrint Archive*, 2022.

A Proofs for Our Construction

In this section, we provide the missing proofs for the completeness, soundness, and t-zero-knowledge for both the honest verifier protocol and the malicious verifier protocol.

A.1 Proofs for the Honest Verifier Protocol

A.1.1 Completeness Proof

Theorem A.1 (Completeness)**.** *The protocol satisfies completeness: for any $\lambda, n \in \mathbb{N}$, for any $(\mathbf{x}, \mathbf{w})$ such that $\boldsymbol{\mathcal{R}}(\mathbf{x}, \mathbf{w}) = 1$,*

$$\Pr\left[\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}}),\ \Pi(\mathsf{pp}, \mathbf{x}, \mathbf{w}) = 1\right] = 1.$$

Proof. The completeness is straightforward.

□

A.1.2 Soundness Proof

Lemma A.2. *Let $\mathbb{F}$ be a finite field. Let $x_1, \ldots, x_k \in \mathbb{F}$, $\alpha \in \mathbb{F}\backslash\{0\}$, and let $x'_1, \ldots, x'_k, \alpha' \in \mathbb{F}$. Suppose that $\mathsf{Multiset}(x_1, \ldots, x_k) \neq \mathsf{Multiset}(x'_1, \ldots, x'_k)$. Then, it holds that*

$$\Pr_{r \xleftarrow{\$} \mathbb{F}}\left[\alpha \cdot \prod_{i \in [k]}(r - x_i) = \alpha' \cdot \prod_{i \in [k]}(r - x'_i)\right] \leq \frac{k}{|\mathbb{F}|}.$$

Proof. Consider the two polynomials $F(R) = \prod_{i\in[n]}(R - x_i)$ and $F'(R) = \prod_{i\in[n]}(R - x_i)$. Since $\mathsf{Multiset}(x_1, \ldots, x_n) \neq \mathsf{Multiset}(x'_1, \ldots, x'_n)$, we know that the two polynomials are not the same, i.e., $F(R) \neq F'(R)$.

- Case 1: $\alpha \neq \alpha'$. Since $F(R)$ and $F(R')$ are both monic polynomials, the two polynomials $\alpha F(R)$ and $\alpha' F'(R')$ are not the same. Thus, due to the Schwartz-Zippel lemma,

$$\Pr_{r\xleftarrow{\$}\mathbb{F}}\left[\alpha F(r) = \alpha' F'(r)\right] \leq \frac{n}{|\mathbb{F}|}.$$

- Case 2: $\alpha = \alpha'$. Since $\alpha \neq 0$, $\alpha F(r) = \alpha' F'(r)$ only happens when $F(r) = F'(r)$. Again, due to the Schwartz-Zippel lemma,

$$\Pr_{r\xleftarrow{\$}\mathbb{F}}\left[\alpha F(r) = \alpha' F'(r)\right] = \Pr_{r\xleftarrow{\$}\mathbb{F}}\left[F(r) = F'(r)\right] \leq \frac{n}{|\mathbb{F}|}.$$

□

Theorem A.3 (Soundness)**.** *Suppose the field size $|\mathbb{F}|$ is superpolynomial w.r.t. the security parameter λ, the commitment scheme is perfectly binding and the* NIZK *scheme satisfies soundness. Then, our protocol satisfies soundness.*

Proof. Before the random challenge is sampled, the server's view contains pp, $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, $\mathsf{Multiset}(\mathbf{y}_1, \ldots, \mathbf{y}_n)$, and $\{\mathsf{com}_i\}_{i\in[n]}$. Except with the negligible probability that some underlying NIZK instance's soundness is broken, if the server outputs "accept", then for every $i \in [n]$, it must be that there exists $(\mathbf{x}'_i, \gamma_i, w_i, \rho'_i)$ such that $\mathcal{R}_i(\mathbf{x}'_i, w_i) = 1$, $\mathsf{commit}(1^\lambda, (\mathbf{x}'_i, \rho'_i); \gamma_i) = \mathsf{com}_i$; and $\prod_{j\in[m]}(x'_{i,j} - r) \cdot \rho'_i = z_i$.

Henceforth we ignore the negligibly small probability that the NIZK's soundness is broken, and assume that the above equations hold. Since the commitment scheme is perfectly binding, the $\mathbf{x}'_i$ and ρ'_i satisfying the above equations are already uniquely determined given com_i, which must be sent before seeing the random challenge r — we need this because later, to apply Lemma A.2, the challenge r must be sampled independently of the $\mathbf{x}'_i$s and ρ'_is.

Now, if the server passes verification, it must be that

$$\prod_{i\in[n],j\in[m]} (x'_{i,j} - r) \cdot \alpha' = \prod_{i\in[n],j\in[m]} (x_{i,j} - r) \cdot \alpha$$

where

$$\alpha' := \prod_{i\in[n]} \rho'_i, \quad \alpha := \prod_{i\in[n],j\in[d]} y_{i,j}$$

Due to Lemma A.2, we conclude that except with the negligibly small probability that a bad challenge r is sampled, it must be that $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \mathsf{Multiset}(\mathbf{x}'_1, \ldots, \mathbf{x}'_n)$ where $(\mathbf{x}'_1, \ldots, \mathbf{x}'_n)$ are uniquely determined by $\mathsf{com}_1, \ldots, \mathsf{com}_n$.

Summarizing the above, except with negligible probability, if the server accepts, then there exists a partitioning $(\mathbf{x}'_1, \ldots, \mathbf{x}'_n)$ of the multiset $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ which is uniquely determined by $\mathsf{com}_1, \ldots, \mathsf{com}_n$, such that for every $i \in [n]$, $\mathcal{R}_i(\mathbf{x}'_i, w_i) = 1$.

□

A.1.3 Honest-Verifier-Zero-Knowledge Proof

Additional preliminary.

Statistical Distance. For any two distributions X, Y on the same finite sample space D, the statistical distance is defined as

$$SD(X,Y) = \frac{1}{2}\sum_{\alpha \in D} |\Pr[X=\alpha] - \Pr[Y=\alpha]|.$$

When the context is clear, given two random variables A, B, we also use the notation $SD(A,B)$ to denote the statistical distance between their empirical distribution.

Lemma A.4 ([BBGN20]). *Suppose $n \geq 19, d \geq 4$. Given an abelian group $(\mathbb{G}, \cdot)$ and two vectors $(\mu_1, \ldots, \mu_n)$ and $(\mu'_1, \ldots, \mu'_n)$ such that $\prod_{i\in[n]} \mu_i = \prod_{i\in[n]} \mu'_i$. For $i \in [n]$, let $y_{i,1}, \ldots, y_{i,d} \xleftarrow{\$} \mathbb{G}^d$ be uniformly random variables subject to $\prod_{j\in[d]} y_{i,j} = \mu_i$. Similarly, let $y'_{i,1}, \ldots, y'_{i,d} \xleftarrow{\$} \mathbb{G}^d$ be uniformly random variables subject to $\prod_{j\in[d]} y'_{i,j} = \mu'_i$. Let $Y = \mathsf{Multiset}(y_{1,1}, \ldots, y_{n,d})$ and $Y' = \mathsf{Multiset}(y'_{1,1}, \ldots, y'_{n,d})$. Then,*

$$SD(Y, Y') \leq 2^{-\sigma(n,d,|G|)},$$

where

$$\sigma(n,d,|G|) = \frac{(d-2)}{2}(\log_2 n - \log_2 e) - \frac{1}{2}\log_2 |\mathbb{G}|.$$

Lemma A.5 (Key lemma for proving zero-knowledge)**.** *Given a security parameter λ, assume $|\mathbb{F}|$ is superpolynomial in λ, $n-t \geq 19$, $d \geq \left\lceil \frac{2\lambda + \log_2(|\mathbb{F}|-1)}{\log_2(n-t) - \log_2 e} + 2 \right\rceil$. Denote $\mathcal{H}$ as the honest client index set. Given any $\{x_{i,j}\}_{i\in\mathcal{H}, j\in[m]} \in \mathbb{F}^{|\mathcal{H}|\cdot m}$ and $\{x'_{i,j}\}_{i\in\mathcal{H}, j\in[m]} \in \mathbb{F}^{|\mathcal{H}|\cdot m}$ such that $\mathsf{Multiset}(\{x_{i,j}\}_{i\in\mathcal{H}, j\in[m]}) = \mathsf{Multiset}(\{x'_{i,j}\}_{i\in\mathcal{H}, j\in[m]})$, given $r \notin \{x_{i,j}\}_{i\in\mathcal{H}, j\in[m]}$ also from $\mathbb{F}$, the following distributions have negligibly small in λ statistical distance:*

- *Distribution 0: Sample $\{y_{i,j}\}_{i\in\mathcal{H}, j\in[d]}$ at random from $\mathbb{F}$, output the following terms:*

$$\mathsf{Multiset}(\{x_{i,j}\}_{i\in\mathcal{H}, j\in[m]}), \quad r, \quad \mathsf{Multiset}(\{y_{i,j}\}_{i\in\mathcal{H}, j\in[d]}),$$
$$\text{for each } i \in \mathcal{H} : z_i := \prod_{j\in[m]}(x_{i,j} - r) \cdot \prod_{j\in[d]} y_{i,j}$$

- *Distribution 1: Same as Distribution 0 except that each $x_{i,j}$ is replaced with $x'_{i,j}$.*

Proof. First, consider the following hybrid experiment which is equivalent to Distribution 1 except that the order in which the random variables are sampled is changed.

Experiment Hyb_0:

- First, sample $\{z_i\}_{i\in\mathcal{H}}$ at random from $\mathbb{F}/\{0\}$.
- Next, for each $i \in \mathcal{H}$, compute $\mu_i := z_i / \prod_{j\in[m]}(x_{i,j} - r)$, basically μ_i corresponds to the the product of the terms $\{y_{i,j}\}_{j\in[d]}$.
- Next, for each $i \in \mathcal{H}$, sample $\{y_{i,j}\}_{j\in[d]}$ at random subject to the constraint that their product is μ_i.
- Finally, compute and output the following terms:

$$\mathsf{Multiset}(\{x_{i,j}\}_{i\in\mathcal{H}, j\in[m]}), r,$$
$$\mathsf{Multiset}(\{y_{i,j}\}_{i\in\mathcal{H}, j\in[d]}), \{z_i\}_{i\in\mathcal{H}}$$

Experiment Hyb_1: same as Hyb_0 except that each $x_{i,j}$ is replaced with $x'_{i,j}$. Hyb_1 is equivalent to Distribution 1 except that the order in which the random variables are sampled is changed.

The key lemma follows directly from the following claim which we prove below.

Claim A.6. *Hyb_0 and Hyb_1 have statistical distance negligibly small in λ.*

It suffices to show that conditioned on any fixed $\{z_i\}_{i\in\mathcal{H}}$, Hyb_0 and Hyb_1 are statistically close. Since $\mathsf{Multiset}(\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]}) = \mathsf{Multiset}(\{x'_{i,j}\}_{i\in\mathcal{H},j\in[m]})$, we have

$$\prod_{i\in\mathcal{H}} \frac{z_i}{\prod_{j\in[m]}(x_{i,j}-r)} = \prod_{i\in\mathcal{H}} \frac{z_i}{\prod_{j\in[m]}(x'_{i,j}-r)}.$$

Then, we can directly apply Lemma A.4 and get that the statistical distance of the distribution of $\mathsf{Multiset}(\{y_{i,j}\}_{i\in\mathcal{H},j\in[d]})$ in Hyb_3 and Hyb_4 is bounded by

$$2^{-\frac{(d-2)}{2}(\log_2(n-t)-\log_2 e)+\frac{1}{2}\log_2(|\mathbb{F}|-1)}.$$

Therefore, when $d \geq \left\lceil \frac{2\lambda+\log_2(|\mathbb{F}|-1)}{\log_2(n-t)-\log_2 e} + 2 \right\rceil$, the statistical distance of $\mathcal{A}$'s view between Hyb_3 and Hyb_4 is bounded by $2^{-\lambda}$.

□

Theorem A.7 (t-honest-verifier-zero-knowledge)**.** *Given a security parameter λ, assume $|\mathbb{F}|$ is superpolynomial in λ, $n-t \geq 19$, $d \geq \left\lceil \frac{2\lambda+\log_2(|\mathbb{F}|-1)}{\log_2(n-t)-\log_2 e} + 2 \right\rceil$. Moreover, suppose that the NIZK scheme satisfies zero-knowledge, and the commitment scheme is computationally hiding. Then, our shuffle-model ZKP protocol satisfies t-zero-knowledge.*

Proof. Let $\mathcal{H}$ be the set of at least $n-t$ honest users, and let $\mathcal{C}$ be the set of corrupt clients.

Real-world experiment. Below we first write down the real-world experiment where the adversary $\mathcal{A}$ can see the view of the server and the clients in $\mathcal{C}$.

1. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}})$.
2. $\mathcal{A}(1^\lambda, n, \mathsf{pp})$ outputs $\{\mathbf{x}_i, w_i\}_{i\in[n]}$ where $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$ for $i \in [n]$ and some state st.
3. output st and the adversary's view in an honest execution of the protocol, consisting of the following terms:
   (a) $\mathsf{Multiset}(\{y_{i,j}\}_{i\in\mathcal{H},j\in[d]})$ where the terms $\{y_{i,j}\}_{i\in\mathcal{H},j\in[d]}$ are all sampled at random from $\mathbb{F}/\{0\}$;
   (b) $\{\mathsf{com}_i\}_{i\in\mathcal{H}}$ where $\mathsf{com}_i := \mathsf{commit}((\mathbf{x}_i, \prod_{j\in[d]} y_{i,j}); \gamma_i)$;
   (c) a random challenge r; and
   (d) (z_i, π_i) for each $i \in \mathcal{H}$, where $z_i := \prod_{j\in[m]}(x_{i,j}-r)\cdot\prod_{j\in[d]} y_{i,j}$, and π_i is an honestly computed NIZK proof by client i.
   (e) All the other messages sent by the corrupt clients, including $\{y_{i,j}\}_{i\in\mathcal{C},j\in[d]}$, and $\{\mathsf{com}_i, z_i, \pi_i\}_{i\in\mathcal{C}}$ — without loss of generality, we can omit this part in our subsequent proofs due to Remark A.8.

Remark A.8. Since we are consider a semi-honest adversary, Item 3e above can always be generated honestly using $\{\mathbf{x}_i, w_i\}_{i\in\mathcal{C}}$. Henceforth in our proof, we shall omit this part from the view. Also, for part (a), the adversary actually gets from the shuffler $\mathsf{Multiset}(\{y_{i,j}\}_{i\in[n],j\in[d]})$. However, since the adversary knows the part $\{y_{i,j}\}_{i\in\mathcal{C},j\in[d]}$, the new information the adversary can derive from $\mathsf{Multiset}(\{y_{i,j}\}_{i\in[n],j\in[d]})$ is $\mathsf{Multiset}(\{y_{i,j}\}_{i\in\mathcal{H},j\in[d]})$.

Experiment Hyb_1. Experiment Hyb_1 is otherwise identical to the real-world experiment, except that

- Inside the Setup algorithm, for any honest user $i \in \mathcal{H}$, instead of calling the real $\mathsf{NIZK}_i.\mathsf{Gen}$ algorithm for honest users, now call the simulator of NIZK_i to generate simulated common reference strings and the trapdoors (crs_i, τ_i).
- Whenever the experiment needs to compute a NIZK proof on behalf of an honest user $i \in \mathcal{H}$, it calls the simulator of NIZK_i to generate a proof without using any witness.

Claim A.9. *Suppose that the NIZK scheme satisfies zero-knowledge, then the real-world experiment and* Hyb_1 *are computationally indistinguishable.*

Proof. We can prove this through sequence of hybrids, such that one honest user at a time, we can replace its crs_i and NIZK proof with simulated ones. Any pair of adjacent hybrids are computationally indistinguishable through a straightforward reduction to the zero-knowledge of the underlying NIZK. □

Experiment Hyb_2. Hyb_2 is almost the same as Hyb_1 except that when the experiment needs to compute com_i on behalf of an the honest client $i \in \mathcal{H}$, it now computes a commitment of 0 instead.

Claim A.10. *Suppose that the commitment scheme is computationally hiding, then* Hyb_1 *and* Hyb_2 *are computationally indistinguishable.*

Proof. We can prove this through sequence of hybrids, such that one honest user at a time, we can replace its commitment with a commitment of 0. Any pair of adjacent hybrids are computationally indistinguishable through a straightforward reduction to the computational hiding property of the underlying commitment scheme. □

Experiment Hyb_3. Hyb_3 is is almost identical to Hyb_2 except that we now sample r from $\mathbb{F}\backslash\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]}$ rather than the entire field $\mathbb{F}$.

Claim A.11. Hyb_3 *and* Hyb_2 *have statistical distance at most* $(n-t)\cdot m/|\mathbb{F}|$.

Proof. Straightforward to see. □

Experiment Hyb_4. Hyb_4 is almost identical as Hyb_3 except with the following modifications.

- Given $\mathsf{Multiset}(\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]})$, the experiment reorders the terms $\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]}$ in any arbitrary canonical order (e.g., from small to large). The reordered set is now denoted $\{x'_{i,j}\}_{i\in\mathcal{H},j\in[m]}$.
- Whenever the challenger needs to use $x_{i,j}$ to compute a response to $\mathcal{A}$, use $x'_{i,j}$ instead.

Claim A.12. *Suppose* $n-t \geq 19$, $d \geq \left\lceil \frac{2\lambda+\log_2(|\mathbb{F}|-1)}{\log_2(n-t)-\log_2 e} + 2 \right\rceil$. *Then,* Hyb_4 *and* Hyb_3 *have statistical distance at most* $2^{-\lambda}$.

Proof. Recall that the NIZK proofs and commitments for honest clients have been replaced with simulated proofs and commitments of 0, so they are the same in Hyb_3 and Hyb_4. The remaining terms in Hyb_3 and Hyb_4 have negligibly small in λ statistical distance by Lemma A.5. □

Last but not the least, observe that in Hyb_4, to compute the adversary's view, the experiment only needs to know $\mathsf{Multiset}(\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]})$ and pp but not the honest clients' witnesses. Therefore, the description of Hyb_4 uniquely defines a simulator Sim, such that Hyb_4 can be equivalently viewed as the ideal experiment.

□

A.2 Proofs for the Malicious Verifier Protocol

We modify our shuffle-ZKP definitions in Section 2 to account for a *non-programmable* random oracle:

- Completeness must hold regardless of the choice of the random oracle RO.
- For the soundness definition, one can view RO as an exponentially long random string appended to pp, and the computationally bounded adversary can only query polynomially many locations of RO.
- For the t-zero-knowledge definition, for the real-world experiment, one can also imagine that the RO is an exponentially long random string appended to pp, but the adversary's view only consists of the polynomially many locations it has queried. For the ideal experiment, the simulator Sim is allowed to query polynomially many locations of RO, too.

Theorem A.13 (Soundness of the upgraded protocol)**.** *Suppose the field size $|\mathbb{F}|$ is superpolynomial w.r.t. the security parameter λ, the commitment scheme is perfectly binding, the NIZK scheme satisfies soundness, and $\mathsf{RO}(\cdot)$ is a non-programmable random oracle. Then, the above upgraded protocol satisfies soundness.*

Proof. The first part of the proof is the same as that of Theorem A.3, and we redescribe it below for the reader's convenience. Before the random challenge r is sampled, the server's view contains pp, $\mathsf{Multiset}(\mathbf{x}_1,\ldots,\mathbf{x}_n)$, $\mathsf{Multiset}(\mathbf{y}_1,\ldots,\mathbf{y}_n)$, and $\{\mathsf{com}_i\}_{i\in[n]}$. Except with the negligibly small probability that some underlying NIZK instance's soundness is broken, if the server outputs "`accept`", then for every $i\in[n]$, it must be that there exists $(\mathbf{x}'_i,\gamma_i,w_i,\rho'_i)$ such that

- $\mathcal{R}_i(\mathbf{x}'_i,w_i)=1$;
- $\mathsf{comm}(1^\lambda,(\mathbf{x}'_i,\rho'_i);\gamma_i)=\mathsf{com}_i$; and
- $\prod_{j\in[m]}(x'_{i,j}-r)\cdot\rho'_i=z_i$.

Henceforth we ignore the negligibly small probability that the underlying NIZK's soundness is broken, and assume that the following expressions hold. Since the commitment scheme is perfectly binding, $\mathbf{x}'_i$ and ρ'_i are uniquely determined given com_i. If the server passes verification and the underlying NIZK's soundness is not broken, it must be that

$$\prod_{i\in[n],j\in[m]}(x'_{i,j}-r)\cdot\alpha'=\prod_{i\in[n],j\in[m]}(x_{i,j}-r)\cdot\alpha$$

where

$$\alpha':=\prod_{i\in[n]}\rho'_i,\quad \alpha:=\prod_{i\in[n],j\in[d]}y_{i,j}$$

The main difference from the proof of Theorem A.3 is that to apply Lemma A.2, we need to argue that the even when conditioned on the provers' view right before the challenge r is opened,

r is sampled at random from a sufficiently large random subset of $\mathbb{F}_p$ (unless some negligibly small probability event happens). We give the argument below. Suppose that the provers query the random oracle $\mathsf{RO}(\cdot)$ on a set of inputs denoted Q whose size $|Q|$ is upper bounded by some polynomial in λ. Let Bad be the bad event that there exists $r' \in Q$, such that $\mathsf{RO}(r') = \widetilde{r}$. It is not hard to see that $\Pr[\mathsf{Bad}] \leq \frac{|Q|}{2^\lambda}$ which is negligibly small.

If the event Bad does not happen, then even when conditioned on the provers' view, the challenge r is chosen at random from $\{\lfloor p/2 \rfloor + 1, \ldots, p-1\} \backslash Q$. In this case, using the same argument as the proof of Theorem A.3, except with the negligibly small probability that some bad challenge r is sampled, it must be that $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \mathsf{Multiset}(\mathbf{x}'_1, \ldots, \mathbf{x}'_n)$ where $(\mathbf{x}'_1, \ldots, \mathbf{x}'_n)$ are uniquely determined by $\mathsf{com}_1, \ldots, \mathsf{com}_n$.

Summarizing the above, except with negligible probability, if the server accepts, then there exists a partitioning $(\mathbf{x}'_1, \ldots, \mathbf{x}'_n)$ of the multiset $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ which is uniquely determined by $\mathsf{com}_1, \ldots, \mathsf{com}_n$, such that for every $i \in [n]$, $\mathcal{R}_i(\mathbf{x}'_i, w_i) = 1$.

□

Theorem A.14 (t-zero-knowledge of the upgraded protocol)**.** *Given a security parameter λ, assume $|\mathbb{F}|$ is superpolynomial in λ, $n - t \geq 19$, $d \geq \left\lceil \frac{2\lambda + \log_2(|\mathbb{F}|-1)}{\log_2(n-t) - \log_2 e} + 2 \right\rceil$. Moreover, suppose that the NIZK scheme satisfies zero-knowledge, the commitment scheme is computationally hiding, and $\mathsf{RO}(\cdot)$ is a non-programmable random oracle. Then, the upgraded protocol satisfies t-honest-verifier-zero-knowledge.*

Proof. We now prove t-zero-knowledge of the upgraded protocol.

Real-world experiment. In the real-world experiment, the adversary $\mathcal{A}$ interacts with a challenger who acts on behalf of the honest clients denoted $\mathcal{H}$, and implements the shuffler for $\mathcal{A}$. The experiment is described below:

1. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}})$.
2. $\mathcal{A}(1^\lambda, n, \mathsf{pp})$ outputs $\{\mathbf{x}_i, w_i\}_{i \in \mathcal{H}}$ where $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$ for $i \in \mathcal{H}$ and some state st.
3. Run the protocol Π where the honest clients' inputs are $\{\mathbf{x}_i, w_i\}_{i \in \mathcal{H}}$ and pp, and they interact with the adversary $\mathcal{A}$ acting as the server and the corrupt clients:
   (a) During the experiment, $\mathcal{A}$ can query $\mathsf{RO}(\cdot)$.
   (b) $\mathcal{A}$ outputs a commitment to a challenge $\widetilde{r}$.
   (c) The challenger sends to $\mathcal{A}$ $\mathsf{Multiset}(\{y_{i,j}\}_{i \in \mathcal{H}, j \in [d]})$ where the terms $\{y_{i,j}\}_{i \in \mathcal{H}, j \in [d]}$ are all sampled at random from $\mathbb{F}/\{0\}$;
   (d) The challenger sends to $\mathcal{A}$ the commitments $\{\mathsf{com}_i\}_{i \in \mathcal{H}}$ where $\mathsf{com}_i := \mathsf{commit}((\mathbf{x}_i, \prod_{j \in [d]} y_{i,j}); \gamma_i)$.
   (e) $\mathcal{A}$ outputs a random challenge r.
   (f) If r passes the checks, the challenger sends to the adversary (z_i, π_i) for each $i \in \mathcal{H}$, where $z_i := \prod_{j \in [m]} (x_{i,j} - r) \cdot \prod_{j \in [d]} y_{i,j}$, and π_i is an honestly computed NIZK proof by honest client i.

The first two hybrids Hyb_1 and Hyb_2 basically make the same switches as the earlier honest-verifier zero-knowledge proof, where we switch the NIZK setup for honest clients to simulated setup, switch NIZK proofs for honest clients to simulated proofs, and switch commitments for honest clients to commitments of 0. Assuming the NIZK scheme satisfies zero-knowledge, the commitment scheme is computationally hiding, we can prove the Hyb_0, Hyb_1, Hyb_2 are computationally indistinguishable with similar proofs of Claim A.9 and Claim A.10.

Experiment Hyb_3. Hyb_3 is almost identical as Hyb_2 except with the following modifications.

- Given $\mathsf{Multiset}(\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]})$, the experiment reorders the terms $\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]}$ in any arbitrary canonical order (e.g., from small to large). The reordered set is now denoted $\{x'_{i,j}\}_{i\in\mathcal{H},j\in[m]}$.
- Replace each $x_{i,j}$ with $x'_{i,j}$.

Claim A.15. Hyb_3 *is statistically indistinguishable from* Hyb_2.

Proof. For $i \in \{2,3\}$, define the intermediate hybrid Hyb'_i that makes the following modification: if the adversary ever finds two distinct inputs r_0 and r_1 such that $\mathsf{RO}(r_0) = \mathsf{RO}(r_1)$, the experiment simply aborts outputting `collision` and we redefine the adversary's view to be $\perp$. This intermediate hybrid Hyb'_i has negligibly small statistial distance from Hyb_i if the field size is super-polynomial.

To conclude the proof, observe that Hyb'_2 and Hyb'_3 have negligibly small statistial distance due to Lemma A.5. In particular, in Hyb'_2 and Hyb'_3, conditioned on not aborting outputting `collision`, effectively the adversary has uniquely committed to the challenge r when it submits $\widetilde{r}$. Note that to apply Lemma A.5, the $\{y_{i,j}\}_{i\in\mathcal{H},j=[d]}$ values have to be sampled independently of the challenge r, and r has to be different than any $x_{i,j}$ (otherwise $x_{i,j} - r = 0$ will not be in the multiplicative group $\mathbb{F}/\{0\}$). Also, by construction, as long as the opened challenge r passes the check, it is guaranteed that r does not collide with any $x_{i,j}$ since r is from a different half of the field $\mathbb{F}$. □

Last but not the least, observe that in Hyb_3, to compute the response to $\mathcal{A}$, the challenger only needs to know $\mathsf{Multiset}(\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]})$ and pp but not the honest clients' witnesses. Therefore, the description of Hyb_3 uniquely defines a simulator Sim, such that Hyb_3 can be equivalently viewed as the ideal experiment.

□

B Proofs for the Blame Protocol

We modify the definition of the real and the ideal experiments in the definition of t-zero-knowledge to fit the new syntax of the blame protocol. We say that a $\mathsf{Shuffle\text{-}ZKP}$ scheme (with blame attribution) for the NP relations $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ satisfies t-zero-knowledge, iff there exists a PPT simulator Sim such that for any non-uniform PPT adversary $\mathcal{A}$ controlling the server and a set of at most t clients denoted $\mathcal{C} \subset [n]$, the adversary's views in the following two experiments are computationally indistinguishable where $\mathcal{H} := [n]\backslash\mathcal{C}$:

1. *Real*:
   - $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}})$; honestly sample the honest players' $\{\mathsf{sx}_{i,j}, \beta_{i,j}, \widetilde{\mathsf{sx}}_{i,j}\}_{i\in\mathcal{H},j\in[m]}$;
   - $\mathcal{A}(1^\lambda, n, \mathsf{pp}, \{\widetilde{\mathsf{sx}}_{i,j}\}_{i\in\mathcal{H},j\in[m]})$ outputs $\{\mathbf{x}_i, w_i\}_{i\in\mathcal{H}}$ where it is required that $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$ for $i \in \mathcal{H}$;
   - run the protocol Π where $\mathcal{A}$ controls $\mathcal{C}$ and the server, and honest clients inputs are $\{\mathbf{x}_i, w_i\}_{i\in\mathcal{H}}$ as well as $\{\mathsf{sx}_{i,j}, \beta_{i,j}\}_{i\in\mathcal{H},j\in[m]}$.
2. *Ideal*:
   - $(\mathsf{pp}, \tau, \{\widetilde{\mathsf{sx}}_{i,j}\}_{i\in\mathcal{H},j\in[m]}) \leftarrow \mathsf{Sim}(1^\lambda, n, \boldsymbol{\mathcal{R}})$;
   - $\mathcal{A}(1^\lambda, n, \mathsf{pp}, \{\widetilde{\mathsf{sx}}_{i,j}\}_{i\in\mathcal{H},j\in[m]})$ outputs $\{\mathbf{x}_i, w_i\}_{i\in\mathcal{H}}$ where it is required that $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$ for $i \in \mathcal{H}$;

- $\mathcal{A}$ now interacts with the simulator $\mathsf{Sim}(\tau, \{\mathbf{x}_i\}_{i\in\mathcal{H}})$ which is provided only the *multiset* of the honest clients' input items $\mathsf{Multiset}(\{\mathbf{x}_i\}_{i\in\mathcal{H}})$ but not the witnesses.

Theorem B.1 (Restatement of Theorem 6.2)**.** *The full protocol is zero-knowledge.*

Proof. Consider the following sequence of hybrid.

Experiment Real**.** This is the real-world protocol.

Experiment Hyb_0**.** Almost the same as Real except that we use the simulated setup algorithm for the NIZK schemes and get a trapdoor. All NIZK proofs are now simulated with the trapdoor and the simulated CRS.

Hyb_0 is computationally indistinguishable from Real through a straightforward reduction to the zero-knowledge property of the NIZK scheme.

Experiment Hyb_1**.** Almost the same as Hyb_0 except that the PIR queries are simulated and the response is ignored.

Hyb_1 is computationally indistinguishable from Hyb_0 through a reduction to the security of the PIR scheme.

Experiment Hyb_2**.** Almost identical as Hyb_1 except for any honest client i, we replace every $\widetilde{\mathsf{sx}}_{i,j}$ and $\widetilde{\mathsf{sy}}_{i,j}$ with a fresh commitment of 0.

Hyb_2 is computationally indistinguishable from Hyb_1 through a reduction to the computational hiding property of the commitment scheme.

Note that in Hyb_2, the honest clients' serial numbers $\{\mathsf{sx}_{i,j}\}_{i\in\mathcal{H},j\in[m]}$ and $\{\mathsf{sy}_{i,j}\}_{i\in\mathcal{H},j\in[d]}$ are no longer dependent on the committed serial numbers $\{\widetilde{\mathsf{sx}}_{i,j}\}_{i\in\mathcal{H},j\in[m]}$ and $\{\widetilde{\mathsf{sy}}_{i,j}\}_{i\in\mathcal{H},j\in[d]}$. Therefore, the challenger need not sample the serial numbers $\{\mathsf{sx}_{i,j}\}_{i\in\mathcal{H},j\in[m]}$ upfront. In fact, it can delay the sampling of $\{\mathsf{sx}_{i,j}\}_{i\in\mathcal{H},j\in[m]}$ and $\{\mathsf{sy}_{i,j}\}_{i\in\mathcal{H},j\in[d]}$ to exactly when the honest clients actually have to send their serial numbers to the server.

Experiment Hyb_3**.** Almost identical as Hyb_2 except that during the main phase of the protocol, we call the simulator of the main phase (see Appendix A.2) with the input $\mathsf{Multiset}(\{\mathbf{x}_i\}_{i\in\mathcal{H}})$ to interact with the adversary. Moreover, the simulator needs to bind a commitment of 0 to every message it needs to send to $\mathcal{F}_{\text{shuffle}}$.

Hyb_3 is computationally indistinguishable from Hyb_2 through a straightforward reduction to the t-zero-knowledge property of the main phase protocol (proven in Appendix A.2).

Moreover, in Hyb_3, the challenger only needs to know $\mathsf{Multiset}(\{\mathbf{x}_i\}_{i\in\mathcal{H}})$ to interact with the adversary, so it specifies the simulator and our ideal-world experiment.

□

Theorem B.2 (Restatement of Theorem 6.3)**.** *The protocol satisfies misbehavior identifiability.*

Proof. We first prove no false implication given the underlying commitment scheme is computationally hiding. Specifically, a client i can only be included in the report if one of the following bad events happen: 1) its proof does not verify; or 2) a collision of the form $\mathsf{sx}_{i,j} = \mathsf{sx}_{i',j'}$ or $\mathsf{sy}_{i,j} = \mathsf{sy}_{i',j'}$ is detected. If the underlying NIZK satisfies completeness, the first bad event will never occur.

We now prove that the second bad event does not happen except with negligible probability. First, since honest client i's the serial numbers are sampled at random, an internal collision $\mathsf{sx}_{i,j} =$

$\mathsf{sx}_{i,j'}$ or $\mathsf{sy}_{i,j} = \mathsf{sy}_{i,j'}$ cannot happen except with negligible probability. We now show that an external collision $\mathsf{sx}_{i,j} = \mathsf{sx}_{i',j'}$ or $\mathsf{sy}_{i,j} = \mathsf{sy}_{i',j'}$ also does not happen except with negligible probability.

Imagine that a challenger simulates the behavior of all honest clients and the server, and interacts with the adversary controlling the corrupt clients. The adversary can query the whole $\widetilde{S}_x$ and $\widetilde{S}_y$. The adversary wins if any corrupt player i' can pass the proof verification, and moreover, it submits a serial number that collides with any serial number chosen by the challenger.

We can build a hybrid experiment that instead of revealing the true $\widetilde{\mathsf{sx}}_{i,j}$ and $\widetilde{\mathsf{sy}}_{i,j}$ values to the adversary during the PIR phase, we simply reveal fresh commitments of 0.

The probability that any efficient adversary wins in Hyb_0 is negligibly apart from the probability that it wins in Real through a straightforward reduction to the computational hiding property of the commitment scheme, assuming the honest parties are using secure communication channels.

Now, the view of the adversary is not related to the true serial numbers chosen by the challenger. Therefore, we can equivalent imagine that the true honest serial numbers $\mathsf{sx}_{i,j}$ and $\mathsf{sy}_{i,j}$ are only sampled at the very end when the experiment needs to determine if the adversary wins. Therefore, the probability that the adversary wins is negligibly small.

Blame completeness. Consider the main stage and the blame protocols. Let each client i's input be $\{\mathbf{x}_i, w_i\}$ as well as $\{\mathsf{sx}_{i,j}, \beta_{i,j}\}_{i\in[n],j\in[m]}$. We assume that for any honest client i, its input w_i is a valid witness for the relation $\mathcal{R}_i$. We want to show that assuming that the Merkle hash tree satisfies collision resistance, the NIZK scheme is sound, and the commitment scheme is perfectly binding, except with negligible probability, the following holds: if the server is honest but the main stage protocol fails verification, then, at least one corrupt client will be reported.

Only the following bad events can cause the main protocol to fail verification:

1. $\mathcal{F}_{\text{shuffle}}$ aborted. As long as $\mathcal{F}_{\text{shuffle}}$ satisfies identifiable abort, our protocol reports whoever $\mathcal{F}_{\text{shuffle}}$ reports.

2. There is any invalid proof. In this case, our protocol those who submits it.

3. All proofs verify but the product check failed.

It suffices to show that except with negligible probability, in Case 3, our protocol will report at least one corrupt client too. We prove this below. If some corrupt client's proof in the blame phase fails to verify, then the claim trivially holds. Ignoring the negligible probability that the adversary can break the collision resistance of the Merkle hash tree, or the soundness of NIZK, or the computational binding property of the commitment scheme, as long as all corrupt clients' proofs verify, the following claims must be true for every corrupt client i:

- com_i commits some to $\mathbf{x}'_i = (x'_{i,1}, \ldots, x'_{i,m})$ and ρ_i such that the z_i revealed to the server during the main stage is the correct product $z_i = \prod_{j\in[m]}(x'_{i,j} - r) \cdot \rho_i$; moreover, there exist $\{\widetilde{\mathsf{sx}}_{i,j}\}_{j\in[m]}$ such that each $(x'_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ belong to $\widetilde{S}_x$ and $\widetilde{\mathsf{sx}}_{i,j}$ is a correct commitment of $\mathsf{sx}_{i,j}$;
- there exist $(y'_{i,1}, \widetilde{\mathsf{sy}}_{i,1}), \ldots, (y'_{i,d}, \widetilde{\mathsf{sy}}_{i,d})$, such that each $(y'_{i,j}, \widetilde{\mathsf{sy}}_{i,j})$ belongs to $\widetilde{S}_y$, and $\rho_i = \prod_{j\in[d]} y'_{i,j}$, and moreover each $\widetilde{\mathsf{sy}}_{i,j}$ is a correct commitment of $\mathsf{sy}_{i,j}$.

We now consider the following cases. First, suppose that in the above, there is some corrupt $(x'_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ tuple or some some corrupt $(y'_{i,j}, \widetilde{\mathsf{sy}}_{i,j})$ tuple equal to an honest client's submission. Since the $\widetilde{\mathsf{sx}}_{i,j}$ (or $\widetilde{\mathsf{sy}}_{i,j}$) binds to a unique serial number and the NIZK is sound, the server must detect a collision in the revealed serial numbers, and will therefore report corrupt client i. Henceforth, we may assume that the corrupt $(x'_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ tuples and $(y'_{i,j}, \widetilde{\mathsf{sy}}_{i,j})$ tuples are not equal to any

tuple submitted by an honest client. Since the verification of the main phase failed but all proofs verified, it must be that $\prod_{i\in\mathcal{C},j\in[m]}(x'_{i,j}-r)\cdot\prod_{i\in\mathcal{C},j\in[d]}y'_{i,j}\neq\prod_{i\in\mathcal{C},j\in[m]}(x_{i,j}-r)\prod_{i\in\mathcal{C},j\in[d]}y_{i,j}$ where $\mathcal{C}\subset[n]$ denotes the set of corrupt clients, and $\{x_{i,j}\}_{i\in\mathcal{C},j\in[m]}$ and $\{y_{i,j}\}_{i\in\mathcal{C},j\in[d]}$ are the corrupt clients' contributions to $\widetilde{S}_x$ and $\widetilde{S}_y$ respectively. Further, we also know that each $(x'_{i,j},\widetilde{\mathsf{sx}}_{i,j})$ and $(y'_{i,j},\widetilde{\mathsf{sy}}_{i,j})$ must come from the corrupt clients' contributions to $\widetilde{S}_x$ and $\widetilde{S}_y$ respectively. This means that there must be some $(x'_{i_0,j_0},\widetilde{\mathsf{sx}}_{i_0,j_0})=(x'_{i_1,j_1},\widetilde{\mathsf{sx}}_{i_1,j_1})$ or $(y'_{i_0,j_0},\widetilde{\mathsf{sy}}_{i_0,j_0})=(y'_{i_1,j_1},\widetilde{\mathsf{sy}}_{i_1,j_1})$ where $(i_0,j_0)\neq(i_1,j_1)$ and $i_0,i_1\in\mathcal{C}$. Therefore, the server can detect a collision when the corrupted clients open their serial numbers and the corrupt clients i_0 and i_1 will be reported.

□

C Extension: Using a Computationally Binding Commitment

In practice, we often want to use a computationally binding commitment instead of a perfectly binding one, since such commitments can be succinct. In this section, we provide the modified proofs for our protocol for the case of a computationally binding commitment.

Computationally binding, non-interactive commitment. A non-interactive computationally binding commitment scheme for messages of length ℓ has the following possibly randomized algorithms:

- $\mathsf{pp}\leftarrow\mathsf{Gen}(1^\lambda)$: takes in a security parameter in unary form 1^λ and samples the public parameters pp;
- $\mathsf{com}\leftarrow\mathsf{commit}(\mathsf{pp},x)$: takes in the public parameters pp, a message $x\in\{0,1\}^\ell$, and outputs a committed value com. Sometimes we write $\mathsf{commit}(\mathsf{pp},x;r)$ to mean that r is the random coins consumed by the commit algorithm.

We want the following security properties.

- *Perfectly hiding.* For any λ, with probability 1 over the choice of $\mathsf{pp}\leftarrow\mathsf{Gen}(1^\lambda)$, it must be that $\mathsf{commit}(\mathsf{pp},x)$ is identically distributed as $\mathsf{commit}(\mathsf{pp},x')$.
- *Computationally binding.* For any non-uniform PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[\mathsf{pp}\leftarrow\mathsf{Gen}(1^\lambda),(x,r,x',r')\leftarrow\mathcal{A}(1^\lambda,\mathsf{pp}):\mathsf{commit}(\mathsf{pp},x;r)=\mathsf{commit}(\mathsf{pp},x';r')\wedge x\neq x'\right]\leq\mathsf{negl}(\lambda)$$

Modified proofs when using a computationally binding commitment. If we switch the non-interactive commitment scheme to a computationally binding one, the honest-verifier-zero-knowledge or zero-knowledge proofs still work. We now give a modified proof of soundness assuming that the underlying NIZK satisfies knowledge soundness.

Theorem C.1 (Soundness (variant))**.** *Suppose that the field size $|\mathbb{F}|$ is superpolynomial in the security parameter λ, the commitment scheme is perfectly hiding and computationally binding, and the NIZK satisfies knowledge soundness. Then, our* Shuffle-ZKP *protocol of Section 5.1 satisfies soundness.*

Proof. Suppose that the underlying NIZK satisfies knowledge soundness. This means that there exists an efficient extractor $\mathcal{X}_\mathcal{A}$ such that except with negligible probability, if the server outputs "`accept`", the extractor can output $\{(\mathbf{x}'_i,\gamma_i,w_i,\rho'_i)\}_{i\in[n]}$ such that

- $\mathcal{R}_i(\mathbf{x}'_i, w_i) = 1$;
- $\mathsf{commit}(1^\lambda, (\mathbf{x}'_i, \rho'_i); \gamma_i) = \mathsf{com}_i$; and
- $\prod_{j\in[m]}(x'_{i,j} - r) \cdot \rho'_i = z_i$.

The rest of the proof follows in the same manner as the proof of Theorem A.3. □

Theorem C.2 (Soundness of the upgraded protocol (variant))**.** *Suppose that the field size $|\mathbb{F}|$ is superpolynomial in the security parameter λ, the commitment scheme is perfectly hiding and computationally binding, and the NIZK satisfies knowledge soundness. Then, the upgrade protocol of Section 5.2 satisfies soundness.*

Proof. Suppose that the underlying NIZK satisfies knowledge soundness. This means that there exists an efficient extractor $\mathcal{X}_\mathcal{A}$ such that except with negligible probability, if the server outputs "`accept`", the extractor can output $\{(\mathbf{x}'_i, \gamma_i, w_i, \rho'_i)\}_{i\in[n]}$ such that

- $\mathcal{R}_i(\mathbf{x}'_i, w_i) = 1$;
- $\mathsf{commit}(1^\lambda, (\mathbf{x}'_i, \rho'_i); \gamma_i) = \mathsf{com}_i$; and
- $\prod_{j\in[m]}(x'_{i,j} - r) \cdot \rho'_i = z_i$.

The rest of the proof follows in a similar way as the proof of Theorem A.13 by arguing that unless the adversary submitted a random oracle query and found the pre-image of the committed challenge, the random challenge r must be sampled at random from a sufficiently large space even when conditioned on the adversary's view right before the challenge r is opened. □

D Shuffle-DP Mechanisms with Verifiable Randomness

Differentially private mechanisms intrinsically rely on randomness to perturb the secrets and achieve privacy. The random pertubation poses a challenge for compliance checking and the following example shows the dilemma.

Suppose there are multiple clients and each client has a secret value, denoted as v. As in the shuffle-DP summation protocol [BBGN20], the mechanism adds a random noise y to the value and then the server and the clients run some secure summation protocol based on shuffling, which outputs the sum of all the noisy values. Now, suppose a malicious client wants to perturb the final outcome. It can simply have a valid input v (e.g. within a certain range), but instead of drawing the noise from the right distribution, it can abitrarily generate a random noise y. However, with some small probability, an honest client could also randomly draw a noise that is significantly deviating from the mean. It seems like we do not have a good way to distinguish an anomaly noise is from malicious behaviors or just from the honest behavior with bad luck.

Biswas and Cormode [BC22] proposed verifiable differential privacy in the trusted curator model. Inspired by their work, we briefly introduce a generic construction for verifiability for the shuffle-DP model, combining Shuffle-ZKP protocol and Blum's coin tossing protocol [Blu83]. Since we aim to remove malicious client behaviors, this protocol assumes a semi-honest server.

We abstract the local perturbation function of any shuffle-DP mechanism as $\mathcal{M}(v; \rho)$ where x is the secret input and ρ is a binary random string consumed by the mechanism, such that $\mathcal{M}(v; \rho)$ is a deterministic function. Each client sends all the outputs of $\mathcal{M}(v; \rho)$ to the shuffler, and the server only observes the shuffled results. The server then computes some useful statistics based on the shuffled set. For example, in the private summation protocol [BBGN20], the randomness is used to generate two discrete negative binomial noises and for random rounding and the mechanism outputs

many additive shares of the sum of the noise and the original input. Or, in the private histogram protocol [GGK+21], the randomness is used to generate dummy items and the mechanism outputs the dummies along with the original input item.

Now, suppose ρ is a private random string that is guaranteed to be unbiased, a client can use our Shuffle-ZKP protocol to prove that: 1) v is a valid input; 2) given ρ, $x_1, \ldots, x_m$ are indeed the correct outputs of $\mathcal{M}(v; \rho)$. Assuming the soundness of the underlying ZK proof system, we enforce each client to follow the right perturbation process, and malicious client's only hope is to perturb the random coin ρ.

We can use a variant of Blum's coin tossing protocol to avoid this loophole. A client is required to first locally sample a random string, say ρ_1 and send the commitment of ρ_1 to the server. The server then sends back a random string ρ_2 to the client. Now, the client uses $\rho = \rho_1 \oplus \rho_2$ as the random string in the mechanism $\mathcal{M}$. In addition, during the execution of Shuffle-ZKP protocol, the client also needs prove that: 1) the commitment of ρ_1 is computed correctly; 2) ρ is computed correctly, $\rho = \rho_1 \oplus \rho_2$. The hiding property of the commitment and the zero-knowledge property of the underlying proof system ensure that the server still learns nothing about the random string ρ, thus the shuffle-DP mechanism remains private. The binding property of the commitment ensures that $\rho = \rho_1 \oplus \rho_2$ is unbiased, as long as the server indeed samples an unbiased ρ_2.

This protocol adds one extra round of communication that the clients need to send a commitment first and the server has to send a random string back to each client.

Remark. Although being clean in terms of theoretical description, the protocol might not be directly practical. One of the main obstacles is that the most of the randomized functions used in shuffle-DP mechanisms, i.e., those $\mathcal{M}(v; \rho)$, heavily rely on floating number operations (e.g. to sample noise from some distributions). Supporting floating number operations in ZK proof systems are notoriously challenging [GJJZ22]. Therefore, we believe that designing simple mechanisms tha can be efficiently proven in ZK systems is an interesting open problem for future shuffle-DP research.