CONAN: Distributed Proofs of Compliance for Anonymous Data Collection

Mingxun Zhou Elaine Shi Giulia Fanti

Carnegie Mellon University

Abstract

We consider how to design an anonymous data collection protocol that enforces compliance rules. Imagine that each client contributes multiple data items (e.g., votes, location crumbs, or secret shares of its input) to an anonymous network, which mixes all clients' data items so that the receiver cannot determine which data items belong to the same user. Now, each user must prove to an auditor that the set it contributed satisfies a compliance predicate, without identifying *which* items it contributed. For example, the auditor may want to ensure that no voter voted for the same candidate twice, or that a user's location crumbs are not too far apart in a given time interval.

Our main contribution is a novel *anonymous, compliant data collection* protocol that realizes the above goal. In comparison with naïve approaches such as generic multi-party computation or earlier constructions of collaborative zero-knowledge proofs, the most compelling advantage of our approach is that each client's communication and computation overhead do not grow with respect to the number of clients n. In this sense, we save a factor of at least n over prior work, which allows our technique to scale to applications with a large number of clients, such as anonymous voting and privacy-preserving federated learning.

We first describe our protocol using generic cryptographic primitives that can be realized from standard assumptions. We then suggest a concrete instantiation called CONAN which we implement and evaluate. In this concrete instantiation, we are willing to employ SNARKs and the random oracle model for better practical efficiency. Notably, in this practical instantiation, each client's additional communication overhead (not counting the overhead of sending its data items over the anonymous network) is only $\widetilde{O}(1)$. We evaluated our technique in various application settings, including secure voting, and secure aggregation protocols for histogram, summation, and vector summation. Our evaluation results show that in all scenarios, each client's additional communication overhead is only 2.2KB or 2.6KB, depending on which SNARK implementation we use. Further, each client's computation only 0.2s - 0.5s for almost all cases, except for the vector summation application where the data items are high-dimensional and each client's computation is 8.5-10.6s.

1 Introduction

Anonymous data collection [Cha88, Cha81, Abe99, DMS04, GRS99, FM02, MB09, RR98, DD08, EY09, SSA+18, ZZZR05, SBS02, SW21] has been used in many privacy-enhancing applications. For example, in anonymous voting [MV98] or anonymous surveys [HMPS14], users' votes and opinions are collected over an anonymous network. In privacy-preserving federated learning [BIK+17], the popular "shuffle model" [CJMP21, BEM+17, GDD+21, BBGN20, BBGN19, GGK+21, GMPV20, Che21, BC19, LCC+21] anonymously collects noisy data from participating clients, such that the server can perform statistical analysis and learning tasks without learning which client contributed what data. Previous works have shown that, anonymity significantly amplifies privacy in the context of differential privacy: if we fix the amount of noise each client adds to their data, then the privacy guarantee is proven to be much stronger if the data collection is performed anonymously rather than in the plain model without anonymity [CJMP21, BEM+17, GDD+21, BBGN20, BBGN19, GGK+21, GMPV20, Che21, BC19, LCC+21].

In this paper, we focus on how to check the compliance of the data contributed by anonymous participants. Specifically, imagine that each client $i \in [n]$ submits a set of data items $\{x_{i,1}, \ldots, x_{i,m}\}$, and the anonymous network (also called the shuffler) randomly permutes all data items $\{x_{i,j}\}_{i\in[n],j\in[m]}$, and sends the unordered multiset $\mathsf{Multiset}(\{x_{i,j}\}_{i\in[n],j\in[m]})$ to the data collector (also called the server). The server wants to ensure that each client i's contributions $\{x_{i,j}\}_{j\in[m]}$ satisfy some compliance predicate C. For example, in an anonymous voting scenario, the server wants to check that the multiple votes cast by the same voter must vote for distinct candidates. In a privacy-preserving federated learning scenario, we might want to check that the data items contributed by each client satisfies some robustness condition. For example, in a multi-message shuffle-model protocol for secure summation or frequency estimation [BBGN20, BBGN19, GGK+21, GMPV20], each client adds noise to its input and submits secret shares of its noisy input to the shuffler. In this case, we may want to verify that the summation of each client's shares lies within some appropriate range.

Enforcing compliance is challenging because the shuffler breaks linkability among the multiple items contributed by the same client and mixes them together with all other clients' contributions. In particular, breaking up the linkability among the same client's items is essential for numerous privacy-preserving protocols [IKOS06, BBGN20, GGK+21, BBGN19]. Our goal is to check compliance of each client's contributions without breaking anonymity. This means that we cannot reveal which data items belong to which user, or even whether two data items belong to the same user. Henceforth, we refer to this task as an *anonymous, compliant data collection protocol.*

To get a better feel of this problem, it helps to consider a couple naïve solutions and see why they do not work.

A flawed solution. A straightforward idea is the following. Suppose that the server receives the unordered multiset $\mathsf{Pool} := \mathsf{Multiset}(\ \{x_{i,j}\}_{i\in[n],j\in[m]})$ from the shuffler, where $\{x_{i,j}\}_{j\in[m]}$ are contributions from the i-th client. Now, the server and the clients run an audit protocol: each client i proves in zero knowledge that it knows a set of items $\mathbf{x}'_i := (x'_{i,1}, \ldots, x'_{i,m})$ such that

1. for $j \in [m]$, item $x'_{i,j}$ belongs to Pool, e.g., by showing that there exists a valid Merkle tree proof for $x'_{i,j}$ w.r.t. the Merkle digest of Pool; and
2. the set $\mathbf{x}'_i$ satisfy the compliance predicate C.

Unfortunately, this approach is flawed due to the following reason. Suppose that the adversary $\mathcal{A}$ controls a subset of the clients. As long as one of the colluding clients i submitted compliant data $\mathbf{x}_i$ in the data collection phase, all the colluding clients can use client i's contribution $\mathbf{x}_i$ to pass the audit protocol. In other words, the problem is that this protocol did not verify the $\mathbf{x}'_1, \ldots, \mathbf{x}'_n$ purported in the audit phase is indeed a disjoint partitioning of the Pool of data items collected earlier — henceforth for convenience, we call this property *set consistency.*

Besides the security flaw, the above protocol may also be inefficient since each client needs to find the Merkle proof for each data item it contributed. The straightforward approach is for each client to also download the entire Pool and compute the Merkle proofs, but this incurs per-client communication that is linear in n.

Naïve MPC-based solution. Another generic but inefficient approach is to use a *maliciously* secure multi-party computation (MPC) protocol for the audit, where each client i's input is the set $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,m})$, and the server's input is the unordered multiset Pool obtained from the shuffler. Now, all parties engage in an MPC protocol to securely evaluate a circuit which outputs a bit indicating whether the following conditions are both satisfied: 1) $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is a disjoint partitioning of Pool, and 2) for each $i \in [n]$, $\mathbf{x}_i$ satisfies the desired compliance predicate C.

Unfortunately, generic MPC is expensive. Using known MPC implementations [EKR18, Ors22, KSS13], the per-client communication would be linear in n. Although techniques exist for MPC with sublinear communication (e.g. using Fully Homomorphic Encryption), it is not clear how to make these techniques work when the server is potentially malicious, without incurring linear in n communication per client — see Appendix E for further discussions.

Question. We ask the following question:

> Assuming an underlying anonymous network (modelled as a shuffler ideal functionality), can one design an efficient *anonymous, compliant data collection* protocol such that each client's communication and computation do *not grow w.r.t. n*?

1.1 Results and Contributions

We give an affirmative answer to the above question. We design a novel anonymous, compliant data collection protocol assuming that the underlying network is anonymous. Our protocol proceeds in two phases: 1) a *data collection phase* where each client simply submits its data items through the anonymous network, and 2) an *audit phase* where the server and the clients engage in an interactive protocol to check compliance. Notably, in comparison with other generic approaches such as those based on MPC [EKR18, Ors22, KSS13] or collaborative ZKP [OB22, DPP⁺22], the most compelling advantage of our approach is that per-client communication and computation do not grow w.r.t. the number of clients n, but depend only on the number of data items each client contributes m and the circuit for checking compliance. This is crucial for scaling to applications with large n, e.g., anonymous voting or privacy-preserving federated learning.

We make novel contributions both on the theoretical and practical fronts.

Theoretical contribution: a succinct anonymous, compliant data collection protocol. To state our theoretical contribution, we use generic cryptographic primitives which can be realized from standard cryptographic assumptions. Specifically, we prove the following theorem where $T(c)$, $S(c)$, and $V(c)$ denote the prover time, proof size, and verification time of a non-interactive zero-knowledge (NIZK) proof system when proving a circuit of size c.

Theorem 1.1 (Informal). *Assume a NIZK scheme with the above costs, and a committing public-key encryption scheme. Moreover, assume the existence of an anonymous network (modeled as a shuffler ideal functionality). Then, there exists an anonymous, compliant data collection protocol that is sound even when all of the clients are corrupted, and t-anonymous*[1] *when all but one client is corrupted with the following costs:*

- Each client's communication is upper bounded by $O(m) + S(|C| + O(m)) + \widetilde{O}(1)$ and its computation is upper bounded by $O(m) + T(|C| + O(m)) + \widetilde{O}(1)$, where m is the length of the client's input, and $|C|$ denotes the size of the circuit that encodes the compliance predicate C.

[1] We define t-anonymity in Section 3.3 to mean that the adversary only learns the multiset of the $(n - t)$ honest clients' data items even when it controls the server and t clients $(0 \le t \le n - 1)$.

- The server's computation is upper bounded by $n \cdot V(|C| + O(m)) + O(n \cdot m) + \widetilde{O}(n)$ and its communication is bounded by $n \cdot S(|C| + O(m)) + O(n \cdot m) + \widetilde{O}(n)$.
- The protocol has only *3 rounds of client-server interaction*, and there is *no client-client interaction.*

Note that throughout, the $\widetilde{O}(1)$ term in the client's communication and computation actually spells out to $\omega\left(\frac{\log \lambda}{\log(n-t)}\right)$ where t is the number of corrupted clients. So in fact, *each client's communication and computation decrease as the number of honest clients grows* — see also our evaluation results Figure 3. Intuitively, this is because with a larger n, the mixing effect of the anonymous network allows us to achieve the same level of security with cheaper cost (and jumping ahead, using a smaller number of decoy terms). Similarly, the $\widetilde{O}(n)$ additive term in the server's communication and computation spells out to $n \cdot \omega\left(\frac{\log \lambda}{\log(n-t)}\right)$.

Practical instantiation and concrete efficiency. For our practical implementation, we do not restrict ourselves to using only standard assumptions for better efficiency. Specifically, we will instantiate the NIZKs using Succinct Non-Interactive Zero-Knowledge Arguments of Knowledge (SNARKs), and we allow the random oracle model. In this case, we can state even tighter bounds on the cost, that is,

- Each client's *extra* communication (besides sending the data items to the shuffler) is only $\widetilde{O}(1)$;
- Each client's computation is $\widetilde{O}(|C|)$ where $\widetilde{O}(\cdot)$ hides a logarithmic factor;
- The server's communication and computation are upper bounded by $O(nm) + \widetilde{O}(n)$.

We created an implementation of our protocol which we call CONAN (short for COmpliant N ANonoymous[2]). Our code is open sourced at `https://github.com/shufflezkp/shuffle-zkp-open`. We evaluated our technique in various application settings, including secure voting, and secure aggregation protocols for histogram, summation, and vector summation — see Section 1.2 for more details about these applications. Our evaluation results show that in all scenarios, each client's additional communication overhead is only 2.2 KB or 2.6 KB, depending on whether we use Groth16 [Gro16] or Plonk [GWC19] to instantiate the SNARK. Further, each client's computation only 0.2s to 0.5s for almost all cases, except for the vector summation application where the data items are high-dimensional and as a result, each client's computation is 8.5-10.6s.

New techniques. Interestingly, our techniques are inspired by techniques from the distributed differential privacy (DP) literature [BBGN20], which is also related to Ishai et al.'s result for building cryptographic protocols for anonymous communication networks [IKOS06]. Importantly, we stress that although we use DP-inspired techniques, we actually prove *cryptographically strong notions of security, not differential privacy.* We give an informal overview of our ideas in Section 2.

Definitional contribution. We also make a new conceptual contribution by formulating the anonymous, compliant data collection problem and the corresponding security definitions. We believe that this abstraction can be useful in numerous application scenarios — see Section 1.2 for more discussion.

[2]In our protocol, the server plays Detective Conan [Aoy] and will detect any cheating behavior.

Extension: attributing blame. In our basic anonymous, compliant data collection protocol, there is no recourse if the audit phase fails. In reality, it may be desirable for the server to identify a subset of the cheating clients that caused the protocol to fail. In Appendix C, we propose an extension of our basic protocol that supports *identifiable abort*, such that should the audit fail, the auditor can run an additional "blame protocol" to catch a subset of the cheating clients. We also implement this extension for one application (shuffle-DP sum) and evaluate it in our experiments. The results show that the additional per-client communication and computation overheads are within reasonable bounds, being 1.0 MB and 4.6s, respectively.

1.2 Potential Applications

We discuss some potential applications of our anonymous, compliant data collection protocol. These are also the applications used in our experimental evaluation (Section 7):

- *Privacy-preserving federated analytics in the shuffle model.* In privacy-preserving federated analytics, an untrusted server wants to learn some statistics over of n clients' inputs, without compromising each client's privacy. Earlier work [BBGN20,BBGN20,BBGN19,GGK$^+$21,GMPV20] showed that the multi-message shuffle-model is a promising approach for designing differentially private aggregation protocols. In this model, each client uses multiple messages that jointly encodes its data, and send them to a trusted shuffler. The shuffler mixes all data items and send them to the server who then performs statistical analytics on the received data. Our work can be viewed as a cryptographic protocol that upgrades a traditional shuffler without compliance checking to a *robust shuffler with compliance checking.* Therefore, an immediate application of our work is to make multi-message shuffle-model protocols [BBGN20, BBGN20, BBGN19, GGK$^+$21, GMPV20] robust to data corruption attacks.

  In our evaluation (Section 7), we implement the multi-message shuffle-model protocols proposed Balle et al. [BBGN20]. Specifically, each client adds noise to its input, splits the noisy input into random shares, and sends the shares to the shuffler. We use our Conan protocol to verify that each user's noisy input (i.e., the summation of its shares) is within some appropriate range.

- *Secure histogram protocol.* Imagine that each user has watched a set of movies. A server wants to compute the popularity of each movie (i.e., how many users have watched it). To achieve this, each user sends all the movies it has watched to the shuffler, and the shuffler randomly permutes all entries and sends them to the server. Note that in this protocol, the server learns only the resulting histogram and nothing else. In the above secure histogram protocol, we can use our Conan protocol to ensure that each client cannot submit duplicate entries.

- *Anonymous Condorcet voting.* We also consider an anonymous Condorcet voting [You88] scenario in our evaluation. Specifically, each user has a ranking among the candidates. Based on this ranking, the user submits to the shuffler a set of votes that encodes its preference over every pair of candidates. For example, if the ranking is $A > B > C > D$, the votes submitted are $A > B$, $A > C$, $A > D$, $B > C$, $B > D$ and $C > D$. After obtaining the shuffled votes, the server can tally for each pair of candidates, how many votes rank each candidate over the other, and this will be used to decide the outcome of the election. It is not hard to see that the server learns only the result of the tally and nothing else. In such an application, we can use our Conan protocol to ensure that each user submits only one vote for each pair of candidates; moreover, the votes submitted by each user should be internally consistent, i.e., its rankings should not form a cycle.

1.3 Additional Related Work

Since our work can be viewed as an efficient distributed ZKP protocol over an anonymous network where the witness is partitioned across multiple clients, we review other related notions of distributed ZKPs and explain why they fail to solve our problem.

A line of work considered distributed zero-knowledge proofs (ZKP) where the witness is partitioned or secretly-shared across n provers [OB22, DPP$^+$22]. Although these works can be used in our context for the clients to jointly generate a compliance proof, the communication overhead per client would be at least linear in n.

Another line of work [WZC$^+$18, XZC$^+$22, LXZ$^+$23] considered how to use a cluster of machines to accelerate the prover of a ZKP system. However, these approaches are not applicable to our setting because they do not address the privacy requirement.

Prio [CB17] and others [BBCG$^+$19, AGJ$^+$22, DPRS23] considered statistic aggregation protocols where there are multiple servers and each client sends secret-shared versions of its input to the servers. Their compliance checking protocols can be considered as distributed ZKP protocols where the prover knows the statement and each verifier (the servers) only knows a secret share of the statement. In their model, there is a single prover and multiple verifiers, whereas there are multiple provers and one verifier in our model. So they can be considered as the dual of our model.

Bell et al. [BGL$^+$23] proposed ACORN, a robust secure aggregation protocol with input validation. One building block they used is a protocol for multiple clients to collectively prove that their committed values are correlated in the correct manner. However, their protocol is restricted to a specific relation (i.e., the sum of all provers' secret witnesses is equal to a public value) and does not fit our setting.

2 Technical Roadmap

We give a high-level overview of our novel techniques.

First attempt: representing sets as polynomials for set consistency check. Recall that the flawed solution mentioned earlier is unsound because it fails to check the "set consistency" property, i.e., it cannot guarantee that the purported sets $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in the audit phase are a disjoint partitioning of the shuffled Pool the server received during the data collection phase.

As a first attempt to fix this problem, we can use the polynomial interpolation technique to prove set consistency [GWC19]. Suppose each client i represents its set of items $\mathbf{x}_i := (x_{i,1}, \ldots, x_{i,m})$ using a polynomial $f_i(x) = (x - x_{i,1}) \cdot \ldots \cdot (x - x_{i,m})$. During the audit, each client commits to its purported set $\mathbf{x}_i$. Next, the server sends a random challenge r, and each client responds with $f_i(r)$, and proves in zero-knowledge that 1) the committed $\mathbf{x}_i$ satisfies the predicate C, and 2) the purported outcome $f_i(r)$ is correct w.r.t. the committed $\mathbf{x}_i$. The server now verifies all clients' proofs. Moreover, to verify set consistency, the server additionally checks that $\prod_i f_i(r) = \prod_{a \in \mathsf{Pool}} (r - a)$. This technique is a perfect fit for our distributed setting: every client can compute their polynomial evaluations locally without communicating with other clients, making this technique efficient in both communication and computation.

Unfortunately, even though this approach indeed enforces the set consistency check, it breaks privacy. Specifically, the server can perform a "subset-style" attack: it can pick a subset $X^* \subseteq \mathsf{Pool}$, and test if a some client i's $f_i(r)$ agrees with $\prod_{a \in X^*} (r - a)$. This allows the server to learn if client i's set $\mathbf{x}_i = X^*$.

Novel idea: preventing leakage with decoy terms. Our main novel idea is to introduce decoy terms to provably prevent the aforementioned leakage. Specifically, during the audit phase, instead of sending $f_i(r)$ directly to the server, client i generates random decoy terms $y_{i,1}, \ldots y_{i,d}$, and sends $p_i := f_i(r) \cdot \prod_{j \in [d]} y_{i,j}$ to the server, where the term $\prod_{j \in [d]} y_{i,j}$ masks the true value of $f_i(r)$. Further, each client sends its decoy terms $y_{i,1}, \ldots y_{i,d}$ to a trusted shuffler (denoted $\mathcal{F}_{\mathsf{shuffle}}$). The shuffler $\mathcal{F}_{\mathsf{shuffle}}$ randomly permutes all decoy terms, and sends an unordered set Y of all clients' decoy terms to the server. Instead of checking that $\prod_i p_i = \prod_{a \in \mathsf{Pool}}(r - a)$, the server now checks $\prod_i p_i = \prod_{a \in \mathsf{Pool}}(r - a) \cdot \prod_{y' \in Y} y'$. Intuitively, the "subset-style" attack is now much harder for the server: the server needs to find both the subset X^* and a correct subset of decoy terms Y^* to break an individual client's privacy. Formally, we show that for an appropriate choice of the field size, it suffices to set the number of decoy terms $d = \mathsf{poly}\log\lambda$ to achieve negligibly small security failure probability.

New proof techniques. Henceforth, let $\mathcal{H} \subseteq [n]$ denote the set of honest clients, and we use the notation $\mathsf{Multiset}(\{y_{i,j}\}_{i \in \mathcal{H}, j \in [d]})$ to denote the shuffled decoy terms $\{y_{i,j}\}_{i \in \mathcal{H}, j \in [d]}$.

To prove that the protocol satisfies zero-knowledge, a key step is to argue that even when the server has seen $\mathsf{Multiset}(\{y_{i,j}\}_{i \in \mathcal{H}, j \in [d]})$ for each honest client $i \in \mathcal{H}$, the randomizing term $\prod_{j \in [d]} y_{i,j}$ still serves as a good mask for the partial product $\prod_{j \in [m]}(r - x_{i,j})$, i.e., the mask $\prod_{j \in [d]} y_{i,j}$ is sufficient for hiding which subset of Pool belong to client i.

To show this, we rely on a technical lemma shown by Balle et al. [BBGN20] in the context of a secure summation protocol over an anonymous network. Specifically, imagine that n clients each have an input denoted $x_1, x_2, \ldots, x_n$ from some appropriate domain. They want to jointly compute the summation of their private values without leaking each individual's input. Ishai et al. [IKOS06] suggested a simple protocol: each client splits its private input into d additive shares, and sends all shares to a shuffler which mixes all $n \cdot d$ shares, and presents them to a server. The server simply sums up all the shares it receives. Ishai et al. [IKOS06] showed that if we set the number of shares to $d = O(\log q + \sigma + \log n)$, then for any input vectors $(x_1, \ldots, x_n)$ and $(x'_1, \ldots, x'_n)$ such that $\sum_{i \in [n]} x_i = \sum_{i \in [n]} x'_i$, the views of the adversary have statistical distance bounded by $2^{-\sigma}$. Balle et al. [BBGN20] observed that the bound by Ishai et al. [IKOS06] is not tight when $n \geq 19$. In particular, the number of shares d grows with n, which is counterintuitive since having more parties should intuitively strengthen the privacy guarantees, allowing us to use a smaller number of shares. Thus, Balle et al. showed a stronger version of the theorem where the number of shares d only needs to be $d > 2 + \frac{2\sigma + \log_2 q}{\log_2 n - \log_2 e}$ to get $2^{-\sigma}$ statistical distance.

We now give an informal proof roadmap. Fix the Pool of data items, and consider two different ways to partition Pool across the n clients. Specifically, imagine that in world 0, the partitioning is $\{x_{i,j}\}_{i \in [n], j \in [m]}$, and in world 1, the partitioning is $\{x'_{i,j}\}_{i \in [n], j \in [m]}$. Let $z_i = \prod_{j \in m}(r - x_{i,j}) \prod_{j \in [d]} y_{i,j}$ and $z'_i = \prod_{j \in m}(r - x'_{i,j}) \prod_{j \in [d]} y'_{i,j}$ be the masked partial product from each client i, where the decoy terms $\{y_{i,j}\}_{i \in [n], j \in [d]}$ and $\{y'_{i,j}\}_{i \in [n], j \in [d]}$ are all chosen independently at random. We want to show that

$$\begin{aligned} &\left(\{z_i\}_{i \in [n]}, \mathsf{Multiset}(\{y_{i,j}\}_{i \in [n], j \in [d]})\right) \\ \approx &\left(\{z'_i\}_{i \in [n]}, \mathsf{Multiset}(\{y'_{i,j}\}_{i \in [n], j \in [d]})\right) \end{aligned}$$

where $\approx$ denotes statistical indistinguishability. To show this, observe that conditioned on $z_i = z'_i$ for all $i \in [n]$ in the two worlds, it must be that $\prod_{i \in [n], j \in [d]} y_{i,j} = \prod_{i \in [n], j \in [d]} y'_{i,j}$. Using Balle et al.'s theorem, we have that conditioned on $z_i = z'_i$ for all $i \in [n]$, the terms $\mathsf{Multiset}(\{y_{i,j}\}_{i \in [n], j \in [d]})$ and $\mathsf{Multiset}(\{y'_{i,j}\}_{i \in [n], j \in [d]})$ are statistically indistinguishable.

Our technical sections later will formalize the above intuition. Specifically, to make the proof formal, we need to 1) add computationally sound reasoning for the cryptographic primitives used; 2) change the above argument to work for the subset of honest parties rather than all parties; and 3) correctly set parameters of the scheme to get negligibly small security failure. We defer the details to Appendix A.

Concretely efficient upgrade to the malicious-server setting. The protocol mentioned so far only achieves anonymity only if the server is semi-honest. To get rid of this assumption, one approach is to use standard theoretical techniques for converting an honest-verifier ZKP to a malicious-verifier ZKP [Dam], or to rely on a random coin toss protocol to jointly generate the server's challenge. However, these generic techniques are not concretely efficient. Instead, we propose a new upgrade that incurs minimal additional overhead in comparison with the semi-honest-server setting. We defer the detailed description to Section 6.

3 Formal Problem Definition

3.1 Notations

Shuffler notation. We use the notation $\mathcal{F}_{\mathsf{shuffle}}$ to denote a shuffler ideal functionality. Assume there are n clients and each client i submits to $\mathcal{F}_{\mathsf{shuffle}}$ a multiset of m data items denoted $\mathbf{x}_i := (x_{i,1}, \ldots, x_{i,m})$, then $\mathcal{F}_{\mathsf{shuffle}}$ outputs an *unordered* multiset of all the data items, that is, $\{x_{i,j}\}_{i\in[n],j\in[m]}$. Henceforth, we use the notation $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to denote th multiset $\{x_{i,j}\}_{i\in[n],j\in[m]}$.

Notation for NP relation. Henceforth, we use the notation $\mathcal{R}_i(\mathbf{x}_i, w_i)$ to represent the NP relation corresponding to the compliance predicate that the server wants to check for each client i, where $\mathbf{x}_i$ is the data items contributed by client i, and w_i is the witness. Specifically, $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$ means that $\mathbf{x}_i$ is in the NP language, i.e., it satisfies the compliance predicate. In the most general form, this compliance predicate need not be the same for all clients.

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 $i \in [n]$, $\mathcal{R}_i(\mathbf{x}_i, w_i) = 1$.

3.2 Syntax

An *anonymous, compliant data collection* has the following syntax:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}})$:
- Pool or $\perp \leftarrow \Pi(\mathsf{pp}, \mathbf{x}_1, \ldots, \mathbf{x}_n, w_1, \ldots, w_n)$: All parties have the input pp and moreover, each client i has a multiset of data items $\mathbf{x}_i := \{x_{i,j}\}_{j\in[m]}$, and witness w_i. The client and the server then engage in protocol, such that at the end of the protocol, the server either outputs a multiset Pool of data items, or outputs $\perp$ indicating failure.

Completeness. Completeness is a natural correctness requirement, it stipulates the following: for any $\lambda, n \in \mathbb{N}$, any $\boldsymbol{\mathcal{R}}$, for any $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, any $\mathbf{w} = (w_1, \ldots, w_n)$ such that $\boldsymbol{\mathcal{R}}(\mathbf{x}, \mathbf{w}) = 1$,

$$\Pr\left[\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}}),\ \Pi(\mathsf{pp}, \mathbf{x}, \mathbf{w}) = \mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)\right] = 1$$

3.3 Security Definitions

Henceforth, let $\mathcal{C} \subseteq [n]$ denote the set of corrupt clients, and let $\mathcal{H} = [n] \backslash \mathcal{C}$ denote the set of honest clients. Consider the following random experiment denoted $\mathsf{Expt}^{n,\mathcal{A},\boldsymbol{\mathcal{R}}}(1^\lambda)$:

- run $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}})$, and $\mathcal{A}$ receives pp;
- $\mathcal{A}$ outputs $\{\mathbf{x}_i, w_i\}_{i \in \mathcal{H}}$ which is required to satisfy $\mathcal{R}_i(\mathbf{x}_i, w_i)$ for any $i \in \mathcal{H}$;
- run the protocol Π with $\mathcal{A}$ who controls the set $\mathcal{C}$ of clients and possibly the server, where the honest clients use the inputs $\{\mathbf{x}_i, w_i\}_{i \in \mathcal{H}}$.

Soundness. Soundness requires that for any n that is polynomially bounded in λ, any $\mathcal{C} \subseteq [n]$, any $\boldsymbol{\mathcal{R}}$ (where the circuits for checking the NP relations are polynomially bounded in λ), for any non-uniform PPT adversary $\mathcal{A}$ that controls the set $\mathcal{C}$ of corrupt clients (but not the server), there exists a negligible function $\mathsf{negl}(\cdot)$, such that in the above randomized experiment $\mathsf{Expt}^{n,\mathcal{A},\boldsymbol{\mathcal{R}}}(1^\lambda)$, except with $1 - \mathsf{negl}(\lambda)$ probability, if the server did not reject outputting $\perp$, the multiset Pool it outputs must satisfy the following:

1. $\mathsf{Multiset}(x_{\mathcal{H}}) \subseteq \mathsf{Pool}$;
2. there exists a disjoint partitioning $\{\mathbf{x}_j\}_{j \in \mathcal{C}}$ of $\mathsf{Pool} \backslash \mathsf{Multiset}(\{\mathbf{x}_i\}_{i \in \mathcal{H}})$, such that for any $j \in \mathcal{C}$, there exists some w_j such that $\mathcal{R}_j(\mathbf{x}_j, w_j) = 1$.

The first condition says that honest clients' contributions must show up in Pool, and the second condition says that for the corrupt clients' contributions, there must be a way to partition these data items among the corrupt clients $\mathcal{C}$, such that every corrupt client $j \in \mathcal{C}$ submitted a compliant multiset of data items.

***t*-anonymity.** t-anonymity requires that for any non-uniform PPT adversary $\mathcal{A}$ controls at most t clients and possibly the server, there exists a PPT simulator $(\mathsf{SimSetup}, \mathsf{SimProt})$ such that for any n that is polynomially bounded in λ, any $\boldsymbol{\mathcal{R}}$ (where the circuits for checking the NP relations are polynomially bounded in λ), the adversary's view in the above real-world experiment $\mathsf{Expt}^{n,\mathcal{A},\boldsymbol{\mathcal{R}}}(1^\lambda)$ is computationally indistinguishable from the output of the following ideal experiment.

- run $(\mathsf{pp}, st) \leftarrow \mathsf{SimSetup}(1^\lambda, n, \boldsymbol{\mathcal{R}})$, and $\mathcal{A}$ receives pp;
- $\mathcal{A}$ outputs $\{\mathbf{x}_i, w_i\}_{i \in \mathcal{H}}$ which is required to satisfy $\mathcal{R}_i(\mathbf{x}_i, w_i)$ for any $i \in \mathcal{H}$;
- output the simulated view $\mathsf{SimProt}(st, \mathsf{Multiset}(\{\mathbf{x}_i\}_{i \in \mathcal{H}}))$.

Notice that the simulator only sees $\mathsf{Multiset}(\{\mathbf{x}_i\}_{i \in \mathcal{H}}$ of honest clients' contributions. This implies that a computationally bounded adversary does not learn who contributed which data items.

Remark 3.1 (t-anonymity in the presence of a semi-honest server)**.** Later, as a stepping stone, we will first construct a scheme that satisfies t-anonymity in the presence of a semi-honest server. In this model, we assume that the adversary controls a subset of the clients, and possibly the server. The server is guaranteed to behave honestly, but the adversary can observe the server's view including its internal coins and all the messages it sends and receives. On the other hand, the corrupted clients can behave arbitrarily including in a manner dependent on the server's internal coins.

4 Preliminaries

4.1 Technical Lemma for Secure Summation

As mentioned, the most interesting technique in our construction and proof is the introduction of decoy terms to allow a privacy-preserving set consistency check. To prove the security of this approach, we rely on a technical lemma from Balle et al. [BBGN20], which is derived from a simple secure summation protocol first proposed by Ishai et al. [IKOS06].

Imagine that there are n parties each with an input $x_i \in \mathbb{Z}_q$. Each party i splits its input x_i into d random, additive shares and sends them to a shuffler. The server receives all the shares from the shuffler and sums them up. The lemma shows that as long as the server only sees the unordered, shuffled shares, it learns only the sum of the inputs and nothing else, ignoring a small statistical security loss. Balle et al. [BBGN20] observed that the lemma also works when the inputs are from a finite abelian group. We will formally state the lemma in the context of the case when the inputs are from a multiplicative abelian group.

Lemma 4.1 ([BBGN20]). *Suppose that $n \geq 19, d \geq 3$, and $\sigma \geq 1$. Let $\mathbb{G}$ be a multiplicative abelian group of order q. Suppose we are given two arbitrary vectors $(\mu_1, \ldots, \mu_n) \in \mathbb{G}^n$ and $(\mu'_1, \ldots, \mu'_n) \in \mathbb{G}^n$, such that $\prod_{i\in[n]} \mu_i = \prod_{i\in[n]} \mu'_n$. Now, for $i \in [n]$, randomly sample $(y_{i,1}, \ldots, y_{i,d})$ such that $\mu_i = \prod_{j\in[d]} y_{i,j}$. Similarly, randomly sample $(y'_{i,1}, \ldots, y'_{i,d})$ such that $\mu'_i = \prod_{j\in[d]} y'_{i,j}$. Then, the two multisets $\mathsf{Multiset}(\{y_{i,j}\}_{i\in[n],j\in[d]})$ and $\mathsf{Multiset}(\{y'_{i,j}\}_{i\in[n],j\in[d]})$ have statistical distance at most $2^{-\sigma}$ where*

$$\sigma = \frac{(d-1)(\log_2 n - \log_2 e) - \log_2 |q|}{2}$$

4.2 Cryptographic Building Blocks

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 $r \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and then calls $\mathsf{commit}(1^\lambda, x; r)$.
- *Perfectly binding.* There does not exist λ, (x, r) and (x', r') where $x \neq x'$, such that $\mathsf{commit}(1^\lambda, x; r) = \mathsf{commit}(1^\lambda, x'; r')$.

Non-interactive zero-knowledge. A non-interactive argument system for a family 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 Warmup: Protocol for a Semi-Honest Server

We first present a protocol assuming a semi-honest server; however, a subset of the clients may be under the control of the adversary and behave arbitrarily mailiciously. Later in Section 6, we will discuss how to upgrade the protocol to the malicious-server setting with minimal additional overhead.

5.1 Construction

We assume a prime-order field $\mathbb{F}$ whose size is superpolynomial in the security parameter λ. Suppose that each data item is encoded in the field $\mathbb{F}$, i.e., each $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,m}) \in \mathbb{F}^m$. Our protocol relies on an underlying anonymous network modelled as a shuffler ideal functionality denoted $\mathcal{F}_{\text{shuffle}}$ — see Section 3.1.

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

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

1. **Data collection phase:** Every client $i \in [n]$ sends $\mathbf{x}_i$ to $\mathcal{F}_{\text{shuffle}}$, and $\mathcal{F}_{\text{shuffle}}$ sends $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to the server.
2. **Audit phase:**
   - *Each client i*:
     - 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*:
     - Sample a random challenge $r \xleftarrow{\$} \mathbb{F}$ and broadcast r to all clients.
   - *Each client i*:
     - Parse $\mathbf{x}_i := (x_{i,1}, \ldots, x_{i,m})$, and compute $z_i := \rho_i \cdot \prod_{j \in [m]} (x_{i,j} - r)$.
     - 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 $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ if the following checks pass, else output 0:
     - *NIZK verification*: For $i \in [n]$, $\mathsf{NIZK}_i.\mathsf{V}(\mathsf{crs}_i, (z_i, \mathsf{com}_i, r), \pi_i) = 1$.
     - *Set consistency check*: $\prod_{i \in [n]} z_i = \prod_{i \in [n], j \in [m]} (x_{i,j} - r) \cdot \prod_{i \in [n], j \in [d]} y_{i,j}$. The server can compute both sides of the equation knowing $z_1, \ldots, z_n$, $\mathsf{Multiset}(\{x_{i,j}\}_{i \in [n], j \in [m]})$, and $\mathsf{Multiset}(\{y_{i,j}\}_{i \in [n], j \in [d]})$. Also, check all $y_{i,j}$'s are not 0.

Figure 1: Our anonymous, compliant data collection protocol for a semi-honest server.

Protocol. During the setup, we call the NIZKs' setup and outputs the resulting common reference strings as the public parameter. Our protocol then proceeds with a data collection phase and an audit phase as follows:

1. **Data collection phase.** All clients send their data items over an anonymous network to the server. More formally, every client $i \in [n]$ sends its data items $\mathbf{x}_i := (x_{i,1}, \ldots, x_{i,m})$ to $\mathcal{F}_{\text{shuffle}}$, and $\mathcal{F}_{\text{shuffle}}$ sends $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to the server.
2. **Audit phase.**[3] Each client proves compliance to the server without identifying which data items it has contributed using the following protocol.
   - First, each client samples d random decoy terms $y_{i,1}, \ldots, y_{i,d}$ from $\mathbb{F} \backslash \{0\}$, and sends $\{y_{i,1}, \ldots, y_{i,d}\}$ to $\mathcal{F}_{\text{shuffle}}$, which in turn sends $\mathsf{Multiset}(\{y_{i,j}\}_{i \in [n], j \in [d]})$ to the server.
   - Additionally, client i sends a commitment com_i of $(\mathbf{x}_i, \rho_i)$ to the server where $\rho_i := \prod_{j \in [d]} y_{i,j}$.
   - Next, the server sends a random challenge $r \in \mathbb{F}$ to all clients.

[3]Here, we focus on the case when the audit phase is run only once. Our protocol can be extended to run multiple audit phases on the same collected data with different predicates naturally.

- Now, each client i sends $z_i := \rho_i \cdot \prod_{j\in[m]}(x_{i,j} - r)$ to the server, along with a NIZK proof attesting to the following facts: 1) z_i is computed correctly using the tuple $(\mathbf{x}_i, \rho_i)$ under the commitment com_i, and the data items $\mathbf{x}_i$ under the commitment com_i satisfy the compliance predicate.
- Finally, the server outputs $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ if all n NIZK proofs verify, and moreover, $\prod_{i\in[n]} z_i = \prod_{i\in[n],j\in[m]}(x_{i,j} - r) \cdot \prod_{i\in[n],j\in[d]} y_{i,j}$; otherwise, it outputs $\perp$. Notice that the server can efficiently compute $\prod_{i\in[n],j\in[m]}(x_{i,j} - r)$ and $\prod_{i\in[n],j\in[d]} y_{i,j}$, since it knows $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and $\mathsf{Multiset}(\{y_{i,j}\}_{i\in[n],j\in[d]})$.

We give a formal protocol description in Figure 1 where the NP relation for the NIZK proofs are defined below.

NP relation for the NIZK proofs. In Figure 1, we use NIZK_i to denote a NIZK scheme for the NP relation $\widetilde{R}_i$:

- Public statement (z_i, com_i, r): the client's masked evaluation z_i, the commitment com_i, and the evaluation point r;
- Private witness $(\mathbf{x}_i, \gamma_i, w_i, \rho_i)$: the client's data items $\mathbf{x}_i$, witness w_i, the randomness γ_i used in the commitment, as well as the product of client i's decoy terms 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
  - $\prod_{j\in[m]}(x_{i,j} - r) \cdot \rho_i = z_i$.

Parameter choices. Throughout, we use λ to denote the security parameter, and we assume that n is upper bounded by a fixed polynomial in λ. We set the field size of $\mathbb{F}$ to be superpolynomial in λ. Let $\sigma = \omega(\log \lambda)$ be a super-logarithmic function in the security parameter. Assume n clients and at most t corrupted clients, by setting the number of decoy terms per client as

$$d \geq \frac{2\sigma + \log_2 |\mathbb{F}|}{\log_2(n-t) - \log_2 e} + 2, \tag{1}$$

the *statistical* security loss is upper bounded by $2^{-\sigma}$ which is negligibly small in λ since $\sigma = \omega(\log \lambda)$. Additionally, the cryptographic primitives we employ introduce a separate *computational* security loss that is also negligibly small in λ. Note that Equation (1) also shows that the number of decoy terms per client d decreases as n grows. Further, if we set the field size to be $\exp(\log^c n)$ for some constant $c > 1$, then d is upper bounded by $O(\log^c \lambda)$.

Remark 5.1 (Parameter choices depend on the number of honest clients)**.** Our choice of d in Equation (1) assumes that there are at least 19 honest clients — this is inherited from the technical lemma (Lemma 4.1) proven by Balle et al. [BBGN20]. Note that the total number of clients n can be much larger than 19, i.e., security holds even when a large majority of the clients can be maliciously corrupted. However, if there are fewer than 19 honest clients, we can set the number of decoys d to be $d = \lceil 1.5 \log_2(|\mathbb{F}|) + \log_2 n + \sigma \rceil$ due to the lemma of Ishai et al. [IKOS06] to achieve the same level of security loss.

In Appendix A, we prove the following theorem.

Theorem 5.2 (Protocol for a semi-honest server)**.** *Suppose that $|\mathbb{F}|$ is superpolynomial in λ, $n - t \geq 19$ and $d = \omega\left(\frac{\log|\mathbb{F}|+\log\lambda}{\log(n-t)}\right)$. Further, suppose that the underlying* NIZK *satisfies completeness, soundness, and zero-knowledge, and the commitment scheme* comm *is perfectly binding and computationally hiding. Then, the protocol described in this section satisfies completeness, soundness, and t-anonymity in the presence of a semi-honest server.*

In our 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, 1000, and 10000 honest clients, we can choose $d = 82$, $d = 51$, and $d = 37$, respectively.

Cost analysis. Based on the above parameter choices, we can now give the asymptotic performance bounds. Suppose we use a SNARK scheme to realize the underlying NIZK, and suppose that the SNARK scheme has $\widetilde{O}(c)$ prover time for proving a size-c circuit, $O(1)$ proof size and verification time. Further, suppose the commitment scheme comm has $O(m)$ computation time and commitment size for a message of size $O(m)$. Then, the above protocol satisfies the following performance bounds:

- Each client i incurs $\widetilde{O}(|\mathcal{R}_i|)$ computation and $O(m)+\widetilde{O}(1)$ communication (see also Remark 5.3), where the notation $|\mathcal{R}_i|$ denotes the size of the circuit that checks the NP relation $\mathcal{R}_i$. Specifically, the $O(m)$ part comes from sending the m data items to the shuffler and committing to them again during the audit, and the $\widetilde{O}(1)$ accounts for sending the decoy terms to the shuffler and all other communication.
- The server's computation and communication are upper bounded by $O(nm) + \widetilde{O}(n)$.

Remark 5.3 (Regarding client communication)**.** The client communication includes sending the m data items to the shuffler during data collection, and sending one commitment and one ZKP in the audit phase. If we use a perfectly binding commitment, the commitment size is linear in m for committing to a length-m message. In our actual implementation, we use a random-oracle-based commitment scheme that the commitment size is $O(1)$ even for committing to a length-m message; and we use a SNARK whose proof size is also $O(1)$. In this case, the client's *extra* communication overhead (besides sending the m data items to the shuffler) is actually bounded by $\widetilde{O}(1)$. Later in Section 6 and Appendix B, we will argue that our security proofs still hold when we replace the commitment scheme with a random-oracle-based one (see also Remark B.1).

5.2 Anonymity for a Semi-Honest Adversary

We now prove a key lemma that is needed for proving anonymity in the presence of a semi-honest server. This key lemma represents the most interesting step in our security proof, since it captures the statistical steps of reasoning why the decoy terms give us strong privacy. We defer the full proof of anonymity for a semi-honest server to Appendix A.

Key lemma for anonymity. Intuitively, the above lemma says that suppose the zero-knowledge proofs and commitments leak nothing, then the server cannot distinguish whether the honest clients' inputs are $\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]} \in \mathbb{F}^{|\mathcal{H}|\cdot m}$ or $\{x'_{i,j}\}_{i\in\mathcal{H},j\in[m]} \in \mathbb{F}^{|\mathcal{H}|\cdot m}$, as long as $\mathsf{Multiset}(\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]}) = \mathsf{Multiset}(\{x'_{i,j}\}_{i\in\mathcal{H},j\in[m]})$. In other words, the server cannot learn how the permuted data items are partitioned across the honest clients. This intuition can be captured by the following lemma.

Lemma 5.4 (Key lemma for proving anonymity)**.** *Given a security parameter $\sigma = \omega(\log \lambda)$, and assume $|\mathbb{F}|$ is superpolynomial in λ, $n - t \geq 19$, and $d \geq \left\lceil \frac{2\sigma + \log_2 |\mathbb{F}|}{\log_2(n-t) - \log_2 e} + 2 \right\rceil$. Let $\mathcal{H}$ denote the set of honest clients. Fix arbitrary $\{x_{i,j}\}_{i\in\mathcal{H}, j\in[m]} \in \mathbb{F}^{|\mathcal{H}|\cdot m}$ and $\{x'_{i,j}\}_{i\in\mathcal{H}, j\in[m]} \in \mathbb{F}^{|\mathcal{H}|\cdot m}$ such that $\mathsf{Multiset}(\{x_{i,j}\}_{i\in\mathcal{H}, j\in[m]}) = \mathsf{Multiset}(\{x'_{i,j}\}_{i\in\mathcal{H}, j\in[m]})$, and fix some $r \notin \{x_{i,j}\}_{i\in\mathcal{H}, j\in[m]}$ also from $\mathbb{F}$. Then, 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:*

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

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

Proof. To prove the key lemma, we shall rely on Lemma 4.1, the technical lemma from Balle et al. [BBGN20].

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 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:

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

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.

Claim 5.5. *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 4.1 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 $2^{-\sigma}$ where

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

Therefore, fixing $\sigma = \omega(\log \lambda)$, when $d \geq \left\lceil \frac{2\sigma + \log_2 |\mathbb{F}|}{\log_2(n-t) - \log_2 e} + 2 \right\rceil$, the statistical distance of $\mathcal{A}$'s view between Hyb_3 and Hyb_4 is bounded by $2^{-\sigma}$ which is negligibly small in λ. □

Completing the proof of anonymity against a semi-honest server. As mentioned, the key lemma essentially shows that the server does not learn how the data items are partitioned across the honest clients, assuming that the zero-knowledge proofs and commitments are perfectly secret. In reality, however, the cryptographic primitives satisfy only computational notions of security. Therefore, to formally prove anonymity in the presence of a computationally bounded adversary, we need to go through a sequence of hybrid experiments, such that we first replace the zero-knowledge proofs and commitments with simulated ones using the security of the cryptographic primitives, and then apply Lemma 5.4 to switch the honest clients' inputs from $\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]}$ to $\{x'_{i,j}\}_{i\in\mathcal{H},j\in[m]}$ where the latter is an arbitrary partition of $\mathsf{Multiset}(\{x_{i,j}\}_{i\in[n],j\in[m]})$. Afterwards, the view of the adversary in the hybrid experiment can be fully simulated by a simulator that only knows the multiset of the honest clients' inputs. We present the full proof in Appendix A.

5.3 Proof of Soundness

We now prove a key lemma that captures the essence of the soundness proof. This key lemma captures the essential statistical reasoning in the soundness proof, when we imagine that all the cryptographic primitives were ideal. The full soundness proof requires a computationally sound treatment of the cryptographic primitives used, and we defer the full soundness proof to Appendix A.

Key lemma for soundness. We now state the key lemma (Lemma 5.6) needed for proving soundness. To better understand the lemma below, it helps to imagine that $\{x_{i,j}\}_{i\in[n],j\in[m]}$ (represented by $\{x_1,\ldots,x_k\}$ in Lemma 5.6) represent the data items the server has obtained during the data collection phase, and $\{x'_1,\ldots,x'_k\}$ (represented by $\{x_1,\ldots,x_k\}$ in Lemma 5.6) represent the data items the clients commit to during the audit. Moreover, imagine that $\alpha = \prod_{i\in[n],j\in[d]} y_{i,j}$, and $\alpha' = \prod_{i\in[n]} \rho_i$. Recall that during the audit (Figure 1), the clients submit the decoy terms $\{y_{i,j}\}$ and commit to $\{x'_{i,j}\}_{i\in[n],j\in[m]}$ and $\{\rho_i\}_{i\in[n]}$. This is why in Lemma 5.6 below, we imagine that $\{x_{i,j}\}_{i\in[n],j\in[m]}$, $\{x'_{i,j}\}_{i\in[n],j\in[m]}$, α, α' are fixed; however, the challenge r is randomly chosen.

Lemma 5.6 (Key lemma for proving soundness)**.** *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,*

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

Proof. Since $\mathsf{Multiset}(x_1,\ldots,x_n) \neq \mathsf{Multiset}(x'_1,\ldots,x'_n)$, the polynomials $F(R) = \prod_{i\in[n]}(R-x_i)$ and $F'(R) = \prod_{i\in[n]}(R-x_i)$ are not the same. In the case of $\alpha \neq \alpha'$, since $F(R)$ and $F'(R)$ are both monic polynomials, $\alpha F(R)$ and $\alpha' F'(R)$ are two different polynomials. In the case of $\alpha = \alpha'$, $\alpha F(R)$ and $\alpha' F'(R)$ are different because $F(R) \neq F'(R)$. Therefore, the lemma follows from a direct application of the Schwartz-Zippel lemma. □

Intuitively, the above key lemma for soundness says that as long as the corrupted clients use inconsistent data items in the audit phase, the audit will fail with overwhelming probability regardless of how the corrupted clients generate their decoys.

Completing the soundness proof. The full proof of soundness makes use of the perfect binding property of the commitment scheme and the soundness of the zero-knowledge proof, and then reaches a step where applying the key lemma, i.e., Lemma 5.6 would be sufficient. We defer the full proof to Appendix A.

6 A Simple Upgrade to the Malicious-Server Setting

In this section, we propose a simple upgrade with minimal overhead that lifts our warmup protocol to the malicious-server setting. Specifically, for soundness, we always assume an honest server; however, we want our anonymity guarantees to hold even when the server can be malicious. In this setting, the challenge for proving anonymity is that the server may not choose the challenge r at random[4]. To upgrade our protocol to the malicious-server setting, one naïve way is the use a generalization of the transformation described by Damgård [Dam] that converts an honest-verifier zero-knowledge proof to a malicious-verifier zero-knowledge proof. Another naïve way is to use a random coin toss protocol to jointly generate the server's challenge. However, these standard approaches result in relatively high concrete overhead. Instead, we propose a simple upgrade with minimal overhead.

Intuition. The construction in Section 5.1 provides anonymity only in the presence of a semi-honest server, relies on the facts that 1) the challenge r is chosen independently of the decoy terms $\{y_{i,j}\}_{i\in\mathcal{H},j\in[d]}$ — however, r need not be chosen uniformly at random; and 2) the challenge r is not equal to any of the data items $\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]}$. However, a malicious server may choose r based on its guess of the data items $\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]}$ or the decoy terms $\{y_{i,j}\}_{i\in\mathcal{H},j\in[d]}$ that have already been revealed. For example, 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. Therefore, the intuition of our upgrade is to ensure that even a malicious server cannot choose the challenge r that collides with any honest client's data item, or depends on the decoy terms. For the former, we can split the field $\mathbb{F}$ into two halves and enforce the data items and the challenge r to be sampled from different halves. For the latter, we require the server to commit to r upfront, so r cannot be chosen based on the decoy terms.

Upgrade to the malicious-server setting. In the upgraded protocol, the server commits to the challenge r upfront. Next, the clients commits to their data items and submits the decoy terms using an *extractable commitment scheme*, and the server then opens r. The clients check if the opening r is valid, and that r must be 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 audit phase, the server picks a random challenge r from $\{\lfloor p/2\rfloor + 1,\ldots,p-1\}$, and it computes $\widetilde{r} := \mathsf{commit}(r;\mathsf{coins})$ where coins denotes the random coins used in the commitment scheme[5]. The server sends $\widetilde{r}$ to all clients over a broadcast channel (see Remark 6.1).
- Now, run the earlier protocol in Figure 1 except with the following modifications. First, the clients now use an extractable commitment scheme when committing to $(\mathbf{x}_i,\rho_i)$[6] Second, instead of directly sending the challenge r to all clients, the server now sends opening r, coin to all clients. All clients check that $\mathsf{commit}(r;\mathsf{coins}) = \widetilde{r}$ and that $r \geq \lfloor p/2\rfloor + 1$. They continue with the protocol if the check passes, and else they abort.

[4]Notice that the Fiat-Shamir heuristic [FS86] does not work in our setting, because the provers are distributed.
[5]This commitment need not be extractable.
[6]The common reference string (CRS) of the extractable commitment scheme is included in the CRS of our protocol.

$\underline{\mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}})\textbf{:}}$ Same as before.

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

1. **Data collection phase:** Same as before, except that now we require that each data item $x_{i,j} \in \{0, \ldots, \lfloor p/2 \rfloor\}$.

2. **Audit phase:**

   - *Server*:
     - Sample a random challenge $r \xleftarrow{\$} \{\lfloor p/2 \rfloor + 1, \ldots, p-1\}$;
     - Let $\widetilde{r} := \mathsf{commit}(r; \mathsf{coins})$ for some randomly sampled coins;
     - Send $\widetilde{r}$ to all clients over a broadcast channel.
   - *Each client* i:
     - 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 $\mathsf{com}_i = \mathsf{commit}((\mathbf{x}_i, \rho_i); \gamma_i)$ to the server where γ_i denotes sampled random coins and commit is an extractable commitment scheme.
   - *Server*:
     - Send the opening (r, coins) to all clients.
   - *Each client* i:
     - Check that (r, coin) is a correct opening of $\widetilde{r}$, and further, check that $r \in \{\lfloor p/2 \rfloor + 1, \ldots, p-1\}$. Abort if the check fails.
     - Parse $\mathbf{x}_i := (x_{i,1}, \ldots, x_{i,m})$, and compute $z_i := \rho_i \cdot \prod_{j \in [m]} (x_{i,j} - r)$.
     - 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 $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ 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 $\mathsf{Multiset}(\mathbf{y}_1, \ldots, \mathbf{y}_n)$. Also check that all $y_{i,j}$ are non-zero.

Figure 2: Our anonymous, compliant data collection protocol for a malicious server. The modifications are highlighted in blue.

For completeness, we describe the full protocol formally in Figure 2. We prove that this upgraded protocol satisfies soundness and t-anonymity in the presence of a malicious server in Appendix B.

Remark 6.1 (Necessity of the broadcast channel)**.** Note that the broadcast channel is needed to ensure that the server sends the same challenge r to all clients. Otherwise, there is an explicit attack where the server can distinguish between the following two cases: 1) client 1 has x and client 2 has x'; and 2) the two clients data items are swapped. The server can send r_1 to client 1 and r_2 to client 2, and from its view, it can recover the product $(x - r_1) \cdot (x' - r_2)$ in the former case; or the product $(x' - r_1) \cdot (x - r_2)$ in the latter case. The server can then learn the source of the data items, violating the anonymity property. One way to instantiate the broadcast channel is to have the server send $\widetilde{r}$ to all clients; all clients then sign $\widetilde{r}$ using a threshold signature scheme. The server aggregates all clients' signatures and sends the aggregated signature to all clients. The clients accept the challenge $\widetilde{r}$ if the aggregated signature verifies.

Remark 6.2 (Necessity of using extractable commitment scheme for clients' data items)**.** When we make this change, the soundness proof becomes more challenging. Our earlier soundness proof (Lemma 5.6) relies on the fact that the malicious clients cannot choose the committed $\{x_{i,j}\}_{i \in \mathcal{H}, j \in [m]}$ values and the decoy terms $\{y_{i,j}\}_{i \in \mathcal{H}, j \in [d]}$ that depend on the challenge r. However, now the server has committed some challenge r upfront, we effectively need to argue that it is not possible for malicious clients to commit to values that are related to the committed challenge r. To deal with this issue, we require that the clients commit to their data items $\{x_{i,j}\}$ using an *extractable* commitment scheme which is easy to construct assuming a common reference string (see Appendix B.1).

Cost analysis. In comparison with the semi-honest-server protocol in Section 5, this new upgrade introduces minimal overhead, and all the asymptotic bounds for the semi-honest-server protocol still hold for the upgraded protocol. More specifically, in comparison with the semi-honest-server protocol, each client incurs only $O_\lambda(1)$ additive overhead in terms of client computation and bandwidth. Further, the server needs to incur only $O_\lambda(1)$ additional computation relative to the scheme in Section 5, broadcast an additional message of size $O_\lambda(1)$ to all clients, and send $O_\lambda(1)$ extra information to each client over a point-to-point channel.

In Appendix B, we prove the following theorem.

Theorem 6.3 (Protocol for a malicious server)**.** *Suppose that $|\mathbb{F}|$ is superpolynomial in λ, $n - t \geq 19$ and $d = \omega\left(\frac{\log|\mathbb{F}| + \log\lambda}{\log(n-t)}\right)$. Further, suppose that the underlying* NIZK *satisfies completeness, soundness, and zero-knowledge, the commitment scheme used by the server is perfectly binding and computationally hiding, and the commitment scheme used by the client is computationally hiding and extractable. Then, the protocol described in this section satisfies completeness, soundness, and t-anonymity in the presence of a malicious server.*

7 Implementation and Evaluation

Implementation. We use gnark [BPH+22], an open-sourced NIZK library to create an implementation of our protocol. We separate the implementation into two phases, which include the audit phase and the blame phase that can catch the non-compliant clients (See Appendix C). We assume a semi-honest server throughout the experiments. 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.

7.1 Application Settings and Parameters Used in our Evaluation

We evaluate the application scenarios mentioned earlier in Section 1.2. Here, we provide more details about the parameters we choose for all applications. Unless otherwise noted, we assume there are 1000 clients with 500 corrupted clients. This matches the common settings in shuffle-model data analytics and federated learning scenarios, such as in [BBGN20, LCC+21, GD24]. We list the concrete settings for each application here.

Secure histogram protocol. We implement a simple anonymous histogram protocol.

- Each client has 60 data items from a domain of $\{0, 1, \ldots, 9999\}$.
- The clients send their data items to the shuffler, who shuffles the data items and sends them to the server. The server then computes the histogram of the data items.
- *Constraint:* The contribution from each client cannot include repeated data items.

Shuffle-DP summation protocol. 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 a Polya noise as specified in [BBGN20], which 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.[7]

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.

Secure Condorcet voting protocol. In Condorcet voting [You88], each client's preference is ranking list of candidates and a candidate is chosen if it beats every other candidate in a pairwise comparison. For example, if there are three candidates Alice, Bob, and Charlie, a client's preference can be Bob $>$ Alice $>$ Charlie. Then, if there are five voters, where Bob is preferred over Alice from three voters and is preferred over Charlie from four voters, Bob will be the winner.

- We implement a secure Condorcet voting protocol with 10 candidates and 1000 voters.
- Each voter has a secret ranking among the 10 candidates, represented as a permutation of 1 to 10.

[7] 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.

- Given a ranking $(c_1, \ldots, c_{10})$, a voter will submit 45 comparison pairs to the shuffler, i.e., $\{(c_i, c_j)\}_{1\le i\le 10, i<j\le 10}$, indicating that it prefers c_i over c_j.
- *Constraint:* Each voter's submitted pairs form a valid ranking.

7.2 Audit Phase Costs

Use Case	# Constraints	Public Param	Client Time	Amortized Server Time	Communication Per Client
			Based on Groth16		
Histogram	2.4×10^4	4.0MB	0.2s	3.2ms	2.2KB
Voting	1.6×10^4	2.4MB	0.2s	3.1ms	2.2KB
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
			Based on Plonk		
Histogram	3.2×10^4	22.5MB	0.5s	2.7ms	2.6KB
Voting	2.1×10^4	10.8MB	0.3s	3.0ms	2.6KB
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

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.

We measure the computation and the communication cost for the 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 amortized by the number of clients. Since the clients also need to store the public parameters of the NIZK to generate the proof, the public parameter size captures each client's storage costs. The per-client communication cost counts only the additional overhead incurred by the audit, and does not include the overhead of sending the original data items to the shuffler. Moreover, we also measure the number of constraints needed 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.

Communication cost. We observe that the per-client communication cost is dominated by the decoys used to mask the polynomial evaluations, which is independent of the concrete use cases. Therefore, the use cases have nearly the same communication cost when the underlying proof systems are the same. Recall that each client needs to send $d = \omega\left(\frac{\log \lambda}{\log(n-t)}\right)$ decoys to the shuffler, so given more honest clients, the decoys sent to the shuffler per client are less. To better demonstrate the communication cost, we plot the communication overhead for our protocol in Fig 3 given different honest client numbers and two different proof systems. With our parameter setting, each client will not send more than 60 decoys 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. Therefore, given a wide range of client numbers, the communication cost is no more than 6KB.

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

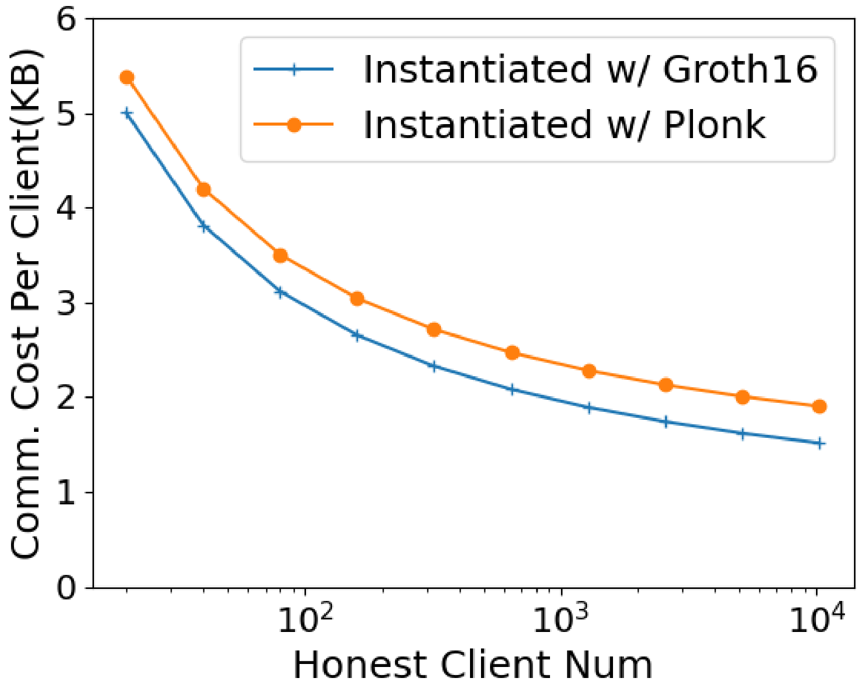

Figure 3: **Per-client communication overhead.** The number of corrupted clients is fixed to 500. The per-client communication decreases with a larger n since with more honest clients, the client needs to send fewer decoy terms.

generate the proof. The constraint numbers for the histogram, voting and Shuffle-DP sum use cases are around 1.6×10^4 to 3.6×10^4, depending on the proof systems. The per client time ranges from 0.2s to 0.5s. The vector sum use cases have more constraints, around 10^6, and the per client time is around 8.5s and 10.6s for the Groth16 and Plonk instantiations, respectively.

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 public parameter size reflects the storage cost of each client and also scale with the constraint number. The Groth16 system has smaller public parameters that are no more than 5MB for the histogram, voting and shuffle-DP summation use cases, and no more than 200MB for the vector summation. The Plonk system has larger public parameters that are 10.8 - 22.5MB for the three simpler cases and around 700MB for the vector sum use case.

7.3 Blame Phase Costs

We also evaluate the extension of our protocol with identifiable abort (described in Appendix C) for the Shuffle-DP Sum use case. Specifically, we use $\mathcal{F}_{\text{shuffle}}$ plus Private Information Retrieval (PIR) for instantiating the enhanced $\mathcal{F}^*_{\text{shuffle}}$ functionality described in Appendix C.2, which works assuming a semi-honest server.

Use Cases	Compliance	Commitment	Consistency	Total
Histogram	3540	$\underline{2.0 \times 10^4}$	61	2.4×10^4
Voting	109	$\underline{1.6 \times 10^4}$	46	1.6×10^4
DP Sum	4.0×10^3	$\underline{2.0 \times 10^4}$	61	2.4×10^4
Vector Sum	2.0×10^4	$\underline{9.9 \times 10^5}$	3001	9.9×10^5

Table 2: Breakdown of the constraints required by each component during the proof generation process for each task. The results are based on the implementation with Groth16 proof system.

In summary, the per client time is 4.6s, the amortized server time is 27.2ms, and the communication cost is 1.0MB. The costs remain reasonable for the use case. We list the implementation and evaluation details in Appendix D.

References

[Abe99] Masayuki Abe. Mix-networks on permutation networks. In *ASIACRYPT*, 1999.

[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.

[Aoy] Gosho Aoyama. Case closed. https://en.wikipedia.org/wiki/Case_Closed.

[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 *Advances in Cryptology – CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III*, page 67–97, 2019.

[BBGN19] Borja Balle, James Bell, Adria Gascon, and Kobbi Nissim. Improved summation from shuffling, 2019.

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

[BC19] Victor Balcer and Albert Cheu. Separating local & shuffled differential privacy via histograms. *arXiv preprint arXiv:1911.06879*, 2019.

[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.

[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.

[BIK$^+$17] Kallista A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1175–1191. ACM, 2017.

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

[CB17] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 259–282. USENIX Association, 2017.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, February 1981.

[Cha88] David L. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, March 1988.

[Che21] Albert Cheu. Differential privacy in the shuffle model: A survey of separations. *arXiv preprint arXiv:2107.11839*, 2021.

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

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

[DD08] George Danezis and Claudia Diaz. A survey of anonymous communication channels. Technical Report MSR-TR-2008-35, Microsoft Research, 2008.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, 2004.

[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. In *Proceedings on Privacy Enhancing Technologies (PET)*, 2023.

[EKR18] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Found. Trends Priv. Secur.*, 2(2–3):70–246, dec 2018.

[EY09] Matthew Edman and Bülent Yener. On anonymity in an electronic society: A survey of anonymous communication systems. *ACM Comput. Surv.*, 42(1), December 2009.

[FM02] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, page 193–206, 2002.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[GD24] Antonious M Girgis and Suhas Diggavi. Multi-message shuffled privacy in federated learning. *IEEE Journal on Selected Areas in Information Theory*, 5:12–27, 2024.

[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.

[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.

[GMPV20] Badih Ghazi, Pasin Manurangsi, Rasmus Pagh, and Ameya Velingker. Private aggregation from fewer anonymous messages. In *Advances in Cryptology – EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II*, page 798–827, 2020.

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

[GRS99] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42:39–41, 1999.

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

[HHCG$^+$23] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In *Usenix Security*, 2023.

[HMPS14] Susan Hohenberger, Steven A. Myers, Rafael Pass, and Abhi Shelat. ANONIZE: A large-scale anonymous survey system. In *IEEE S & P*, 2014.

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

[KSS13] Marcel Keller, Peter Scholl, and Nigel P. Smart. An architecture for practical actively secure MPC with dishonest majority. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 549–560. ACM, 2013.

[LCC$^+$21] Ruixuan Liu, Yang Cao, Hong Chen, Ruoyang Guo, and Masatoshi Yoshikawa. Flame: Differentially private federated learning in the shuffle model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8688–8696, 2021.

[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.

[MB09] Prateek Mittal and Nikita Borisov. Shadowwalker: Peer-to-peer anonymous communication using redundant structured topologies. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, page 161–172, 2009.

[MV98] Yi Mu and Vijay Varadharajan. Anonymous secure e-voting over a network. In *14th Annual Computer Security Applications Conference (ACSAC 1998), 7-11 December 1998, Scottsdale, AZ, USA*, pages 293–299. IEEE Computer Society, 1998.

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

[Ors22] Emmanuela Orsini. Efficient, actively secure mpc with a dishonest majority: a survey. Cryptology ePrint Archive, Paper 2022/417, 2022. https://eprint.iacr.org/2022/417.

[RR98] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, November 1998.

[SBS02] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P5: A protocol for scalable anonymous communication. In *IEEE S & P*, 2002.

[SSA$^+$18] Fatemeh Shirazi, Milivoj Simeonovski, Muhammad Rizwan Asghar, Michael Backes, and Claudia Diaz. A survey on routing in anonymous communication protocols. *ACM Comput. Surv.*, 51(3), June 2018.

[SW21] Elaine Shi and Ke Wu. Non-interactive anonymous router. In *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 489–520. Springer, 2021.

[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.

[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.

[You88] H Peyton Young. Condorcet's theory of voting. *American Political science review*, 82(4):1231–1244, 1988.

[ZZZR05] Li Zhuang, Feng Zhou, Ben Y. Zhao, and Antony Rowstron. Cashmere: Resilient anonymous routing. In *NSDI*, 2005.

A Proofs for the Semi-Honest-Server Setting

In this section, we provide the detailed proofs for our warmup protocol in Section 5, assuming a semi-honest server. The completeness proof is straightforward, so we will focus on proving soundness and anonymity.

A.1 Soundness Proof

Recall that earlier, we proved a key lemma (Lemma 5.6) which captures the core statistical reasoning for proving soundness. The full soundness proof needs to additionally make use of the security of the cryptographic primitives, and we present the full proof below.

Theorem A.1 (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 passes the verification, 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 5.6, 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

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

Given that the field size is superpolynomial in λ, and the server's check ensures that $\alpha \neq 0$, due to Lemma 5.6, 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 $\mathsf{com}_1, \ldots, \mathsf{com}_n$. Because for any honest client $i \in \mathcal{H}$, $\mathbf{x}'_i = \mathbf{x}_i$, it means that $\mathsf{Multiset}(\{\mathbf{x}_i\}_{i\in\mathcal{C}}) = \mathsf{Multiset}(\{\mathbf{x}'_i\}_{i\in\mathcal{C}})$, i.e., the corrupt clients' purported data items in the audit phase must be equal to their contributions in the data collection phase.

Summarizing the above, except with negligible probability, if the server did not reject outputting $\perp$, then there exists a partitioning $\{\mathbf{x}'_i\}_{i\in\mathcal{C}}$ (which is uniquely determined by $\{\mathsf{com}_i\}_{i\in\mathcal{C}}$) of the corrupt clients' contributions $\mathsf{Multiset}(\{\mathbf{x}_i\}_{i\in\mathcal{C}})$, such that for every corrupt client $i \in \mathcal{C}$, $\mathcal{R}_i(\mathbf{x}'_i, w_i) = 1$. □

A.2 Anonymity Proof

Recall that there are three data-dependent components in the protocol that we need to handle: the NIZK proofs, the commitments, and the masked partial products. From a high level view, our proof goes first by replacing the honest clients' NIZK proofs and commitments with simulated ones. Then, the most challenging part is to show that the joint distribution of the honest clients' masked partial polynomial evaluations and the shuffled decoy terms can be simulated just by knowing the multiset of the honest clients' data items. We proved a key lemma (Lemma 5.4) that captures this core statistical reasoning step in the proof. We provide the full proof below.

Theorem A.2 (t-anonymity)**.** *Given a security parameter λ, assume $|\mathbb{F}|$ is superpolynomial in λ, $n-t \geq 19$, $d \geq \omega\left(\frac{\log|\mathbb{F}|+\log\lambda}{\log(n-t)}\right)$. Moreover, suppose that the NIZK scheme satisfies zero-knowledge, and the commitment scheme is computationally hiding. Then, our shuffle-model ZKP protocol satisfies t-anonymity in the presence of a semi-honest server.*

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. The real-world experiment is the same as the experiment $\mathsf{Expt}^{n,\mathcal{A}}(1^\lambda)$ as defined in Section 3.3. Recall that we assume that if the server is corrupted, it will still act honestly except that the adversary can see the server's view (including its internal coins and messages sent and received). However, the corrupted clients can act arbitrarily and in a way possibly dependent on the server's internal coins.

We can imagine that in the real-world experiment, there is a challenger acting on behalf of the trusted setup and all honest parties and interacting with adversary who controls the corrupted parties. The challenger also implements the oracle $\mathcal{F}_{\text{shuffle}}$ for the adversary.

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.3. *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.4. *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 almost identical to Hyb_2 except that if the challenge r sampled happens to be one of $\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]}$, we simply abort the current execution, and retry till we encounter a run in which the random challenge r does not collide with $\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]}$.

Claim A.5. *Hyb_3 and Hyb_2 have statistical distance at most $(n-t)\cdot m/|\mathbb{F}|$.*

Proof. The random challenge r is always sampled honestly when the server is semi-honest. Therefore, the probability of retrying is at most $(n-t)\cdot m/|\mathbb{F}|$. Thus, the claim follows by the definition of statistical distance. □

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

- After the adversary chooses $\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]}$, 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.6. *Suppose $n-t\geq 19$, $d\geq\omega\left(\frac{\log|\mathbb{F}|+\log\lambda}{\log(n-t)}\right)$. Then, Hyb_4 and Hyb_3 have negligibly small in λ statistical distance.*

Proof. Recall that the NIZK proofs and commitments for honest clients have been replaced with simulated proofs and commitments of 0. Also, since the challenger simulates $\mathcal{F}_{\mathsf{shuffle}}$ for the adversary, from what $\mathcal{F}_{\mathsf{shuffle}}$ outputs to the adversary, effectively the adversary can see the honest clients' multisets $\mathsf{Multiset}(\{x'_{i,j}\}_{i\in\mathcal{H},j\in[m]})$ and $\mathsf{Multiset}(\{y_{i,j}\}_{i\in\mathcal{H},j\in[d]})$. By Lemma 5.4, Hyb_3 and Hyb_4 have statistical distance at most $2^{-\sigma}$ as long as $d\geq\left\lceil\frac{2\sigma+\log_2(|\mathbb{F}|-1)}{\log_2(n-t)-\log_2 e}+2\right\rceil$. This means that the statistical distance is negligibly small in λ as long as $d\geq\omega\left(\frac{\log|\mathbb{F}|+\log\lambda}{\log(n-t)}\right)$. □

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.

□

B Proofs for the Malicious-Server Setting

In this section, we prove the security of our upgraded protocol in the presence of a potentially malicious server. Specifically, in the soundness proof, we still assume that the server is honest; however, in the anonymity proof, we need to take into account the possibility of a malicious server.

B.1 Additional Preliminaries: Extractable Commitment

Recall that earlier in Section 4.2, we defined a non-interactive commitment scheme. We now define a non-interactive extractable commitment scheme under a CRS model. In particular, there is a simulated CRS generation algorithm that generates a trapdoor trap along with a simulated CRS, such that using trap, one can extract the witness from the commitment. Such an extractable commitment scheme can be implemented using any committing public-key encryption scheme where the CRS is the public key, the commitment is an encryption of the message, and the trapdoor is the decryption key.

A non-interactive extractable commitment scheme consists of the following possibly randomized algorithms:

- $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$: takes in the security parameter 1^λ and samples a common reference string crs.
- $c \leftarrow \mathsf{commit}(\mathsf{crs}, x)$: given crs and some message x from an appropriate message space, outputs a commitment c. Whenever needed, we use the notation $c \leftarrow \mathsf{commit}(\mathsf{crs}, x; \mathsf{coins})$ to explicitly denote the random coins denoted coins consumed by the commitment algorithm.

We now define the desired security properties.

Perfectly binding. A non-interactive commitment scheme $(\mathsf{Gen}, \mathsf{commit})$ is said to be perfectly binding, iff for any crs in the support of Gen, there does not exist (x, coins) and (x', coins') where $x \neq x'$, such that $\mathsf{commit}(\mathsf{crs}, x; \mathsf{coins}) = \mathsf{commit}(\mathsf{crs}, x'; \mathsf{coins}')$.

Computationally hiding. A non-interactive commitment scheme $(\mathsf{Gen}, \mathsf{commit})$ is said to be computationally hiding, iff for any x and x', the following probability ensembles are computationally indistinguishable:

- $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$, $c \leftarrow \mathsf{commit}(\mathsf{crs}, x)$, output crs, c.
- $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$, $c' \leftarrow \mathsf{commit}(\mathsf{crs}, x')$, output crs, c'.

Extractability under a simulated CRS. There exists a simulated CRS generation algorithm denoted $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{Gen}'(1^\lambda)$, and an extractor algorithm $\mathsf{Ex}(\mathsf{trap}, c)$ such that the first outcome of $\mathsf{Gen}'(1^\lambda)$ is identically distributed as the outcome of $\mathsf{Gen}(1^\lambda)$, and moreover, the following holds for any x:

$$\Pr\Big[(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{Gen}'(1^\lambda), c \leftarrow \mathsf{commit}(\mathsf{crs}, x) : \mathsf{Ex}(\mathsf{trap}, c) = x\Big] = 1$$

It is not hard to see that extractability implies perfectly binding.

Remark B.1 (Using an RO-based commitment scheme in our implementation)**.** In our actual implementation, we use a random-oracle-based commitment scheme. Let H denote a random oracle. To commit to x, simply compute $H(x, \mathsf{coins})$ for some randomly chosen coins. To open the commitment, simply reveal the pair (x, coins). Using this random-oracle-based construction has the advantage that we can have a succinct commitment even when the message is long. Further, with a random oracle, extraction is easily accomplished using standard techniques by having the reduction implement the random oracle for the adversary (rather than using a trapdoor).

B.2 Soundness Proof

Soundness. We first prove the soundness property of the protocol.

Theorem B.2 (Soundness of the upgraded protocol)**.** *Suppose the field size $|\mathbb{F}|$ is superpolynomial w.r.t. the security parameter λ, the commitment scheme used by the clients satisfies computationally hiding and extractability, the commitment scheme used by the server is perfectly binding, and the* NIZK *scheme satisfies soundness. Then, the above upgraded protocol satisfies soundness.*

Proof. We consider a hybrid experimented henceforth denoted Hyb_1, in which a challenger interacts with an adversary who acts on behalf of the clients and attempt to break soundness. The challenge follows the honest protocol except that during the CRS setup, it calls $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{Gen}'(1^\lambda)$ to choose a simulated CRS crs and a trapdoor trap for the extractable commitment scheme. Since the crs output by Gen' is identically distributed as the crs output by Gen, the adversary's probability in breaking soundness in Hyb_1 is identically distributed as in the real-world experiment.

Therefore, it suffices to prove soundness for the Hyb_1 experiment. Let $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ be the data items the server obtained during the data collection phase. Let $\mathsf{Multiset}(\mathbf{y}_1, \ldots, \mathbf{y}_n)$ and $\{\mathsf{com}_i\}_{i\in[n]}$ be the decoy terms and commitments the server has received from the clients before the random challenge r is opened. Except with the negligibly small probability that some underlying NIZK instance's soundness is broken, if the server accepts, 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. 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

$$\prod_{i\in[n],j\in[m]} (x'_{i,j} - r) \cdot \alpha' = \prod_{i\in[n],j\in[m]} (x_{i,j} - r) \cdot \alpha \tag{2}$$

where

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

If $\mathcal{A}$ can break soundness, it means that for a non-negligible fraction of the challenges (henceforth denoted the set B), there is a non-negligible probability that the above Equation (2) holds after the adversary has submitted the commitment and the decoy terms, and $\mathsf{Multiset}(\mathbf{x}'_1, \ldots, \mathbf{x}'_n) \neq \mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

Now consider the following efficient reduction $\mathcal{R}$ that tries to break the commitment scheme by leveraging a soundness adversary $\mathcal{A}$. Imagine that in world 0, the reduction $\mathcal{R}$ receives c which is a commitment of an arbitrary fixed $r_0 \in B$, and in world 1, the reduction also receives c, but it is now a commitment of a randomly chosen challenge r_1. The reduction $\mathcal{R}$ wants to distinguish which world it is in. To achieve this, the reduction $\mathcal{R}$ runs Hyb_1 with the adversary $\mathcal{A}$ plugging in committed challenge c it has received, and it waits for the adversary to output the commitments $\{\mathsf{com}_i\}_{i\in[n]}$ and the decoy terms $\{y_{i,j}\}_{i\in[n],j\in[d]}$. It then uses the trapdoor to extract the $x'_{i,j}$ and ρ'_i values under the commitment. Now, $\mathcal{R}$ outputs 0 if $\mathsf{Multiset}(\mathbf{x}'_1, \ldots, \mathbf{x}'_n) \neq \mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and Equation (2) holds for the challenge r_0; else it outputs 1.

Since $\mathcal{A}$ can break soundness, in world 0, the probability that the reduction $\mathcal{R}$ outputs 0 is non-negligible. By the computational hiding property of the commitment scheme, the probability that $\mathcal{R}$ outputs 0 in world 1 is also non-negligible. In world 1, the distribution of $\{x_{i,j}\}$, $\{x'_{i,j}\}$ and $\{y_{i,j}\}$ is independent of r_0 in Hyb_1 since the challenge r_1 is sampled completely at random and does not encode any information of r_0. Since the size of B is a non-negligible fraction, if we run the experiment of world 1 for a randomly chosen r_0, the probability that $\mathcal{R}$ outputs 0 should also be non-negligible; however, this contradicts Lemma 5.6.

□

B.3 Anonymity Proof

We now prove that the upgraded protocol satisfies anonymity in the presence of a malicious server.

Theorem B.3 (t-anonymity in the presence of a malicious server)**.** *Given a security parameter λ, assume $|\mathbb{F}|$ is superpolynomial in λ, $n - t \geq 19$, $d \geq \omega\left(\frac{\log|\mathbb{F}|+\log\lambda}{\log(n-t)}\right)$. Moreover, suppose that the NIZK scheme satisfies zero-knowledge, the commitment scheme used by the client is computationally hiding, and the commitment scheme used by the server is perfectly binding. Then, the upgraded protocol satisfies t-anonymity (even in the presence of a malicious server).*

Proof. The proof follows in the same fashion as that of Theorem A.2, except that to show the statistical indistinguishability of Hyb_2 and Hyb_3, we need to rely on a couple more observations. Specifically, observe that Lemma 5.4 holds as long as the challenge r is chosen independently of the decoy terms $\{y_{i,j}\}_{i\in\mathcal{H},j\in[d]}$ and the challenge r is not equal to any of the data items $\{x_{i,j}\}_{i\in\mathcal{H},j\in[m]}$. In our upgraded protocol, the latter fact is guaranteed by construction by requiring r to come from a different half of the field than the data items. Because the adversary has to commit to r upfront without having observed the decoy terms $\{y_{i,j}\}_{i\in\mathcal{H},j\in[d]}$, and the commitment scheme is perfectly binding, the choice of r is independent of $\{y_{i,j}\}_{i\in\mathcal{H},j\in[d]}$. □

C Extension: Protocol with Identifiable Abort

In our protocol so far, if the verification fails, the server simply outputs $\perp$. Ideally, if the protocol fails, we want to catch some of the cheating clients. In this section, we introduce an extension of the basic protocol that supports identification of cheating clients, a property also called *identifiable abort.*

Syntax. We modify the syntax defined in Section 3.2 to support identifiable abort, where the changes are highlighted in blue. An *anonymous, compliant data collection* with *identifiable abort* has the following syntax:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \boldsymbol{\mathcal{R}})$:
- Pool or $\mathcal{I} \leftarrow \Pi(\mathsf{pp}, \mathbf{x}_1, \ldots, \mathbf{x}_n, w_1, \ldots, w_n)$: All parties have the input pp and moreover, each client i has a multiset of data items $\mathbf{x}_i := \{x_{i,j}\}_{j\in[m]}$, and witness w_i. The client and the server then engage in protocol, such that at the end of the protocol, the server either outputs a multiset Pool of data items, or outputs $\mathcal{I}$ which is a subset of the cheating clients.

Completeness and t-anonymity are defined as before. We strengthen the previous soundness definition to the following version:

Soundness with identifiable abort. Soundness with identifiable abort requires that for any n that is polynomially bounded in λ, any $\mathcal{C} \subseteq [n]$, any $\boldsymbol{\mathcal{R}}$ (where the circuits for checking the NP relations are polynomially bounded in λ), for any non-uniform PPT adversary $\mathcal{A}$ that controls the set $\mathcal{C}$ of corrupt clients (but not the server), there exists a negligible function $\mathsf{negl}(\cdot)$, such that in the above randomized experiment $\mathsf{Expt}^{n,\mathcal{A},\boldsymbol{\mathcal{R}}}(1^\lambda)$, except with $1 - \mathsf{negl}(\lambda)$ probability, either the server outputs a *non-empty* $\mathcal{I} \subseteq \mathcal{C}$, or it outputs a multiset Pool which must satisfy the following:

1. $\mathsf{Multiset}(x_{\mathcal{H}}) \subseteq \mathsf{Pool}$;
2. there exists a disjoint partitioning $\{\mathbf{x}_j\}_{j\in\mathcal{C}}$ of $\mathsf{Pool}\backslash\mathsf{Multiset}(\{\mathbf{x}_i\}_{i\in\mathcal{H}})$, such that for any $j \in \mathcal{C}$, there exists some w_j such that $\mathcal{R}_j(\mathbf{x}_j, w_j) = 1$.

C.1 High Level View

In our original protocol, if the server detects an invalid proof, it can identify the cheating clients. However, a smart adversary can still cheat by using inconsistent data items during the audit phase, and just fails the consistency check. Our polynomial based consistency check, although being highly efficient, is not feasible to identify the cheating clients.

The main purpose of our extension is to identify the clients who are using inconsistent data items. Say the set of data items collected during the collection phase is S_x and the (multi)set of data items used during the audit phase is S'_x. For now, we assume S_x does not contain any duplicates. We know the size of these two sets are the same, but they are not consistent with each other. Therefore, there could be two reasons for this inconsistency:

- *There exists alien data items in S'_x.* For example, $S_x = \{1, 2, 3, 4\}$ but $S'_x = \{1, 2, 3, 5\}$.
- *There exists duplicate data items in S'_x.* For example, $S_x = \{1, 2, 3, 4\}$ but $S'_x = \{1, 2, 3, 3\}$.

To handle the first case, we can require an extra zero-knowledge proof from each client attesting that the data items they used in the audit phase are from S_x. This can be done with any efficient membership proof scheme, such as Merkle tree. The second case is more challenging. Our solution is to require each client to attach a commitment of a random serial number to each data item they send to the shuffler. Then, during the blame attributing phase, we require the clients to *directly send* these random serial numbers to the server, while proving the commitment of these serial numbers are indeed attached to a data item in S_x, again using the same membership proof approach. Then, if the server sees one serial number being used multiple times, it can identify the cheating clients. Notice that an adversary that does not control the server cannot cause an honest client to be blamed except with negligible probability, because it can only guess the honest clients' serial numbers. This also preserves anonymity by the hiding property of the commitment scheme.

C.2 Additional Preliminaries

Our protocol with identifiable abort will make use of an anonymous network functionality that additionally supports private retrieval, as represented by the ideal functionality $\mathcal{F}^*_{\text{shuffle}}$. $\mathcal{F}^*_{\text{shuffle}}$ can be viewed as a stronger version of $\mathcal{F}_{\text{shuffle}}$ enhanced with some additional capabilities. Below we first define this functionality, and then we describe how to instantiate it.

Syntax of $\mathcal{F}^*_{\text{shuffle}}$. The functionality $\mathcal{F}^*_{\text{shuffle}}$ receives m data items $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,m})$ from each client $i \in [n]$. If some clients abort without sending m data items to $\mathcal{F}^*_{\text{shuffle}}$, then $\mathcal{F}^*_{\text{shuffle}}$ outputs $\mathcal{I}$ which is a non-empty subset of the aborting clients. Otherwise, $\mathcal{F}^*_{\text{shuffle}}$ sends $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to the server. Meanwhile, it computes a Merkle hash tree on top of the unordered multiset[8] $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, and offers the following query interface to everyone:

- On receiving a query `digest` from anyone, it returns the Merkle digest `dt`.
- On receiving a query of the form (`mproof`, $x_{i,j}$) from anyone, it returns a Merkle proof for $x_{i,j}$ w.r.t. the digest `dt`. If $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ contains multiple copies of $x_{i,j}$, $\mathcal{F}_{\text{retrieval}}$ may return the Merkle proof for an arbitrary copy.

Intuitively, besides being an anonymous shuffler, $\mathcal{F}^*_{\text{shuffle}}$ additionally offers the following capabilities: 1) consensus on the Merkle digest of the multiset of data items collected from all users;

[8]To achieve this, we can order $\mathsf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ using some canonical ordering, and build a Merkle tree on top of the resulting vector of items.

and 2) a private retrieval functionality where anyone can retrieve the Merkle proof of any data item that belong to the multiset. Importantly, when an honest client retrieves a Merkle proof for one of its own data items, the adversary cannot observe which data item it is querying.

Instantiating $\mathcal{F}^*_{\text{shuffle}}$. We give two different instantiations depending on whether the server is semi-honest or malicious.

If the server is assumed to be semi-honest, we can instantiate $\mathcal{F}^*_{\text{shuffle}}$ as follows — for our evaluation results in Appendix D, we use this instantiation. Below we assume that $\mathcal{F}_{\text{shuffle}}$ supports identifiable abort too, that is, if some client did not contribute data items, $\mathcal{F}_{\text{shuffle}}$ will report a non-empty subset $\mathcal{I}$ of cheating clients.

- Every client $i \in [n]$ sends $\mathbf{x}_i$ to $\mathcal{F}_{\text{shuffle}}$, and $\mathcal{F}_{\text{shuffle}}$ sends $\mathbf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to the server. If some clients aborted, then everyone outputs the set $\mathcal{I}$ reported by $\mathcal{F}_{\text{shuffle}}$.
- The server computes a Merkle digest dt of $\mathbf{Multiset}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and send dt to every client.
- Using a Private Information Retrieval (PIR) protocol that supports key-value search, every one can privately retrieve the Merkle digest for some data item $x_{i,j}$ that belongs to the resulting multiset.

If the server is malicious, then we can realize $\mathcal{F}^*_{\text{shuffle}}$ assuming an anonymous public bulletin board, which can be instantiated by having players send messages over an anonymous network and post them on a blockchain. Similarly, players can also query any content posted to the blockchain through an anonymous network.

C.3 Protocol with Identifiable Abort

Intuition. To identify cheating clients, each client will additionally attach a committed serial number to every data item $x_{i,j}$ and every decoy term $y_{i,j}$ it posts to $\mathcal{F}^*_{\text{shuffle}}$. Henceforth, we will use $\mathsf{sx}_{i,j}$ to denote the serial number associated with $x_{i,j}$, and use $\mathsf{sy}_{i,j}$ to denote the serial number associated with $y_{i,j}$. We use $\widetilde{\mathsf{sx}}_{i,j}$ and $\widetilde{\mathsf{sy}}_{i,j}$ to denote the commitment of $\mathsf{sx}_{i,j}$ and $\mathsf{sy}_{i,j}$, respectively. Using this notation, each client $i \in [n]$ will post to $\mathcal{F}^*_{\text{shuffle}}$ $\{(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})\}_{j\in[m]}$ and $\{(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})\}_{j\in[d]}$.

During the audit phase, if any client's zero-knowledge proof fails to verify, the server will report these clients. However, if all clients' proofs verify but the set consistency check fails, then the server and the clients engage in an additional blame phase to identify the cheating clients. During the blame phase, each client i will privately retrieve from $\mathcal{F}^*_{\text{shuffle}}$ a Merkle proof for each $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ pair and each $(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})$ pair it posted to $\mathcal{F}^*_{\text{shuffle}}$ earlier. It will then open the corresponding serial numbers $\{\mathsf{sx}_{i,j}\}_{j\in[m]}$ and $\{\mathsf{sy}_{i,j}\}_{j\in[d]}$ to the server, and prove in zero knowledge that it knows $\{(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})\}_{j\in[m]}$ and $\{(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})\}_{j\in[d]}$, such that

1. every $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ and every $(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})$ has a valid Merkle proof;
2. $\{\mathsf{sx}_{i,j}\}_{j\in[m]}$ and $\{\mathsf{sy}_{i,j}\}_{j\in[d]}$ are correct openings of $\{\widetilde{\mathsf{sx}}_{i,j}\}_{j\in[m]}$ and $\{\widetilde{\mathsf{sy}}_{i,j}\}_{j\in[d]}$, respectively; and
3. the commitment com_i it sent earlier correctly encodes $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,m})$ and $\rho_i = \prod_{j\in[d]} y_{i,j}$.

Intuitively, by revealing the underlying serial numbers committed to earlier and with the zero-knowledge proof, each client is privately declaring ownership of the pairs $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ and $(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})$ it contributed earlier, without actually identifying which pairs. Since each client talks to the server through a private channel and the commitment scheme reveals nothing about the committed serial numbers, a malicious client is unable to falsely guess and claim any serial number belonging to an honest client (except with negligible probability). This means that if the set consistency check

- **Data Collection Phase**:
  - Each client i samples serial numbers $(\mathsf{sx}_{i,1},\ldots,\mathsf{sx}_{i,m}) \xleftarrow{\$} \{0,1\}^{\lambda\cdot m}$, $(\mathsf{sy}_{i,1},\ \ldots,\ \mathsf{sy}_{i,d}) \xleftarrow{\$} \{0,1\}^{\lambda\cdot d}$, and computes $\widetilde{\mathsf{sx}}_{i,j} = \mathsf{commit}(\mathsf{sx}_{i,j};\beta_{i,j})$, $\widetilde{\mathsf{sy}}_{i,j} = \mathsf{commit}(\mathsf{sy}_{i,j};\beta'_{i,j})$ where $\beta_{i,j}, \beta'_{i,j}$ are sampled randomly;
  - Each client i sends $\{(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})\}_{j\in[m]}$ to $\mathcal{F}^*_{\text{shuffle}}$. Then, $\mathcal{F}^*_{\text{shuffle}}$ sends $\mathsf{Multiset}(\{(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})\}_{i\in[n],j\in[m]}$ to the server or reports aborting clients $\mathcal{I}$.
  - Each client i samples the decoy $\{y_{i,j}\}_{j\in[d]}$ at random from $\{\lfloor p/2\rfloor+1,\ldots,p-1\}$. It then sends $\{(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})\}_{j\in[d]}$ to $\mathcal{F}^*_{\text{shuffle}}$. Then, $\mathcal{F}^*_{\text{shuffle}}$ sends $\mathsf{Multiset}(\{(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})\}_{i\in[n],j\in[d]}$ to the server or reports aborting clients $\mathcal{I}$.

- **Audit Phase**:
  - Execute the audit phase of Figure 2, except that we can skip the step of collecting the decoy terms $\{y_{i,j}\}_{i\in[n],j\in[d]}$ since we have front-loaded this in the data collection phase.
  - If the server detects an invalid proof or fails to hear from any client i, report the clients.
  - If the set consistency check fails, proceed to the blame phase.

- **Blame Phase**: // $\widetilde{S}_x := \mathsf{Multiset}\left((x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})_{i\in[n],j\in[m]}\right)$, $\widetilde{S}_y := \mathsf{Multiset}\left((y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})_{i\in[n],j\in[d]}\right)$

  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 $\mathcal{F}^*_{\text{shuffle}}$.
  - For $j \in [m]$, privately retrieve a membership proof $\mathsf{mkx}_{i,j}$ for $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ from $\mathcal{F}^*_{\text{shuffle}}$;
  - For $j \in [d]$, privately retrieve a membership proof $\mathsf{mky}_{i,j}$ for $(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})$ from $\mathcal{F}^*_{\text{shuffle}}$;
  - Call the NIZK prover to generate a proof π'_i:
    - * Public statement: $\mathsf{com}_i, \mathsf{digest}(\widetilde{S}_x)$, $\mathsf{digest}(\widetilde{S}_y)$, $\mathsf{sx}_{i,1},\ldots,\mathsf{sx}_{i,m}$, $\mathsf{sy}_{i,1},\ldots,\mathsf{sy}_{i,d}$;
    - * 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 $i' = i$).

Figure 4: **Extension: protocol with identifiable abort.** The variables com_i, γ_i, and ρ_i are inherited from the main stage of the protocol described earlier.

fails, it must be that either some data item $x_{i,j}$ or decoy term $y_{i,j}$ contributed by a malicious client $i \in \mathcal{C}$ is claimed in two commitments com_j and $\mathsf{com}_{j'}$ in the audit phase where $j, j' \in \mathcal{C}$. Due to the soundness of the zero-knowledge proofs, if both j, j' sent valid proofs in the blame phase, they must reuse some serial number $\mathsf{sx}_{i,j}$ or $\mathsf{sy}_{i,j}$. In this case, the server can catch j and j' by noticing that their purported serial numbers collided.

Formal description. A formal description of the protocol is shown in Figure 4, where we assume that each client talks to the server through a private channel (which can easily be realized through key exchange and encrypting the messages).

Cost analysis. Suppose we use the same parameter choices as Section 5, and use a SNARK scheme with the same performance bounds as stated in Section 5. Below we use $\widetilde{O}(\cdot)$ to hide logarithmic factors. In the $\mathcal{F}^*_{\text{shuffle}}$-hybrid model, each client i incurs $\widetilde{O}(|\mathcal{R}_i|)$ computation $\widetilde{O}(m)$ communication, where the notation $|\mathcal{R}_i|$ denotes the size of the circuit that checks the NP relation $\mathcal{R}_i$. Further, the server's computation and communication are upper bounded by $O(nm) + \widetilde{O}(n)$.

The above costs are stated for the $\mathcal{F}^*_{\text{shuffle}}$-hybrid model. When $\mathcal{F}^*_{\text{shuffle}}$ is actually instantiated, we also need to charge for the overhead of instantiating $\mathcal{F}^*_{\text{shuffle}}$ itself. In our evaluation in Section 7, we use an instantiation with a semi-honest server where the retrieval is accomplished through a PIR scheme (see Appendix C.2). Specifically, we use SimplePIR [HHCG+23] to instantiate the PIR in our evaluation. In this case, the server's computation will increase to $\widetilde{O}(n^2m^2)$ due to the cost of the PIR. If we employ known batch PIR techniques [ACLS18], the server's computation cost can be reduced to $\widetilde{O}(n^2m)$, since the m queries from a client can be served with $\widetilde{O}(n)$ server computation.

In Appendix C.4 and Appendix C.5, we prove the following theorem.

Theorem C.1 (Protocol for a malicious server)**.** *Suppose that $|\mathbb{F}|$ is superpolynomial in λ, $n - t \geq 19$ and $d = \omega\left(\frac{\log|\mathbb{F}| + \log\lambda}{\log(n-t)}\right)$. Further, suppose that the underlying* NIZK *satisfies completeness, soundness, and zero-knowledge, the commitment scheme used by the server is perfectly binding and computationally hiding, and the commitment scheme used by the client is computationally hiding and extractable. Then, the protocol described in this section satisfies completeness, soundness with identifiable abort, and t-anonymity in the presence of a malicious server.*

C.4 Proof of Anonymity

The anonymity proof is the same as the proof in Appendix B.3, except that with the following modifications: 1) in Hyb_1 where we switch the NIZK's CRS and NIZK proofs to simulated ones, we additionally switch the proofs in the blame phase to simulated ones as well; and 2) in Hyb_2 where we switch the honest clients' commitments to commitments of 0, we additionally switch the honest clients' committed serial numbers to commitments of 0.

C.5 Proof of Soundness with Identifiable Abort

If the protocol aborted identifying some subset $\mathcal{I}$ before the blame phase, due to the completeness of the NIZK and the definition of $\mathcal{F}^*_{\text{shuffle}}$, it is straightforward to see that the set $\mathcal{I}$ identified must be non-empty and contain only malicious clients. The soundness proof in Appendix B.2 directly implies that if the server accepts without invoking the blame phase, then the multiset output by the server must satisfy the conditions required by the soundness definition. Therefore, it suffices to show that except with negligible probability, if the protocol enters the blame phase, then the set $\mathcal{I}$ reported must be non-empty and contain only malicious clients. We prove this below.

Hybrid experiment with simulated commitments and proofs. First, we switch to Hyb_3 in the anonymity proof (see Appendix C.4), where the honest clients' commitments and zero-knowledge proofs are all replaced with commitments of 0 and simulated proofs. Since the adversary cannot distinguish whether it is in the real world or Hyb_3, it suffices to prove that in Hyb_3, except with negligible probability, if the protocol enters the blame phase, then the set $\mathcal{I}$ reported must be non-empty and contain only malicious clients.

Proving soundness in the hybrid experiment. In Hyb_3, the corrupt clients have no knowledge of the honest clients' serial numbers in an information theoretic sense (recall that we assume that the clients communicate with the server over a private channel). Therefore, except with negligible probability, the serial numbers $\{\mathsf{sx}_{i,j}\}_{i\in\mathcal{C},j\in[m]}$ and $\{\mathsf{sy}_{i,j}\}_{i\in\mathcal{C},j\in[d]}$ revealed by the corrupt clients in the blame phase do not collide with honest clients' revealed serial numbers. This guarantees that except with negligible probability, if anyone is reported, it cannot be an honest client.

It remains to show that except with negligible probability, if the protocol enters the blame phase, indeed it will report a non-empty set $\mathcal{I}$. Ignore the negligibly small probability that the adversary breaks the soundness of the NIZK scheme or the collision resistance property of the Merkle tree, then the following must be true. Let $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,m})$ and ρ_i be the values under the commitment com_i submitted by a corrupt client $i \in \mathcal{C}$ during the audit phase. Suppose during the blame phase, each corrupt client $i \in \mathcal{C}$ reveals the serial numbers $\{\mathsf{sx}_{i,j}\}_{j\in[m]}$ and $\{\mathsf{sy}_{i,j}\}_{j\in[d]}$. It must be that

1. for each $j \in [m]$, $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ belongs to the multiset of data items received by the server from $\mathcal{F}^*_{\text{shuffle}}$ during the data collection phase, where $\widetilde{\mathsf{sx}}_{i,j}$ is a correct commitment of $\mathsf{sx}_{i,j}$;
2. for each $j \in [d]$, $(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})$ belongs to the multiset of decoy terms received by the server from $\mathcal{F}^*_{\text{shuffle}}$ during the data collection phase, where $\widetilde{\mathsf{sy}}_{i,j}$ is a correct commitment of $\mathsf{sy}_{i,j}$; and
3. $\rho_i = \prod_{j\in[d]} y_{i,j}$.

Ignore the negligible probability that the corrupt clients reveal some serial number that collides with an honest client's serial number. The above also implies that each $(x_{i,j}, \widetilde{\mathsf{sx}}_{i,j})$ or $(y_{i,j}, \widetilde{\mathsf{sy}}_{i,j})$ belongs to the corrupt clients' contributions during the data collection phase.

We claim that if all the above hold and yet the set consistency check during the audit phase failed, it must be that there exist two corrupt clients $j, j' \in \mathcal{C}$, such that j and j' opened the same serial number during the blame phase — this is sufficient for concluding the proof. To see why this above claim is true, one can easily verify that if all corrupt clients opened distinct serial numbers during the blame phase, and assuming the above conditions, then the corrupt clients' committed data items during the audit phase $\{x_{i,j}\}_{i\in\mathcal{C},j\in[m]}$ must form a disjoint partitioning of the corrupt clients' contributions in the data collection phase, and the same also holds for the decoy terms $\{y_{i,j}\}_{i\in\mathcal{C},j\in[d]}$. Now, ignore the probability that the adversary breaks the NIZK of the audit phase, it must be that the set consistency check will pass during the audit phase.

D Implementation and Evaluation for the Blame Phase

We implement the blame phase for the Shuffle-DP Sum use case, based on Groth16 proof system. We use an open-sourced PIR implementation of SimplePIR [HHCG+23] to obtain benchmarks about the PIR-related costs. We report the detailed breakdown numbers in Table 3. We divide the costs into three parts, the NIZK-related costs, the PIR-related costs and other costs. The NIZK-related costs denote the proving costs for the clients, and denote the verifying time for the server. The other costs include the costs of building the Merkle tree and checking repeated serial numbers for the server, and include the costs of preparing the assignment (used by the prover algorithm) for the client. The other cost generating the constraint assignments, and the other costs for the server include The NIZK-related client time corresponds to the proof generation time, and the NIZK-related server time corresponds to the verification time; the communication cost includes the cost to transmit the proof. The PIR-related client time includes the PIR query generation time and the time to recover the PIR response, while the server time includes the query response time; the communication cost includes both the upload and download costs.

Client time. The proof generation process still dominates the cost, and it takes 4.6 seconds to generate the proof. The PIR and the other costs only take a few milliseconds.

Server time. The server generates the PIR responses upon receiving clients' queries, and this time dominates the cost. In our setup, we need a batched PIR query (including 118 sub-queries) to a database with 1.2×10^5 entries, where each entry is a 528-byte Merkle proof. By running a state-of-the-art open-sourced PIR implementation [HHCG+23], we estimate the server time cost to be 22.5ms per client. The server also needs to verify the proofs and finds repeated serial numbers. These operations only take a few milliseconds.

Communication. The communication cost is dominated by the PIR queries and the PIR query size is estimated to be 1.0MB. The NIZK proof size only takes 0.2KB, and the other parts take 11KB.

Components	**Client Time**	**Amt. Server Time**	**Comm.**
	Based on Groth16		
NIZK-related	<u>4.6s</u>	1.7ms	0.2KB
PIR-related	2.0ms	<u>22.5ms</u>	<u>1.0MB</u>
Others	3.9ms	3.0ms	11KB
Total	4.6s	27.2ms	1.0MB

Table 3: The breakdown costs of the blame phase for the Shuffle-DP Sum use case. The bottleneck components are highlighted by underlining. "Amt. Server Time" denotes the server time cost spent on each client. "Comm." denotes the communication cost per client.

E Discussions: Other Failed Attempts

To illustrate why our solution is technically non-trivial, we discuss some other naïve approaches to achieve shuffle-model ZKP using generic primitives.

One possible approach is to rely on Fully Homomorphic Encryption (FHE) and a succinct zero-knowledge proof protocol. Suppose that the server is semi-honest. During the audit, each client uses a threshold FHE to encrypt its set, and sends the encrypted set to the server. The server homomorphically evaluates a circuit that outputs 1 iff all clients' sets are a disjoint partitioning the pool of data items it has collected, and moreover, each client's set satisfies the compliance predicate C. It sends the encrypted output to all clients. Now, each client computes a decryption share, and proves in zero-knowledge that the decryption share is computed correctly. The server recovers the output from the decrypted shares.

This approach achieves $\widetilde{O}(m)$ communication and computation per client; however, it has two drawbacks 1) it is not concretely efficient; and 2) it only works for a semi-honest server. If the server is malicious, it can substitute the input pool to break clients' privacy. We are not aware of any way (without assuming a powerful functionality like the blockchain) where the clients need not download the entire pool or incur communication linear in the n, and yet can verify that the server used the correct set.