Improving Utility and Security of the Shuffler-based Differential Privacy

Tianhao Wang
Purdue University
tianhaowang@purdue.edu

Bolin Ding
Alibaba Group
bolin.ding@alibaba-inc.com

Min Xu
University of Chicago
xum@cs.uchicago.edu

Zhicong Huang
Alibaba Group
zhicong.huang@alibaba-inc.com

Cheng Hong
Alibaba Group
vince.hc@alibaba-inc.com

Jingren Zhou
Alibaba Group
jingren.zhou@alibaba-inc.com

Ninghui Li
Purdue University
ninghui@cs.purdue.edu

Somesh Jha
University of Wisconsin
jha@cs.wisc.edu

ABSTRACT

When collecting information, local differential privacy (LDP) alleviates privacy concerns of users because their private information is randomized before being sent it to the central aggregator. LDP imposes large amount of noise as each user executes the randomization independently. To address this issue, recent work introduced an intermediate server with the assumption that this intermediate server does not collude with the aggregator. Under this assumption, less noise can be added to achieve the same privacy guarantee as LDP, thus improving utility for the data collection task.

This paper investigates this multiple-party setting of LDP. We analyze the system model and identify potential adversaries. We then make two improvements: a new algorithm that achieves a better privacy-utility tradeoff; and a novel protocol that provides better protection against various attacks. Finally, we perform experiments to compare different methods and demonstrate the benefits of using our proposed method.

PVLDB Reference Format:
Tianhao Wang, Bolin Ding, Min Xu, Zhicong Huang, Cheng Hong, Jingren Zhou, Ninghui Li, Somesh Jha. Improving Utility and Security of the Shuffler-based Differential Privacy. *PVLDB*, 13(13): xxxx-yyyy, 2020.
DOI: https://doi.org/10.14778/3424573.3424576

1. INTRODUCTION

To protect data privacy in the context of data publishing, differential privacy (DP) [26] is proposed and widely accepted as the standard of formal privacy guarantee. DP mechanisms allow a server to collect users' data, add noise to the aggregated result, and publish the result. More recently, local differential privacy (LDP) has been proposed [25]. LDP differs from DP in that random noise is added by each user before the data is sent to the central server. Thus, users do not need to trust the server. This desirable feature of LDP has led to wider deployment by industry [31, 1, 23, 53]. Meanwhile, DP is still deployed in settings where the centralized server can be trusted (e.g., the US Census Bureau deployed DP for the 2020 census [4]). However, removing the trusted central party comes at the cost of utility. Since every user adds some independently generated noise, the effect of noise adds up when aggregating the result. As a result, while noise of scale (standard deviation) $\Theta(1)$ suffices for DP, LDP has noise of scale $\Theta(\sqrt{n})$ on the aggregated result (n is the number of users). This gap is fundamental for eliminating the trust in the centralized server, and cannot be removed by algorithmic improvements [18].

Recently, researchers introduced settings where one can achieve a middle ground between DP and LDP, in terms of both privacy and utility. This is achieved by introducing an additional party [19, 30, 9, 20]. The setting is called the *shuffler model*. In this model, each user adds LDP noise to data, encrypt it, and then send it to the new party called the shuffler. The shuffler permutes the users' reported data, and then sends them to the server. Finally the server decrypts the reports and obtains the result. In this process, the shuffler only knows which report comes from which user, but does not know the content. On the other hand, the server cannot link a user to a report because the reports are shuffled. The role of the shuffler is to break the linkage between the users and the reports. Intuitively, this anonymity can provide some privacy benefit. Therefore, users can add less noise while achieving the same level of privacy.

In this paper, we study this new model from two perspectives. First, we examine from the algorithmic aspect, and make improvement to existing techniques. More specifically, in [9], it is shown the essence of the privacy benefit comes from a "noise" whose distribution is independent of the input value, also called privacy blanket. While existing work leverages this, it only works well when each user's value is drawn from a small domain. To obtain a similar privacy benefit when the domain is large, we propose to use the local hashing idea (also considered in the LDP setting [12, 52, 11, 5]). That is, each user selects a random hash function, and uses LDP to report the hashed result, together with the selected hash function. By analyzing the utility and optimizing the parameters with respect to the utility metric (mean squared error), we present an algorithm that achieves accuracy orders of magnitude better than existing method. We call it Shuffler-Optimal Local Hash (SOLH).

We then work from the security aspect of the model. We review the system setting of this model and identify two types of attack that were overlooked: collusion attack and data-poisoning attack. Specifically, as there are more parties involved, there might exist collusions. While existing work assumes non-collusion, we explicitly consider the consequences of collusions among different parties and propose a protocol Private Encrypted Oblivious Shuffle (PEOS) that is safer under these colluding scenarios. The other attack considers the setting where the additional party introduces calibrated noise to bias the result or break the privacy protection.

Proceedings of the VLDB Endowment, Vol. 13, No. 13
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3424573.3424576

To overcome this, our protocol PEOS takes advantage of cryptographic tools to prevent the shufflers from adding arbitrary noise.

To summarize, we provide a systematic analysis of the shuffler-based DP model. Our main contributions are:

- We improve the utility of the model and propose SOLH.
- We design a protocol PEOS that provides better trust guarantees.
- We provide implementation details and measure utility and execution performance of PEOS on real datasets. Results from our evaluation are encouraging.

2. BACKGROUND

We assume each user possesses a value v from a finite, discrete domain $\mathscr{D}$, and the goal is to estimate frequency of $v \in \mathscr{D}$.

2.1 Differential Privacy

Differential privacy is a rigorous notion about individual's privacy in the setting where there is a trusted data curator, who gathers data from individual users, processes the data in a way that satisfies DP, and then publishes the results. Intuitively, the DP notion requires that any single element in a dataset has only a limited impact on the output.

DEFINITION 1 (DIFFERENTIAL PRIVACY). *An algorithm* $\mathbf{A}$ *satisfies* (ϵ, δ)*-DP, where* $\epsilon, \delta \geq 0$*, if and only if for any neighboring datasets* D *and* D'*, any set* $\mathbf{R}$ *of possible outputs of* $\mathbf{A}$*,*

$$\mathsf{Pr}\left[\mathbf{A}(D) \in \mathbf{R}\right] \leq e^{\epsilon}\, \mathsf{Pr}\left[\mathbf{A}(D') \in \mathbf{R}\right] + \delta$$

Denote a dataset as $D = \langle v_1, v_2, \ldots, v_n\rangle$, where each v_i is from some domain $\mathscr{D}$. Two datasets $D = \langle v_1, v_2, \ldots, v_n\rangle$ and $D' = \langle v'_1, v'_2, \ldots, v'_n\rangle$ are said to be neighbors, or $D \simeq D'$, iff there exists at most one $i \in [n] = \{1, \ldots, n\}$ such that $v_i \neq v'_i$, and $v_j = v'_j$ for any other $j \neq i$. When $\delta = 0$, we simplify the notation and call $(\epsilon, 0)$-DP as ϵ-DP.

2.2 Local Differential Privacy

Compared to the centralized setting, the local version of DP offers a stronger level of protection, because each user only reports the noisy data rather than the true data. Each user's privacy is still protected even if the server is malicious.

DEFINITION 2 (LOCAL DIFFERENTIAL PRIVACY). *An algorithm* $\mathbf{A}(\cdot)$ *satisfies* (ϵ, δ)*-local differential privacy* ((ϵ, δ)*-LDP), where* $\epsilon, \delta \geq 0$*, if and only if for any pair of input values* $v, v' \in \mathscr{D}$*, and any set* $\mathbf{R}$ *of possible outputs of* $\mathbf{A}$*, we have*

$$\mathsf{Pr}\left[\mathbf{A}(v) \in \mathbf{R}\right] \leq e^{\epsilon}\, \mathsf{Pr}\left[\mathbf{A}(v') \in \mathbf{R}\right] + \delta$$

Typically, $\delta = 0$ in LDP (thus ϵ-LDP). We review the perturbation-based LDP mechanisms that will be used in the paper.

Generalized Randomized Response. The basic mechanism in LDP is called randomized response [57]. It was introduced for the binary case (i.e., $\mathscr{D} = \{0, 1\}$), but can be easily generalized. Here we describe the generalized version of random response (GRR).

In GRR, each user with private value $v \in \mathscr{D}$ sends $\mathsf{GRR}(v)$ to the server, where $\mathsf{GRR}(v)$ outputs the true value v with probability p, and a randomly chosen $v' \in \mathscr{D}$ where $v' \neq v$ with probability $1 - p$. Denote the size of the domain as $d = |\mathscr{D}|$, we have

$$\forall_{y \in \mathscr{D}}\ \mathsf{Pr}\left[\mathsf{GRR}(v) = y\right] = \begin{cases} p = \frac{e^{\epsilon}}{e^{\epsilon}+d-1}, & \text{if } y = v \\ q = \frac{1}{e^{\epsilon}+d-1}, & \text{if } y \neq v \end{cases} \tag{1}$$

This satisfies ϵ-LDP since $\frac{p}{q} = e^{\epsilon}$. To estimate the frequency of $\tilde{f}_v$ for $v \in \mathscr{D}$, one counts how many times v is reported, denoted by $\sum_{i\in[n]} \mathbb{1}_{\{y_i = v\}}$, and then computes

$$\tilde{f}_v = \frac{1}{n} \sum_{i \in [n]} \frac{\mathbb{1}_{\{y_i = v\}} - q}{p - q} \tag{2}$$

where $\mathbb{1}_{\{y_i = v\}}$ is the indicator function that tells whether the report of the i-th user y_i equals v, and n is the total number of users.

Local Hashing. When d is large, the p value in Equation (1) becomes small, making the result inaccurate. To overcome this issue, the local hashing idea [12] lets each user map v to one bit, and then use GRR to perturb it. More formally, each user reports $\langle H, \mathsf{GRR}(H(v))\rangle$ to the server, where H is the mapping (hashing) function randomly chosen from a universal hash family. In this protocol, both the hashing step and the randomization step result in information loss. Later, Wang et al. [52] realized H does not necessarily hashes v to one bit. In fact, the output domain size d' of H is a tradeoff. The optimal d' is $e^{\epsilon} + 1$. The method is called Optimized Local Hash (OLH), and it is frequently used in LDP tasks (e.g., [54, 56, 58, 59]).

Similar to GRR, the result of OLH needs to be calibrated. Let $\langle H_i, y_i\rangle$ be the report from the i'th user. For each value $v \in \mathscr{D}$, to compute its frequency, one first computes $\sum_{i\in[n]} \mathbb{1}_{\{H_i(v) = y_i\}} = |\{i \mid H_i(v) = y_i\}|$, and then computes

$$\tilde{f}_v = \frac{1}{n} \sum_{i \in [n]} \frac{\mathbb{1}_{\{H_i(v) = y_i\}} - 1/d'}{p - 1/d'} \tag{3}$$

2.3 Cryptographic Primitives

We briefly review the cryptographic primitives that will be used.

Additive Homomorphic Encryption. In Additive Homomorphic Encryption (AHE) [45], one can apply an algebraic operation (denoted by $\oplus$, e.g., multiplication) to two ciphertexts c_1, c_2, and get the ciphertext of the addition of the corresponding plaintexts. More formally, there are two functions, encrypt function Enc and decrypt function Dec. Given two ciphertexts $c_1 = \mathrm{Enc}(v_1)$ and $c_2 = \mathrm{Enc}(v_2)$, we have $c_1 \oplus c_2 = \mathrm{Enc}(v_1 + v_2)$.

Additive Secret Sharing. In this technique, a user splits a secret value $v \in \{0, \ldots, d-1\}$ into $r > 1$ shares $\langle s_i\rangle_{i\in[r]}$, where $r - 1$ of them are randomly selected, and the last one is computed so that $\sum_i s_i \mod d = v$. The shares are then sent to r parties, so that each party only sees a random value, and v cannot be recovered unless all the r parties collaborate.

Oblivious Shuffle. In order to prevent the shuffler from knowing the mapping between the input and the output, multiple shufflers are introduced. A natural method is to connect the shufflers sequentially; and each shuffler applies a random shuffle. Another way of achieving oblivious shuffle is the resharing-based shuffle [16, 39] which utilizes secret sharing. Suppose there are r shufflers. The users send their values to the shufflers using secret sharing. Define $t = \lfloor r/2\rfloor + 1$ as the number of "hiders", and $r - t$ as the number of "seekers". The resharing-based oblivious shuffle [39] proceeds like a "hide and seek" game. In particular, the protocol runs in $\binom{r}{t}$ iterations (because there are $\binom{r}{t}$ ways to partition shufflers into hiders and seekers). For each iteration, the seekers each secretly shares its local value (a vector of shares) to the t hiders, respectively. Then the hiders accumulate the shares and shuffle their vectors using an agreed permutation (only the t hiders know the permutation order). The shuffled vectors are then distributed to all of the r auxiliary servers (each of the t hiders secret shares its local value to $r - t$

seekers). After $\binom{r}{t}$ rounds, none of the colluding $r-t$ auxiliary servers know about the final permutation order.

3. PROBLEM DEFINITION AND EXISTING TECHNIQUES

3.1 Problem Definition

Throughout the paper, we focus on the problem of histogram estimation, which is typically used for solving other problems in the LDP setting. We assume there are n users; each user i possesses a value v_i from a discret domain $\mathscr{D}$. The frequency of value $v \in \mathscr{D}$ is represented by $f_v = \frac{1}{n}\sum_{i\in[n]} \mathbb{1}_{\{v_i=v\}}$. The server's goal is to estimate the frequency for each v, denoted by $\tilde{f}_v$. We consider the shuffler model, which is the middle ground between DP and LDP. In particular, an auxiliary server called the shuffler is introduced. Users need to trust that the auxiliary server does not collude with the server. Our goal is to improve the shuffler model in terms of (1) accuracy of estimating $\tilde{f}_v$, and (2) security of the model itself. Given a fixed privacy guarantee, we use the mean squared error of the estimation, i.e., $\frac{1}{|\mathscr{D}|}\sum_{v\in\mathscr{D}}(f_v-\tilde{f}_v)^2$, as the metric.

3.2 Privacy Amplification via Shuffling

The shuffling idea was originally proposed in Prochlo [15], where a shuffler is inserted between the users and the server to break the linkage between the report and the user identification. The privacy benefit was investigated in [19, 30, 9]. It is proven that when each user reports the private value using GRR with ϵ_l-LDP, applying shuffling ensures centralized (ϵ_c, δ)-DP, where $\epsilon_c < \epsilon_l$. Table 1 gives a summary of these results. Among them, [9] provides the strongest result in the sense that the ϵ_c is the smallest, and the proof technique can be applied to other LDP protocols.

Table 1: Privacy amplification results. Each row corresponds to a method. The amplified ϵ_c only differs in constants. The circumstances under which the method can be used are different.

Method	Condition	ϵ_c
[30]	$\epsilon_l < 1/2$	$\sqrt{144\ln(1/\delta)\cdot\frac{\epsilon_l^2}{n}}$
[19]	$\sqrt{\frac{192}{n}\ln(4/\delta)} < \epsilon_c < 1$, binary	$\sqrt{32\ln(4/\delta)\cdot\frac{e^{\epsilon_l}+1}{n}}$
[9]	$\sqrt{\frac{14\ln(2/\delta)d}{n-1}} < \epsilon_c \le 1$	$\sqrt{14\ln(2/\delta)\cdot\frac{e^{\epsilon_l}+d-1}{(n-1)}}$

Privacy Blanket. The technique used in [9] is called *blanket decomposition*. The idea is to decompose the probability distribution of an LDP report into two distributions, one dependent on the true value and the other independently random; and this independent distribution forms a "privacy blanket". In particular, the output distribution of GRR given in Equation (1) is decomposed into

$$\forall_{y\in\mathscr{D}}\ \Pr[\mathsf{GRR}(v)=y] = (1-\gamma)\Pr[y\mid v] + \gamma\Pr[\mathsf{Uni}(\mathscr{D})=y]$$

where $\Pr[y\mid v]$ is the distribution that depends on v, and $\mathsf{Uni}(\mathscr{D})$ is uniformly random with $\Pr[\mathsf{Uni}(\mathscr{D})=y] = 1/d$. With probability $1-\gamma$, the output is dependent on the true input; and with probability γ, the output is random. Given n users, the $n-1$ (except the victim's) such random variables can be seen as containing some uniform noise (i.e., the $\gamma\ \Pr[\mathsf{Uni}(\mathscr{D})=y]$ part). For each value $v\in\mathscr{D}$, the noise follows $\mathsf{Bin}(n-1,\gamma/d)$. Intuitively, this noise makes the output uncertain. The following theorem, which is derived from Theorem 3.1 of [9], formalizes this fact.

THEOREM 1 (BINOMIAL MECHANISM). *Binomial mechanism adds independent noise $\mathsf{Bin}(n,p)$ to each component of the histogram. It satisfies (ϵ_c,δ)-DP where*

$$\epsilon_c = \sqrt{\frac{14\ln(2/\delta)}{np}}$$

In Theorem 1, the larger γ is, the better the privacy. Given GRR, we can maximize γ by setting $\Pr[y\mid v] = \mathbb{1}_{\{v=y\}}$, which gives $\gamma = \frac{d}{e^{\epsilon_l}+d-1}$. The binomial noise $\mathsf{Bin}(n-1, \frac{1}{e^{\epsilon_l}+d-1})$ thus provides $(\sqrt{14\ln(2/\delta)\cdot\frac{e^{\epsilon_l}+d-1}{(n-1)}},\delta)$-DP [9]. One limitation of [9] is that as GRR is used, the accuracy degrades with domain size d.

4. IMPROVING UTILITY OF THE SHUFFLER MODEL

This section focuses on improving utility of the shuffler model.

4.1 Unary Encoding for Shuffling

We first revisit the unary-encoding-based methods, also known as basic RAPPOR [31], and show that this method gives better utility when d is large. However, its communication is linear in d. In particular, in unary-encoding, the value v is transformed into a vector B of size d, where $B[v]=1$ and the other locations of B are zeros (note that this requires values of the domain $\mathscr{D}$ be indexed from 1 to d). Then each bit b of B is perturbed to $1-b$ independently. To satisfy LDP, the perturbation probability is set to $\frac{1}{e^{\epsilon/2}+1}$. Note that we use $\epsilon/2$ because for any two values v and v', their corresponding unary encodings differ by two bits. We can apply the privacy blanket argument and prove that an ϵ_l-LDP unary-encoding method satisfies (ϵ_c,δ)-DP after shuffling.

THEOREM 2. *Given an ϵ_l-LDP unary-encoding method, after shuffling, the protocol is (ϵ_c,δ)-DP, where*

$$\epsilon_c = 2\sqrt{14\ln(4/\delta)\cdot\frac{e^{\epsilon_l/2}+1}{n-1}}$$

PROOF. For any two neighboring datasets $D\simeq D'$, w.l.o.g., we assume they differ in the n-th value, and $v_n=1$ in D, $v_n=2$ in D'. As each bit is perturbed independently, we can ignore other bits and focus on location 1 and 2. For each location, there are $n-1$ users, each reporting a bit with probability

$$\forall_{y\in\{0,1\}}\ \Pr[B[j]\to y] = (1-\gamma)\mathbb{1}_{\{B[j]=y\}} + \gamma\Pr[\mathsf{Uni}(2)=y]$$

where we slight abuse the notation and use $\mathsf{Uni}(2)$ for $\mathsf{Uni}(\{0,1\})$. Given the perturbation probability $\Pr[1\to0] = \Pr[0\to1] = \frac{1}{e^{\epsilon_l/2}+1} = \gamma/2$, we can calculate that $\gamma = \frac{2}{e^{\epsilon_l/2}+1}$. After shuffling, the histogram of $n-1$ (except the victim's) such random variables follows $\mathsf{Bin}(n-1,\gamma/2)$. As there are two locations, by Theorem 1, we have $\epsilon_c = 2\sqrt{14\ln(4/\delta)\cdot\frac{e^{\epsilon_l/2}+1}{n-1}}$. □

4.2 Local Hashing for Shuffling

While sending B when d is large is fine for each user; with n users, receiving B's from the server side is less tolerable as it incurs $O(d\cdot n)$ bandwidth. To reduce the communication cost, we propose a hashing-based method. Its utility is worse than the unary-encoding based method (from the experiment, its MSE is at most twice as that for unary-encoding); but the overall communication bandwidth is smaller. In what follows, we prove the hashing-based method is private in the shuffler model.

We remind the readers that in local hashing, each user reports H and $y = \mathsf{GRR}(H(v))$. The hash function H is chosen randomly

from a universal hash family and hashes v from a domain of size d into another domain of size $d' \le d$; and GRR will report $H(v)$ with probability $\frac{e^{\epsilon_l}}{e^{\epsilon_l}+d'-1}$, and any other value (from the domain of size d') with probability $\frac{1}{e^{\epsilon_l}+d'-1}$ (Equation (1)). Here, whether user i reports *truthfully* or *randomly* are two random events, whose probabilities are denoted by $\mathsf{Pr}[\mathrm{Tru}_i]$ and $\mathsf{Pr}[\mathrm{Rnd}_i]$, respectively. More specifically, the user flips a coin with $P(\text{heads}) = (e^{\epsilon_l}-1)/(e^{\epsilon_l}+d'-1)$. If it lands heads, the user reports $H(v)$ and we call this event Tru_i. If it lands tails, the user picks a value uniformly at random from $0 \dots d'-1$. We call this event Rnd_i. We call this method SOLH, which stands for Shuffler-Optimal Local Hash.

THEOREM 3. *Given the ϵ_l-LDP* SOLH *method, after shuffling, the protocol is (ϵ_c, δ)-DP, where*

$$\epsilon_c = \sqrt{\frac{14\ln(2/\delta)(e^{\epsilon_l}+d'-1)}{n-1}}$$

PROOF. Denote $\mathbf{A}$ as the algorithm of SOLH in the shuffler model. Let $[\langle H_i, y_i\rangle]_{i\in[n]}$ be the outputs of all users before shuffling, and let $[\langle \hat{H}_j, \hat{y}_j\rangle]_{j\in[n]}$ be the output of $\mathbf{A}(D)$. W.l.o.g., we assume D and D' differ in the n-th value, i.e., $v_n \neq v'_n$. We denote R as the output from $\mathbf{A}(D)$. To prove $\mathbf{A}$ is (ϵ_c, δ)-DP, it suffices to show

$$\mathsf{Pr}_{R\sim\mathbf{A}(D)}\left[\frac{\mathsf{Pr}[\mathbf{A}(D)=R]}{\mathsf{Pr}[\mathbf{A}(D')=R]} \ge e^{\epsilon_c}\right] \le \delta$$

where the randomness is on coin tosses of all users' LDP mechanism and the shuffler's random shuffle. We first upper bound $\frac{\mathsf{Pr}[\mathbf{A}(D)=R]}{\mathsf{Pr}[\mathbf{A}(D')=R]}$ by assuming user n also report truthfully. That is (we shorten the notation and use $\mathsf{Pr}[X(D)]$ to denote $\mathsf{Pr}[\mathbf{A}(D)=R]$),

$$\begin{aligned}
&\frac{\mathsf{Pr}[X(D)]}{\mathsf{Pr}[X(D')]}\\
=&\frac{\mathsf{Pr}[X(D) \mid \mathrm{Tru}_n]\cdot\mathsf{Pr}[\mathrm{Tru}_n] + \mathsf{Pr}[X(D) \mid \mathrm{Rnd}_n]\cdot\mathsf{Pr}[\mathrm{Rnd}_n]}{\mathsf{Pr}[X(D') \mid \mathrm{Tru}_n]\cdot\mathsf{Pr}[\mathrm{Tru}_n] + \mathsf{Pr}[X(D') \mid \mathrm{Rnd}_n]\cdot\mathsf{Pr}[\mathrm{Rnd}_n]}\\
=&\frac{\mathsf{Pr}[\mathsf{Pr}[X(D)] \mid \mathrm{Tru}_n]\cdot\mathsf{Pr}[\mathrm{Tru}_n] + c}{\mathsf{Pr}[X(D') \mid \mathrm{Tru}_n]\cdot\mathsf{Pr}[\mathrm{Tru}_n] + c} \le \frac{\mathsf{Pr}[\mathsf{Pr}[X(D)] \mid \mathrm{Tru}_n]}{\mathsf{Pr}[X(D') \mid \mathrm{Tru}_n]}
\end{aligned}$$

where $c = \mathsf{Pr}[X(D) \mid \mathrm{Rnd}_n]\cdot\mathsf{Pr}[\mathrm{Rnd}_n] = \mathsf{Pr}[X(D') \mid \mathrm{Rnd}_n]\cdot\mathsf{Pr}[\mathrm{Rnd}_n]$ is a constant. Thus we assume user n reports truthfully, and omit the conditional part for simplicity. The rest of the proof proceeds in 5 steps:

• Step 1 (expand the probability expression):

Denote T as indices of the first $n-1$ users who report truthfully (i.e., with probability $1-\gamma = \frac{e^{\epsilon_l}-1}{e^{\epsilon_l}+d'-1}$), and let R_T denote their chosen hash functions and hashed results ($R_T = [\langle H_i, y_i\rangle]_{i\in T}$). We examine the conditional probability $\mathsf{Pr}[\mathbf{A}(D)=R \mid (T, R_T)]$:

$$\begin{aligned}
&\mathsf{Pr}[\mathbf{A}(D)=R \mid (T,R_T)]\\
=&\sum_\pi \mathsf{Pr}[\pi]\,\mathsf{Pr}[\mathbf{A}(D)=R \mid (T,R_T,\pi)]\\
=&\sum_\pi \mathsf{Pr}[\pi]\left(\underbrace{\prod_{i\in T}\mathsf{Pr}\left[H_{\pi(i)}\right]\mathbb{1}_{\left\{H_{\pi(i)}=\hat{H}_i\wedge y_{\pi(i)}=\hat{y}_i\right\}}}_{\text{reports from users in } T}\cdot\right. \qquad (4)\\
&\left.\underbrace{\prod_{i\in[n-1]\setminus T}\mathsf{Pr}\left[H_{\pi(i)}\right]\frac{1}{d'}}_{\text{reports from users in } [n-1]\setminus T}\cdot\underbrace{\mathsf{Pr}\left[H_{\pi(n)}\right]\mathbb{1}_{\left\{H_{\pi(n)}(v_n)=y_{\pi(n)}\right\}}}_{\text{report from user } n}\right)
\end{aligned}$$

$\mathsf{Pr}[\pi]$ denotes the probability a specific random permutation is chosen ($\mathsf{Pr}[\pi] = 1/n!$), $\mathsf{Pr}\left[H_{\pi(i)}\right]$ (short for $\mathsf{Pr}\left[\hat{H}_i = H_{\pi(i)}\right]$) is the probability user i chooses hash function $H_{\pi(i)}$ (assuming there are h possible hash functions, $\mathsf{Pr}\left[H_{\pi(i)}\right] = 1/h$), and the summation is over all permutation π. The users are divided into three groups. For $i \in T$, we know from R_T that his/her report is $\langle \hat{H}_i, \hat{y}_i\rangle$, and it must match $\langle H_{\pi(i)}, y_{\pi(i)}\rangle$ (otherwise $\mathsf{Pr}[\mathbf{A}(D)=R \mid (T,R_T,\pi)] = 0$). We use the indicator function to denote this. Note that here as the user reports truthfully, $\hat{y}_i = \hat{H}_i(v_i)$, and $\mathbb{1}_{\left\{H_{\pi(i)}=\hat{H}_i\wedge y_{\pi(i)}=\hat{y}_i\right\}} = \mathbb{1}_{\left\{H_{\pi(i)}=\hat{H}_i\wedge H_{\pi(i)}(v_i)=y_{\pi(i)}\right\}}$. For user n and users who report randomly, their probabilities can also be analyzed similarly.

• Step 2 (convert probabilities to counts):

Denote $P = \{\pi \mid \forall i\in T, H_{\pi(i)} = \hat{H}_i \wedge y_{\pi(i)} = \hat{y}_i\}$. Here P is the set of all possible permutations that make the $i \in T$ part of Equation (4) non-zero (i.e., all the indicator functions for $i \in T$ equal 1). Assuming the reports in R are distinct (i.e., $\nexists i, j \in [n]$ s.t. $H_i = H_j \wedge y_i = y_j$), such permutations must map $i \in T$ to $\pi(i)$ s.t. $H_{\pi(i)} = \hat{H}_i$ and $y_{\pi(i)} = \hat{y}_i$. P can be partitioned into $n - |T|$ equal-sized subsets each with $\pi(n) = i$. That is, for each $i \in [n]\setminus T$, define $P_i = \{\pi \mid \pi \in P \wedge \pi(n) = i\}$. Each P_i is of size $\mathbb{1}_{\left\{\hat{H}_i(v_n)=\hat{y}_i\right\}}\cdot(n-1-|T|)!$ because P_i left the mapping of $[n-1]\setminus T$ unspecified (and any random permutation is possible). We now have:

$$\begin{aligned}
\frac{\mathsf{Pr}[\mathbf{A}(D)=R \mid (T,R_T)]}{\mathsf{Pr}[\mathbf{A}(D')=R \mid (T,R_T)]} &= \frac{c_1\sum_{\pi\in P}\mathbb{1}_{\left\{H_{\pi(n)}(v_n)=y_{\pi(n)}\right\}}}{c_1\sum_{\pi\in P}\mathbb{1}_{\left\{H_{\pi(n)}(v'_n)=y_{\pi(n)}\right\}}}\\
=\frac{\sum_{i\in[n]\setminus T}\sum_{\pi\in P_i}\mathbb{1}_{\left\{\hat{H}_i(v_n)=\hat{y}_i\right\}}}{\sum_{i\in[n]\setminus T}\sum_{\pi\in P_i}\mathbb{1}_{\left\{\hat{H}_i(v'_n)=\hat{y}_i\right\}}} &= \frac{\sum_{i\in[n]\setminus T}\mathbb{1}_{\left\{\hat{H}_i(v_n)=\hat{y}_i\right\}}}{\sum_{i\in[n]\setminus T}\mathbb{1}_{\left\{\hat{H}_i(v'_n)=\hat{y}_i\right\}}}
\end{aligned} \qquad (5)$$

where $c_1 = \mathsf{Pr}[\pi]\left(\prod_{i\in[n]}\mathsf{Pr}\left[H_{\pi(i)}\right]\right)\left(\prod_{i\in[n-1]\setminus T}\frac{1}{d'}\right)$ is a constant that does not depend on v_n or v'_n. Note that we previously assumed the reports in R are unique. If there are duplicated reports, P could be larger, but the ratio stays the same. To see this, define $R_{-T} = [\langle \hat{H}_i, \hat{y}_i\rangle]_{i\in[n]\setminus T}$ as reports from $[n]\setminus T$. We model a valid permutation in P as a two-step process: for any report from user $i \in [n]\setminus T$, suppose there are $a_i \ge 0$ reports in R_T that is the same (both the hash function and the hash result are same) as user i's report, and $b_i \ge 1$ duplicated reports in R_{-T}. We first choose a_i from $a_i + b_i$ reports and "put" them to R_T; then we permute R_T (there are $c \ge 1$ valid permutations within R_T) and R_{-T} (there are $\sum_{i\in[n]\setminus T}\mathbb{1}_{\left\{\hat{H}_i(v_n)=\hat{y}_i\right\}}\cdot(n-1-|T|)!$ valid permutations in R_{-T}). It can be verified that this modeling covers exactly all permutations in P. Now for each $i \in [n]\setminus T$: If $a_i = 0$, there are $x_i = \mathbb{1}_{\left\{\hat{H}_i(v_n)=\hat{y}_i\right\}}\cdot\prod_{i\in[n]\setminus T}\binom{a_i+b_i}{a_i}\cdot c\cdot(n-1-|T|)!$ possible permutations in P, where $\prod_{i\in[n]\setminus T}\binom{a_i+b_i}{a_i}$ denotes the number of possible choices for the duplicated reports. If $a_i > 0$, denote $y_i = x_i/\binom{a_i+b_i}{a_i}$. We consider all these a_i+b_i duplicate reports together. Index n can be mapped to match any of the a_i+b_i duplicated reports. For each report, there are $\binom{a_i+b_i-1}{a_i}$ choices (because the permutation will first choose a_i reports and put them into R_T, and the current report which n is mapped to cannot be put to R_T; thus we choose a_i from the remaining a_i+b_i-1 reports to put to R_T). Overall, we have $y_i\cdot(a_i+b_i)\cdot\binom{a_i+b_i-1}{a_i} = y_i\cdot b_i\cdot\binom{a_i+b_i}{a_i} = x_i\cdot b_i$ valid permutations, which equals to the case when we sum all the b_i values each with x_i permutations. Therefore, there are $x_i = \mathbb{1}_{\left\{\hat{H}_i(v_n)=\hat{y}_i\right\}}\cdot c'$ valid permutations for each $i \in [n]\setminus T$. Summarizing all x_i's gives us Equation (5).

• Step 3 (model the counts with Binomial RVs):

So far, we have proved that, fixing R, T and R_T, the ratio only depends on the numbers of reports that are random and matches v_n and v'_n, respectively. The high level idea is to show that knowing

T and R_T fixes the permutation on values from T; and any valid permutation only shuffles values from $[n] \setminus T$ (informally, this can be thought of as the server removes reports from T). Now define

$$N_{R,T,R_T} = \sum_{i \in [n] \setminus T} \left(\mathbb{1}_{\{\hat{H}_i(v_n) = \hat{y}_i\}} \right)$$
$$\text{and } N'_{R,T,R_T} = \sum_{i \in [n] \setminus T} \left(\mathbb{1}_{\{\hat{H}_i(v'_n) = \hat{y}_i\}} \right),$$

we want to prove

$$\mathsf{Pr}_{(R,T,R_T) \sim \mathbf{A}(D)} \left[\frac{\mathsf{Pr}\left[\mathbf{A}(D) = R \mid (T, R_T)\right]}{\mathsf{Pr}\left[\mathbf{A}(D') = R \mid (T, R_T)\right]} \geq e^{\epsilon_c} \right]$$

(omit the $(R, T, R_T) \sim \mathbf{A}(D)$ part to simplify notations)

$$= \mathsf{Pr}\left[\frac{N_{R,T,R_T}}{N'_{R,T,R_T}} \geq e^{\epsilon_c} \right]$$
$$\leq 1 - \mathsf{Pr}\left[N_{R,T,R_T} \leq \theta e^{\epsilon_c/2} \wedge N'_{R,T,R_T} \geq \theta e^{-\epsilon_c/2} \right]$$
$$\leq \mathsf{Pr}\left[N_{R,T,R_T} \geq \theta e^{\epsilon_c/2} \right] + \mathsf{Pr}\left[N'_{R,T,R_T} \leq \theta e^{-\epsilon_c/2} \right]$$

where θ is some constant. For (R, T, R_T) generated from a random run of $\mathbf{A}(D)$, we can show N_{R,T,R_T} and N'_{R,T,R_T} follow Binomial distributions. In particular, as we assumed user n always report truth, there must be $H_n(v_n) = y_n$; the remaining $n-1$ users will first decide whether to report truthfully (i.e., with probability $(e^{\epsilon_l} - 1)/(e^{\epsilon_l} + d' - 1)$), and if user i's report $\langle H_i, y_i \rangle$ is random, we have $\mathsf{Pr}\left[H_i(v_n) = y_i\right] = 1/d'$. Each user's reporting process are thus modeled as two Bernoulli processes. As a result, N_{R,T,R_T} follows the Binomial distribution $\mathsf{Bin}(n-1, 1/(e^{\epsilon_l} + d' - 1))$ plus a constant 1. Similarly, $N'_{R,T,R_T} \sim \mathsf{Bin}(n-1, 1/(e^{\epsilon_l} + d' - 1)) + \mathbb{1}_{\{H_n(v'_n) = y_n\}} \geq \mathsf{Bin}(n-1, 1/(e^{\epsilon_l} + d' - 1))$.

• **Step 4 (bound the ratio of Binomial RVs with Chernoff bounds):** Following the later part of the proof of Theorem 3.1 from [9]: set $\theta = \frac{n-1}{e^{\epsilon_l} + d' - 1} = \mathbb{E}\left[N'_{R,T,R_T} \right] = \frac{14 \log(2/\delta)}{\epsilon^2}$,

$$\mathsf{Pr}\left[N_{R,T,R_T} \geq \theta e^{\epsilon_c/2} \right] + \mathsf{Pr}\left[N'_{R,T,R_T} \leq \theta e^{-\epsilon_c/2} \right]$$
$$= \mathsf{Pr}\left[N'_{R,T,R_T} \geq \theta e^{\epsilon_c/2} - 1 \right] + \mathsf{Pr}\left[N'_{R,T,R_T} \leq \theta e^{-\epsilon_c/2} \right]$$
$$\leq \mathsf{Pr}\left[N'_{R,T,R_T} - \mathbb{E}\left[N'_{R,T,R_T} \right] \geq \theta(e^{\epsilon/2} - 1 - 1/\theta) \right]$$
$$+ \mathsf{Pr}\left[N'_{R,T,R_T} - \mathbb{E}\left[N'_{R,T,R_T} \right] \leq \theta(e^{-\epsilon/2} - 1) \right]$$
$$\leq \exp(-\theta(e^{\epsilon/2} - 1 - 1/\theta)^2/3) + \exp(-\theta(1 - e^{-\epsilon/2})^2/2)$$

Assuming $\epsilon \leq 1$, both of them are less than or equal to $\delta/2$: For the first term, $\theta \geq \frac{27}{\epsilon}$ implies $e^{\epsilon/2} - 1 - 1/\theta \geq \frac{25}{54}\epsilon$ and $14 \geq \frac{3 \cdot 54^2}{25^2}$. For the second term, $1 - e^{\epsilon/2} \geq (1 - e^{1/2})\epsilon \geq \epsilon/\sqrt{7}$.

• **Step 5 (put things together):** We have bound the conditional probability ratio. It also implies a bound on joint probability ratio, because $\frac{\mathsf{Pr}[\mathbf{A}(D)=R|(T,R_T)]}{\mathsf{Pr}[\mathbf{A}(D')=R|(T,R_T)]} = \frac{\mathsf{Pr}[\mathbf{A}(D)=R \wedge (T,R_T)]\mathsf{Pr}[T,R_T]}{\mathsf{Pr}[\mathbf{A}(D')=R \wedge (T,R_T)]\mathsf{Pr}[T,R_T]} = \frac{\mathsf{Pr}[\mathbf{A}(D)=R \wedge (T,R_T)]}{\mathsf{Pr}[\mathbf{A}(D')=R \wedge (T,R_T)]}$. For any R, we say (T, R_T) is "good" if $e^{\epsilon_c} \geq \frac{\mathsf{Pr}[\mathbf{A}(D)=R \wedge (T,R_T)]}{\mathsf{Pr}[\mathbf{A}(D')=R \wedge (T,R_T)]}$ and "bad" otherwise. Consider any possible set S of output, we finally prove

$$\mathsf{Pr}\left[\mathbf{A}(D) \in S\right] = \sum_{(T,R_T)} \sum_{R \in S} \mathsf{Pr}\left[\mathbf{A}(D) = R \wedge (T, R_T)\right]$$
$$= \sum_{(T,R_T) \text{ is good}} \sum_{R \in S} \mathsf{Pr}\left[\mathbf{A}(D) = R \wedge (T, R_T)\right]$$
$$+ \sum_{(T,R_T) \text{ is bad}} \sum_{R \in S} \mathsf{Pr}\left[\mathbf{A}(D) = R \wedge (T, R_T)\right]$$
$$\leq \sum_{(T,R_T) \text{ is good}} \sum_{R \in S} e^{\epsilon} \mathsf{Pr}\left[\mathbf{A}(D') = R \wedge (T, R_T)\right]$$
$$+ \sum_{(T,R_T) \text{ is bad}} \sum_{R} \mathsf{Pr}\left[\mathbf{A}(D) = R \wedge (T, R_T)\right]$$
$$\leq \sum_{(T,R_T)} \sum_{R \in S} e^{\epsilon} \mathsf{Pr}\left[\mathbf{A}(D') = R \wedge (T, R_T)\right] + \delta$$
$$= e^{\epsilon} \mathsf{Pr}\left[\mathbf{A}(D') \in S\right] + \delta$$

□

4.3 Utility Analysis

Now we analyze the utility of different methods. We utilize the framework of Theorem 2 from [52] to analyze the accuracy of estimating the frequency of each value in the domain (i.e., Equations (2) and (3)). In particular, we measure the expected squared error of the estimation $\tilde{f}_v$, which equals variance, i.e.,

$$\sum_{v \in \mathscr{D}} \mathbb{E}\left[(\tilde{f}_v - f_v)^2 \right] = \sum_{v \in \mathscr{D}} \mathsf{Var}\left[\tilde{f}_v \right]$$

Fixing the local ϵ_l, the variances are already summarized in [52]. We first restate results from [52], and then extends it to the shuffler setting.

LEMMA 4. *Given the domain size d and the LDP parameter ϵ_l, the variance of* GRR *is* $\frac{e^{\epsilon_l} + d - 2}{n(e^{\epsilon_l} - 1)^2}$.

LEMMA 5. *Given the hashing domain size d' and the LDP parameter ϵ_l, the variance of local hash is* $\frac{(e^{\epsilon_l} + d' - 1)^2}{n(e^{\epsilon_l} - 1)^2(d' - 1)}$.

Utility of Generalized Randomize Response. We first prove the variance of GRR.

PROPOSITION 6. *Given ϵ_c in the shuffler model, the variance of using* GRR *is bounded by* $\frac{\frac{\epsilon_c^2(n-1)}{14 \ln(2/\delta)} - 1}{n\left(\frac{\epsilon_c^2(n-1)}{14 \ln(2/\delta)} - d\right)^2}$.

PROOF. From [9], we have $e^{\epsilon_l} + d - 1 = \frac{\epsilon_c^2(n-1)}{14 \ln(2/\delta)}$. Plugging it to Lemma 4 the variance is $\frac{\frac{\epsilon_c^2(n-1)}{14 \ln(2/\delta)} - 1}{n\left(\frac{\epsilon_c^2(n-1)}{14 \ln(2/\delta)} - d\right)^2}$. □

Utility of Unary Encoding (RAPPOR). Similarly, we can prove the variance of unary encoding.

PROPOSITION 7. *Given ϵ_c in the shuffler model, the variance of using unary encoding (RAPPOR) is bounded by* $\frac{\frac{\epsilon_c^2(n-1)}{56 \ln(4/\delta)} - 1}{n\left(\frac{\epsilon_c^2(n-1)}{56 \ln(4/\delta)} - 2\right)^2}$.

PROOF. For each value, the estimate is based on the number of 1's in the corresponding location. Thus we can apply Lemma 4 with $d = 2$. From Theorem 2, we have $e^{\epsilon_l/2} + 1 = \frac{\epsilon_c^2(n-1)}{56 \ln(4/\delta)}$. Thus the variance becomes $\frac{\frac{\epsilon_c^2(n-1)}{56 \ln(4/\delta)} - 1}{n\left(\frac{\epsilon_c^2(n-1)}{56 \ln(4/\delta)} - 2\right)^2}$. □

Utility of Local Hashing. Now we prove the variance of SOLH and instantiate d'.

PROPOSITION 8. *Given ϵ_c in the shuffler model, the variance of using* SOLH *is bounded by* $\frac{\left(\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}\right)^2}{n\left(\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}-d'\right)^2(d'-1)}$.

PROOF. From Theorem 3, we have $e^{\epsilon_l}+d'-1=\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}$. Plugging in Lemma 5, the variance is $\frac{\left(\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}\right)^2}{n\left(\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}-d'\right)^2(d'-1)}$. □

Optimizing Local Hashing. Note that d' is unspecified. We can tune d' to optimize variance given a fixed ϵ_c. Denote m as $\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}$, our goal is to choose d' that minimize this variance $\frac{m^2}{n(m-d')^2(d'-1)}$. By making its partial derivative to 0, we can obtain that when

$$d'=\frac{m+2}{3}=\frac{\epsilon_c^2(n-1)}{42\ln(2/\delta)}+\frac{2}{3} \tag{6}$$

the variance is minimized. Note that d' can only be an integer. In the actual implementation, we choose d' to be $\lfloor(m+2)/3\rfloor$.

Comparison of the Methods. We first observe that the variance of GRR grows with d (as shown in Proposition 6). When d is large, we should use unary encoding or local hashing. Between the two, the variance of unary encoding is slightly better, however, its communication cost is higher. Thus, between GRR and SOLH, we can choose the one with better utility by comparing Proposition 6 and $\mathsf{Var}(m,\lfloor(m+2)/3\rfloor)$.

4.4 Comparison with Parallel Work

Parallel to our work, [8, 34] also propose mechanisms to improve utility in this model. Among them [8] gives better utility which does not depend on $|\mathscr{D}|$. Similar to our method, its proof also utilizes Theorem 1. But the approach is different. In particular, [8] first transforms the data using one-hot encoding, then independently increment values in each location with probability $p=1-\frac{200}{\epsilon_c^2 n}\ln(4/\delta)$. We call this method AUE for appended unary encoding. As each location is essentially a Bernoulli bit, its variance is $p(1-p)=\frac{200}{\epsilon_c^2 n}\ln(4/\delta)\left(1-\frac{200}{\epsilon_c^2 n}\ln(4/\delta)\right)$. Compared with Lemma 8, this gives comparable results (differing by only a constant). But this protocol itself is not LDP. Moreover, as one-hot encoding is used, the communication cost for each user is linear in $|\mathscr{D}|$, which is even worse than GRR. We will empirically compare with [8] in the experimental evaluation section.

More recently, [29] also proposed a similar unary-encoding-based method. We note that [29] operate on a novel removal LDP notion. More specifically, previous (ours included) LDP and shuffler-based LDP literature works with Definition 2, which ensures that for each user, if his/her value changes, the report distribution is similar. [29] introduces a novel removal LDP notion inspired by the removal DP. In particular, removal DP states that for any two datasets D and D_-, where D_- is obtained by removing any one record from D, the output distributions are similar. Extending that idea to the local setting, removal LDP states that for each user, whether his/her value is empty or not, the report distribution is similar. Given that, a unary-encoding-based method similar to RAPPOR [31] is proposed. The method is similar to the method we described in Section 4.1, except that privacy budget ϵ_l is not divided by 2. Interestingly, any ϵ-Removal LDP algorithm is also a 2ϵ-Replacement LDP algorithm, because

$$\Pr[\mathbf{A}(v)\in\mathbf{R}]\le e^{\epsilon}\Pr[\mathbf{A}(\perp)\in\mathbf{R}]\le e^{2\epsilon}\Pr[\mathbf{A}(v')\in\mathbf{R}]$$

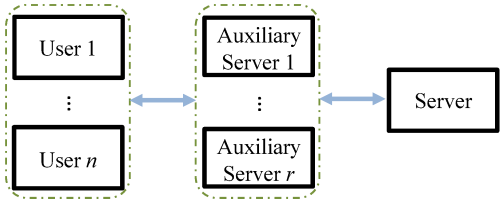

Figure 1: Overview of parties and interactions. Users communicate with the auxiliary servers. The auxiliary servers processes the users' data, and communicate with the server.

where $\perp$ is a special "empty" input introduced in Removal-LDP. Thus, in our LDP setting, the two methods achieves the same utility.

5. SECURITY ANALYSIS

This section focuses on the analyzing the security implications of the shuffler model. We identify different parties and potential attacks. Then we propose countermeasures using secret sharing and oblivious shuffle in next section.

5.1 Parties and Attackers

There are three types of parties in the shuffler model: *users*, the *server*, and the *auxiliary servers* (shufflers). The auxiliary servers do not exist in the traditional models of DP and LDP; and in DP, the server may share result with some external parties. Figure 1 provides an overview of the system model.

The Attackers. From the point of view of a single user, other parties, including the auxiliary server, the server, and other users, could all be adversaries. We assume all parties have the same level of background knowledge, i.e., all other users' information except the victim's. This assumption essentially enables us to argue DP-like guarantee for each party.

The prominent adversary is the server. Other parties can also be adversaries but are not the focus because they have less information. For example, in the shuffler-based approach, there is only one auxiliary server. It knows nothing from the ciphertext.

Additional Threat of Collusion. We note that in the multi-party setting, one needs to consider the consequences when different parties collude. In general, there are many combinations of colluding parties. And understanding these scenarios enables us to better analyze and compare different approaches.

In particular, the server can collude with the auxiliary servers. If all the auxiliary servers are compromised, the model is reduced to that for LDP. Additionally, the server can also collude with other users (except the victim), but in this case the model is still LDP. On the other hand, if the server only colludes with other users, the basic shuffler model degrades to LDP; but it seems we can add more functionality to the shuffler to provide better guarantee. Other combinations are possible but less severe. Specifically, there is no benefit if the auxiliary servers collude with the users. We consider collusions and highlight three important (sets of) adversaries:

- Adv: the server itself.
- Adv_a: the server with the auxiliary servers.
- Adv_u: the server colluding with other users.

5.2 Privacy Guarantees of Existing Methods

Having identified the potential adversaries and the proving technique, now we examine the shuffler-based DP. The key ideas are (1) We model each attack's view using an algorithm, such that we can prove the DP guarantee. (2) We prove the DP guarantee for

each party separately. Existing work focuses on Adv, but we examine the privacy guarantee also for Adv_a and Adv_u. This gives a comprehensive understanding of the system's privacy guarantee.

In particular, existing work showed that if each user executes an ϵ_l-LDP protocol, the view of Adv is (ϵ_c, δ)-DP. If the users collude with the server, the server's view is composed of two parts: the shuffled reports as in Adv, and all users' reports except the victim's. By subtracting each user's reports from the shuffled result, the server now knows the victim's LDP report; thus the model falls back to the LDP setting. Finally, if the shuffler colludes with the server, the model also degrade to the LDP setting.

Note that we assume the cryptographic primitives are safe (i.e., the adversaries are computationally bounded and cannot learn any information from the ciphertext) and there are no side channels such as timing information. In some cases, the whole procedure can be interactive, i.e., some part of the observation may depend on what the party sends out. For this, one can utilize composition theorems to prove the DP guarantee. Moreover, the parties are assumed to follow the protocol in the privacy proofs. If the parties deviate from the prescribed procedure, we examine the possible deviations and their influences in the next subsection.

5.3 Robustness to Malicious Parties

There could be multiple reasons for each party to be malicious to (1) interrupt the data collection process, (2) infer more sensitive information from the users, and (3) degrade the utility (estimation accuracy) of the server. In what follows, for each of the reasons, we analyze the consequence and potential mitigation of different parties. Note that the server will not deviate from the protocol as it is the initiator, unless to infer more information of the users.

First, any party can try to interrupt the process; but it is easy to mitigate. If a user blocks the protocol, his/her report can be ignored. If the auxiliary server denies the service, the server can find another auxiliary server and redo the protocol. Note that in this case, users need to remember their report to avoid averaging attacks.

Second, it is possible that the auxiliary server deviates from the protocol (e.g., by not shuffling LDP reports). In this case, the auxiliary server does not have benefits except saving some computational power. Thus we assume the auxiliary server will not deviate in order to infer sensitive information. For the server, as it only sees and evaluates the final reports, there is nothing the server can do to obtain more information from the users.

Third, we note that any party can degrade the utility. Any party other than the server has the incentive to do so. For example, when the server is interested in learning the popularity of websites, different parties can deviate to promote some targeted website. This is also called the data poisoning attack. To do this, the most straightforward way is to generate many fake users, and let them join the data collection process. This kind of Sybil attack is hard to defend against without some kind of authentication, which is orthogonal to the focus of this paper. Another unavoidable attack is that users can change their private values for reporting. We note that any ability beyond these is undesirable. In addition, the protocol should restrict the impact of the auxiliary server on the result. Thus the major concern is that the users or auxiliary servers disrupt utility.

5.4 Discussion and Key Observations

Several observations and lessons are worth noting.

When Auxiliary Server Colludes: No Amplification. When the server colludes with the auxiliary servers, the privacy guarantee falls back to the original LDP model. When using the shuffler model, we need to reduce the possibility of this collusion, e.g., by introducing more auxiliary servers.

When Users Collude: Possibility Missed by Previous Literature. When proving privacy guarantees against the server, existing work assumes the adversary has access to users' sensitive values but not the LDP output. While this is possible, we note that if an adversary already obtains users' sensitive values, it may also have access to the users' LDP reports. Such cases include the users (except the victim) collude with the server; or the server is controlling the users (except the victim). To handle this challenge, we propose to have the auxiliary servers add noise (shown in the next section).

When Parties Deviates: Avoid Utility Disruption. The protocol should be designed so that each individual user or auxiliary server has limited impact on the estimation result.

6. DEFENDING AGAINST ATTACKS

We present a protocol that improves the security guarantee of existing work. The goal is to simultaneously defend against three threats: (1) the server colludes with the users; (2) the server colludes with the auxiliary servers; (3) data poisoning from each party.

6.1 Fake Response from Auxiliary Servers

To defend against the threat when the server colludes with the users, we propose to have the auxiliary servers inject noise. There can be different ways to do this. Our approach utilizes uniform fake reports. That is, the auxiliary servers draw n_r reports uniformly distributed in the range of the LDP protocol and report them. These reports are indistinguishable from the n reports contributed from users. On the server side, after obtaining the estimated frequency $\tilde{f}$ (given by Equation (2) or (3)), the server recovers the frequency for the original dataset by subtracting the expected noise, i.e.,

$$f'_v = \frac{n + n_r}{n} \tilde{f}_v - \frac{n_r}{n} \frac{1}{d} \tag{7}$$

where d is the domain size. Building on top of this, we present efforts to defend against the other two threats, i.e., the server colluding with the auxiliary servers, and data poisoning attack.

6.1.1 First Attempt: Sequential Shuffle

To improve the trust model of the shuffler-based model, one idea is to introduce a sequence of shufflers, so that as long as one shuffler is trusted, the privacy guarantee remains. In this case, the task of inserting n_r fake reports can be divided equally among the r auxiliary servers (shufflers). More specifically, the first shuffler receives the users' LDP reports as input, and draws $n_u = n_r/r$ fake reports. It then shuffles all the reports and sends them to the second shuffler, who draws another n_u fake reports, shuffles all the reports, and sends them to the next shuffler. This procedure proceeds until the last shuffler sends the result to the server. Onion encryption is used during the process; each party decrypts one layer of encryption, and the server obtains $n + n_r$ reports.

However, this approach is vulnerable to poison attacks by the shufflers. That is, the auxiliary servers can replace the users' reports with any report of their choice to change the final result, and the fake reports each shuffler inserts can be chosen arbitrarily.

To mitigate the first threat, we can use an idea of spot-checking. That is, the server can add dummy accounts before the system setup, then it can check whether the reports from its accounts are tampered. For the second threat, we find that it hard to handle. Specifically, a dishonest auxiliary server may draw fake reports from some skewed (instead of uniform) distribution in order to mislead the analyzer and achieve a desired result; and there is no way to self-prove the randomness he/she used is truly random.

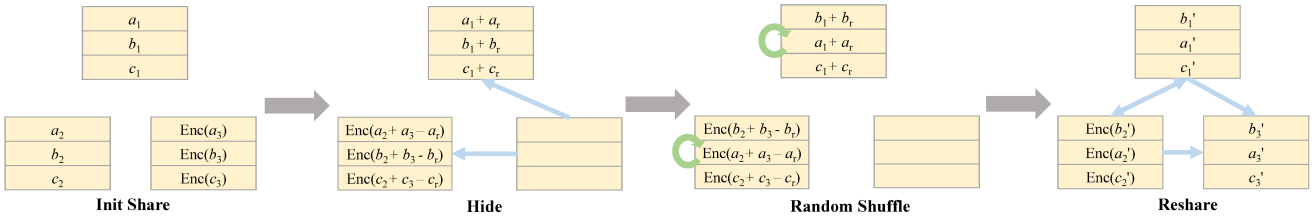

Figure 2: Overview of EOS with $r = 3$ shufflers and $n = 3$ values a, b, c. Each shuffler receives n shares; and one shuffler's shares are encrypted by additive homomorphic encryption. During hiding, one shuffler sends its shares to the other two shufflers, who then shuffle the aggregated shares with an agreed permutation. To reshare, each of the shufflers splits its shares and send them to the other shufflers.

6.1.2 *Second Attempt: Oblivious Shuffle*

To overcome the data poisoning attack, our approach is to construct the fake reports using secret sharing, which ensures that as long as one shuffler is honest, the inserted fake reports are uniformly random. To share an LDP report, we note that for both GRR and SOLH, the domain of the report can be mapped to an ordinal group $\{0, 1, \ldots, x\}$, where each index represents one different LDP report. Thus the LDP reports can be treated as numbers and shared with additive secret sharing.

In order to shuffle shares of secret, we utilize the oblivious shuffle protocol described in Section 2.3. More specifically, the n users each splits his/her LDP reports into r shares among the r shufflers. Each of the shufflers then uniformly draws one share for each of the n_r fake reports. Thus the shufflers each has $n + n_r$ shares. An oblivious shuffle protocol is then executed among the shufflers to shuffle the $n + n_r$ shares of reports. Finally the r shufflers send their shares to the server, who combines the shares to obtain the results. Note that the communication is assumed to be secure.

This solution suffers from a threat that, even without the server, half of the shufflers can collude to recover the user reports. To mitigate this concern, we design a new oblivious shuffle protocol EOS that uses additive homomorphic encryption (AHE).

6.1.3 *Proposal: Private Encrypted Oblivious Shuffle*

To ensure that the shufflers cannot infer the users' reported data, a natural solution is to encrypt the shares using the server's public key. Moreover, the encryption needs to be additively homomorphic in order to be compatible with the secret-sharing operations. In what follows, we present a new protocol Encrypted Oblivious Shuffle (EOS) that utilizes additive homomorphic encryption (AHE) in oblivious shuffle. We then present our proposal Private Encrypted Oblivious Shuffle (PEOS) that uses EOS for DP.

Encrypted Oblivious Shuffle. Encrypted Oblivious Shuffle (EOS) works similarly to oblivious shuffle. One difference is that in each round, one shuffler will possess the encrypted shares. The encrypted shares can be shuffled and randomized just like normal shares except that they are then processed under AHE.

Denote the shuffler who possess encrypted shares as E. In each round, E splits its encrypted vector of shares into t new vectors so that $t - 1$ of them are plaintexts, and the remaining one is in the ciphertext form (this can be done because of AHE). The t shares are randomly sent to the t hiders. Only one hider receives the ciphertext share and becomes the next E. After the group shuffling, the new E splits its vector of shares and sends them to r parties. An example of EOS with $r = 3$ is demonstrated in Figure 2. EOS strengthens oblivious shuffle in that even if the t shufflers collude, they cannot figure out the users' original reports, because one share is encrypted under the server's public key.

Note that the AHE scheme should support a plaintext space of $\mathbb{Z}_{2^\ell}$ where ℓ is normally 32 or 64 in our case. This is because the fake reports are sampled by shufflers as random ℓ-bit shares. Note that the range of the LDP report, denoted by z, is smaller than 2^ℓ; thus each user should add a random number from $\{z, \ldots, 2^\ell - 1\}$ to his/her report, otherwise the fakeness will be detected by the server. Such an AHE scheme can be instantiated to be the full-decryption variant of DGK [22] using Pohlig-Hellman algorithm [46].

COROLLARY 9. *Encrypted oblivious shuffle, instantiated with additive homomorphic encryption of plaintext space $\mathbb{Z}_{2^\ell}$, is a secure oblivious shuffle protocol in the semi-honest model.*

Proof Sketch: The difference of EOS from oblivious shuffle is that AHE is used for one hider's computation in each round. As long as AHE does not leak additional information, similar proof about the final shuffling order can be derived from oblivious shuffle [39].

For AHE, note that although we use AHE for one hider's computation in each round, the computation is translated into modulo 2^ℓ in the plaintext space, which is exactly the same as normal secret sharing computation. Therefore, AHE does not leak additional information as long as the security assumption of the AHE holds (hardness of integer factorization in the case of DGK). □

Using EOS for Differential Privacy. Algorithm 1 describes the full description of this protocol. There are three kinds of parties, users, shufflers, and the server. They all agree to use some method FO with the same parameter (e.g., ϵ, domain $\mathscr{D}$, etc); the FO can be either GRR or SOLH, depending on the utility, as described in Section 4.3. The users split their LDP reports into r shares, encrypt only the r-th shares using AHE, and send them to the shufflers. Each shuffler generate n_r shares for fake reports; only the r-th shuffler encrypts the shares with AHE. In this case, a malicious shuffler can draw its shares from a biased distribution; but those shares will then be "masked" by other honest shufflers' random shares and become uniformly random. By Corollary 9, the users' reports are protected from the shufflers; and the server cannot learn the permutation unless he can corrupt more than half of the auxiliary servers.

6.2 Privacy Analysis

In the EOS protocol, the server knows all the fake reports and each user's LDP report if it can corrupt more than $\lfloor r/2 \rfloor$ of the shufflers. And in this case, each user's privacy is only protected by ϵ_l-DP. On the other hand, as long as the server corrupt no more than $\lfloor r/2 \rfloor$ shufflers, the server cannot gain useful information.

In what follows, we assume the server cannot corrupt more than $\lfloor r/2 \rfloor$ shufflers and examine the privacy guarantee of PEOS. The focus is on how the privacy guarantees change after the addition of n_r fake reports. With these injected reports, what the server can observe is the reports from both users and the shufflers. If the users

Algorithm 1 PEOS

User i: Value v_i, server's public key pk
1: $Y_i = \mathsf{FO}(v_i)$ ▷ FO can be GRR or SOLH
2: Split Y_i into r shares $\langle Y_{i,j}\rangle_{j\in[r]}$
3: **for** $j \in [r-1]$ **do**
4: Send $Y_{i,j}$ to auxiliary server j
5: Send $c_{i,r} \leftarrow \mathrm{Enc}_{pk}(Y_{i,r})$ to auxiliary server r

Shuffler $j \in [r-1]$: Shares $\langle Y_{i,j}\rangle_{i\in[n]}$
1: **for** $k \in [n_r]$ **do** ▷ Generate shares of fake reports
2: Sample $Y'_{k,j}$ uniformly from output space of FO
3: Participate in EOS with $\langle Y_{i,j}\rangle_{i\in[n]}$ and $\langle Y'_{k,j}\rangle_{k\in[n_r]}$ and send the shuffled result to the server

Shuffler r: Encrypted shares $\langle c_{i,r}\rangle_{i\in[n]}$
1: **for** $k \in [n_r]$ **do** ▷ Encrypted shares of fake reports
2: Sample $Y'_{k,r}$ uniformly from output space of FO
3: $c'_{k,r} \leftarrow \mathrm{Enc}_{pk}(Y'_{k,r})$
4: Participate in EOS with $\langle c_{i,r}\rangle_{i\in[n]}$ and $\langle c'_{k,r}\rangle_{k\in[n_r]}$ and send the shuffled result to the server

Server: Shares from auxiliary servers
1: Decrypt and aggregate the shares to recover Y
2: For any $v \in \mathscr{D}$, estimate f'_v using Y and Equation (7)

collude, the server can subtract all other users' contribution and the privacy comes from the fake reports. The following corollaries give the precise privacy guarantee:

COROLLARY 10. *If* SOLH *is used and* SOLH *is ϵ_l-LDP, then* PEOS *is ϵ_c-DP against the server; and if other users collude with the server, the protocol is ϵ_s-DP, where*

$$\epsilon_s = \sqrt{14\ln(2/\delta)\cdot\frac{d'}{n_r}}, \epsilon_c = \sqrt{14\ln(2/\delta)/\left(\frac{n-1}{e^{\epsilon_l}+d'-1}+\frac{n_r}{d'}\right)} \tag{8}$$

PROOF. The proof is similar to the setting of with SOLH, but with n_r more random reports. More specifically, when other users collude, privacy is provided by the n_r reports that follow uniform distribution over $[d']$. Plugging the argument into Equation (5), these can be viewed as a random variable that follows Binomial distribution with $\mathsf{Bin}\left(n_r, \frac{1}{d'}\right)$. The rest of the proof follows from that for Theorem 3.

Similarly, for the privacy guarantee against the server, there are $n-1$ random reports from users, and n_r reports from the auxiliary server. The effect of both can be viewed as one Binomial random variable: $\mathsf{Bin}\left(n-1, 1/(e^{\epsilon_l}+d'-1)\right) + \mathsf{Bin}\left(n_r, 1/d'\right) = \mathsf{Bin}\left(n-1+n_r, \frac{(n-1)/(e^{\epsilon_l}+d'-1)+n_r/d'}{n-1+n_r}\right)$. □

One can also use GRR in PEOS, and we have a similar theorem:

COROLLARY 11. *If* GRR *is used and* GRR *is ϵ_l-LDP, then* PEOS *is ϵ_c-DP against the server; and if other users collude with the server, the protocol is ϵ_s-DP, where*

$$\epsilon_s = \sqrt{14\ln(2/\delta)\cdot\frac{d}{n_r}}, \epsilon_c = \sqrt{14\ln(2/\delta)/\left(\frac{n-1}{e^{\epsilon_l}+d-1}+\frac{n_r}{d}\right)}$$

The proof is similar to that for Corollary 10 and is thus omitted.

6.3 Utility Analysis

In Section 4.3, we analyze the accuracy performance of different methods under the basic shuffling setting. In this section, we further analyze the utility of these methods in PEOS. The difference mainly comes from the fact that n_r dummy reports are inserted, and the server runs a further step (i.e., Equation (7)) to post-process the results. In what follows, we first show that Equation (7) gives an unbiased estimation; based on that, we then provide a general form of estimation accuracy.

We first show f'_v is an unbiased estimation of f_v, where $f_v = \frac{1}{n}\sum_{i\in[n]}\mathbb{1}_{\{v_i=v\}}$ is the true frequency of value v.

LEMMA 12. *The server's estimation f'_v from Equation (7) is an unbiased estimation of f_v, i.e., $\mathbb{E}\left[\tilde{f}_v\right] = f_v$.*

PROOF.

$$\mathbb{E}\left[f'_v\right] = \mathbb{E}\left[\frac{n+n_r}{n}\tilde{f}_v - \frac{n_r}{n}\frac{1}{d}\right] = \frac{n+n_r}{n}\mathbb{E}\left[\tilde{f}_v\right] - \frac{n_r}{n}\frac{1}{d} \tag{9}$$

Here $\tilde{f}_v$ is the estimated frequency of value v given the $n+n_r$ reports; among them, n of them are from the true users, and n_r are from the randomly sampled values. For the n reports from users, nf_v of them have original value v; and for the n_r reports, in expectation, n_r/d of them have original value v. Thus we have

$$\mathbb{E}\left[\tilde{f}_v\right] = \frac{nf_v + n_r/d}{n+n_r}$$

Putting it back to Equation (9), we have $\mathbb{E}\left[\tilde{f}_v\right] = f_v$. □

Given that, we prove the expected squared error of f'_v:

$$\mathsf{Var}\left[f'_v\right] = \mathsf{Var}\left[\frac{n+n_r}{n}\tilde{f}_v - \frac{n_r}{n}\frac{1}{d}\right] = \frac{(n+n_r)^2}{n^2}\mathsf{Var}\left[\tilde{f}_v\right]$$

Now plugging in the results of $\mathsf{Var}\left[\tilde{f}_v\right]$ from Section 4.3 (note that we replace n with $n+n_r$ in the denominator as there are $n+n_r$ total reports), we obtain the specific variance of different methods after inserting n_r dummy reports.

One thing to note is that as the utility expression changes, the optimal parameter d' in SOLH becomes different as well. In particular, Corollary 10 gives both ϵ_s and ϵ_c. Given n_r and δ, d' is determined given ϵ_s; but d' can change given ϵ_c. In particular, we can also derive the optimal value of d' following the similar to the analysis of Section 4.3 (after Proposition 8):

Given $\epsilon_c = \sqrt{14\ln(2/\delta)/\left(\frac{n-1}{e^{\epsilon_l}+d'-1}+\frac{n_r}{d'}\right)}$, we have

$$e^{\epsilon_l} + d' - 1 = \frac{n-1}{14\ln(2/\delta)/\epsilon_c^2 - n_r/d'}$$

We denote it as m, and (to simplify the notations) use a to represent $14\ln(2/\delta)/\epsilon_c^2$ and b to represent $n-1$. By the variance derived above, we have $\mathsf{Var} = \frac{m^2}{(m-d)^2(d-1)}\frac{n+n_r}{n^2}$. Note that this formula is similar to the previous one in Section 4.3; but here m also depends on d'. Thus we need to further simplify Var:

$$\mathsf{Var} = \frac{(n+n_r)\left(\frac{b}{a-n_r/d'}\right)^2}{n^2\left(\frac{b}{a-n_r/d'}-d\right)^2(d'-1)} = \frac{(n+n_r)b^2}{n^2\left(b-(a-n_r/d)d'\right)^2(d'-1)}$$

$$= \frac{(n+n_r)b^2}{n^2a^2\left(d' - (b+n_r)/a\right)^2(d'-1)}$$

To minimize Var, we want to maximize $\left(d' - (b+n_r)/a\right)^2(d'-1)$. By making its partial derivative to 0, we can obtain that when

$$d' = \frac{(b+n_r)/a+2}{3} = \frac{\epsilon_c^2(n-1-n_r)}{42\ln(2/\delta)} + \frac{2}{3}$$

the variance is minimized. Comparing to Equation (6), introducing n_r will reduce the optimal d'. We use the integer component of d' in the actual implementation.

6.4 Discussion and Guideline

PEOS strengthens the security aspect of the shuffler model from three perspectives: First, it provides better privacy guarantee when users collude with the server, which is a common assumption made in DP. Second, it makes it more difficult for the server to collude with the shufflers. Third, it limits the ability of data poisoning of the shufflers. We discuss criteria for initiating PEOS.

Choosing Parameters. Given the desired privacy level $\epsilon_1, \epsilon_2, \epsilon_3$ against the three adversaries Adv, Adv_u, Adv_a, respectively. Also given the domain size d, number of users n, and δ, we want to configure PEOS so that it provides $\epsilon_c \le \epsilon_1$, $\epsilon_s \le \epsilon_2$, and $\epsilon_l \le \epsilon_3$.

Local perturbation is necessary to satisfy ϵ_3-DP against Adv_a. To achieve ϵ_2 when other users collude, noise from auxiliary servers are also necessary. Given that, to satisfy $\epsilon_c \le \epsilon_1$, if we have to add more noise, we have two choices. That is, the natural way is to add noisy reports from the auxiliary server, but we can also lower ϵ_l at the same time. As we have the privacy and utility expressions, we can numerically search the optimal configuration of n_r and ϵ_l. Finally, given ϵ_l, we can choose to use either GRR or SOLH by comparing Theorem 3 and Proposition 6.

7. EVALUATION

7.1 Experimental Setup

Datasets. We run experiments on three real datasets.

- IPUMS [49]: The US Census data for the year 1940. We sample 1% of users, and use the city attribute (N/A are discarded). This results in $n = 602325$ users and $d = 915$ cities.
- Kosarak [2]: A dataset of 1 million click streams on a Hungarian website that contains around one million users with 42178 possible values. For each stream, one item is randomly chosen.
- AOL [3]: The AOL dataset contains user queries on AOL website during the first three months in 2006. We assume each user reports one query (w.l.o.g., the first query), and limit them to be 6-byte long. This results a dataset of around 0.5 million queries including 0.12 million unique ones. It is used in the succinct histogram case study in Section 7.3.

Competitors. We compare the following methods:

- OLH: The local hashing method with the optimal d' in LDP [52].
- Had: The Hadamard transform method used in [5]. It can be seen as OLH with $d' = 2$ (utility is worse than OLH); but compared to OLH, its server-side evaluation is faster by a constant factor.
- SH: The shuffler-based method for histogram estimation [9].
- AUE: Method from [8]. It first transforms each user's value using one-hot encoding. Then the values (0 or 1) in each location is incremented w/p $p = 1 - \frac{200}{\epsilon_c^2 n}\ln(4/\delta)$. Note that it is not an LDP protocol, and its communication cost is $O(d)$.
- RAP: The unary-encoding-based idea described in Section 4.1. Its local side method is equivalent to RAPPOR [31]. Similar to AUE, it has large communication cost.
- RAP_R: Method from [29]. Similar to AUE and RAP, it transforms each user's value using one-hot encoding. The method works in the removal setting of DP. When converting to the replacement definition, it has the same utility as RAP.
- SOLH: The hashing-based idea introduced in Section 4.2.
- PEOS: We focus on the perspective of the computation and communication complexity in Section 7.4.
- SS: As a baseline, we also evaluate the complexity of the sequential shuffling method presented in 6.1.1; we call it SS.

Implementation. The prototype was implemented using Python 3.6 with fastecdsa 1.7.4, pycrypto 2.6.1, python-xxhash 1.3.0 and numpy 1.15.3 libraries. For SS, we generate a random AES key to encrypted the message using AES-128-CBC, and use the ElGamal encryption with elliptic curve secp256r1 to encrypt the AES key. For the AHE in PEOS, we use DGK [21] with 3072-bits ciphertext. All of the encryption used satisfy 128-bit security.

Metrics. We use mean squared error (MSE) of the estimates as metrics. For each value v, we compute its estimated frequency $\tilde{f}_v$ and the ground truth f_v, and calculate their squared difference. Specifically, $\text{MSE} = \frac{1}{|\mathscr{D}|}\sum_{v\in\mathscr{D}}(f_v - \tilde{f}_v)^2$.

Methodology. For each dataset and each method, we repeat the experiment 100 times, with result mean and standard deviation reported. The standard deviation is typically very small, and barely noticeable in the figures. By default, we set $\delta = 10^{-9}$.

7.2 Frequency Estimation Comparison

We first show the utility performance of SOLH. We mainly compare it against other methods in the shuffler model, including SH, AUE, RAP, and RAP_R. We also evaluate several kinds of baselines, including LDP methods OLH and Had, centralized DP method Laplace mechanism (Lap) that represents the lower bound, and a method Base that always outputs a uniform distribution.

Figure 3 shows the utility comparison of the methods. We vary the overall privacy guarantee ϵ_c against the server from 0.1 to 1, and plot MSE. First of all, there is no privacy amplification for SH when ϵ_c is below a threshold. In particular, when $\epsilon_c < \sqrt{\frac{14\ln(2/\delta)d}{n-1}}$, $\epsilon_l = \epsilon_c$. We only show results on the IPUMS dataset because for the Kosarak dataset, d is too large so that SH cannot benefit from amplification. When there is no amplification, the utility of SH is poor, even worse than the random guess baseline method. Compared to SH, our improved SOLH method can always enjoy the privacy amplification advantage, and gets better utility result, especially when ϵ_c is small. The three unary-encoding-based methods AUE, RAP, and RAP_R all perform slightly better than SOLH. But the communication cost of them are higher. The best-performing method is RAP_R; but it works in the removal-LDP setting. Because of this, its performance with ϵ_c is equivalent to RAP with $2\epsilon_c$.

Moving to the LDP methods, OLH and Had perform very similarly (because in these settings, OLH mostly chooses $d' = 2$ or 3, which makes it almost the same as Had), and are around 3 orders of magnitude worse than the shuffler-based methods. For the central DP methods, we observe Lap outperforms the shuffler-based methods by around 2 orders of magnitude.

In Table 2, we list the value of d' of SOLH and the utility of SOLH and RAP_R for some ϵ_c values. We also fix d' in SOLH and show how sub-optimal choice of d' makes SOLH less accurate. The original domain d is more than 40 thousand, thus RAP_R introduces a larger communication cost compared to SOLH (5KB vs 8B). The computation cost for the users is low for both methods; but for the server, estimating frequency with SOLH requires evaluating hash functions. We note that as this takes place on server, some compu-

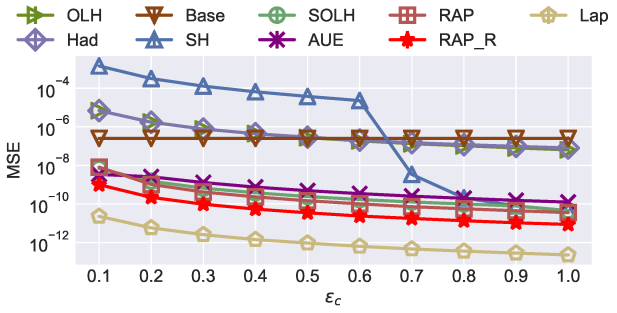

Figure 3: Results of MSE varying ϵ_c on the IPUMS dataset. Base always outputs $1/d$ for each estimation. Lap stands for Laplace mechanism for DP.

Table 2: Comparison of SOLH and RAP_R in Kosarak.

Metric	Method \ ϵ_c	0.2	0.4	0.6	0.8
d'	SOLH	45	177	397	705
Utility	SOLH	5.27e-8	1.30e-8	5.76e-9	3.24e-9
	RAP_R ($d' = 10$)	1.31e-7	1.17e-7	1.14e-7	1.13e-7
	RAP_R ($d' = 100$)	1.73e-7	1.55e-8	1.22e-8	1.22e-8
	RAP_R ($d' = 1000$)	1.02e-4	2.60e-5	4.02e-8	3.66e-9
	RAP_R	7.82e-9	1.92e-9	8.53e-10	4.78e-10

tational cost is tolerable, especially the hashing evaluation nowadays is efficient. For example, our machine can evaluate the hash function 10 million times within 1 second on a single thread.

7.3 Succinct Histograms

In this section, we apply shuffle model to the problem of succinct histogram as a case study. The succinct histogram problem still outputs the frequency estimation; but different from the ordinary frequency or histogram estimation problem, on which we focused in the last section, it handles the additional challenge of a much larger domain (e.g., domain size greater than 2^{32}). The problem is extensively investigated. We use the tree-style approach [11, 50, 55]. We first present TreeHist from [11]. It assumes the domain to be composed of fixed-length bitstrings and constructs a binary prefix tree. The root of the tree denotes the empty string. Each node has two children that append the parent string by 0 and 1. For example, the children of root are two prefixes $0*$ and $1*$, and the grand children of root are $00*$, $01*$, $10*$, and $11*$. The leaf nodes represent all possible strings in the domain. Parallel to TreeHist, PEM [55] advocates using a the fan-out number as large as possible.

To find the frequent strings, we traverse the tree in a breadth-first-search style: We starts from the root and checks whether the prefixes at its children are frequent enough. If a prefix is frequent, its children will be checked in the next round. For each round of checking, an LDP mechanism is used. Note that the mechanism can group all nodes in the same layer into a new domain (smaller than the original domain because many nodes will be infrequent and ignored). Each user will check which prefix matches the private value, and report it (or a dummy value if there is no match). To demonstrate the utility gain of the shuffler model, we use the methods SH, SOLH, AUE, and RAP as the frequency estimator.

We consider the AOL dataset assuming each user's value is 48 bits. We run PEM in 6 rounds, each for 8 bits (fan-out 256). We set the goal to identify the the top 32 strings, and in each intermediate round, we identify the top 32 prefixes. In the LDP setting, PEM divides the users into 6 groups, as that gives better results. In the shuffler case, a better approach is to avoid grouping users, but rather

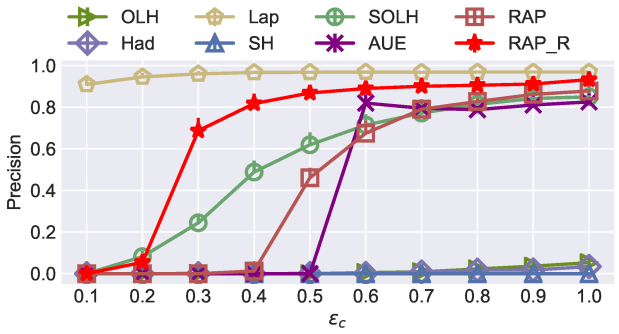

Figure 4: Comparison on the succinct histogram problem. The target is to identify the top 32 most frequent values.

dividing ϵ_c and δ_c by 6 for each round.

Figure 4 shows the results. We can observe that the except SH, the other shuffler-based methods outperforms the LDP PEM (OLH and Had) . In addition to the capability of reducing communication cost, another advantage of SOLH we observe here is that SOLH enables non-interactive execution of PEM (note that this is also one reason why the original PEM algorithm uses the local hashing idea). In particular, the users can encode all their prefixes and report together. The server, after obtaining some frequent prefix, can directly test the potential strings in the next round. On the other hand, using the unary-encoding-based methods, users cannot directly upload all their prefixes, because the size of a report can be up to 2^{48} bits. Instead, the server has to indicate which prefixes are frequent to the users and then request the users to upload.

7.4 Performance Evaluation

We evaluate the computational and communication costs of SS and PEOS, focusing on the overhead introduced by the encryption and shuffling. We run the experiments on servers running Linux kernel version 5.0 with Intel Xeon Silver 4108 CPU @ 1.80GHz and 128GB memory. We assume there are $r = 3$ and $r = 7$ shufflers. The results are listed in Table 3. As both methods scales with $n + n_r$, we fix n to be 1 million and ignore n_r.

Note that we the results are only for SOLH with report size fixed at 64 bits. If we use RAP in this case, the communication cost will increase proportional to the size of the domain d (by $d/64$).

Table 3: Computation and communication overhead of SS and PEOS for each user, each shuffler, and the server. We assume $n = 10^6$ and $r = 3$ or 7.

Metric \ Method	SS		PEOS	
	$r = 3$	$r = 7$	$r = 3$	$r = 7$
User comp. (ms)	0.24	0.49	1.6	1.6
User comm. (Byte)	416	800	400	432
Aux. comp. (s)	49	50	0.2	0.7
Aux. comm. (MB)	224	416	429.8	3293.3
Server comp. (s)	49	49	65	65
Server comm. (MB)	128	128	392	408

User Overhead. Overall, the user-side computation and communication overhead are small for both methods. The computation only involves sampling, secret-sharing, and r times of encryption operations. All of them are fast. Note that in SS, as onion encryption is used, its overhead is larger and grows linearly with respect to r. The communication cost for each user is also very limited.

Shuffler Overhead. For each shuffler in SS, the computation cost

lies in n decryption (for one layer), sampling n_u random reports (with necessary encryption), and then shuffling. Note that the decryption is done in parallel. We use 32 threads for demonstration. With more resources, the processing time can be shortened.

In SS, an ElGamal ciphertext is a tuple $\langle P, C\rangle$, where P is a point in the secp256r1 curve represented by 256×2 bits, and C is a number in $\{0,1\}^{256}$. Thus, we need 96 bytes for the AES key in each layer. For SOLH, we let each user randomly select an 4-byte seed as the random hash function. After padding, each message is $32 + 96(r+1)$ bytes, where r is the number of layers used for shufflers. One additional layer is used for the server. Given $n = 1$ million users and r shufflers, there will be on average $\frac{1}{r} \times n \times \sum_{k=1}^{r}(32+96(k+1)) = 672$ MB data sent to the three shufflers.

PEOS consists of $\binom{r}{\lfloor r/2 \rfloor + 1}$ rounds of sorting. Since a well-implemented sorting on 1 million elements takes only several milliseconds, the computation cost of shuffling is minor for the shufflers. In addition, our protocol require each shuffler do $\binom{r}{\lfloor r/2 \rfloor + 1} \cdot n/r$ homomorphic additions during shuffling. As Table 3 indicates, all of these cryptographic operations are efficient. The cost is no more than one second with $n = 1$ million reports.

According to the analysis of oblivious shuffle from [39], each shuffler's communication cost is $O(2^r\sqrt{rn})$. In addition, our protocol sends n encrypted shares each round, which introduces another communication cost of $O(2^r n/\sqrt{r})$ by similar analysis (multiplied with a larger constant factor because of the 3072-bit DGK ciphertexts). In experiments with 1 million users and 3 shufflers, each shuffler needs to send 430 MB. In a more expensive case with 7 shufflers, it becomes 3.3 GB. While the communication cost is higher than that of SS, we note that the cost is tolerable in our setting, as the data collection does not happen frequently.

Server Overhead. For SS, the server computation overhead is similar to that of the shufflers, as they all decrypt one layer. The server's communication cost (measured by amount of data received) is lower though, as there is only one layer of encryption on the data.

In PEOS, the server needs to collect data from all r shufflers. The communication overhead is mostly DGK ciphertext and grows slowly with r. The computation overhead is also dominated by decrypting the DGK ciphertexts.

8. RELATED WORK

Privacy Amplification by Shuffling. The shuffling idea was originally proposed in Prochlo [15]. Later the formal proof was given in [30, 19, 9]. Parallel to our work, [8, 33] propose mechanisms to improve utility in this model. They both rely on the privacy blanket idea [9]. More recently, [29] considered an intriguing removal-based LDP definition and work in the shuffler model. Besides estimating histograms, the problem of estimating the sum of numerical values are also extensively investigated [34, 10].

Crypto-aided Differential Privacy. Different from using shufflers, researchers also proposed methods that utilize cryptography to provide differential privacy guarantees, including [32, 28, 42]. One notable highlight is [20], which proposes Cryptϵ. In this approach, users encrypt their values using homomorphic encryption, and send them to the auxiliary party via a secure channel. The auxiliary server tallies the ciphertext and adds random noise in a way that satisfies centralized DP, and sends the result to the server. The server decrypts the aggregated ciphertext. More recently, researchers in [48] introduce several security features including verification and malice detection. This line of work does not require LDP protection, thus differs from our approach. Moreover, to handle the histogram estimation when $|\mathscr{D}|$ is larger, the communication overhead is larger than that of ours.

Relaxed Definitions. Rather than introducing the shuffler, another direction to boost the utility of LDP is to relax its *semantic meaning*. In particular, Wang et al. propose to relax the definition by taking into account the distance between the true value and the perturbed value [51]. More formally, given the true value, with high probability, it will be perturbed to a nearby value (with some pre-defined distance function); and with low probability, it will be changed to a value that is far apart. A similar definition is proposed in [37, 35]. Both usages are similar to the geo-indistinguishability notion in the centralized setting [7]. In [44], the authors consider the setting where some answers are sensitive while some not (there is also a DP counterpart called One-sided DP [24]). The work [36] is a more general definition that allows different values to have different privcay level. Our work applied to the standard LDP definition, and we conjecture that these definitions can also benefit from introducing a shuffler without much effort.

There also exist relaxed models that seem incompatible with the shuffler model, i.e., [13] considers the inferring probability as the adversary's power; and [53] utilizes the linkage between each user's sensitive and public attributes.

Distributed DP. In the distributed setting of DP, each data owner (or proxy) has access to a (disjoint) subset of users. For example, each patient's information is possessed by a hospital. The DP noise is added at the level of the intermediate data owners (e.g., [41]). A special case (two-party computation) is also considered [38, 47]. [40] studies the limitation of two-party DP. In [27], a distributed noise generation protocol was proposed to prevent some party from adding malicious noise. The protocol is then improved by [17]. [43] lays the theoretical foundation of the relationship among several kinds of computational DP definitions.

We consider a different setting where the data are held by each individual users, and there are two parties that collaboratively compute some aggregation information about the users.

DP by Trusted Hardware. In this approach, a trusted hardware (e.g., SGX) is utilized to collect data, tally the data, and add the noise within the protected hardware. The result is then sent to the analyst. Google propose Prochlo [15] that uses SGX. Note that the trusted hardware can be run by the server. Thus [18] and [6] designed oblivious DP algorithms to overcome the threat of side information (memory access pattern may be related to the underlying data). These proposals assume the trusted hardware is safe to use. However, using trusted hardware has potential risks (e.g., [14]). This paper considers the setting without trusted hardware.

9. CONCLUSIONS

In this paper, we study the shuffler model of differential privacy from two perspectives. First, we examine from the algorithmic aspect, and make improvement to existing techniques. Second, we work from the security aspect of the model, and emphasize two types of attack, collusion attack and data-poisoning attack; we then propose PEOS that is safer under these attacks. Finally, we perform experiments to compare different methods and demonstrate the advantage of our proposed method. For the problem of histogram estimation, our proposed protocol is both more accurate and more secure than existing work, with a reasonable communication/computation overhead. We also demonstrate the applicability of our results in the succinct histogram problem.

Acknowledgement. We sincerely thank the reviewers for their insightful comments. This work is supported by NSF grant 1640374 and 1931443. Tianhao's work was partly done at Alibaba.

10. REFERENCES

[1] Apple differential privacy team, learning with privacy at scale. Available at https://machinelearning.apple.com/docs/learning-with-privacy-at-scale/appledifferentialprivacysystem.pdf.

[2] Frequent itemset mining dataset repository. Available at http://fimi.ua.ac.be/data/.

[3] Web search query log downloads. Available at http://www.radiounderground.net/aol-data/.

[4] J. M. Abowd. Protecting the confidentiality of america's statistics: Adopting modern disclosure avoidance methods at the census bureau. https://www.census.gov/newsroom/blogs/research-matters/2018/08/protecting_the_confi.html, 2018.

[5] J. Acharya, Z. Sun, and H. Zhang. Hadamard response: Estimating distributions privately, efficiently, and with little communication. In *AISTATS*, 2019.

[6] J. Allen, B. Ding, J. Kulkarni, H. Nori, O. Ohrimenko, and S. Yekhanin. An algorithmic framework for differentially private data analysis on trusted processors. In *Advances in Neural Information Processing Systems*, pages 13635–13646, 2019.

[7] M. Andrés, N. Bordenabe, K. Chatzikokolakis, and C. Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. In *20th ACM Conference on Computer and Communications Security*, pages 901–914. ACM, 2013.

[8] V. Balcer and A. Cheu. Separating local & shuffled differential privacy via histograms. *arXiv preprint arXiv:1909.06879*, 2019.

[9] B. Balle, J. Bell, A. Gascon, and K. Nissim. The privacy blanket of the shuffle model. In *CRYPTO*, 2019.

[10] B. Balle, J. Bell, A. Gascon, and K. Nissim. Private summation in the multi-message shuffle model. In *CCS*, 2020.

[11] R. Bassily, K. Nissim, U. Stemmer, and A. G. Thakurta. Practical locally private heavy hitters. In *NIPS*, 2017.

[12] R. Bassily and A. Smith. Local, private, efficient protocols for succinct histograms. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 127–135. ACM, 2015.

[13] A. Bhowmick, J. Duchi, J. Freudiger, G. Kapoor, and R. Rogers. Protection against reconstruction and its applications in private federated learning. *arXiv preprint arXiv:1812.00984*, 2018.

[14] A. Biondo, M. Conti, L. Davi, T. Frassetto, and A.-R. Sadeghi. The guard's dilemma: Efficient code-reuse attacks against intel sgx. In *27th USENIX Security Symposium*, 2018.

[15] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *SOSP*. ACM, 2017.

[16] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.

[17] J. Champion, J. Ullman, et al. Securely sampling biased coins with applications to differential privacy. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–614. ACM, 2019.

[18] T. H. Chan, K.-M. Chung, B. M. Maggs, and E. Shi. Foundations of differentially oblivious algorithms. In *SODA*. SIAM, 2019.

[19] A. Cheu, A. D. Smith, J. Ullman, D. Zeber, and M. Zhilyaev. Distributed differential privacy via shuffling. In *EUROCRYPT*, 2019.

[20] A. R. Chowdhury, C. Wang, X. He, A. Machanavajjhala, and S. Jha. Cryptε: Crypto-assisted differential privacy on untrusted servers. *SIGMOD*, 2020.

[21] I. Damgård, M. Geisler, and M. Krøigaard. Efficient and secure comparison for on-line auctions. In *Australasian Conference on Information Security and Privacy*, pages 416–430. Springer, 2007.

[22] I. Damgard, M. Geisler, and M. Kroigard. Homomorphic encryption and secure comparison. *Int. J. Appl. Cryptol.*, 1(1):22–31, Feb. 2008.

[23] B. Ding, J. Kulkarni, and S. Yekhanin. Collecting telemetry data privately. In *Advances in Neural Information Processing Systems*, pages 3574–3583, 2017.

[24] S. Doudalis, I. Kotsogiannis, S. Haney, A. Machanavajjhala, and S. Mehrotra. One-sided differential privacy. *arXiv preprint arXiv:1712.05888*, 2017.

[25] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Local privacy and statistical minimax rates. In *FOCS*, pages 429–438, 2013.

[26] C. Dwork. Differential privacy. In *ICALP*, pages 1–12, 2006.

[27] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503. Springer, 2006.

[28] T. Elahi, G. Danezis, and I. Goldberg. Privex: Private collection of traffic statistics for anonymous communication networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1068–1079, 2014.

[29] Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, S. Song, K. Talwar, and A. Thakurta. Encode, shuffle, analyze privacy revisited: Formalizations and empirical evaluation. *arXiv preprint arXiv:2001.03618*, 2020.

[30] Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, K. Talwar, and A. Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *SODA*, pages 2468–2479, 2019.

[31] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*, pages 1054–1067. ACM, 2014.

[32] D. Froelicher, P. Egger, J. S. Sousa, J. L. Raisaro, Z. Huang, C. Mouchet, B. Ford, and J.-P. Hubaux. Unlynx: a decentralized system for privacy-conscious data sharing. *Proceedings on Privacy Enhancing Technologies*, 2017(4):232–250, 2017.

[33] B. Ghazi, N. Golowich, R. Kumar, R. Pagh, and A. Velingker. On the power of multiple anonymous messages. *arXiv preprint arXiv:1908.11358*, 2019.

[34] B. Ghazi, P. Manurangsi, R. Pagh, and A. Velingker. Private aggregation from fewer anonymous messages. In *EUROCRYPT*, 2020.

[35] X. Gu, M. Li, Y. Cao, and L. Xiong. Supporting both range queries and frequency estimation with local differential privacy. In *2019 IEEE Conference on Communications and*

Network Security (CNS), pages 124–132. IEEE, 2019.

[36] X. Gu, M. Li, L. Xiong, and Y. Cao. Providing input-discriminative protection for local differential privacy. In *ICDE*, 2020.

[37] M. E. Gursoy, A. Tamersoy, S. Truex, W. Wei, and L. Liu. Secure and utility-aware data collection with condensed local differential privacy. *arXiv preprint arXiv:1905.06361*, 2019.

[38] X. He, A. Machanavajjhala, C. Flynn, and D. Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1389–1406. ACM, 2017.

[39] S. Laur, J. Willemson, and B. Zhang. Round-efficient oblivious database manipulation. In *International Conference on Information Security*, pages 262–277. Springer, 2011.

[40] A. McGregor, I. Mironov, T. Pitassi, O. Reingold, K. Talwar, and S. Vadhan. The limits of two-party differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 81–90. IEEE, 2010.

[41] B. McMahan and D. Ramage. Federated learning: Collaborative machine learning without centralized training data. *Google Research Blog*, 3, 2017.

[42] L. Melis, G. Danezis, and E. De Cristofaro. Efficient private statistics with succinct sketches. *arXiv preprint arXiv:1508.06110*, 2015.

[43] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan. Computational differential privacy. In *Annual International Cryptology Conference*, pages 126–142. Springer, 2009.

[44] T. Murakami and Y. Kawamoto. Utility-optimized local differential privacy mechanisms for distribution estimation. In *28th USENIX Security Symposium*, 2019.

[45] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[46] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms overgf(p)and its cryptographic significance (corresp.). *IEEE Transactions on Information Theory*, 1978.

[47] F.-Y. Rao, J. Cao, E. Bertino, and M. Kantarcioglu. Hybrid private record linkage: Separating differentially private synopses from matching records. *ACM Transactions on Privacy and Security (TOPS)*, 22(3):15, 2019.

[48] E. Roth, D. Noble, B. H. Falk, and A. Haeberlen. Honeycrisp: large-scale differentially private aggregation without a trusted core. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 196–210. ACM, 2019.

[49] S. Ruggles, S. Flood, R. Goeken, J. Grover, E. Meyer, J. Pacas, and M. Sobek. Integrated public use microdata series: Version 9.0 [database], 2019.

[50] N. Wang, X. Xiao, Y. Yang, T. D. Hoang, H. Shin, J. Shin, and G. Yu. Privtrie: Effective frequent term discovery under local differential privacy. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 821–832. IEEE, 2018.

[51] S. Wang, Y. Nie, P. Wang, H. Xu, W. Yang, and L. Huang. Local private ordinal data distribution estimation. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2017.

[52] T. Wang, J. Blocki, N. Li, and S. Jha. Locally differentially private protocols for frequency estimation. In *26th USENIX Security Symposium*, 2017.

[53] T. Wang, B. Ding, J. Zhou, C. Hong, Z. Huang, N. Li, and S. Jha. Answering multi-dimensional analytical queries under local differential privacy. In *SIGMOD*, 2019.

[54] T. Wang, N. Li, and S. Jha. Locally differentially private frequent itemset mining. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 127–143. IEEE, 2018.

[55] T. Wang, N. Li, and S. Jha. Locally differentially private heavy hitter identification. *IEEE Trans. Dependable Sec. Comput.*, 2019.

[56] T. Wang, M. Lopuhaä-Zwakenberg, Z. Li, B. Skoric, and N. Li. Locally differentially private frequency estimation with consistency. In *NDSS*, 2020.

[57] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[58] M. Xu, B. Ding, T. Wang, and J. Zhou. Collecting and analyzing data jointly from multiple services under local differential privacy. *Proceedings of the VLDB Endowment*, 13(12):2760–2772, 2020.

[59] J. Yang, T. Wang, N. Li, X. Cheng, and S. Su. Answering multi-dimensional range queries under local differential privacy. *arXiv preprint arXiv:2009.06538*, 2020.