LDPGuard: Defenses against Data Poisoning Attacks to Local Differential Privacy Protocols

Kai Huang[§], Gaoya Ouyang[§], Qingqing Ye[‡], Haibo Hu[‡], Bolong Zheng[†], Xi Zhao[§], Xiaofang Zhou[§]
[§]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology
[‡]Department of Electronic and Information Engineering, Hong Kong Polytechnic University
[†]School of Computer Science and Technology, Huazhong University of Science and Technology
ustkhuang|ouyanggaoya|xizhao|zxf@ust.hk, qqing.ye|haibo.hu@polyu.edu.hk, zblchris@gmail.com

***Abstract*—The protocols that satisfy Local Differential Privacy (LDP) enable untrusted third parties to collect aggregate information about a population without disclosing each user's privacy. In particular, each user locally encodes and perturbs his private data before sending it to the data collector, who aggregates and estimates the statistics about the population based on the collected perturbed values from individuals. Owing to their growing importance, LDP protocols have been widely studied and deployed in real-world scenarios (*e.g.,* Chrome and Windows). However, as data poisoning attacks may be injected by attackers who introduce many fake users, the utility of the statistics is heavily poisoned.**

In this paper, we present a generic and extensible framework called LDPGuard to address the problem. LDPGuard provides effective defenses against data poisoning attacks to LDP protocols for frequency estimation, a basic query of most data analytics tasks. In particular, it first precisely estimates the percentage of fake users and then provides adversarial schemes to defend against particular data poisoning attacks. Experimental study on real-world and synthetic datasets demonstrates the superiority of LDPGuard compared to existing techniques.

I. Introduction

Local Differential Privacy (LDP) that enables an untrusted data collector to collect aggregate information about a population has been accepted as a *de facto* standard for data privacy in the local setting. In particular, each user locally encodes and perturbs his private data before sending it to the untrusted data collector. After receiving the distributed perturbed data from users, the data collector designs detailed methods to aggregate and estimate statistics about the population without knowing the true values of each user. As the standard supports not only privacy-preserving data collection for untrusted third parties but also plausible deniability for each user, it has received a great amount of attention from both academia and industry. In recent years, a lot of protocols [4]–[11], [13]–[15], [19] that satisfy Local Differential Privacy (*a.k.a.,* LDP) have been developed. Some of the protocols have been deployed in industries such as Google, Apple, and Microsoft. In particular, Google integrated RAPPOR [7] into the Chrome browser to collect user responses to questions such as the default Chrome homepages while preserving their privacy. Apple [20] deployed LDP protocols in iOS to gather user preferences for emojis to identify popular emojis and recommend them to more users. Microsoft [23] developed LDP protocols to enable Windows 10 systems to aggregate statistics such as application usage. Most of these protocols are designed for frequency estimation, a basic query of most data analytics tasks (*e.g.,* marginal release [5] and heavy hitter identification [11]). Since each user involved in the protocol is required to perturb his value before sending it to the data collector, it largely sacrifices the utility of analytics results obtained by the data collector. Therefore, almost all existing protocols focused on designing effective strategies to improve the utility of data.

However, these protocols are exposed to data poisoning attacks, which are injected by attackers who introduce fake users to LDP protocols to craft the data and manipulate the data analytics results as they desire. In particular, fake users first choose some items (*a.k.a.,* target items) and adopt various attack schemes to promote the estimated frequencies for target items. Previous studies [25], [61] have already discovered that attackers have access to a number of compromised accounts of web services (*e.g.,* Hotmail, Twitter, and Google). In addition, they can purchase these compromised accounts from underground markets at very low prices (*e.g.,* \$0.004 – \$0.03 for a Hotmail account and \$0.03 – \$0.50 for a phone verified Google account). In general, the mainstream poisoning attacks consist of random perturbed-value attack (RPA), random item attack (RIA) and maximal gain attack (MGA) [25]. For RPA, each fake user randomly selects a value from the encoded space of the LDP protocols and sends it to the data collector; for RIA, each fake user involved selects a value from the target items and perturbs it before sending it to the data collector; as for MGA, each fake user tries to maximize the frequency gains for the target items by crafting his value by solving an optimization problem (detailed in Section III). These attacks are posing serious security problems to existing LDP protocols and threats to LDP-based data analytics. For example, by applying data poisoning attacks to LDP protocols for default homepages in Chrome, attackers can promote a phishing webpage as the default homepage in Chrome. Similarly, they can degrade the user experience on emojis by increasing the frequencies of some emojis through data poisoning attacks. In addition, the popularity of malicious applications in Windows 10 systems could be increased in a similar way. In short, all these attacks can cause damage to user experience and commercial interests. In recent years, two main countermeasures [25] including *normalization* and *fake users detection* are proposed for defending against these attacks. The former is to normalize the estimated item frequencies such that each estimated item frequency is non-negative and the overall summation of item frequencies is 1, while the latter is to detect possible fakes users based on their submitted values and to remove their impact on the final results. However, these countermeasures face two major challenges. The first is that the countermeasures are ad-hoc in nature, i.e., they can take effect on some scenarios but cannot adapt to others. The second is that they have limited effectiveness in reducing the impacts introduced by data poisoning attacks, as can be seen in Section VI.

In this paper, we present a generic and extensible framework called LDPGuard to address the aforementioned challenges. First, LDPGuard is a unified framework that supports different adversarial schemes to offset the adverse effects of various data poisoning attacks. It supports state-of-the-art LDP protocols for frequency estimation such as kRR [10], OUE [13], and OLH [13]. Second, LDPGuard adopts a two-round collection strategy to precisely estimate the percentage of fake users to facilitate effective countermeasures. The paper makes the following contributions.

- We introduce a unified framework called LDPGuard for defenses against data poisoning attacks to local differential privacy protocols. LDPGuard works for all existing data poisoning attacks and state-of-the-art LDP protocols.
- We present an effective two-round collection strategy to precisely estimate the percentage of fake users to facilitate effective countermeasures.
- We conduct an extensive experimental study on real-world and synthetic datasets to demonstrate the superiority of LDPGuard compared to existing techniques.

The rest of the paper is organized as follows. Related research is discussed in Section II, followed by preliminaries in Section III. The LDPGuard framework is described in Section IV and the estimation of the percentage of fake users is detailed in Section V. Experimental results are presented in Section VI. Section VII concludes the paper. Formal proofs of theorems and lemmas are given in Appendix A.

II. Related Work

Most germane to this research are local differential privacy, data poisoning attacks to LDP protocols/machine learning, and corresponding countermeasures.

Local Differential Privacy. Differential privacy (*a.k.a.,* DP) is a standard that provides semantic, information-theoretic privacy guarantees, which is more stringent than traditional privacy techniques such as generalization [26]–[29], [32], [32]. DP was first proposed in the centralized setting with a trusted data collector [30]. In real-world life, users do not fully trust the data curator for privacy. Therefore, much attention has shifted to the local differential privacy (*a.k.a.,* LDP), which can be cast as DP in the local setting and enables an untrusted data curator to collect aggregate information. A long line of research has been introduced for LDP especially frequency/mean estimation and practical data analysis task [2]–[19], [21], [22], [31]. Randomized response is a classical technique for structured survey interview [35] and motivates the first formalized LDP model [36]. [4] provides an efficient protocol called random matrix projection and the matching accuracy with lower bounds for succinct histogram estimation. Some protocols extended from the randomized response mechanism have been developed. For example, RAPPOR framework [7] extends the randomized response mechanism and adopts the Bloom filter to defend against the adversary that infers users' private information. The follow-up work [38] also extends RAPPOR to estimate more advanced statistics (*e.g.,* joint-distributions, association testing, and categorical attributes). For these LDP protocols and frameworks such as RAPPOR and random matrix projection [4], the basic task is frequency estimation. The state-of-the-art LDP protocols for frequency estimation consist of kRR [10], OUE [13], and OLH [13]. These protocols focused on improving data utility but may be exposed to data poisoning attacks, which are injected by attackers who introduce fake users to LDP protocols to craft the data and manipulate the analytics results as they desire.

Data poisoning attacks to LDP protocols. Data poisoning attacks consist of targeted attacks [25] and untargeted attacks [39]. The targeted attack, including random perturbed-value attack (RPA), random item attack (RIA), and maximal gain attack (MGA), is to increase the estimated frequencies for the attacker-chosen items (*i.e.,* targets). These attacks are posing serious security problems to existing LDP protocols and threats to LDP-based data analytics. In contrast, the untargeted attack is to distort the item frequency distribution by manipulating the L_p-norm distance between frequency vectors before and after attacking.

Data poisoning attacks to machine learning. There are a lot of works on data poisoning attacks to machine learning models including traditional machine learning models [40]–[43], deep networks [44]–[46] and recommendation systems [47]–[50]. In particular, [40] presents data poisoning attacks against autoregressive models. [41] develops data poisoning attacks to the classic SVM model. Data poisoning attacks to byzantine robust federated learning are also presented in [42]. The work in [44]–[46] focuses on data poisoning attacks to deep neural networks. Another line of research focus on top-n recommendation systems [47], graph-based recommendation systems [48], factorization-based and collaborative filtering [49]. Observe that most attacks aim to poison the training data to train a machine learning model with bad performance, they are orthogonal to the problem studied in this paper as the latter focuses on developing effective countermeasures against data poisoning attacks to LDP protocols.

Countermeasures against data poisoning attacks to LDP protocols. Although there are some countermeasures against data poisoning attacks to LDP protocols [43], [51] and Sybil attacks [52]–[59], the former does not focus on LDP protocols for frequency estimation but machine learning models, the latter utilizes various information such as content, behavior, and social graphs to detect fake users in social networks. Recently, Xiaoyu et al. present three countermeasures against data poisoning attacks to LDP protocols, namely, *normaliza-*

TABLE I: List of key notations.

Notation	Description
ϵ, ϵ_1, ϵ_2	privacy budgets
d, $\{1, 2, ..., d\}$	number of items, original data space
$\mathcal{D}$	encoded data space
Encode$(\cdot)$, Perturb$(\cdot)$	Encode algorithm, Decode algorithm
PE$(\cdot)$	composition operation of Encode and Perturb
$\textsc{Sup}(y)$	support set of the reported value y
p	$Pr[PE(v) \in \{y \| v \in \textsc{Sup}(y)\}]$
q	$Pr[PE(v') \in \{y \| v \in \textsc{Sup}(y)\}]$
p_1, q_1	p and q in the first round
p_2, q_2	p and q in the second round
p', q'	$p' = p_1$ or p_2, $q' = q_1$ or q_2
$\mathbb{I}_{\textsc{Sup}(y_i)}(v)$	indicator function
N, M	number of genuine users and fake users
Y_{true}, Y_{fake}	values from genuine users and fake users
y_i, Y	reported values from i-th user and all users
H, $\mathcal{H}$	a hash function, hash function family
r, τ	number of target items/ chosen target items
$T = \{t_1, t_2, ..., t_r\}$	target items
$\widetilde{f}_{t,b}$, $\widetilde{f}_{t,a}$	estimated frequencies before/after attacking
f_t	frequency of target t over all genuine users
β, $\widetilde{\beta}$	percentage/estimated percentage of fake users

tion, *fake users detection*, and *conditional probability based detection*. *Normalization* is to normalize the estimated item frequencies such that each estimated item frequency is non-negative and the summation of overall item frequencies is 1. *Fake users detection* is to detect potential fake users based on their submitted values and remove their impact on the final results. *Conditional probability based detection* is to detect possible fake users based on conditional probability, but it is applicable when there is only one target item. Moreover, the former two countermeasures are facing two major challenges. First, they are ad-hoc in nature, i.e., they can take effect on some scenarios but cannot adapt to others. Second, they have limited effectiveness in reducing the impacts introduced by data poisoning attacks.

III. Preliminaries

Table I lists the key notations and acronyms used in this paper. Recall that LDP enables an untrusted **data collector** to collect aggregate information of **users** who are willing to help with data collection but do not fully trust the data collector for privacy concerns. Without loss of generality, we assume that there are N users, each of which has a value $v \in \{1, 2, 3, ..., d\}$ where $\{1, 2, ..., d\}$ is **original data space** and d is the **number of items**. This paper focuses on LDP protocols for frequency estimation. The data collector aims to aggregate and estimate the frequencies of values among the N users. To this end, the following three algorithms are performed one by one:

- **Encode:** A user who holds a value $v \in \{1, 2, ..., d\}$ first encodes v to Encode$(v) \in \mathcal{D}$ where $\mathcal{D}$ is **encoded data space** (*i.e.,* the space of encoded values).
- **Perturb:** The user then perturbs the encoded value Encode(v) to Perturb(Encode(v)) and reports it to the data collector. We can also use PE$(\cdot)$ to denote the composition operation of the Encode and Perturb algorithms for simplification.
- **Aggregate:** The data collector receives the reported values from users and then estimates the statistics of interest.

The Perturb algorithm should satisfy the following ϵ-local differential privacy (ϵ-LDP).

Definition 1: **(Local Differential Privacy, LDP)** A algorithm $\mathcal{A}$ satisfies ϵ-LDP iff for any two inputs v and v',

$$\forall y \in Range(\mathcal{A}) : \frac{Pr[\mathcal{A}(v) = y]}{Pr[\mathcal{A}(v') = y]} \leq e^{\epsilon} \tag{1}$$

where ϵ ($\geq$0) is the privacy budget and $Range(\mathcal{A})$ denotes the set of all possible outputs of the algorithm $\mathcal{A}$.

To analyze the accuracy of LDP protocols, the **Pure LDP** that supports simple and fast aggregation is developed [13].

Definition 2: **(Pure Local Differential Privacy (Pure LDP))** A protocol $\mathcal{A}$ with an Encode and Perturb function (denoted by PE$(\cdot)$) satisfies Pure Local Differential Privacy (or ϵ-Pure LDP) iff for any input value v a user holds,

$$\forall y \in Range(\mathcal{A}) : \frac{p}{q} \leq e^{\epsilon}$$

such that

$$\begin{aligned} Pr[PE(v) \in \{y|v \in \textsc{Sup}(y)\}] = p \\ \forall v' \neq v : Pr[PE(v') \in \{y|v \in \textsc{Sup}(y)\}] = q \end{aligned} \tag{2}$$

where $\textsc{Sup}(y)$ is the set of values that y supports [13], ϵ ($\geq$0) is the privacy budget and $Range(\mathcal{A})$ is the set of all possible outputs of the protocol $\mathcal{A}$.

In particular, $\{y|v \in \textsc{Sup}(y)\}$ contains all outputs $\{y\}$ that "support" the occurrences of v (the support set of v for short). Note that the definition of the support set depends on a particular LDP protocol. For example, for the basic RAPPOR protocol [7] that encodes a value to a binary vector B whose v-th bit is 1, $\textsc{Sup}$(B) $= \{v|B[v] = 1\}$.

When each user follows a pure LDP to report his value y_i ($i \in \{1, 2, ..., d\}$) to the data collector, the estimated frequency of v is

$$\widetilde{f}_v = \frac{\frac{1}{N}\sum_{i=1}^{N} \mathbb{I}_{\textsc{Sup}(y_i)}(v) - q}{p - q} \tag{3}$$

where $\mathbb{I}_{\textsc{Sup}(y_i)}(v)$ is an indicator function. If $v \in \textsc{Sup}(y_i)$, $\mathbb{I}_{\textsc{Sup}(y_i)}(v) = 1$, and 0 otherwise. The estimated frequency $\widetilde{f}_v$ is an unbiased estimator of true frequency of v (Lemma 1), thus, $\mathbb{E}[\widetilde{f}_v] = f_v$ and

$$\sum_{i=1}^{N} \mathbb{E}[\mathbb{I}_{\textsc{Sup}(y_i)}(v)] = N(f_v(p - q) + q). \tag{4}$$

Lemma 1: $\widetilde{f}_v = \frac{\frac{1}{N}\sum_{i=1}^{N} \mathbb{I}_{\textsc{Sup}(y_i)}(v) - q}{p-q}$ is an unbiased estimator of the true frequency f_v of the item v.

A. Local Differential Privacy Protocols

In this subsection, we present three state-of-the-art LDP protocols for frequency estimation, namely kRR [10], OUE [13], and OLH [13], with a focus on their definitions and estimation procedures.

1) Randomized Response (kRR):

Encode: Each user encodes his value v to itself, *i.e.,* Encode(v) = v. The encoded data space $\mathcal{D} = \{1, 2, ..., d\}$, which is the same as the original data space.

Perturb: The user keeps the true value v with probability p and perturbs it to a different value $v' \in \mathcal{D}$ with probability q, then reports the perturbed value y =PE(v) to the data collector according to the following probability distribution.

$$Pr[y = a] = \begin{cases} \frac{e^\epsilon}{e^\epsilon+d-1} \triangleq p & \text{if } a = v \\ \frac{1}{e^\epsilon+d-1} \triangleq q & \text{otherwise} \end{cases} \tag{5}$$

Aggregate: The data collector receives the reported values $\{y_i | i \in \{1, 2, ..., d\}\}$ from users, and estimates the frequency of a particular item v with Equation (3). In particular, the support set $\textsc{Sup}(y_i)(v)$ is $\{y | y = v\}$.

2) Optimized Unary Encoding (OUE):

Encode: Each user encodes his value v to a d-dimensional binary vector B where v-th bit is 1, *i.e.,* B[v] = 1 and B[v'] = 0 for $v' \neq v$. The encoded data space $\mathcal{D} = \{0, 1\}^d$ since each bit could be 0 or 1.

Perturb: The user keeps each bit with value 1 (resp. 0) in B with probability p (resp. $1 - q$) and flips it with probability $1-p$ (resp. q), then reports the perturbed value y = PE(v) (or equally, y = Perturb(B)) to the data collector. Formally,

$$Pr[y[pos] = 1] = \begin{cases} \frac{1}{2} \triangleq p & \text{if } B[pos] = 1 \\ \frac{1}{e^\epsilon+1} \triangleq q & \text{if } B[pos] = 0 \end{cases} \tag{6}$$

Aggregate: The data collector receives the reported values $\{y_i | i \in \{1, 2, ..., d\}\}$ from users, and estimates the frequency of a particular item v with Equation (3). The support set $\textsc{Sup}(y_i)(v)$ for OUE is $\{y | y[v] = 1\}$.

3) Optimized Local Hashing (OLH):

Encode: Given a universal hash function family $\mathcal{H}$ such that the range of $H \in \mathcal{H}$ belongs to $\{1, 2, .., d'\}$, each user randomly selects a hash function H from $\mathcal{H}$ and encodes his value v to $\langle H, H(v) \rangle$. The encoded data space $\mathcal{D} = \{\langle H, h \rangle | H \in \mathcal{H}, h \in \{1, 2, ..., d'\}\}$. Note that $\mathcal{H}$ is usually generated by the hash algorithm called xxHash [60], which implements the hash function family with different random seeds. For a better estimation performance, we follow existing work [13], [25] to set $d' = e^\epsilon + 1$.

Perturb: The user keeps the $H(v)$ with probability p and perturbs it to a different value $H(v)' \in \{1, 2, ..., d'\}$ with probability q, and then reports the perturbed value y = PE(v) = $\langle H, H(v)' \rangle$ to the data collector. Formally,

$$Pr[y = \langle H, h \rangle] = \begin{cases} \frac{e^\epsilon}{e^\epsilon+d'-1} \triangleq p^* & \text{if } h = H(v) \\ \frac{1}{e^\epsilon+d'-1} \triangleq q^* & \text{if } h \neq H(v) \end{cases} \tag{7}$$

Aggregate: The data collector receives the reported values $\{y_i | i \in \{1, 2, ..., d\}\}$ from users, and estimates the frequency of a particular item v by Equation (3) with the support set $\textsc{Sup}(y_i)(v) = \{y | y = \langle H, h \rangle \text{ and } H(v) = h\}$. Note that p^* and q^* are for hash function H instead of the pair of $\langle H, H(\cdot) \rangle$. As each hash function is randomly selected from $\mathcal{H}$, the overall perturbation probability parameters are $p = p^* = \frac{e^\epsilon}{e^\epsilon+d'-1}$ and $q = \frac{1}{d'}p^* + (1 - \frac{1}{d'})q^* = \frac{1}{d'}$.

B. Threat Model and Data Poisoning Attacks

Data poisoning attack is to increase the estimated frequencies for the attacker-chosen items (*i.e.,* targets). In particular, given a target set with r target items, $T = \{t_1, t_2, ..., t_r\}$ where $t_j \in \{1, 2, ..., d\}$, suppose the estimated frequencies of a particular item t before and after attacking are $\widetilde{f}_{t,b}$ and $\widetilde{f}_{t,a}$ respectively, data poisoning attack is to injects M fake users to increase the **overall frequency gain** over T, *i.e.,* $\sum_{t \in T} \triangle\widetilde{f}_t = \widetilde{f}_{t,a} - \widetilde{f}_{t,b}$ where $\triangle\widetilde{f}_t$ is the **frequency gain** of a single target item t.

In general, it consists of **random perturbed-value attack (RPA)**, **random item attack (RIA)**, and **maximal gain attack (MGA)** [25]. *For RPA, each fake user randomly selects a value from the encoded space of LDP protocols and sends it to the data collector; for RIA, each fake user involved selects a value from the target item set and perturbs it before sending it to a data collector; as for MGA, each fake user tries to maximize the frequency gains for the target items by crafting his value via solving the following optimization problem.* Let reported values of fake users be Y_{fake}, the overall frequency gain over T be $Gain(Y_{fake})$, to maximize the gain, we have

$$\begin{aligned}
&\underset{Y_{fake}}{Max} \quad Gain(Y_{fake}) \iff \underset{Y_{fake}}{Max} \sum_{t \in T} \mathbb{E}[\triangle\widetilde{f}_t] \\
&\iff \underset{Y_{fake}}{Max} \sum_{t \in T} \mathbb{E}[\widetilde{f}_{t,a} - \widetilde{f}_{t,b}] \iff \underset{Y_{fake}}{Max} \sum_{t \in T} \mathbb{E}[\widetilde{f}_{t,a}] - \mathbb{E}[\widetilde{f}_{t,b}] \\
&\overset{Equ(3)}{\iff} \underset{Y_{fake}}{Max} \sum_{t \in T} \mathbb{E}\left[\frac{\frac{\sum_{i=1}^{N+M} \mathbb{I}_{\textsc{Sup}(y_i)}(t)}{N+M} - q}{p-q}\right] - \mathbb{E}\left[\frac{\frac{\sum_{i=1}^{N} \mathbb{I}_{\textsc{Sup}(y_i)}(t)}{N} - q}{p-q}\right] \\
&\iff \underset{Y_{fake}}{Max} \sum_{t \in T} \frac{\sum\limits_{i=N+1}^{N+M} \mathbb{E}[\mathbb{I}_{\textsc{Sup}(y_i)}(t)]}{(N+M)(p-q)} - \frac{M\sum\limits_{i=1}^{N} \mathbb{E}[\mathbb{I}_{\textsc{Sup}(y_i)}(t)]}{N(N+M)(p-q)} \\
&\overset{Equ(4)}{\iff} \underset{Y_{fake}}{Max} \frac{\sum\limits_{i=N+1}^{N+M} \sum_{t \in T} \mathbb{E}[\mathbb{I}_{\textsc{Sup}(y_i)}(t)]}{(N+M)(p-q)} - \frac{\sum\limits_{t \in T} MN(f_t(p-q)+q)}{N(N+M)(p-q)},
\end{aligned} \tag{8}$$

where f_t is the frequency of the target t over all genuine users. As $\frac{\sum_{t \in T} MN(f_t(p-q)+q)}{N(N+M)(p-q)}$ only depends on genuine users, MGA is to craft Y_{fake} such that $\frac{\sum_{i=N+1}^{N+M} \sum_{t \in T} \mathbb{E}[\mathbb{I}_{\textsc{Sup}(y_i)}(t)]}{(N+M)(p-q)}$ is maximized. To this end, different implementation strategies are adopted for kRR, OUE, and OLH as listed below.

MGA for kRR. For kRR, observe that $\sum_{t \in T} \mathbb{E}[\mathbb{I}_{\textsc{Sup}(y_i)}(t)] \leq 1$ and $\sum_{t \in T} \mathbb{E}[\mathbb{I}_{\textsc{Sup}(y_i)}(t)] = 1$ if and only if y_i belongs to T. Therefore, MGA for kRR is obtained by randomly selecting a target item for each fake user.

MGA for OUE. For OUE, observe that $\sum_{t \in T} \mathbb{E}[\mathbb{I}_{\textsc{Sup}(y_i)}(t)] \leq r$ and $\sum_{t \in T} \mathbb{E}[\mathbb{I}_{\textsc{Sup}(y_i)}(t)] = r$ if and only if $\forall t \in T$, $y_i[t] = 1$. Intuitively, implementing MGA for OUE only needs to craft each fake user's value with $y_i[t] = 1$ for each target t, but this will result in only r bits 1 in the encoded binary vector. In contrast, each genuine user may report y_i with $\lfloor p + (d-1)q \rfloor$ bits 1. As such, MGA for OUE reports the binary vector with $y_i[t] = 1$ for each target t and $l = \lfloor p + (d-1)q - r \rfloor$ bits 1 for non-target bits.

MGA for OLH. For OLH, observe that $\sum_{t\in T}\mathbb{E}[\mathbb{I}_{\text{SUP}(y_i)}(t)] \leq r$ and $\sum_{t\in T}\mathbb{E}[\mathbb{I}_{\text{SUP}(y_i)}(t)] = r$ if and only if the selected hash function hashes all targets to the same value. In our implementation, we follow [25] to sample 1,000 hash functions from xxHash and select the one that results in the most targets to the same value.

IV. LDPGuard: The framework

In this section, we overview our LDPGuard framework for effective defenses against data poisoning attacks to local differential privacy protocols. In particular, LDPGuard first adopts a two-round collection method to accurately estimate the percentage of fake users, it then utilizes the estimated percentage to provide adversarial schemes to defend against particular data poisoning attacks such as RPA, RIA, and MGA.

A. Overview of LDPGuard

The workflow of LDPGuard is depicted in Figure 1. Suppose there are N **genuine users**, each user holds a value v_i ($i \in \{1, 2, ..., N\}$) in the **original data space** (*i.e.,* $v_i \in \{1, 2, ..., d\}$), encodes v_i to a value in **encoded data space** $\mathcal{D}$, perturbs the encoded value under ϵ_1-LDP, and finally reports the perturbed value $y_i = PE(v_i)$ to the data collector (step ①). In addition, there are M **fake users**, each fake user holds a value v_i ($i \in \{N + 1, N + 2, ..., N + M\}$) in the original data space, and adopts one of the following procedures to obtained the reported value y_i (step ②):

- **Encode only.** The fake user only encodes the input v_i to obtain y_i, *i.e.,* $y_i = \text{Encode}(v_i)$. This applies to both RPA and MGA attacks, which do not utilize any LDP protocols to perturb user inputs. In particular, for RPA, each fake user randomly selects a value from encoded data space and sends it to the data collector directly; for MGA, a fake user crafts a value via solving the optimization problem (see Equation (8)) and reports it without perturbation.
- **Encode and Perturb.** The fake user first encodes the input v_i and then perturbs the encoded value to obtain y_i, *i.e.,* $y_i = \text{PE}(v_i)$. This applies to RIA attacks.

Therefore, the reported value y_i from a fake user is either $y_i = \text{PE}(v_i)$ or $y_i = \text{Encode}(v_i)$. The values $\{y_i | i \in \{1, 2, ..., N+M\}\}$ are then reported to the data collector (step ③) for further processing.

Upon receiving the reported values Y consisting of $Y_{true} = \{y_i | i \in \{1, 2, ..., N\}\}$ from genuine users and $Y_{fake} = \{y_i | i \in \{N + 1, N + 2, ..., N + M\}\}$ from fake users in the first round, the second round with the same process is launched by the data collector (steps ①, ② and ③). Note that the privacy budget in the second round is ϵ_2 and $\epsilon_1 + \epsilon_2 = \epsilon$ (by default, $\epsilon_1 = \epsilon_2 = \frac{1}{2}\epsilon$). *We call this entire process "Two-Round Collection".* Then, the data collector utilizes **percentage estimator (detailed in Section V)** to estimate the percentage of fake users (step ④) by comparing the reported values Y in the two rounds . The estimated percentage (denoted by $\widetilde{\beta}$) is then used for **defenses against attacks (detailed in Section IV-B)** (step ⑤). Finally, the data collector

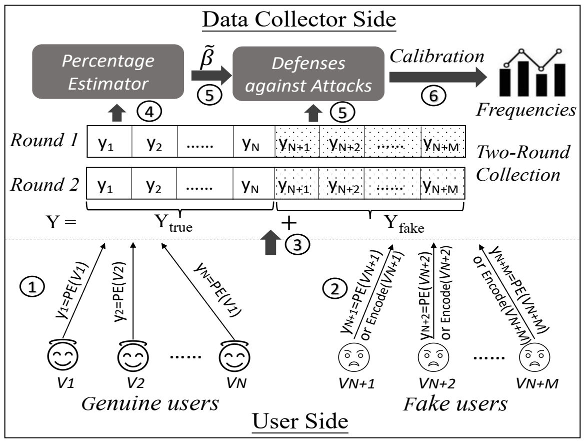


Fig. 1: The LDPGuard framework.

adopts calibrations (*i.e.,* Equation 3) to derive item frequencies for publications (step ⑥).

B. Defenses against Data Poisoning Attacks

In this section, we assume the estimated percentage $\widetilde{\beta}$ of fake users is already derived. We will remove this assumption in Section V. Based on $\widetilde{\beta}$, LDPGuard provides effective adversarial schemes to defend against data poisoning attacks including RPA, RIA, and MGA to start-of-the-art LDP protocols such as kRR, OUE, and OLH. The main idea behind the adversarial schemes is to adopt the operation in the opposite direction of the one attacks adopted to offset the adverse effects of data poisoning attacks. In particular, LDPGuard first generates $(N + M)\beta$ records from possible distributions from which attacks sampled their values. It then removes the records from reported values of genuine and fake users to alleviate adverse effects of attacks. Detailed defenses for different attacks and LDP protocols are discussed below.

1) Defenses against Data Poisoning Attacks to kRR:

We begin by setting $p_1 = \frac{e^{\epsilon_1}}{e^{\epsilon_1}+d-1}$, $q_1 = \frac{1}{e^{\epsilon_1}+d-1}$, and $p_2 = \frac{e^{\epsilon_2}}{e^{\epsilon_2}+d-1}$, $q_2 = \frac{1}{e^{\epsilon_2}+d-1}$ where $\epsilon_1 + \epsilon_2 = \epsilon$. In addition, $p = \frac{e^{\epsilon}}{e^{\epsilon}+d-1}$, $q = \frac{1}{e^{\epsilon}+d-1}$.

Defenses against RPA to kRR. First, LDPGuard randomly samples $(N + M)\widetilde{\beta}$ values (denoted by $\widetilde{Y}_{fake}$) with replacement from encoded data space $\mathcal{D} = \{1, 2, ..., d\}$. For each sampled value $v \in \widetilde{Y}_{fake}$, LDPGuard then removes a corresponding record from all reported values Y if $v \in Y$. Finally, the data collector calibrates the results to obtain the estimated frequency for item v via

$$\widetilde{f}_v = \frac{\frac{1}{|Y\setminus\widetilde{Y}_{fake}|}\sum_{y_i\in Y\setminus\widetilde{Y}_{fake}}\mathbb{I}_{\text{SUP}(y_i)}(v) - q'}{p' - q'} \tag{9}$$

where p'=p_1 or p_2, q'=q_1 or q_2, $Y \setminus \widetilde{Y}_{fake}$ is the remaining records in Y after removing $\widetilde{Y}_{fake}$. The expectation of $\widetilde{f}_v$ is

$$\begin{aligned}\mathbb{E}[\widetilde{f}_v] &= \frac{\mathbb{E}\Big[\frac{1}{|Y\setminus\widetilde{Y}_{fake}|}\sum_{y_i\in Y\setminus\widetilde{Y}_{fake}}\mathbb{I}_{\text{SUP}(y_i)}(v)\Big] - q'}{p' - q'} \\ &= \frac{\frac{N_1(f_v(p'-q')+q')+N_2*f_v^*}{N_1+N_2} - q'}{p' - q'}\end{aligned} \tag{10}$$

where $N_1 = |Y_{true} \cap (Y \setminus \widetilde{Y}_{fake})|$, $N_2 = |Y \setminus \widetilde{Y}_{fake} \setminus Y_{true}|$, and $f_v^* = \frac{1}{d}$ is the probability that v equals the value sampled from $Y \setminus \widetilde{Y}_{fake} \setminus Y_{true}$. Observe that if $N_1 = N$ (*i.e.,* $Y \setminus \widetilde{Y}_{fake} = Y_{true}$), $N_2 = 0$, $\widetilde{f}_v$ is an unbiased estimator of f_v since $\mathbb{E}[\widetilde{f}_v] = f_v$. The overall frequency gain is

$$\begin{aligned}
Gain(Y_{fake}) &= \sum_{t\in T} \mathbb{E}[\widetilde{f}_{t,a} - \widetilde{f}_{t,b}] = \sum_{t\in T} \mathbb{E}[\widetilde{f}_{t,a}] - \mathbb{E}[\widetilde{f}_{t,b}] \\
&\overset{Equ(3)}{\underset{Equ(10)}{=\!=}} \sum_{t\in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2*1/d}{N_1+N_2} - q'}{p'-q'} - \mathbb{E}\left[\frac{\frac{\sum_{i=1}^{N} \mathbb{I}_{\textsc{Sup}(y_i)}(t)}{N} - q}{p-q}\right] \\
&\overset{Equ(4)}{=\!=} \sum_{t\in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2*1/d}{N_1+N_2} - q'}{p'-q'} - \frac{(f_t(p-q)+q)-q}{p-q} \\
&=\!= \sum_{t\in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2*1/d}{N_1+N_2} - f_t(p'-q')}{p'-q'}.
\end{aligned} \tag{11}$$

If $N_1 = N$, $N_2 = 0$, the gain $Gain(Y_{fake})$ is 0.

Defenses against RIA to kRR. For RIA, LDPGuard first randomly samples $(N+M)\widetilde{\beta}$ values (denoted by $\widetilde{Y}_{fake}$) with replacement from target set $T = \{t_1, t_2, ..., t_r\}$, and then encodes each value to the encoded data space $\mathcal{D} = \{1, 2, ..., d\}$. For the encoded data, it keeps the true value with probability $p = p'$ ($p' = p_1$ or p_2) and perturbs it to a different value in $\mathcal{D}$ with probability $q = q'$ ($q' = q_1$ or q_2). Finally, LDPGuard removes the perturbed value from Y if the perturbed value is found in Y, and calibrates the results to obtain the estimated frequency $\widetilde{f}_v$ of the item v using Equation (9). According to Equation (10), the expectation of $\widetilde{f}_v$ is

$$\mathbb{E}[\widetilde{f}_v] = \frac{\frac{N_1(f_v(p'-q')+q')+N_2 f_v^*}{N_1+N_2} - q'}{p'-q'}, f_v^* = \begin{cases} \frac{1}{r}(p'-q')+q', v \in T \\ q', v \notin T \end{cases} \tag{12}$$

where $N_1 = |Y_{true} \cap (Y \setminus \widetilde{Y}_{fake})|$, $N_2 = |Y \setminus \widetilde{Y}_{fake} \setminus Y_{true}|$, and f_v^* is the probability that v equals the value sampled from $Y \setminus \widetilde{Y}_{fake} \setminus Y_{true}$. If $N_1 = N$, $N_2 = 0$, $\widetilde{f}_v$ is an unbiased estimator of f_v since $\mathbb{E}[\widetilde{f}_v] = f_v$, and overall frequency gain $Gain(Y_{fake})$ shown below is 0.

$$\begin{aligned}
Gain(Y_{fake}) &= \sum_{t\in T} \mathbb{E}[\widetilde{f}_{t,a} - \widetilde{f}_{t,b}] = \sum_{t\in T} \mathbb{E}[\widetilde{f}_{t,a}] - \mathbb{E}[\widetilde{f}_{t,b}] \\
&\overset{Equ(4)}{\underset{Equ(12)}{=\!=}} \sum_{t\in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2(\frac{p'-q'}{r}+q')}{N_1+N_2} - q'}{p'-q'} - \frac{f_t(p-q)+q-q}{p-q} \\
&=\!= \sum_{t\in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2(1/r(p'-q')+q')}{N_1+N_2} - f_t(p'-q')}{p'-q'}.
\end{aligned}$$

Defenses against MGA to kRR. For RPA, LDPGuard first randomly samples $(N+M)\widetilde{\beta}$ values (denoted by $\widetilde{Y}_{fake}$) with replacement from from target set $T = \{t_1, t_2, ..., t_r\}$ instead of $\mathcal{D}$. For each value sampled, LDPGuard then removes the record $v \in \widetilde{Y}_{fake}$ from Y if $v \in Y$. Finally, the data collector calibrates the results to obtain the estimated frequency $\widetilde{f}_v$ for the item v using Equation (9). According to Equation (10), the expectation of $\widetilde{f}_v$ is

$$\mathbb{E}[\widetilde{f}_v] = \frac{\frac{N_1(f_v(p'-q')+q')+N_2 f_v^*}{N_1+N_2} - q'}{p'-q'}, f_v^* = \begin{cases} \frac{1}{r} & \text{if } v \in T \\ 0 & \text{if } v \notin T \end{cases} \tag{13}$$

The overall frequency gain

$$\begin{aligned}
Gain(Y_{fake}) &= \sum_{t\in T} \mathbb{E}[\widetilde{f}_{t,a} - \widetilde{f}_{t,b}] = \sum_{t\in T} \mathbb{E}[\widetilde{f}_{t,a}] - \mathbb{E}[\widetilde{f}_{t,b}] \\
&\overset{Equ(4)}{\underset{Equ(13)}{=\!=}} \sum_{t\in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2*1/r}{N_1+N_2} - q'}{p'-q'} - \frac{(f_t(p-q)+q)-q}{p-q} \\
&=\!= \sum_{t\in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2*1/r}{N_1+N_2} - f_t(p'-q')}{p'-q'}.
\end{aligned}$$

Accordingly, $Gain(Y_{fake})$ is 0 if $N_1 = N$ and $N_2 = 0$.

2) *Defenses against Data Poisoning Attacks to OUE:*

For OUE, we set $p_1 = \frac{1}{2}$, $q_1 = \frac{1}{e^{\epsilon_1}+1}$, and $p_2 = \frac{1}{2}$, $q_2 = \frac{1}{e^{\epsilon_2}+1}$ where $\epsilon_1 + \epsilon_2 = \epsilon$. Also, let $p = \frac{1}{2}$, $q = \frac{1}{e^{\epsilon}+1}$, $p' = p_1$ or p_2, and $q' = q_1$ or q_2.

Defenses against RPA to OUE. LDPGuard first samples $(N+M)\widetilde{\beta}$ values (denoted by $\widetilde{Y}_{fake}$) randomly with replacement from the encoded data space $\mathcal{D} = \{0,1\}^d$. For each value sampled, LDPGuard then removes the record $v \in \widetilde{Y}_{fake}$ from all reported Y if $v \in Y$. Finally, the data collector calibrates the results to obtain the estimated frequency $\widetilde{f}_v$ for the item v using Equation (9). On average, each value sampled from $\mathcal{D} = \{0,1\}^d$ supports v with probability $\frac{1}{2}$, according to Equation (10), the expectation of $\widetilde{f}_v$ is

$$\mathbb{E}[\widetilde{f}_v] = \frac{\frac{N_1(f_v(p'-q')+q')+N_2*1/2}{N_1+N_2} - q'}{p'-q'} \tag{14}$$

where $N_1 = |Y_{true} \cap (Y \setminus \widetilde{Y}_{fake})|$ and $N_2 = |Y \setminus \widetilde{Y}_{fake} \setminus Y_{true}|$. Observe that if $N_1 = N$ (*i.e.,* $Y \setminus \widetilde{Y}_{fake} = Y_{true}$), $N_2 = 0$, $\widetilde{f}_v$ is an unbiased estimate of f_v since $\mathbb{E}[\widetilde{f}_v] = f_v$ and the following overall frequency gain $Gain(Y_{fake})$ is 0.

$$Gain(Y_{fake}) \overset{Equ(14)}{=\!=} \sum_{t\in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2*1/2}{N_1+N_2} - f_t(p'-q')}{p'-q'}.$$

Defenses against RIA to OUE. For RIA, LDPGuard first randomly samples $(N+M)\widetilde{\beta}$ values (denoted by $\widetilde{Y}_{fake}$) with replacement from target set $T = \{t_1, t_2, ..., t_r\}$, and then encodes each value to the encoded data space $\mathcal{D} = \{0,1\}^d$. Obviously, each encoded value is a d-bits binary vector, where each bit with value 1 (resp. 0) is kept with probability $p' = p_1$ or p_2 (resp. $1-q'$) and flipped with probability $1-p'$ (resp. $q' = q_1$ or q_2). Finally, LDPGuard removes the perturbed value from Y if the perturbed value is found in Y, and calibrates the results to obtain the estimated frequency for the item v using Equation (9). Similar to RIA to kRR, the expectation of $\widetilde{f}_v$ is $\frac{(N_1(f_v(p'-q')+q')+N_2*f_v^*)/(N_1+N_2)-q'}{p'-q'}$ where $f_v^* = (1/r(p'-q')+q')$, if $v \in T$; and $f_v^* = q'$, otherwise. Consequently, the overall frequency gain is

$$Gain(Y_{fake}) = \sum_{t\in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2(1/r(p'-q')+q')}{N_1+N_2} - f_t(p'-q')}{p'-q'}.$$

Similarly, $Gain(Y_{fake})$ is 0 if $N_1 = N$ and $N_2 = 0$.

Defenses against MGA to OUE. As for MGA to OUE, LDPGuard first initializes $(N + M)\widetilde{\beta}$ d-bits zero vectors (denoted by $\widetilde{Y}_{fake}$), for $y_i \in \widetilde{Y}_{fake}$, and then sets $y_i[t] = 1$ for each bit $t \in \mathcal{T}$ and $l = \lfloor p + (d-1)q - r \rfloor$ randomly selected non-target bits. After that, LDPGuard removes the record $y_i \in \widetilde{Y}_{fake}$ from Y if $y_i \in Y$. Finally, the data collector calibrates the results to obtain the estimated frequency of the item v using Equation (9). The expectation of $\widetilde{f}_v$ is $\frac{(N_1(f_v(p'-q')+q')+N_2 * f_v^*)/(N_1+N_2)-q'}{p'-q'}$ where $f_v^* = 1$, if $v \in T$; and $f_v^* = 0$, otherwise. The overall frequency gain is

$$Gain(Y_{fake}) = \sum_{t \in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2}{N_1+N_2} - f_t(p'-q')}{p'-q'}.$$

$\widetilde{f}_v$ is an unbiased estimator of f_v and the overall frequency gain $Gain(Y_{fake})$ is 0 if $N_1 = N$ and $N_2 = 0$.

3) Defenses against Data Poisoning Attacks to OLH: Let $p^* = \frac{e^\epsilon}{e^\epsilon + d' - 1}$, $q^* = \frac{1}{e^\epsilon + d' - 1}$, $p = p^* = \frac{e^\epsilon}{e^\epsilon + d' - 1}$ and $q = \frac{1}{d'}p^* + (1 - \frac{1}{d'})q^* = \frac{1}{d'}$. Similarly, p_1^*, q_1^*, p_1 and q_1 (resp. p_2^*, q_2^*, p_2 and q_2) are defined w.r.t. ϵ_1 (resp. ϵ_2). In addition, let $p' = p_1$ or p_2, $q' = q_1$ or q_2.

Defenses against RPA to OLH. First, LDPGuard crafts $(N+M)\widetilde{\beta}$ key-value pairs $\{\langle H, h\rangle | H \in \mathcal{H}, h \in \{1, 2, ..., d'\}\}$ (denoted by $\widetilde{Y}_{fake}$) where H is randomly selected from $\mathcal{H}$ and h is randomly drawn from $\{1, 2, ..., d'\}$ ($d' = e^\epsilon + 1$). Each record in $\widetilde{Y}_{fake}$ is removed from Y if it exists in Y. Finally, the data collector calibrates the results to obtain the frequency of the item v using Equation (9). The expectation of $\widetilde{f}_v$ is $\frac{(N_1(f_v(p'-q')+q')+N_2 * f_v^*)/(N_1+N_2)-q'}{p'-q'}$ where $f_v^* = \frac{1}{d'}$, if $v \in T$; and $f_v^* = 0$, otherwise. The overall frequency gain

$$Gain(Y_{fake}) = \sum_{t \in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2 * 1/d'}{N_1+N_2} - f_t(p'-q')}{p'-q'}.$$

If $N_1 = N$, $N_2 = 0$, $\widetilde{f}_v$ is an unbiased estimator of f_v since $\mathbb{E}[\widetilde{f}_v] = f_v$, and the overall frequency gain $Gain(Y_{fake})$ is 0.

Defenses against RIA to OLH. To defend against RIA to OLH, LDPGuard first samples $(N + M)\widetilde{\beta}$ values from target set $\mathcal{T}$. For each sampled value v, a hash function H is randomly selected from $\mathcal{H}$. Then, the key-value pair $\langle H, h\rangle$ is perturbed to $\langle H, h'\rangle$ ($h' \neq h$) with probability q_1^* or q_2^*, and remains unchanged with probability p_1^* or p_2^*. All these perturbed values are removed from Y, and the data collector calibrates the results to obtain the estimated frequency for the item v using Equation (9). The expectation of $\widetilde{f}_v$ is $\frac{(N_1(f_v(p'-q')+q')+N_2 * f_v^*)/(N_1+N_2)-q'}{p'-q'}$ where $f_v^* = \frac{1}{r}(p'-q') + q'$, if $v \in T$; and $f_v^* = q'$, otherwise. The overall frequency gain is

$$Gain(Y_{fake}) = \sum_{t \in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2 * (\frac{1}{r}(p'-q')+q')}{N_1+N_2} - f_t(p'-q')}{p'-q'}.$$

Similarly, $\widetilde{f}_v$ is an unbiased estimator of f_v since $\mathbb{E}[\widetilde{f}_v] = f_v$, and $Gain(Y_{fake})$ is 0 if $N_1 = N$ and $N_2 = 0$.

Defenses against MGA to OLH. As for MGA to OLH, LDPGuard crafts $(N + M)\widetilde{\beta}$ key-value pairs $\{\langle H, h\rangle | H \in \mathcal{H}, h \in \{1, 2, ..., d'\}\}$ (denoted by $\widetilde{Y}_{fake}$) where H is randomly selected from $\mathcal{H}$ and h is the hash function that results in the most targets to the same value. Each record in $\widetilde{Y}_{fake}$ is removed from Y if it exists in Y. Finally, the data collector calibrates the results to obtain the estimated frequency for the item v with Equation (9). The expectation of $\widetilde{f}_v$ is $\frac{(N_1(f_v(p'-q')+q')+N_2 * f_v^*)/(N_1+N_2)-q'}{p'-q'}$ where $f_v^* = 1$, if $v \in T$; and $f_v^* = 0$, otherwise. Obviously, if $N_1 = N$ (*i.e.,* $Y \setminus \widetilde{Y}_{fake} = Y_{true}$), $N_2 = 0$, $\widetilde{f}_v$ is an unbiased estimator of f_v since $\mathbb{E}[\widetilde{f}_v] = f_v$, and the following gain is 0.

$$Gain(Y_{fake}) = \sum_{t \in T} \frac{\frac{N_1(f_t(p'-q')+q')+N_2}{N_1+N_2} - f_t(p'-q')}{p'-q'}.$$

C. Discussions

Communication and Computing Cost. The communication cost of each user is $\mathcal{O}(\log d)$, $\mathcal{O}(d)$, and $\mathcal{O}(\log d)$ for kRR, OUE and OLH, respectively. The computing cost of the data collector is $\mathcal{O}(N + M + d)$, $\mathcal{O}((N + M) * d)$, and $\mathcal{O}((N + M) * d)$ for kRR, OUE and OLH, respectively.

Privacy Analysis. The first round collection satisfies ϵ_1-LDP and the second round collection satisfies ϵ_2-LDP, according to *Sequential Composition* (see Lemma 2), LDPGuard satisfies ϵ-LDP where $\epsilon = \epsilon_1 + \epsilon_2$.

Lemma 2: Given C algorithms $\{\mathcal{A}_i | i \in \{1, 2, ..., C\}\}$, each $\mathcal{A}_i$ satisfies ϵ_i-LDP, the sequence of algorithms $\mathcal{A}_i$ ($i \in \{1, 2, ..., C\}$) collectively satisfies $\sum_{i \in \{1,2,...,C\}} \epsilon_i$-LDP.

Furthermore, observe that if $N_1 = N$ (*i.e.,* $Y \setminus \widetilde{Y}_{fake} = Y_{true}$), $N_2 = 0$, $\widetilde{f}_v$ is an unbiased estimator of f_v since $\mathbb{E}[\widetilde{f}_v] = f_v$, and the overall frequency gain $Gain(Y_{fake})$ is 0. In other words, the effects of poisoning attacks are offset with the proposed defenses if $N_1 = N$ (*i.e.,* $Y \setminus \widetilde{Y}_{fake} = Y_{true}$). To this end, $\widetilde{Y}_{fake}$ should be sampled from the same distribution as the one injected into attacks, as mentioned in this section. Furthermore, the same number of noise data should be removed, *i.e.,* $(N+M)\widetilde{\beta}$ should be close to the real number of fake users (*i.e.,* M). We address this problem by accurately estimating the percentage of fake users in Section V.

V. Estimation of the percentage of fake users

Observe that the reported values of fake users may be statistically abnormal compared to those of genuine users. For example, the values of fake users who engaged in RPA attacks are very randomly distributed in each round of the two-round collection; those of fake users with MGA attacks are too determinate so that they can be distinguished from the reported values from genuine users. This observation motivates us to accurately estimate the percentage of fake users by comparing the values reported in the two-round collection. In particular, let P_1 (resp. P_2) be the probability that the data collector collects the same value from the same genuine user (resp. fake user) in two different rounds, and CNT is the number of users who reported the same value in two different rounds, the percentage of fake users (*i.e.,* $\beta = \frac{M}{N+M}$) can be estimated by

$$\widetilde{\beta} = \frac{(N+M)P_1 - CNT}{(N+M)(P1 - P2)}. \quad (15)$$

Theorem 1: $\widetilde{\beta} = \frac{(N+M)P_1 - CNT}{(N+M)(P1-P2)}$ is an unbiased estimator of the percentage of fake users.

The problem now becomes how to compute P_1 and P_2. We answer this question in the remaining parts of this section.

A. Percentage Estimation for kRR

Recall that each genuine user reports his value with kRR follows the following three steps: 1) encodes his value v to itself, *i.e.,* Encode(v) = v; 2) keeps the true value v with probability p' and perturbs it to a different value $v' \in \mathcal{D} = \{1, 2, ..., d\}$ with probability q'. For the two-round collection, let $p' = p_1$ or p_2 and $q' = q_1$ or q_2. 3) and finally reports the perturbed value y = PE(v) to the data collector. Therefore, a genuine user reports the same value when he 1) reports the true value v in both the first and second round; or 2) reports the same value v' ($v' \neq v$). The probability P_1 that the data collector collects the same value from a genuine user in two rounds is

$$P_1 = (p')^2 + (d-1)(q')^2. \tag{16}$$

where $(p')^2$ (resp. $(q')^2$) is the probability that a genuine user reports the same value v (resp. v').

Computing P_2 for RPA to kRR. Since each fake user engaged in RPA attacks randomly selects a value from the encoded data space $\mathcal{D} = \{1, 2, ..., d\}$, each item $v \in \mathcal{D}$ is selected in each round with probability $\frac{1}{d}$. Therefore, the first and second rounds select the same item v with probability $\frac{1}{d} * \frac{1}{d}$. Furthermore, since v could be any element in $\mathcal{D}$,

$$\begin{aligned} P_2 &= \sum_{v \in \mathcal{D}} Pr(v \text{ in } 1st \text{ round}) * Pr(v \text{ in } 2nd \text{ round}) \\ &= \sum_{v \in \mathcal{D}} \frac{1}{d} * \frac{1}{d} = d * (\frac{1}{d} * \frac{1}{d}) = \frac{1}{d}. \end{aligned}$$

Computing P_2 for RIA to kRR. As a fake user is to select a item v from target set $T = \{t_1, t_2, ..., t_r\}$ and perturbs it to a different value in $\mathcal{D}$ (resp. remains the same value) with probability $q' = q_1$ or q_2 (resp. $p' = p_1$ or p_2), computing P_2 for RIA to kRR should be divided into two cases. The first is that the values reported in the two rounds are the same and belong to T. In this case, the probability that the fake user reports the same value is $\sum_{v \in T} (\frac{1}{r}p' + (1 - \frac{1}{r})q')^2$ where $\frac{1}{r}p'$ (resp. $(1 - \frac{1}{r})q'$) is the probability that the reported value is not perturbed (resp. perturbed). The second is that the values reported in two rounds are the same, and not in T. In this case, the reported value must be perturbed from the true value. Thus, the probability that the fake user reports the same value is $\sum_{v \in \mathcal{D} \setminus T} (q')^2$. Therefore,

$$\begin{aligned} P_2 &= \sum_{v \in T} Pr(v \text{ in } 1st \text{ round}) * Pr(v \text{ in } 2nd \text{ round}) \\ &+ \sum_{v \in \mathcal{D} \setminus T} Pr(v \text{ in } 1st \text{ round}) * Pr(v \text{ in } 2nd \text{ round}) \\ &= \sum_{v \in T} (\frac{1}{r}p' + (1 - \frac{1}{r})q')^2 + \sum_{v \in \mathcal{D} \setminus T} (q')^2 \\ &= r(\frac{1}{r}p' + (1 - \frac{1}{r})q')^2 + (d - r)(q')^2. \end{aligned}$$

Computing P_2 for MGA to kRR. Unlike RPA to kRR, each fake user randomly selects a value from target set T instead of encoded data space $\mathcal{D}$, each item $v \in T$ is selected in each round with a probability $\frac{1}{r}$. Therefore, each fake user selects the same item v in both the first and second rounds with probability $\frac{1}{r} * \frac{1}{r}$. Since v could be any item in T,

$$\begin{aligned} P_2 &= \sum_{v \in T} Pr(v \text{ in } 1st \text{ round}) * Pr(v \text{ in } 2nd \text{ round}) \\ &= \sum_{v \in T} \frac{1}{r} * \frac{1}{r} = r * (\frac{1}{r} * \frac{1}{r}) = \frac{1}{r}. \end{aligned}$$

B. Percentage Estimation for OUE

For OUE, each genuine user encodes his value v to a d-dimensional binary vector B where the v -th bit is 1, *i.e.,* Encode(v) = B where B[v] = 1 and B[v'] = 0 for $v' \neq v$. Each non-zero entry (resp. zero entry) in B is kept unchanged with probability $p' = p_1$ or p_2 (resp. $1 - q'$) and flipped with probability $1 - p'$ (resp. $q' = q_1$ or q_2). Let B_1 and B_2 be binary vectors reported in the first and second rounds, respectively, $B_1 = B_2$ requires that two corresponding bits in the vectors are the same, and each bit i keeps the same in two rounds when both $B_1[i]$ and $B_2[i]$ are perturbed from $B[i]$ or kept as $B[i]$. For $i = v$, the probability of $B_1[i] = B_2[i]$ is $(p')^2 + (1 - p')^2$ where $(p')^2$ is for $B_1[i] = B_2[i] = B[i]$ and $(1 - p')^2$ is for $B_1[i] = B_2[i] \neq B[i]$. For $i \neq v$, the probability of $B_1[i] = B_2[i]$ is $(q')^2 + (1 - q')^2$. Therefore, the probability P_1 that the data collector collects the same value from a genuine user is

$$\begin{aligned} P_1 &= \sum_{i=v} Pr(B_1[i] = B_2[i]) * \sum_{i \neq v} Pr(B_1[i] = B_2[i]) \\ &= \sum_{i=v} \Big((p')^2 + (1-p')^2 \Big) * \sum_{i \neq v} \Big((q')^2 + (1-q')^2 \Big) \\ &= \Big((p')^2 + (d-1)(q')^2 \Big) \Big((q')^2 + (1-q')^2 \Big)^{d-1}. \end{aligned} \tag{17}$$

Observe that the dimension of a binary vector could be very large, so P_1 has a very small value and scarce observations of the same value are obtained in two rounds. To solve this problem, P_1 for any τ bits of B_1 and B_2 (denoted by $P_1(\tau)$) should be given. Since the τ bits are randomly selected from d bits, they contain v-th bit with probability $\frac{\tau}{d}$. As such,

$$\begin{aligned} P_1(\tau) &= \frac{\tau}{d} * \sum_{i=v} Pr(B_1[i] = B_2[i]) * \sum_{i \neq v} Pr(B_1[i] = B_2[i]) \\ &= +(1 - \frac{\tau}{d}) * \sum_{i \neq v} Pr(B_1[i] = B_2[i]) \\ &= \frac{\tau}{d} * ((p')^2 + (d-1)(q')^2)((q')^2 + (1-q')^2)^{\tau - 1} \\ &= +(1 - \frac{\tau}{d})((q')^2 + (1-q')^2)^{\tau}. \end{aligned}$$

Computing P_2 for RPA to OUE. Similar to the computation of P_2 for RPA to kRR, the fake user randomly selects a value from the encoded data space, the difference lie in that the encoded data space is now $\mathcal{D} = \{0, 1\}^d$. Thus, the fake user selects the same binary vector B in the first and second rounds with probability $\frac{1}{2^d} * \frac{1}{2^d}$,

$$P_2 = \sum_{B \in \mathcal{D}} Pr(B \text{ in } 1st \text{ round}) * Pr(B \text{ in } 2nd \text{ round})$$
$$= \sum_{B \in \mathcal{D}} \frac{1}{2^d} * \frac{1}{2^d} = 2^d * (\frac{1}{2^d} * \frac{1}{2^d}) = \frac{1}{2^d}.$$

Consequently, $P_2(\tau)$ is $2^\tau * (\frac{1}{2^\tau} * \frac{1}{2^\tau}) = \frac{1}{2^\tau}$.

Computing P_2 for RIA to OUE. For RIA to OUE, the fake user first selects a value from target set T and then adopts the same encoding and perturbation procedures used by the genuine users. Let B_1 and B_2 be binary vectors reported in the first and second rounds, respectively, $B_1 = B_2$ requires that two corresponding bits in the vectors are the same, and each bit i keeps the same in two rounds when both $B_1[i]$ and $B_2[i]$ are perturbed from $B[i]$ or kept as $B[i]$. For $i \in T$, the probability of $B_1[i] = B_2[i]$ should be divided into two cases. First, the probability of $B_1[i] = B_2[i] = 1$ is $(\frac{1}{r}p' + (1-\frac{1}{r})q')^2$. Second, for $B_1[i] = B_2[i] = 0$, the probability is $(\frac{1}{r}(1-p') + (1-\frac{1}{r})(1-q'))^2$. For $i \notin T$, the probability of $B_1[i] = B_2[i]$ is $(q')^2 + (1-q')^2$. Therefore, the probability P_2 that the data collector collects the same value from a fake user is

$$P_2 = \sum_{i \in T} Pr(B_1[i] = B_2[i]) * \sum_{i \notin T} Pr(B_1[i] = B_2[i])$$
$$= \sum_{i \in T} (\frac{1}{r}p' + (1-\frac{1}{r})q')^2 + (\frac{1}{r}(1-p') + (1-\frac{1}{r})(1-q'))^2$$
$$* \sum_{i \notin T} (q')^2 + (1-q')^2$$
$$= \left[(\frac{1}{r}p' + (1-\frac{1}{r})q')^2 + (\frac{1}{r}(1-p') + (1-\frac{1}{r})(1-q'))^2 \right]^r$$
$$* \left[(q')^2 + (1-q')^2 \right]^{d-r}.$$

The computation of P_2 for any chosen τ bits (*i.e.,* $P_2(\tau)$) depends on how many bits (denoted by j) are chosen from T. Therefore, $P_2(\tau)$ is the expectation over j, that is,

$$\sum_{j=0}^{r} \prod_{i=0}^{d} Pr(B_1[i] = B_2[i], j) = \sum_{j=0}^{r} Pr(j) \prod_{i=0}^{d} Pr(B_1[i] = B_2[i] | j)$$
$$= \sum_{j=0}^{r} Pr(j) \left[\prod_{i \in T} Pr(B_1[i] = B_2[i]|j) * \prod_{i \notin T} Pr(B_1[i] = B_2[i]|j) \right]$$
$$= \sum_{j=0}^{r} \frac{C_r^j C_{d-r}^{\tau-j}}{C_d^\tau} \left[\prod_{i \in T} Pr(B_1[i] = B_2[i]|j) * \prod_{i \notin T} Pr(B_1[i] = B_2[i]|j) \right]$$
$$= \sum_{j=0}^{r} \frac{C_r^j C_{d-r}^{\tau-j}}{C_d^\tau} \left[\left[(\frac{p'}{r} + \frac{r-1}{r}q')^2 + (\frac{1}{r}(1-p') + \frac{r-1}{r}(1-q'))^2 \right]^j * \left[(q')^2 + (1-q')^2 \right]^{\tau-j} \right]$$

where $C_{X_1}^{X_2}$ is the number of X_2-combinations of the set with X_1 elements.

Computing P_2 for MGA to OUE. Recall that each fake user attacks OUE with MGA by sending a binary vector where all target bits and randomly chosen $l = \lfloor p + (d-1)q - r \rfloor$ bits are 1, and the remaining bits are 0. Let B_1 and B_2 be binary vectors reported in the first and second rounds, respectively, whether or not $B_1 = B_2$ depends on the randomly chosen l bits in two different rounds. Since these l bits are randomly selected from all non-target bits, each possible l bits are selected with probability of $1/C_{d-r}^l$. Let $\mathbb{B}_\ell$ be all possible binary vectors where all target bits and additional l bits are 1,

$$P_2 = \sum_{B \in \mathbb{B}_\ell} Pr(B \text{ in } 1st \text{ round}) * Pr(B \text{ in } 2nd \text{ round})$$
$$= \sum_{B \in \mathbb{B}_\ell} \frac{1}{C_{d-r}^l} * \frac{1}{C_{d-r}^l} = C_{d-r}^l * \left(\frac{1}{C_{d-r}^l} * \frac{1}{C_{d-r}^l} \right) = \frac{1}{C_{d-r}^l}.$$

Observe that $P_2(\tau)$ for any chosen τ bits depends on how many bits (denoted by j) are chosen from T. In particular, if j bits are chosen from T, the values in the corresponding entries in B_1 and B_2 are always the same, thus, $P_2(\tau)$ is computed as follows:

$$\sum_{j=0}^{r} \prod_{i=0}^{d} Pr(B_1[i] = B_2[i], j) = \sum_{j=0}^{r} Pr(j) \prod_{i=0}^{d} Pr(B_1[i] = B_2[i] | j)$$
$$= \sum_{j=0}^{r} Pr(j) \left[\prod_{i \in T} Pr(B_1[i] = B_2[i]|j) * \prod_{i \notin T} Pr(B_1[i] = B_2[i]|j) \right]$$
$$= \sum_{j=0}^{r} \frac{C_r^j C_{d-r}^{\tau-j}}{C_d^\tau} \left[1 * \prod_{i \notin T} Pr(B_1[i] = B_2[i]|j) \right]$$
$$= \sum_{j=0}^{r} \frac{C_r^j C_{d-r}^{\tau-j}}{C_d^\tau} \left[(\frac{l}{d-r})^2 + (\frac{d-r-l}{d-r})^2 \right]^{\tau-j}$$

where $C_{X_1}^{X_2}$ is the number of X_2-combinations of the set with X_1 elements, and $\left(\frac{l}{d-r}\right)^2$ (resp. $\left(\frac{d-r-l}{d-r}\right)^2$) is the corresponding probability that the bit i is in (resp. not in) the randomly chosen l bits.

C. Percentage Estimation for OLH

We follow Section IV-B3 to set $p^* = \frac{e^\epsilon}{e^\epsilon + d' - 1}$, $q^* = \frac{1}{e^\epsilon + d' - 1}$, $p = p^* = \frac{e^\epsilon}{e^\epsilon + d' - 1}$ and $q = \frac{1}{d'}p^* + (1 - \frac{1}{d'})q^* = \frac{1}{d'}$. Similarly, p_1^*, q_1^*, p_1 and q_1 (resp. p_2^*, q_2^*, p_2 and q_2) are defined w.r.t. ϵ_1 (resp. ϵ_2). Also, let $p_{1,2}^* = p_1^*$ or p_2^*, $q_{1,2}^* = q_1^*$ or q_2^*. Recall that each genuine user reports his value v with OLH first randomly selects a hash function H from $\mathcal{H}$ to obtained $H(v)$, and then reports the key-value pair $\langle H, H(v)' \rangle$ where $H(v)'$ is perturbed (resp. kept) from $H(v)$ with probability $q_{1,2}^*$ (resp. $p_{1,2}^*$). As such, the genuine user reports the same value when 1) reporting the true value $H(v)$ in both the first and second rounds; 2) reporting the same value $H(v)'$ ($H(v)' \neq H(v)$). The probability P_1 that the data collector collects the same value from a genuine user is

$$P_1 = (p_{1,2}^*)^2 + (d'-1)(q_{1,2}^*)^2 \tag{18}$$

where $p_{1,2}^*$ (resp. $q_{1,2}^*$) is the probability that the genuine user reports the same value $H(v)$ (resp. $H(v)'$, $H(v)' \neq H(v)$).

Computing P_2 for RPA to OLH. For RPA to OLH, each fake user reports a key-value pair $\langle H, h \rangle$ where h is randomly selected from $\mathcal{D} = \{1, 2, ..., d'\}$ ($d' = e^\epsilon + 1$) without perturbation. As such, each item $h \in \mathcal{D}$ is selected in each round with probability $\frac{1}{d'}$. Therefore, each fake user selects the same item h in both the first and second rounds with probability $\frac{1}{d'} * \frac{1}{d'}$,

$$\begin{aligned} P_2 &= \sum_{h \in \mathcal{D}} Pr(h \text{ in } 1st \text{ round}) * Pr(h \text{ in } 2nd \text{ round}) \\ &= \sum_{h \in \mathcal{D}} \frac{1}{d'} * \frac{1}{d'} = d' * (\frac{1}{d'} * \frac{1}{d'}) = \frac{1}{d'}. \end{aligned}$$

Computing P_2 for RIA to OLH. Each fake user randomly selects a value from target set T and perturbs it to a value in $\mathcal{D} = \{1, 2, ..., d'\}$ where $d' = e^{\epsilon} + 1$, he reports the same value in $\mathcal{D} = \{1, 2, ..., d'\}$ in two rounds with probability $\frac{1}{d'} * p^*_{1,2} + (1 - \frac{1}{d'}) * q^*_{1,2}$,

$$\begin{aligned} P_2 &= \sum_{h \in \mathcal{D}} Pr(h \text{ in } 1st \text{ round}) * Pr(h \text{ in } 2nd \text{ round}) \\ &= \sum_{h \in \mathcal{D}} (\frac{1}{d'} * p^*_{1,2} + (1 - \frac{1}{d'}) * q^*_{1,2})^2 \\ &= d' * (\frac{1}{d'} * p^*_{1,2} + (1 - \frac{1}{d'}) * q^*_{1,2})^2. \end{aligned}$$

Computing P_2 for MGA to OLH. Since MGA to OLH is to seek for a hash function that maps all target values to the same encoded value, P_2 is always be 1. But in practice, there may be no such hash function. Hence, we resort to the hash function that results in the most targets to the same value. Suppose that there are maximally NUM targets hashed to the same value, $P_2 = (\frac{\text{NUM}}{r})^2 + (1 - \frac{\text{NUM}}{r})^2$.

VI. Experimental Evaluation

In this section, we investigate the performance of LDPGuard and report the key findings. LDPGuard is implemented with Python 3.10. All experiments are conducted on macOS Monterey 12.2.1 with Quad-Core Intel Core i5 4-Core CPU (2GHz) and 16GB of main memory.

A. Experimental Setup

Datasets. LDPGuard is evaluated on three datasets consisting of two real-world datasets (*i.e.,* IPUMS [62] and Fire [63]) and one synthetic dataset. The datasets are summarized in Table II and their details are listed as follows.

- **IPUMS [62].** IPUMS dataset consists of harmonized census and American Community Survey data over the years. We select the data of 2017 and the geographic variable “CITY” as the item set. The item set contains 102 items, each of which is a city code. The number of users is $389,894$.
- **Fire [63].** Fire calls-for-service contains all fire units’ responses to calls in San Francisco. Each record in the original dataset includes at least the call type, the call number, and the incident number at all relevant time intervals. We follow [25] to filter out other call types but “Alarms” and select the “Unit ID” as the item field to construct the Fire dataset, which has 262 items and covers 632,543 users.
- **Zipf.** Zipf is a synthetic dataset that follows the Zipf distribution. It includes 128 items and $500,000$ users.

TABLE II: Datasets

Dataset	Number of Items	Number of Users
IPUMS [62]	102	$389,894$
Fire [63]	262	$632,543$
Zipf	128	$500,000$

TABLE III: Parameter Settings for Experiments

Parameter	Symbol	Default Value
number of targets	r	10
percentage of fake users	β	0.05
privacy budget	ϵ	1.0
	ϵ_1	$\frac{\epsilon}{2}$
	ϵ_2	$\frac{\epsilon}{2}$

Baselines. We compare **LDPGuard** with two related countermeasures [25] against data poisoning attacks: (1) *Normalization* (denoted by **NORM**) that normalizes the estimated frequency of the items so that each estimated item frequency is non-negative and the overall summation of item frequencies is 1; (2) *Fake users detection* (denoted by **DETECT**) that detects possible fake users based on their reported values to alleviate their impact on the final results. We also compare it with the results where no countermeasure is adopted (denoted by **NO**), and both DETECT and NORM are applied (denoted by **BOTH**). Noted that the resulting frequency of LDPGuard is also normalized to avoid negative results.

Performance Metrics. We use **absolute gain**, *i.e.,* the absolute value of the frequency gain (**denoted by** $|Gain|$), to compare LDPGuard with baselines.[1] We use **NA** to denote the case that a countermeasure is not applicable for a data poisoning attack. We also compare LDPGuard with DETECT in terms of the estimated percentage (*i.e.,* $\widetilde{\beta}$) of fake users.

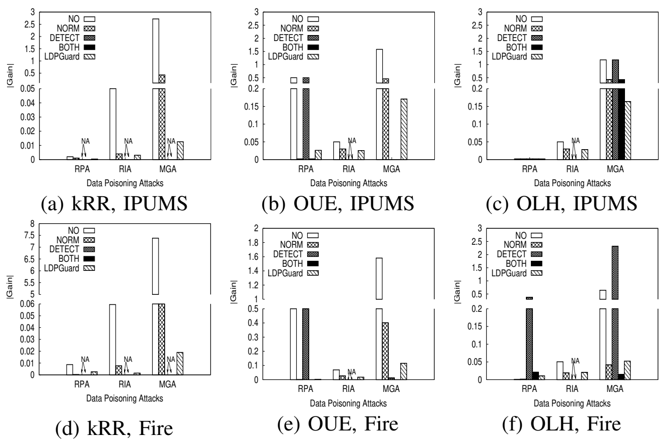
Parameter Settings. Unless otherwise stated, we use the default parameter values in Table III. In addition, we follow existing work [13], [25] to set $d' = e^{\epsilon} + 1$ for the OLH protocol.

B. Experimental Results

Exp 1: Performances of Frequency Estimation. We first compare LDPGuard with baselines on three datasets and plot the results of frequency estimation on IPUMS and Fire in Figure 2. [2] As depicted in Figure 2(a) and (d), LDPGuard always outperforms baselines for kRR regardless of data poisoning attacks. Compared to NO, NORM reduces $|Gain|$ to a relatively smaller value, but the impact is limited as NORM does not reduce $|Gain|$ directly but normalizes each estimated item frequency to a non-negative value. For kRR, both DETECT and BOTH cannot work and their results are denoted by NA. For OUE (Figure 2(b) and (e)) and OLH (Figure 2(c) and (f)), NORM is ineffective in the scenario

[1]One may argue that he can follow the existing work [25] to use frequency gain $Gain$ instead of its absolute value, but he may ignore the fact that the frequency of targets is largely distorted if $Gain$ is a negative number with a large absolute value, which results in unfair experimental comparison for countermeasures.

[2]Results of frequency estimation on Zipf are given in Appendix A.



(a) kRR, IPUMS (b) OUE, IPUMS (c) OLH, IPUMS

(d) kRR, Fire (e) OUE, Fire (f) OLH, Fire

Fig. 2: Absolute Gain $|Gain|$ on IPUMS and Fire.

TABLE IV: Percentage Estimation on IPUMS&Fire, $\beta = 0.05$

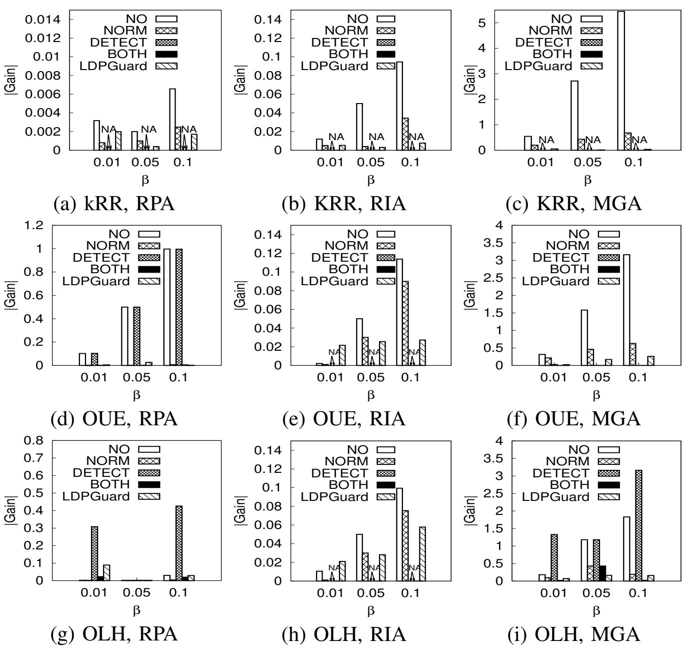
Protocols	Attacks	$\widetilde{\beta}$ (IPUMS)		$\widetilde{\beta}$ (Zipf)	
		DETECT	LDPGuard	DETECT	LDPGuard
KRR	RPA	—	0.117	—	0.593
	RIA	—	0.129	—	0.382
	MGA	—	0.050	—	0.049
OUE	RPA	7.0E-05	0.060	7.0E-05	0.042
	RIA	—	0.232	—	0.292
	MGA	0.050	0.044	0.050	0.041
OLH	RPA	0.029	0.013	0.025	0.081
	RIA	—	0.024	—	0.045
	MGA	0.346	0.048	0.267	0.040

where MGA attacks are deployed, and the reason is discussed above. DETECT and BOTH are effective in this scenario, as they can detect abnormal statistical patterns in the reported values generated by MGA. However, DETECT is ineffective for RPA because each fake user in RPA randomly selects perturbed values in the encoded space, and thus the perturbed values do not show meaningful statistical patterns. In addition, both DETECT and BOTH are not applicable for RIA since each fake user under RIA perturbs the value sampled from the target item set before sending it to a data collector, which results in the incapability of distinguishing fake users from genuine users. In contrast, LDPGuard can obtain relatively less $|Gain|$ than NORM for MGA. Compared to DETECT, it also has better performance for RIA. Moreover, LDPGuard is applicable regardless of LDP protocols and data poisoning attacks.

In summary, LDPGuard can effectively defend against data poisoning attacks and outperforms baselines in terms of absolute gain.

Exp 2: Performances of Percentage Estimation. We further compare LDPGuard with DETECT on IPUMS and Fire [3], and report the estimated percentages of fake users in Table IV. In this experiment, we set the true percentage of fake users to 0.05 and use "–" to denote the case where the countermeasure is not applicable. Observe that DETECT can accurately estimate the percentage of MGA to OUE because each reported value of OUE is a binary vector and the target

[3]Results of percentage estimation on Zipf are given in Appendix A.



(a) kRR, RPA (b) KRR, RIA (c) KRR, MGA

(d) OUE, RPA (e) OUE, RIA (f) OUE, MGA

(g) OLH, RPA (h) OLH, RIA (i) OLH, MGA

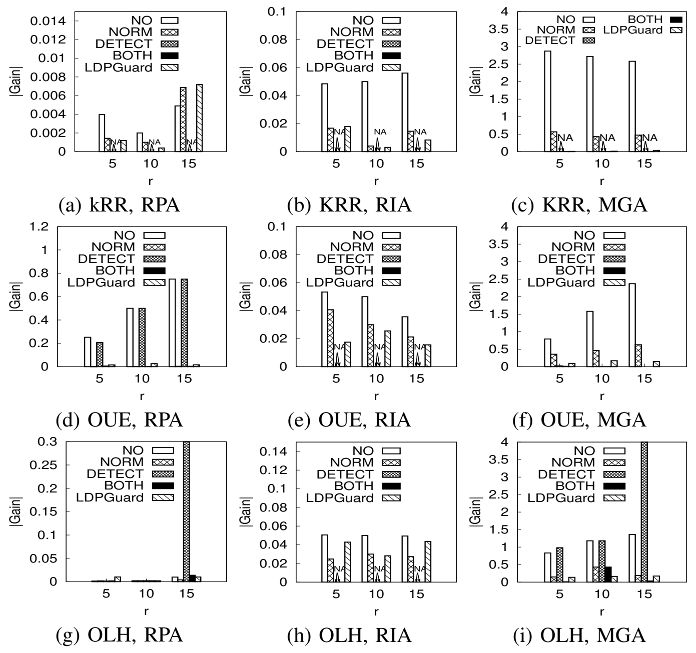
Fig. 3: Effect of β.

TABLE V: Percentage Estimation, β=0.01, 0.05, 0.1

Protocol,Attack	$\widetilde{\beta}$ (DETECT)			$\widetilde{\beta}$ (LDPGuard)		
	β=0.01	β=0.05	β=0.1	β=0.01	β=0.05	β=0.1
KRR,RPA	—	—	—	0.621	0.117	0.020
KRR,RIA	—	—	—	0.189	0.129	0.022
KRR,MGA	—	—	—	0.009	0.050	0.099
OUE,RPA	1E-05	7E-05	9E-05	0.011	0.060	0.143
OUE,RIA	—	—	—	0.204	0.232	0.379
OUE,MGA	0.010	0.050	0.100	0.014	0.044	0.088
OLH,RPA	0.019	0.029	0.032	0.009	0.013	0.104
OLH,RIA	—	—	—	0.006	0.024	0.109
OLH,MGA	0.122	0.346	0.370	0.008	0.267	0.094

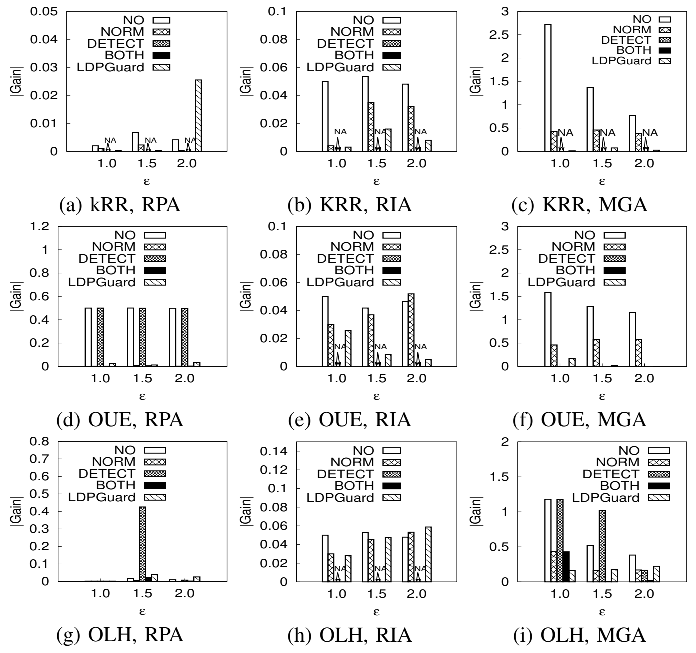
bits in the vector tends to expose abnormal statistical patterns. We also observe that it has limitations in estimating the percentage in the scenarios where there is no deterministic and meaningful statistical patterns. For example, for RPA to OUE, each fake user randomly selects a value from a very large encoding space $\mathcal{D}$ (*i.e.,* $\mathcal{D} = \{0,1\}^d$) and reports it without perturbation. This will result in each vector reported by fake users being with about $d/2$ non-zero bits, and further lead to the difficulty of distinguishing a fake user from a genuine user who reports the value with nearly $1/2 + (d-1)/(e+1)$ non-zero bits when $\epsilon = 1$. Furthermore, DETECT is not applicable for KRR as well as OUE (and OLH) to RIA. In contrast, LDPGuard is applicable for all LDP protocols and data poisoning attacks, although the estimated percentages under RPA and RIA where randomnesses are here and there are not very precise. In general, the estimated percentages by LDPGuard are close to the true percentages (*e.g.,* 0.05) of fake users. Hence, LDPGuard is able to accurately estimate the percentage of fake users.

Exp 3: Effect of β. Next, we vary the value of β and report the impact of β on IPUMS in Figure 3. Observe that $|Gain|$ increases with β if no countermeasure is adopted (*i.e.,* NO,



(a) kRR, RPA (b) KRR, RIA (c) KRR, MGA
(d) OUE, RPA (e) OUE, RIA (f) OUE, MGA
(g) OLH, RPA (h) OLH, RIA (i) OLH, MGA

Fig. 4: Effect of r.



(a) kRR, RPA (b) KRR, RIA (c) KRR, MGA
(d) OUE, RPA (e) OUE, RIA (f) OUE, MGA
(g) OLH, RPA (h) OLH, RIA (i) OLH, MGA

Fig. 5: Effect of ϵ.

TABLE VI: Percentage Estimation, r=5, 10, 15

Protocol,Attack	$\widetilde{\beta}$ **(DETECT)**			$\widetilde{\beta}$ **(LDPGuard)**		
	r**=5**	r**=10**	r**=15**	r**=5**	r**=10**	r**=15**
KRR,RPA	—	—	—	0.055	0.117	0.055
KRR,RIA	—	—	—	0.602	0.129	0.255
KRR,MGA	—	—	—	0.051	0.050	0.050
OUE,RPA	0.003	7.06E-05	4.87E-06	0.134	0.060	0.031
OUE,RIA	—	—	—	0.177	0.232	0.101
OUE,MGA	0.051	0.050	0.050	0.044	0.044	0.046
OLH,RPA	0	0.029	0.176	0.010	0.013	0.068
OLH,RIA	—	—	—	0.021	0.025	0.154
OLH,MGA	0.149	0.346	0.621	0.049	0.048	0.042

TABLE VII: Percentage Estimation, ϵ=1, 1.5, 2

Protocol,Attack	$\widetilde{\beta}$ **(DETECT)**			$\widetilde{\beta}$ **(LDPGuard)**		
	ϵ**=1**	ϵ**=1.5**	ϵ**=2**	ϵ**=1**	ϵ**=1.5**	ϵ**=2**
KRR,RPA	—	—	—	0.117	0.301	0.271
KRR,RIA	—	—	—	0.129	0.054	0.199
KRR,MGA	—	0.050	—	0.050	0.048	0.050
OUE,RPA	7.07E-05	5.36E-05	7.07E-05	0.060	0.035	0.050
OUE,RIA	—	—	—	0.232	0.105	0.065
OUE,MGA	0.050	0.050	0.050	0.044	0.050	0.051
OLH,RPA	0.029	0.049	0	0.013	0.057	0.015
OLH,RIA	—	—	—	0.025	0.025	0.069
OLH,MGA	0.346	0.179	0.058	0.048	0.046	0.038

Figure 3) because more fake users are introduced to distort the true data distribution. Although NORM can decrease $|Gain|$, it has little impact on the RIA and MGA (*e.g.,* Figure 3(b), (c), (e), (f)). DETECT is effective in defending against MGA to OUE (see Figure 3(f)) and has good performance against MGA to OLH when combined with NORM (*i.e.,* BOTH, Figure 3(i)). However, DETECT and BOTH are not applicable for kRR protocols as well as RIA attacks. For MGA to OUE, as shown in Figure 3(f), LDPGuard is comparable to DETECT. Moreover, it outperforms baselines in other scenarios regardless of the value of β. We also report the estimated percentage of fake users (*i.e.,* $\widetilde{\beta}$) in Table V and observe similar results. In the following experiments, we set $\beta = 0.05$.

Exp 4: Effect of r. We also study the performance of LDPGuard with different numbers of targets (*i.e.,* r) on IPUMS. First, for RPA to kRR (Figure 4(a)), we observe that $|Gain|$ of all these countermeasures are close to zero (exactly, less than 0.008). This indicates that RPA has very limited impacts on kRR even if no countermeasure is adopted. For other attacks to kRR, LDPGuard outperforms baselines no matter what the value of r is. Second, observe that DETECT is ineffective in RPA to OUE but very effective in MGA to OUE (Figure 4(d)-(f)). The reason is as discussed above. For OLH, DETECT is incapable of decreasing $|Gain|$ of MGA attacks when $r = 15$ because MGA can only find a hash function among the 1,000 random seeds that hashes a subset of target items to the same value for each fake user. But in general, our LDPGuard can outperforms baselines regardless of r. Moreover, it also outperforms baselines in terms of estimated percentages as shown in Table VI. We set $r = 10$ in the following experiments unless specified.

Exp 5: Effect of ϵ. We finally vary privacy budget ϵ on IPUMS and report the results in Figure 5. As depicted in Figure 5(c), (f) and (i), $|Gain|$ under MGA decreases as the privacy budget ϵ increases. This is because for larger ϵ, genuine users inject less noise to their data, while the values reported by fake users are irrelevant to the privacy budget. As such, fake users contribute less to the frequencies of target items and lead to a decrease in $|Gain|$. In contrast, $|Gain|$ under other attacks are less sensitive to ϵ (*e.g.,* Figure 5(b), (e) and (h)). For example, genuine and fake users under RIA need to perturb their data, and hence $|Gain|$ under RIA fluctuates in a small range. We also report the estimated percentages with different privacy budgets in Table VII and observe that LDPGuard outperforms

DETECT in most scenarios and is comparable to DETECT for MGA to OUE. Hence, LDPGuard outperforms baselines regardless of privacy budgets.

VII. Conclusions

In this paper, we focus on the problem of defending against data poisoning attacks to local differential privacy protocols. We present a novel framework called LDPGuard to this end. In particular, LDPGuard first adopts the two-round collection strategy to accurately estimate the percentage of fake users. Then it utilizes the estimated percentage to provide adversarial schemes to defend against various data poisoning attacks. LDPGuard supports state-of-the-art LDP protocols (kRR, OUE, and OLH) for frequency estimation and can effectively defend against existing data poisoning attacks. Experimental study on real-world and synthetic datasets demonstrates the superiority of LDPGuard compared to existing techniques. As part of our future work, we will explore data poisoning attacks and countermeasures to other data types, such as graph data.

References

[1] Technical Report. Available at: https://github.com/TechReport2023/LDPGuard/blob/main/LDPGuard-TR.pdf.

[2] Brendan Avent, Aleksandra Korolova, David Zeber, Torgeir Hovden, and Benjamin Livshits. BLENDER: Enabling local search with a hybrid differential privacy model. In *USENIX Security*, 2017.

[3] Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Guha Thakurta. Practical locally private heavy hitters. In *NeurIPS*, 2017.

[4] Raef Bassily and Adam Smith. Local, private, efficient protocols for succinct histograms. In *STOC*, 2015.

[5] Graham Cormode, Tejas Kulkarni, and Divesh Srivastava. Marginal release under local differential privacy. In *SIGMOD*, 2018.

[6] John C Duchi, Michael I Jordan, and Martin J Wainwright. Local privacy and statistical minimax rates. In *FOCS*, 2013.

[7] *Ú*lfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*, 2014.

[8] Jinyuan Jia and Neil Zhenqiang Gong. Calibrate: Frequency estimation and heavy hitter identification with local differential privacy via incorporating prior knowledge. In *INFOCOM*, 2019.

[9] Peter Kairouz,Keith Bonawitz, and Daniel Ramage. Discrete distribution estimation under local privacy. In *ICML*, 2016.

[10] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Extremal mechanisms for local differential privacy. In *NeurIPS*, 2014.

[11] Zhan Qin, Yin Yang, Ting Yu, Issa Khalil, Xiaokui Xiao, and Kui Ren. Heavy hitter estimation over set-valued data with local differential privacy. In *CCS*, 2016.

[12] Xuebin Ren, Chia-Mu Yu, Weiren Yu, Shusen Yang, Xinyu Yang, Julie A McCann, and S Yu Philip. LoPub: High-dimensional crowdsourced data publication with local differential privacy. *TIFS*, 2018.

[13] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. Locally differentially private protocols for frequency estimation. In *USENIX Security*, 2017.

[14] Tianhao Wang, Bolin Ding, Jingren Zhou, Cheng Hong, Zhicong Huang, Ninghui Li, and Somesh Jha. Answering multi-dimensional analytical queries under local differential privacy. In *SIGMOD*, 2019.

[15] Tianhao Wang, Ninghui Li, and Somesh Jha. Locally differentially private frequent itemset mining. In *S&P*, 2018.

[16] Tianhao Wang, Ninghui Li, and Somesh Jha. Locally differentially private heavy hitter identification. *TDSC*, 2019.

[17] Tianhao Wang, Milan Lopuhaä-Zwakenberg, Zitao Li, Boris Skoric, and Ninghui Li. Locally differentially private frequency estimation with consistency. In *NDSS*, 2020.

[18] Qingqing Ye, Haibo Hu, Xiaofeng Meng, Huadi Zheng, Kai Huang, Chengfang Fang, Jie Shi. PrivKVM*: Revisiting Key-Value Statistics Estimation with Local Differential Privacy. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[19] Zhikun Zhang, Tianhao Wang, Ninghui Li, Shibo He, and Jiming Chen. Calm: Consistent adaptive local marginal for marginal release under local differential privacy. In *CCS*, 2018.

[20] Apple Differential Privacy Team. Learning with privacy at scale. *Machine Learning Journal*, 2017.

[21] Qingqing Ye, Haibo Hu, Xiaofeng Meng, Huadi Zheng. PrivKV: Key-value data collection with local differential privacy. In *S&P*, 2019.

[22] Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, Xiaokui Xiao. LF-GDPR: A Framework for Estimating Graph Metrics with Local Differential Privacy. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[23] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. In *NeurIPS*, 2017.

[24] Nguyên, T Thông, Xiaokui Xiao, et al. Collecting and analyzing data from smart device users with local differential privacy. *arXiv preprint arXiv:1606.05053*, 2016.

[25] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Data poisoning attacks to local differential privacy protocols. In *USENIX Security*, 2021.

[26] Sweeney, Latanya. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2002.

[27] Machanavajjhala Ashwin, Kifer Daniel, Gehrke Johannes, et al. l-diversity: Privacy beyond k-anonymity. In *ICDE*, 2006.

[28] Ninghui Li, Tiancheng Li, and Venkatasubramanian Suresh. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, 2007.

[29] Chang Zhao, Lei Zou, and Feifei Li. Privacy preserving subgraph matching on large graphs in cloud. In *SIGMOD*, 2016.

[30] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, 2006.

[31] Qingqing Ye, Haibo Hu, Ninghui Li, Xiaofeng Meng, Huadi Zheng, Haotian Yan. Beyond Value Perturbation: Local Differential Privacy in the Temporal Setting. In *INFOCOM*, 2021.

[32] Privacy and efficiency guaranteed social subgraph matching. Kai Huang, Haibo Hu, Shuigeng Zhou, Jihong Guan, Qingqing Ye, and Xiaofang Zhou. *The VLDB Journal*, 2021.

[33] Cynthia DDwork, Aaron Roths. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 2014.

[34] Ninghui Li, Min Lyu, Dong Su, Weining Yang. Differential Privacy: From Theory to Practice. *Synthesis Lectures on Information Security, Privacy, and Trust*, 2016.

[35] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 1965.

[36] Shiva Prasad Kasiviswanathan et al. 2011. What can we learn privately? *SIAM Journal on Computing*, 2011.

[37] Justin Hsu, Sanjeev Khanna, and Aaron Roth. Distributed private heavy hitters. In *Automata, Languages, and Programming*, 2012.

[38] Giulia Fanti, Vasyl Pihur, Úlfar Erlingsson. Building a RAPPOR with the Unknown: Privacy-Preserving Learning of Associations and Data Dictionaries. In *PoPETS*, 2016.

[39] Albert Cheu, Adam Smith, and Jonathan Ullman. Manipulation attacks in local differential privacy. In *S&P*, 2021.

[40] Scott Alfeld, Xiaojin Zhu, and Paul Barford. Data poisoning attacks against autoregressive models. In *AAAI*, 2016.

[41] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *ICML*, 2012.

[42] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to byzantine robust federated learning. In *USENIX Security*, 2020.

[43] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *S&P*, 2018.

[44] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. In *IEEE Access*, 2019.

[45] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *NeurIPS*, 2018.

[46] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *NDSS*, 2018.

[47] Minghong Fang, Neil Zhenqiang Gong, and Jia Liu. Influence function based data poisoning attacks to top-n recommender systems. In *WWW*, 2020.

[48] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. Poisoning attacks to graph-based recommender systems. In *ACSAC*, 2018.

[49] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. In *NeurIPS*, 2016.

[50] Guolei Yang, Neil Zhenqiang Gong, and Ying Cai. Fake co-visitation injection attacks to recommender systems. In *NDSS*, 2017.

[51] Mehran Mozaffari-Kermani, Susmita Sur-Kolay, Anand Raghunathan, and Niraj K Jha. Systematic poisoning attacks on and defenses for machine learning in healthcare. *IEEE journal of biomedical and health informatics*, 2014.

[52] Qiang Cao, Xiaowei Yang, Jieqi Yu, and Christopher Palow. Uncovering large groups of active malicious accounts in online social networks. In *CCS*, 2014.

[53] George Danezis and Prateek Mittal. Sybilinfer: Detecting sybil nodes using social networks. In *NDSS*, 2009.

[54] Neil Zhenqiang Gong, Mario Frank, and Prateek Mittal. Sybilbelief: A semi-supervised learning approach for structure-based sybil detection. *TIFS*, 2014.

[55] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting spammers on social networks. In *ACSAC*, 2010.

[56] Binghui Wang, Jinyuan Jia, and Neil Zhenqiang Gong. Graph-based security and privacy analytics via collective classification with joint weight learning and propagation. In *NDSS*, 2019.

[57] Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y Zhao. You are how you click: Clickstream analysis for sybil detection. In *USENIX Security*, 2013.

[58] Haifeng Yu, Haifeng Yu, Michael Kaminsky, Phillip B Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. In *SIGCOMM*, 2006.

[59] Dong Yuan, Yuanli Miao, Neil Zhenqiang Gong, Zheng Yang, Qi Li, Dawn Song, Qian Wang, and Xiao Liang. Detecting fake accounts in online social networks at the time of registrations. In *CCS*, 2019.

[60] Yann Collet. xxHash: Extremely fast hash algorithm. https://github.com/Cyan4973/xxHash, 2016.

[61] Kurt Thomas, Damon McCoy, Chris Grier, Alek Kolcz, and Vern Paxson. Trafficking fraudulent accounts: The role of the underground market in twitter spam and abuse. In *USENIX Security*, 2013.

[62] Ruggles Steven, Flood Sarah, Goeken Ronald, Grover Josiah, Meyer Erin, Pacas Jose, and Sobek Matthew. IPUMS USA: Version 9.0 [dataset]. minneapolis, mn: Ipums, 2019. https://doi.org/10.18128/D010.V9.0, 2019.

[63] San francisco fire department calls for service. http://bit.ly/336sddL, 2019.

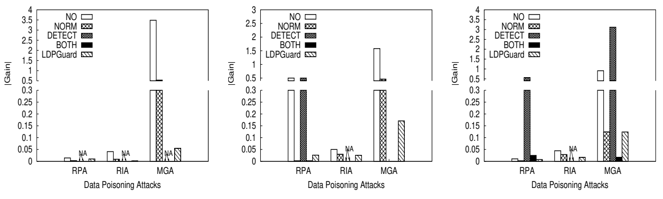
Appendix

A. Proofs

Proof of Lemma 1.

$$\begin{aligned}\mathbb{E}[\widetilde{f}_v] &= \mathbb{E}\left[\frac{\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}_{\text{SUPPORT}(y_i)}(v) - q}{p-q}\right]\\ &= \frac{\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}\left[\mathbb{I}_{\text{SUPPORT}(y_i)}(v)\right] - q}{p-q}\\ &= \frac{\frac{1}{N} * N(f_v(p-q)+q) - q}{p-q} = f_v\end{aligned}$$

$\mathbb{E}[\widetilde{f}_v] = f_v$, $\widetilde{f}_v = \frac{\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}_{\text{SUPPORT}(y_i)}(v)-q}{p-q}$ is an unbiased estimator of the true frequency f_v of the item v.



(a) kRR (b) OUE (c) OLH

Fig. 6: Absolute Gain $|Gain|$ on Zipf.

TABLE VIII: Percentage Estimation on Zipf, $\beta = 0.05$

Protocols	Attacks	$\widetilde{\beta}$ DETECT	$\widetilde{\beta}$ LDPGuard
KRR	RPA	—	0.084
	RIA	—	0.176
	MGA	—	0.049
OUE	RPA	5.7E-05	0.222
	RIA	—	0.442
	MGA	0.050	0.099
OLH	RPA	0.039	0.023
	RIA	—	0.040
	MGA	0.346	0.047

Proof of Lemma 2. Without loss of generality, we prove it by setting $\mathcal{C} = 2$,

$$\begin{aligned}\forall y \in Range(\mathcal{A}_2): &\frac{Pr[\mathcal{A}_2(\mathcal{A}_1(v), v) = y]}{Pr[\mathcal{A}_2(\mathcal{A}_1(v), v') = y]}\\ &= \frac{\sum_{s\in\mathcal{S}} Pr[\mathcal{A}_1(v) = s]Pr[\mathcal{A}_2(s, v) = y]}{\sum_{s\in\mathcal{S}} Pr[\mathcal{A}_1(v') = s]Pr[\mathcal{A}_2(s, v') = y]}\\ &\leq \sum_{s\in\mathcal{S}}\frac{Pr[\mathcal{A}_1(v) = s]Pr[\mathcal{A}_2(v) = y]}{Pr[\mathcal{A}_1(v') = s]Pr[\mathcal{A}_2(v') = y]}\\ &\leq e^{\epsilon_1}e^{\epsilon_2} = e^{\epsilon}\end{aligned}$$

where $\mathcal{S} = Range(\mathcal{A}_1)$ and $\epsilon_1 + \epsilon_2 = \epsilon$.

Proof of Theorem 1.

$$\begin{aligned}\mathbb{E}[\widetilde{\beta}] &= \mathbb{E}\left[\frac{(N+M)P_1 - CNT}{(N+M)(P1-P2)}\right]\\ &= \frac{(N+M)P_1 - \mathbb{E}[CNT]}{(N+M)(P1-P2)}\\ &= \frac{(N+M)P_1 - (NP_1 + MP_2)}{(N+M)(P1-P2)} = \frac{M}{N+M} = \beta\end{aligned}$$

$\mathbb{E}[\widetilde{\beta}] = \beta$, hence, $\widetilde{\beta} = \frac{(N+M)P_1 - CNT}{(N+M)(P1-P2)}$ is an unbiased estimator of the percentage of fake users.

B. Additional Experimental Results

Exp 6: Performances of Frequency Estimation on Zipf. We plot results of frequency estimation on Zipf in Figure 6. Similar to the results on IPUMS and Fire, DETECT performs well for MGA to OLH but is incapable of defending against attacks to kRR and RIA to OUE. Furthermore, NORM has a limited impact on reducing $|Gain|$. In contrast, LDPGuard obtains a relatively smaller $|Gain|$ in most scenarios. This observation further demonstrates the effectiveness of LDPGuard in alleviating the effect of data poisoning attacks.

Exp 7: Performances of Percentage Estimation on Zipf. We also compare LDPGuard with DETECT on Zipf and report the estimated percentages of fake users in Table VIII. Similar to the results on IPUMS and Fire, DETECT can accurately estimate the percentage of MGA to OUE where abnormal statistical patterns may exist, but underperforms for RPA to OUE. In contrast, LDPGuard is applicable for all LDP protocols and data poisoning attacks, and its estimated percentages are close to the true percentages of fake users in most scenarios. This observation demonstrates that LDPGuard is able to accurately estimate the percentage of fake users.