Augmented Shuffle Differential Privacy Protocols for Large-Domain Categorical and Key-Value Data

Takao Murakami
ISM/AIST/RIKEN AIP
tmura@ism.ac.jp

Yuichi Sei
UEC
seiuny@uec.ac.jp

Reo Eriguchi
AIST
eriguchi-reo@aist.go.jp

***Abstract*—Shuffle DP (Differential Privacy) protocols provide high accuracy and privacy by introducing a shuffler who randomly shuffles data in a distributed system. However, most shuffle DP protocols are vulnerable to two attacks: collusion attacks by the data collector and users and data poisoning attacks. A recent study addresses this issue by introducing an augmented shuffle DP protocol, where users do not add noise and the shuffler performs random sampling and dummy data addition. However, it focuses on frequency estimation over categorical data with a small domain and cannot be applied to a large domain due to prohibitively high communication and computational costs.**

In this paper, we fill this gap by introducing a novel augmented shuffle DP protocol called the FME (Filtering-with-Multiple-Encryption) protocol. Our FME protocol uses a hash function to filter out unpopular items and then accurately calculates frequencies for popular items. To perform this within one round of interaction between users and the shuffler, our protocol carefully communicates within a system using multiple encryption. We also apply our FME protocol to more advanced KV (Key-Value) statistics estimation with an additional technique to reduce bias. For both categorical and KV data, we prove that our protocol provides computational DP, high robustness to the above two attacks, accuracy, and efficiency. We show the effectiveness of our proposals through comparisons with twelve existing protocols.

I. Introduction

DP (Differential Privacy) [1] has been widely adopted by industry [2], [3], [4] and government agencies [5] to perform data analysis while protecting individual privacy. DP has been originally studied in the central model, where a single data collector holds all users' personal data and adds noise to the statistical results. Although central DP enables accurate data analysis, all personal data can be leaked by data breach incidents [6]. LDP (Local DP) [7], [8], [9] addresses this issue by adding noise to each user's personal data before sending it to the data collector. However, LDP suffers from low accuracy, as it needs to add a lot of noise to each user's personal data.

The shuffle model of DP [10], [11], [12], [13], [14], [15], [16] has been recently studied to achieve high accuracy and privacy. It assumes a distributed system involving an intermediate server called the *shuffler* and typically works as follows. Each user adds noise to her input data and sends an encrypted version of the noisy data to the shuffler. The shuffler randomly shuffles the noisy data and sends it to the data collector. Finally, the data collector decrypts the shuffled data. Under the assumption that the shuffler does not collude with the data collector, the random shuffling amplifies privacy. Specifically, DP strongly protects privacy when the privacy budget ε is small (e.g., $\varepsilon \leq 1$), and the shuffling significantly reduces ε. Thus, the shuffle model achieves the same value of ε as the local model with less noise, i.e., higher accuracy.

However, most shuffle DP protocols have the following two vulnerabilities. First, they are vulnerable to *collusion attacks by the data collector and users* (or *collusion with users*) [17], [18]. Specifically, the data collector can obtain noisy data of some users by colluding with them or compromising their accounts. In this case, the data collector can reduce the number of shuffled values and thereby reduce the effect of shuffling. As a result, the actual value of ε can be significantly increased (e.g., from around 1 to 8; see Section IV-B). Second, most shuffle protocols are vulnerable to *data poisoning attacks* [18], [19], [20], which inject fake users and carefully craft data sent from the fake users to manipulate the statistical results.

A recent study [18] addresses these two issues by introducing an *augmented* shuffle model, where the shuffler performs additional operations, such as random sampling and adding dummies, before shuffling. Specifically, [18] proposes the LNF (Local-Noise-Free) protocol, in which each user sends her (encrypted) input data to the shuffler without adding noise, and the shuffler performs random sampling, adding dummies, and shuffling. The key idea of this protocol is to prevent malicious users' behavior by adding noise on the shuffler side rather than the user side. The protocol can also be easily implemented using any PKE (Public Key Encryption) scheme, such as RSA and ECIES. It is shown in [18] that the LNF protocol is robust to both collusion with users and poisoning from users while also providing higher accuracy than other shuffle protocols.

Unfortunately, the LNF protocol in [18] is limited to a simple frequency estimation task with a small domain and cannot be applied to data analysis with a large domain. Specifically, both the communication and computational costs

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.231124
www.ndss-symposium.org

of the LNF protocol are linear in the number d of items and are prohibitively large when d is large. For example, our experimental results show that the LNF protocol would require about 100 Terabits of communications and 3 years of run time to calculate a frequency distribution when $d = 10^9$ (see Section VIII). The LNF protocol also cannot be applied to a more complicated task, such as KV (Key-Value) statistics estimation [21], [22], [23], with large d for the same reason.

In this work, we fill this gap by introducing a novel protocol for large-domain data. We first focus on frequency estimation over categorical data with a large domain and consider a simple extension of the LNF protocol that reduces the domain size using a hash function common to all users. We analyze the theoretical properties of this protocol and show that it suffers from low accuracy due to hash collision. We also show that this issue cannot be addressed by introducing a different hash function for each user (or each user group assigned in advance), as dummies need to be added for each hash function.

To achieve high accuracy and efficiency, we propose a novel protocol called the *FME (Filtering-with-Multiple-Encryption) protocol*. Below, we explain its technical overview.

A. Technical Overview

Our Protocol. In our FME protocol, we use a hash function to filter out unpopular items with low (or zero) frequencies rather than to calculate frequencies. Then, we accurately calculate frequencies for the selected items. This can be realized by introducing a two-round interaction between users and the shuffler, where each user sends her hash value for in the first round and her input data or a symbol "$\perp$" representing an unselected item in the second round. However, the two-round interaction significantly reduces the usability, as each user must respond to the shuffler twice. Moreover, it needs *synchronization* [24] in that the shuffler must wait for all users' responses before shuffling in each round. Thus, the two-round protocol would not be practical for many systems.

We overcome this issue by *replacing unselected items with $\perp$ on the data collector side* and *introducing multiple encryption* [25]. Specifically, our FME protocol achieves *one round of interaction* between users and the shuffler as follows. Each user sends her hash value and input data simultaneously to the shuffler. The shuffler performs augmented shuffling for the hash values and sends them and the corresponding input data to the data collector. The data collector filters items based on the shuffled hash values, replaces unselected items in the input data with $\perp$, and sends them back to the shuffler. Finally, the shuffler performs augmented shuffling for the input data, and the data collector calculates frequencies from the shuffled input data. Note that the input data are communicated between the shuffler and the data collector *three times* in this protocol. Thus, a lot of information can be leaked by comparing them. To prevent this leakage, we use multiple encryption for the input data and have the shuffler and data collector decrypt the input data each time they receive them. We rigorously analyze the privacy of our protocol and prove it achieves computational DP [26], [27]. We also show that it achieves high robustness against collusion and poisoning attacks, accuracy, and efficiency. Furthermore, we optimize the range of the hash function in terms of efficiency.

Then, we apply our FME protocol to KV statistics estimation, where each user has KV pairs (e.g., movies and ratings) and the goal is to estimate the frequency and mean value for each key (item). For this data type, we propose an additional technique called *TKV-FK (Transforming KV Pairs and Filtering Keys)*, which transforms KV pairs into one-dimensional data and filters the data at a key level to reduce bias in the estimates. We extensively evaluate our proposals and show they are effective for both categorical and KV data.

Technical Novelty. Our main technical novelty lies in a technique that carefully uses multiple encryption to provide (computational) DP within one round of interaction for users, and in the rigorous proof of DP. To our knowledge, we are the first to use multiple encryption to reduce the number of rounds while providing DP (see Section II for details). In addition, our FME protocol does not use multiple encryption merely to implement onion routing [28]. More importantly, multiple encryption enables the data collector to *remove ciphertexts corresponding to unpopular items without revealing this fact to the shuffler* by replacing them with ciphertexts that encrypt $\perp$. We prove the DP guarantee of such a protocol by reducing it to the security of a PKE scheme. Our work also includes other technical contributions, such as the optimization of the hash range and a technique to reduce bias (i.e., TKV-FK).

Furthermore, we prove many theorems that are not simple extensions of [18]. Examples include our theoretical results for KV data (e.g., Theorems 11 and 12), as [18] does not deal with KV data. In particular, the existing robustness analysis for KV data [19] assumes that the number of each user's KV pairs does not exceed the padding length [23], which may not hold in practice. Theorem 11 removes this assumption and shows the robustness of our protocol in a general setting where the number of KV pairs can exceed the padding length. Another example is Theorem 8, which analyzes the communication cost of our FME protocol based on the size of single, double, or triple ciphertexts and a bound on the expected number of selected items. This theorem is also new and serves as a basis for optimizing the hash range. We also present a new theorem (Theorem 5) that shows the low accuracy of a simple extension of the LNF protocol [18] using the common hash function under the 2-wise independence assumption [29].

B. Our Contributions

We make the following contributions:

- We propose a novel augmented shuffle protocol called the FME protocol for large-domain categorical and KV data. Our protocol achieves high privacy, robustness, accuracy, and efficiency within one round of interaction for users using multiple encryption. For KV data, we propose an additional technique called TKV-FK to reduce bias.
- We demonstrate the effectiveness of our proposals through theoretical analysis and extensive experiments

that compare ours with twelve state-of-the-art shuffle protocols (eight for categorical and four for KV data).

Compared to the LNF protocol [18], our protocol reduces the communication cost from about 100 Terabits to 260 Gigabits and the run time from about 3 years to 1 day ($d = 10^9$). Note that d is much smaller than 10^9 in most practical applications (e.g., there are $d = 8 \times 10^6$ census blocks in the US; Amazon has $d = 6 \times 10^8$ products in total [30]), in which case the communication and computational costs are also smaller. We also show that our protocol can be applied to a system with large-scale users and items (e.g., item rating system [31]) by introducing user sampling. The proofs of all statements are given in our full paper [32]. Our code is available in [33].

II. Related Work

Shuffle DP. The shuffle model of DP can be divided into two models: a *pure shuffle model* and an *augmented shuffle model* [18]. Most existing work (e.g., [11], [12], [13], [14], [15], [16], [34], [35], [36]) assumes the pure shuffle model, where the shuffler performs only shuffling. However, this model is vulnerable to collusion with users and data poisoning. This vulnerability is inevitable in this model because genuine users need to add noise to their input data; in this case, the data collector can increase ε by obtaining noisy data of some users, and fake users can effectively manipulate the statistical results by *not* adding noise to their input data [19].

Beimel *et al.* [37] show a pure shuffle protocol for general tasks, which achieves accuracy comparable to the central one. However, their protocol requires two rounds of interaction for users whereas ours requires only one round of interaction.

Some protocols, including Google's Prochlo [10], assume the augmented shuffle model, where the shuffler performs additional operations, e.g., randomized thresholding [10], random sampling [38], and adding dummies [17]. A recent study [18] proposes the LNF protocol that does not add noise on the user side and shows that it is robust to both collusion and poisoning attacks. However, this protocol cannot be applied to large-domain data. We address this issue by introducing a novel protocol with multiple encryption.

Collusion/Poisoning Attacks. Some studies [17], [18] show that pure shuffle protocols are vulnerable to collusion with users. To address this issue, Wang *et al.* [17] add dummies uniformly at random from the domain of noisy data on the shuffler side. However, their protocol still suffers from the increase in ε by collusion with users, as shown in Appendix E-B.

Data poisoning attacks have been studied in various data types, e.g., categorical [19], [39], KV [20], numerical [40], and set-valued data [41]. Although these attacks assume the local model, they can also be applied to the shuffle model. Defenses have also been studied for categorical or KV data. The defenses in [42], [43], [44], [45] introduce multiple rounds for users and reduce usability. The defenses in [19], [46] have limited effectiveness [18]. We also show that the defense in [20] has limited effectiveness in Appendix E-B.

Cryptographic Protocols. Multiple encryption has been studied in the field of cryptography [25], [47], [48]. Its applications include key-insulated encryption [47], onion routing [28], mix-net [49], and an instant messenger [50]. To our knowledge, we are the first to use multiple encryption to provide DP for distributed systems within one round of interaction for users.

Finally, a DP protocol using secure multi-party computation is proposed in [51] to calculate a frequency distribution. Their protocol requires a PKE scheme with a homomorphic property, whereas ours can use any PKE scheme based on a wider class of assumptions. Moreover, the protocol in [51] cannot be applied to more advanced KV statistics estimation, as (i) each user may hold multiple KV pairs, and (ii) the data collector needs to estimate both frequency and mean for each key. In contrast, our protocol can be used to estimate KV statistics.

III. Preliminaries

A. Notations

Let $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, $\mathbb{N}$, and $\mathbb{Z}_{\geq 0}$ be the sets of real numbers, non-negative real numbers, natural numbers, and non-negative integers, respectively. Let $n \in \mathbb{N}$ be the number of users, and $d \in \mathbb{N}$ be the number of items. For $i \in [n]$ $(= \{1, 2, \ldots, n\})$, let u_i be the i-th user. Let $\mathcal{X}$ be the space of input data, and $x_i \in \mathcal{X}$ be the input data of user u_i. Sections V and VI focus on frequency estimation over categorical data, whereas Section VII focuses on frequency and mean estimation over KV data. Below, we introduce the notations for each case.

Categorical Data. For categorical data, we follow [18], [19], [52] and represent an item as an integer from 1 to d. Each user's input data x_i ($i \in [n]$) is an item, i.e., $\mathcal{X} = [d]$. The data collector estimates the (relative) frequency $f_i \in [0, 1]$ for each item $i \in [d]$. The frequency f_i is given by $f_i = \frac{1}{n} \sum_{j=1}^{n} \mathbb{1}_{x_j = i}$, where $\mathbb{1}_{x_j = i}$ takes 1 if $x_j = i$ and 0 otherwise. We denote the estimate of f_i by $\hat{f}_i \in \mathbb{R}$. Let $\mathbf{f} = (f_1, \ldots, f_d)$ and $\hat{\mathbf{f}} = (\hat{f}_1, \ldots, \hat{f}_d)$. The goal for the data collector is to calculate $\hat{\mathbf{f}}$ as close as possible to $\mathbf{f}$ under DP.

KV Data. For KV data, we follow [20], [21], [22], [23] and represent a key (item) as an integer from 1 to d and a numerical value as a real value between -1 and 1. Note that we can assume that the values are in the range $[-1, 1]$ without loss of generality, as numerical values can be transformed into $[-1, 1]$. In this use case, each user's input data x_i ($i \in [n]$) is a set of KV pairs $\langle k, v \rangle$, where $k \in [d]$ and $v \in [-1, 1]$. Note that each user has at most one KV pair per key, i.e., $\mathcal{X} = \bigcup_{i=1}^{d} ([d] \times [-1, 1])^i$. The data collector estimates the frequency $\Phi_i \in [0, 1]$ and the mean value $\Psi_i \in [-1, 1]$ for each key $i \in [d]$. They are given by

$$\Phi_i = \frac{1}{n} \sum_{j=1}^{n} \mathbb{1}_{\langle k, \cdot \rangle \in x_j}, \quad \Psi_i = \frac{1}{n\Phi_i} \sum_{j \in [n], \langle k, v \rangle \in x_j} v,$$

where $\mathbb{1}_{\langle k, \cdot \rangle \in x_j}$ takes 1 if x_j includes key k and 0 otherwise. Let $\hat{\Phi}_i$ (resp. $\hat{\Psi}_i$) $\in \mathbb{R}$ be the estimates of Φ_i (resp. Ψ_i). Let $\mathbf{\Phi} = (\Phi_1, \ldots, \Phi_d)$, $\mathbf{\Psi} = (\Psi_1, \ldots, \Psi_d)$, $\hat{\mathbf{\Phi}} = (\hat{\Phi}_1, \ldots, \hat{\Phi}_d)$, and $\hat{\mathbf{\Psi}} = (\hat{\Psi}_1, \ldots, \hat{\Psi}_d)$. The goal is to calculate $\hat{\mathbf{\Phi}}$ and $\hat{\mathbf{\Psi}}$ as close as possible to $\mathbf{\Phi}$ and $\mathbf{\Psi}$, respectively, under DP.

We also summarize our notations in Table I of Appendix A.

B. Differential Privacy

In this work, we use DP [1] and its computational version called CDP (Computational DP) [26], [27] as privacy notions:

Definition 1 ((ε, δ)-DP/CDP)**.** *Let $\varepsilon \in \mathbb{R}_{\geq 0}$ and $\delta \in [0, 1]$. We say a randomized algorithm $\mathcal{M}$ with domain $\mathcal{X}^n$ provides* (ε, δ)-DP *if for any neighboring databases $D = (x_1, \ldots, x_n) \in \mathcal{X}^n$ and $D' = (x'_1, \ldots, x'_n) \in \mathcal{X}^n$ that differ on one entry and any $S \subseteq \mathrm{Range}(\mathcal{M})$,*

$$\Pr[\mathcal{M}(D) \in S] \leq e^{\varepsilon} \Pr[\mathcal{M}(D') \in S] + \delta. \tag{1}$$

We also say $\mathcal{M}$ provides (ε, δ)-CDP (Computational DP) *if for any attacker $\mathcal{A}$ running in time polynomial in $\gamma \in \mathbb{N}$,*

$$\Pr[\mathcal{A}(\mathcal{M}(D)) = 1] \leq e^{\varepsilon} \Pr[\mathcal{A}(\mathcal{M}(D')) = 1] + \delta + \mathsf{negl}(\gamma), \tag{2}$$

where negl *is a function that approaches 0 faster than the reciprocal of any polynomial in γ.*

In our work, γ coincides with the security parameter of a PKE scheme. (ε, δ)-CDP (resp. DP) can be used for shuffle DP protocols with (resp. without) PKE schemes. In shuffle protocols, the shuffler or the data collector can be an attacker $\mathcal{A}$. In either case, all messages the attacker receives during the protocols are outputs of $\mathcal{M}$. Note that $\varepsilon \geq 5$ is unsuitable in most applications [53]. δ should be much smaller than $\frac{1}{n}$ [1].

We also introduce LDP [7], [54], which can be used as a building block for pure shuffle DP protocols:

Definition 2 (ε-LDP)**.** *Let $\varepsilon \in \mathbb{R}_{\geq 0}$ and $\delta \in [0, 1]$. We say a randomized algorithm $\mathcal{R}$ with domain $\mathcal{X}$ provides* ε-LDP *if for any input values $x, x' \in \mathcal{X}$ and any $S \subseteq \mathrm{Range}(\mathcal{R})$,*

$$\Pr[\mathcal{R}(x) \in S] \leq e^{\varepsilon} \Pr[\mathcal{R}(x') \in S]. \tag{3}$$

Examples of LDP mechanisms $\mathcal{R}$ include the GRR [17], RAPPOR [2], OUE [52], and OLH [52].

C. Pure Shuffle Model

Below, we explain the pure shuffle model assumed in most existing shuffle DP protocols. Assume that the data collector has a private key and publishes the corresponding public key. In the pure shuffle protocols, each user u_i adds noise to her data x_i using a randomized algorithm $\mathcal{R}$, encrypts it using the public key, and sends the encrypted version of $\mathcal{R}(x_i)$ to the shuffler. The shuffler randomly shuffles (encrypted) noisy data $\mathcal{R}(x_1), \ldots, \mathcal{R}(x_n)$ and sends them to the data collector. The data collector decrypts the shuffled data using the private key.

Under the assumption that the shuffler does not collude with the data collector, the data collector obtains only the shuffled data. The shuffled data provides (ε, δ)-DP, where $\varepsilon = g(n, \delta)$ and g is a monotonically decreasing function of n and δ. For example, when $\mathcal{R}$ provides LDP, g is expressed using the state-of-the-art privacy amplification result [16] as follows:

Theorem 1 (Privacy amplification result in [16])**.** *Let $\varepsilon_0 \in \mathbb{R}_{\geq 0}$ and $D = (x_1, \ldots, x_n) \in \mathcal{X}^n$. Let $\mathcal{R} : \mathcal{X} \to \mathcal{Y}$ be a randomized algorithm. Let $\mathcal{M}_S : \mathcal{X}^n \to \mathcal{Y}^n$ be a pure shuffle algorithm that takes D as input, samples a uniform random permutation π over $[n]$, and outputs $\mathcal{M}_S(D) = (\mathcal{R}(x_{\pi(1)}), \ldots, \mathcal{R}(x_{\pi(n)}))$. If $\mathcal{R}$ provides ε_0-LDP, then for any $\delta \in [0, 1]$, $\mathcal{M}_S$ provides (ε, δ)-DP with $\varepsilon = g(n, \delta)$, where*

$$g(n, \delta) = \ln\left(1 + (e^{\varepsilon_0} - 1)\frac{4\sqrt{2\ln(4/\delta)}}{\sqrt{(e^{\varepsilon_0} + 1)n}} + \frac{4}{n}\right) \tag{4}$$

if $\varepsilon_0 \leq \ln(\frac{n}{8\ln(2/\delta)} - 1)$ and $g(n, \delta) = \varepsilon_0$ otherwise.

By (4), ε is much smaller than ε_0 in the LDP mechanism when n is large. In multi-message protocols [34], [35], [36], $g(n, \delta)$ is different from (4). See [34], [35], [36] for details.

D. Communication Cost and Accuracy

We use the following measures for the communication cost and accuracy (i.e., utility):

Communication Cost. Let C_{U-S} (resp. C_{S-D}) $\in \mathbb{R}_{\geq 0}$ be the expected number of bits sent from users to the shuffler (resp. from the shuffler to the data collector). Then, the expected total number C_{tot} of bits sent from one party to another is given by $C_{tot} = C_{U-S} + C_{S-D}$. We use C_{tot} as a measure of the communication cost.

Accuracy. For categorical data, we follow [17], [18], [52], [55] and measure the expected squared error $\mathbb{E}[(\tilde{f}_i - f_i)^2]$ of the estimate $\tilde{f}_i$ for each item $i \in [d]$. If the estimate $\tilde{f}_i$ is unbiased, then $\mathbb{E}[(\tilde{f}_i - f_i)^2]$ is equal to the variance $\mathbb{V}[\tilde{f}_i]$. Similarly, we follow [23] and measure the expected squared errors $\mathbb{E}[(\tilde{\Phi}_i - \Phi_i)^2]$ and $\mathbb{E}[(\tilde{\Psi}_i - \Psi_i)^2]$ ($i \in [d]$) for KV data.

IV. Collusion and Poisoning Attacks

In this section, we explain collusion and poisoning attacks in detail. Section IV-A defines our threat model and clarifies why we focus on these attacks. Sections IV-B and IV-C explain collusion and poisoning attacks, respectively.

A. Threat Model

We assume that anyone except a single user (victim), including the shuffler, the data collector, and other users, can be an attacker who attempts to infer the input data of the victim. We assume that input data $x_1, \ldots, x_n$ are independent (which is necessary for DP guarantees [56]) and that the attacker can obtain input data of all users except the victim as background knowledge.

We also assume that the data collector can collude with some users except the victim (or compromise their accounts) to obtain their noisy data sent to the shuffler. In addition, the attacker can inject fake users and send an arbitrary message from each fake user to manipulate the statistics. The former and latter attacks are collusion and poisoning attacks, respectively. We focus on these attacks because attacks by malicious users pose a threat in practice – [57] shows that the attacker can inject a large number of fake users in practical systems. In particular, collusion attacks are threatening because ε can be increased from about 1 to 8 in the pure shuffle protocols when 10% of users collude with the data collector; see Section IV-B. It is also difficult to know the number of colluding users (and hence the actual value of ε) in these protocols.

As with most existing shuffle protocols, we assume that the shuffler and the data collector are semi-honest and do not collude with each other. Our protocol cannot achieve DP or accurate estimates when these servers deviate from the protocol (e.g., when the shuffler leaks data sent from users or does not perform shuffling; when the data collector alters the estimates). One way to address this issue is to ensure data confidentiality and enforce the servers to follow the protocol via a legally binding contract or a TEE (Trusted Execution Environment) [10], [58]. Another way is to provide security against the servers who deviate from the protocol (i.e., malicious security) by using MPC (Multi-Party Computation) [59]. We leave the use of the TEE or MPC for future work.

B. Collusion with Users

Wang *et al.* [17] point out that ε in the pure shuffle model can be increased when the data collector colludes with some users to obtain their noisy data sent to the shuffler.

Specifically, let $\Omega \subset [n]$. Assume that the data collector colludes with users $\{u_i | i \in \Omega\}$ and obtains their noisy data sent to the shuffler. Then, the privacy budget ε for the remaining users is increased from $g(n, \delta)$ to $g(n - |\Omega|, \delta)$ [18]. For example, if $n = 10^6$, $|\Omega| = 10^5$, and $\delta = 10^{-12}$ in Theorem 1, then ε can be increased from 1.1 to 8.3.

Ideally, the value of ε should not be increased even if some users share their data sent to the shuffler with the data collector. Below, we formally define such *robustness to collusion with users*. We first introduce Ω-neighboring databases [60]:

Definition 3 (Ω-Neighboring databases)**.** *Let $\Omega \subset [n]$. We say two databases $D = (x_1, \ldots, x_n) \in \mathcal{X}^n$ and $D' = (x'_1, \ldots, x'_n) \in \mathcal{X}^n$ are* Ω-neighboring *if they differ on one entry whose index i is* not *in Ω, i.e., $x_i \neq x'_i$ for some $i \notin \Omega$ and $x_j = x'_j$ for any $j \neq i$.*

Ω-neighboring databases consider an attacker who colludes with users $\{u_i | i \in \Omega\}$. Based on this, we can define the robustness to collusion with users as follows:

Definition 4 (Robustness to collusion with users)**.** *Let $\mathcal{M}$ be a shuffle protocol that provides (ε, δ)-DP (or CDP). For $i \in [n]$, let ν_i be data sent from user u_i to the shuffler in $\mathcal{M}$. For $\Omega \subset [n]$, let $\mathcal{M}_\Omega$ be a protocol that takes a database $D \in \mathcal{X}^n$ as input and outputs $\mathcal{M}_\Omega(D) = (\mathcal{M}(D), (\nu_i)_{i \in \Omega})$. We say $\mathcal{M}$ is* robust to collusion with users *if for any $\Omega \subset [n]$, any Ω-neighboring databases D and D', and any $S \subseteq \mathrm{Range}(\mathcal{M}_\Omega)$, $\mathcal{M}_\Omega$ also satisfies (1) (or (2)), i.e., if ε and δ are not increased by collusion with users.*

The data ν_i sent from user u_i to the shuffler depends on the protocol $\mathcal{M}$. For example, in the baseline protocol in Section V-B, $\nu_i = h(x_i)$, where h is a hash function. In our protocol in Section VI, $\nu_i = (x_i, h(x_i))$.

Unfortunately, pure shuffle protocols cannot provide the robustness in Definition 4, as they need to add noise on the user side. The robustness in Definition 4 can be achieved by introducing the augmented shuffle model and adding noise on the shuffler side, as shown in Sections V to VII.

C. Data Poisoning Attacks

Following [19], [20], we consider the following targeted attacks as data poisoning attacks. Let $\mathcal{T} \subseteq [d]$ be the set of target items. We assume that the attacker injects $n' \in \mathbb{N}$ fake users; there are $n + n'$ users in total, including n genuine users. Each fake user can send an arbitrary message to the shuffler. This is called the output poisoning attack [40]. For $i \in [n']$, let m_i be a message sent from the i-th fake user. Let $\mathbf{m} = (m_1, \ldots, m_{n'})$ be the messages of n' fake users.

Categorical Data. For categorical data, the attacker attempts to increase the estimates for the target items $\mathcal{T}$ (i.e., to promote $\mathcal{T}$) as much as possible. Formally, let $\hat{f}'_i \in \mathbb{R}$ be the estimate of f_i after poisoning. Then, the attacker's *overall gain* is defined as $G_f(\mathbf{m}) = \sum_{i \in \mathcal{T}} \mathbb{E}[\hat{f}'_i - \hat{f}_i]$ [19]. The attacker's goal is to maximize $G_f(\mathbf{m})$.

Let $G_f^{\mathsf{max}} = \max_{\mathbf{m}} G_f(\mathbf{m})$ be the maximum value of $G_f(\mathbf{m})$. Cao *et al.* [19] propose the MGA (Maximal Gain Attack) that crafts the messages $\mathbf{m}$ to achieve G_f^{max}. We use the maximum gain G_f^{max} to measure the robustness to data poisoning attacks in categorical data.

KV Data. For KV data, the attacker attempts to maximize the frequency and mean estimates for the target items $\mathcal{T}$ (i.e., to promote $\mathcal{T}$). Let $\hat{\Phi}'_i$ (resp. $\hat{\Psi}'_i$) $\in \mathbb{R}$ be the estimate of Φ_i (resp. Ψ_i) after poisoning. Then, the *frequency gain* and the *mean gain* are given by $G_\Phi(\mathbf{m}) = \sum_{i \in \mathcal{T}} \mathbb{E}[\hat{\Phi}'_i - \hat{\Phi}_i]$ and $G_\Psi(\mathbf{m}) = \sum_{i \in \mathcal{T}} \mathbb{E}[\hat{\Psi}'_i - \hat{\Psi}_i]$, respectively [20]. The goal is to maximize $G_\Phi(\mathbf{m})$ and $G_\Psi(\mathbf{m})$ simultaneously.

Let $G_\Phi^{\mathsf{max}} = \max_{\mathbf{m}} G_\Phi(\mathbf{m})$ and $G_\Psi^{\mathsf{max}} = \max_{\mathbf{m}} G_\Psi(\mathbf{m})$. Wu *et al.* [20] propose the M2GA (Maximal Gain Attack) that crafts $\mathbf{m}$ to achieve G_Φ^{max} and G_Ψ^{max} simultaneously. We use the maximum frequency gain G_Φ^{max} and the maximum mean gain G_Ψ^{max} as robustness measures in KV data.

V. Baseline Protocols

In this section, we present two baseline protocols for categorical data and explain why they are unsuitable for large-domain data. Section V-A describes the LNF (Local-Noise-Free) protocol in [18] and explains that it suffers from prohibitively high communication and computational costs. Section V-B introduces the CH (Common Hash) protocol to address this issue and shows that it suffers from low accuracy.

A. Local-Noise-Free Protocol

Protocol. Fig. 1 shows the overview of the LNF protocol. We show an algorithmic description of the LNF protocol in Algorithm 2 of Appendix B-A. We denote the LNF protocol by $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$. $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ has a *dummy-count distribution* $\mathcal{D}$ over $\mathbb{Z}_{\geq 0}$ with mean $\mu \in \mathbb{R}_{\geq 0}$ and variance $\sigma^2 \in \mathbb{R}_{\geq 0}$ and a *sampling probability* $\beta \in [0, 1]$ as parameters.

The LNF protocol $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ is simple and works as follows (we omit the encryption/decryption process). First, each user u_i ($i \in [n]$) sends her input value $x_i \in [d]$ without adding noise. Then, the shuffler performs three operations: random sampling, adding dummies, and shuffling. Specifically, the shuffler randomly selects each input value with probability

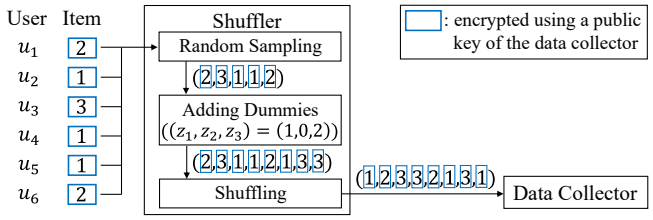

Fig. 1. Overview of the LNF protocol ($n = 6$, $d = 3$). In this example, the shuffler discards input data of u_2 and adds $(z_1, z_2, z_3) = (1, 0, 2)$ dummies.

β. For each item $i \in [d]$, the shuffler randomly generates z_i from the dummy-count distribution $\mathcal{D}$ ($z_i \sim \mathcal{D}$) and adds z_i dummy values. Let $y_1, \ldots, y_{\tilde{n}} \in [d]$ be the selected input values and dummies, where $\tilde{n} \in \mathbb{Z}_{\geq 0}$ is the total number of these values. The shuffler samples a random permutation π over $[\tilde{n}]$ and sends $y_{\pi(1)}, \ldots, y_{\pi(\tilde{n})}$ to the data collector. In Fig. 1, $y_{\pi(1)}, \ldots, y_{\pi(\tilde{n})} = (1, 2, 3, 3, 2, 1, 3, 1)$ ($\tilde{n} = 8$).

After receiving the shuffled values $y_{\pi(1)}, \ldots, y_{\pi(\tilde{n})}$, the data collector calculates their histogram. Specifically, the data collector calculates a count (absolute frequency) $\tilde{c}_i \in \mathbb{Z}_{\geq 0}$ for each item $i \in [d]$ from $y_{\pi(1)}, \ldots, y_{\pi(\tilde{n})}$. In the example of Fig. 1, $(\tilde{c}_1, \tilde{c}_2, \tilde{c}_3) = (3, 2, 3)$. Finally, the data collector calculates an unbiased estimate $\hat{f}_i$ of f_i as $\hat{f}_i = \frac{1}{n\beta}(\tilde{c}_i - \mu)$ and outputs $\hat{\mathbf{f}} = (\hat{f}_1, \cdots, \hat{f}_d)$.

Theoretical Properties. $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ provides DP and is robust to both data poisoning and collusion attacks if a simpler mechanism called the *binary input mechanism* provides DP:

Definition 5 (Binary input mechanism)**.** *Let $\mathcal{D}$ be a dummy-count distribution over $\mathbb{Z}_{\geq 0}$. The binary input mechanism $\mathcal{M}_{\mathcal{D},\beta} : \{0, 1\} \to \mathbb{Z}_{\geq 0}$ with parameters $\mathcal{D}$ and $\beta \in [0, 1]$ takes binary data $x \in \{0, 1\}$ and outputs*

$$\mathcal{M}_{\mathcal{D},\beta}(x) = \alpha x + z,$$

where α and z are random variables that follow the Bernoulli distribution Ber(β) and $\mathcal{D}$, respectively ($\alpha \sim$ Ber(β), $z \sim \mathcal{D}$).

Theorem 2 ([18])**.** *If $\mathcal{M}_{\mathcal{D},\beta}$ provides $(\frac{\varepsilon}{2}, \frac{\delta}{2})$-DP, then $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ provides (ε, δ)-CDP[1] and is robust to collusion with users.*

Theorem 3 ([18])**.** *Let $\lambda = \frac{n'}{n+n'}$ and $f_{\mathcal{T}} = \sum_{i \in \mathcal{T}} f_i$. $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ provides the following robustness against poisoning attacks:*

$$G_f^{\mathsf{max}} = \lambda(1 - f_{\mathcal{T}}). \tag{5}$$

G_f^{max} in (5) does not depend on ε and is smaller than existing pure shuffle protocols [19], [34], [35], [36]. Thus, Theorems 2 and 3 mean that $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ provides DP and is robust to collusion and data poisoning attacks if $\mathcal{M}_{\mathcal{D},\beta}$ provides DP.

In addition, $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ achieves the following accuracy:

[1]Note that $\mathcal{M}_{\mathcal{D},\beta}$ provides CDP, as it uses a PKE scheme. Specifically, it provides (ε, δ)-DP for the data collector and $(0, 0)$-CDP for the shuffler. In addition, ε and δ in $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ are doubled. This is because neighboring data $x, x' \in \{0, 1\}$ in $\mathcal{M}_{\mathcal{D},\beta}$ differ by 1 in one dimension, whereas neighboring databases $D, D' \in [d]^n$ in $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ differ by 1 in two dimensions.

Theorem 4 ([18])**.** *For any $i \in [d]$, $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ outputs an unbiased estimate (i.e., $\mathbb{E}[\hat{f}_i] = f_i$) and achieves the following expected squared error:*

$$\mathbb{E}[(\hat{f}_i - f_i)^2] = \frac{f_i(1-\beta)}{n\beta} + \frac{\sigma^2}{n^2\beta^2}. \tag{6}$$

The error in (6) decreases as σ^2 decreases and β increases.

Dummy-Count Distribution $\mathcal{D}$. Examples of $\mathcal{D}$ providing DP include the binomial distribution and the asymmetric geometric distribution. In particular, [18] shows that the asymmetric geometric distribution provides higher accuracy than the binomial distribution and existing pure shuffle protocols when $\beta = 1$ and achieves pure DP ($\delta = 0$) when $\beta = 1 - e^{-\varepsilon/2}$.

Limitations. The drawback of $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ is that it cannot be applied to large-domain data due to its high communication and computational costs. Specifically, the communication costs of $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ are: $C_{U-S} = \tau n$, $C_{S-D} = \tau(\beta n + \mu d)$, and

$$C_{tot} = \tau((1 + \beta)n + \mu d), \tag{7}$$

where τ is the size of each ciphertext. Thus, C_{tot} can be expressed as $O(n + d)$. Note that we can effectively reduce n by randomly sampling users, as shown in our experiments. However, we cannot reduce d by randomly sampling items without losing accuracy. Moreover, the mean μ of the dummy-count distribution $\mathcal{D}$ is much larger than 1, e.g., $\mu = 108$ in the asymmetric geometric distribution ($\beta = 1$, $\varepsilon = 1$, $\delta = 10^{-12}$). Thus, C_{tot} in (7) can be prohibitively large when d is large. For example, when $n = 10^5$, $d = 10^9$, $\mu = 108$, $\beta = 1$, and the 2048-bit RSA is used, $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ needs $C_{tot} = 220$ Terabits.

Similarly, the computational cost of $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ is $O(n + d)$ and can be prohibitively large when d is large. For example, $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ requires about 3 years when $d = 10^9$ in our experiments.

B. Common Hash Protocol

Protocol. The LNF protocol $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ cannot be applied to categorical data with large domain size d, as both the communication and computational costs are $O(n + d)$. A natural approach to improving the efficiency would be to reduce the domain size using a hash function. Our baseline, called the CH (Common Hash) protocol, simply uses a hash function common to all users. We denote this baseline by $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$.

Specifically, let $\mathcal{H}$ be a universal hash function family whose domain is $[d]$ and whose range is $[b]$ ($b \leq d$). The CH protocol $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ randomly selects a hash function $h : [d] \to [b]$ from $\mathcal{H}$, applies h to input values $x_1, \ldots, x_n$, and runs the LNF protocol $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$. In other words, $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ runs Algorithm 2 (lines 1-11) with inputs $(h(x_1), \ldots, h(x_n))$, b, $\mathcal{D}$, and β. Then, the data collector obtains $(\tilde{c}_1, \ldots, \tilde{c}_b)$, where $\tilde{c}_j$ ($j \in [b]$) is a count of a hash value j calculated from the shuffled values. Finally, the data collector calculates an unbiased estimate $\hat{f}_i$ ($i \in [d]$) of f_i as $\hat{f}_i = \frac{b}{n\beta(b-1)}(\tilde{c}_{h(i)} - \frac{n\beta}{b} - \mu)$.

Theoretical Properties. We show DP and the robustness of $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ in Appendix B-B. Below, we analyze the communication cost of $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$. Since $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ reduces the domain size from d to

b via the hash function h, the communication costs of $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ are: $C_{U-S} = \tau n$, $C_{S-D} = \tau(\beta n + \mu b)$, and

$$C_{tot} = \tau((1+\beta)n + \mu b), \tag{8}$$

where τ is the size of each ciphertext. C_{tot} in (8) can be expressed as $O(n+b)$. Similarly, the computational cost of $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ is $O(n+b)$. Therefore, the CH protocol $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ is much more efficient than the LNF protocol $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ when $b \ll d$.

Limitations. Unfortunately, the CH protocol $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ is also unsuitable for large-domain data because it suffers from low accuracy due to hash collision.

Specifically, assume that the hash function h is 2-wise independent; i.e., for any $i_1, i_2 \in [d]$ and $j_1, j_2 \in [b]$, $\Pr(h(i_1) = j_1 | h(i_2) = j_2) = \Pr(h(i_1) = j_1) = \frac{1}{b}$. For example, let $p \in [d, 2d)$ be a prime, $a_1 \in [p-1]$, and $a_0 \in [p]$. Then, a hash function h defined by $h(x) = ((a_1 x + a_0) \bmod p) \bmod b$ is (almost) 2-wise independent [29]. This hash function is used in [34] and is also used in our experiments. Under this assumption, the accuracy of $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ can be quantified as follows:

Theorem 5. *For any $i \in [d]$, $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ with a 2-wise independent hash function h outputs an unbiased estimate (i.e., $\mathbb{E}[\hat{f}_i] = f_i$) and achieves the following expected squared error:*

$$\mathbb{E}[(\hat{f}_i - f_i)^2] = \frac{b^2}{n^2\beta^2(b-1)^2}\left(n f_i \beta(1-\beta) + \sigma^2 + \omega\right), \tag{9}$$

where

$$\omega = \sum_{j=1, j\neq i}^{d} \left(\frac{(n^2 f_j^2 \beta^2 + n f_j \beta(1-\beta))(b-1)}{b^2} + \frac{n f_j \beta(1-\beta)}{b^2} \right).$$

The first and second terms in (9) are almost the same as the squared error in (6). However, the third term in (9) is introduced by hash collision and is very large when b is small. For example, when $\beta = 1$ and $b = n$, the squared error in (6) is $\frac{\sigma^2}{n^2}$. In contrast, the squared error in (9) is about $\frac{\sigma^2+n}{n^2}$ in the worst case. In other words, the squared error is increased from $O(n^{-2})$ to $O(n^{-1})$ due to the hash collision. We also show that $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ suffers from low accuracy in our experiments.

In Appendix B-C, we describe other baselines than $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ and $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ and explain that they also suffer from low accuracy.

VI. Filtering-with-Multiple-Encryption Protocol

In this section, we propose a novel DP protocol called the *FME (Filtering-with-Multiple-Encryption) protocol* to address the issues in the baselines. Section VI-A explains its technical motivation and overview. Section VI-B describes the details of our FME protocol. Section VI-C analyzes its theoretical properties. Section VI-D proposes a method to optimize the range b of the hash function in our FME protocol.

A. Technical Motivation and Overview

Technical Motivation. As explained in Section V-B, hash values $h(x_1), \ldots, h(x_n)$ cannot be used for frequency estimation due to hash collision. However, they can be used to *filter out* unpopular items with low (or zero) frequencies. This is helpful in large-domain data because the frequency distribution $\mathbf{f}$ is *sparse*; e.g., most items have a count of zero when $n \ll d$.

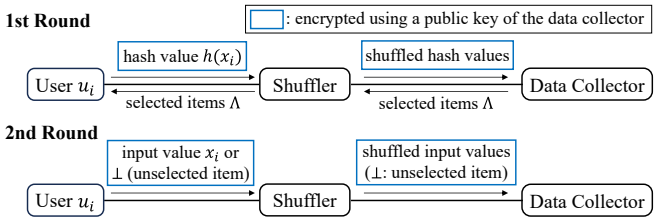

Fig. 2. Two-round protocol using a hash function h.

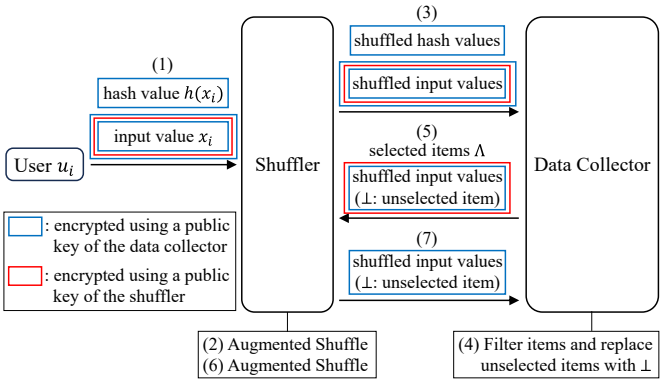

Fig. 3. Overview of our FME protocol. It reduces the number of rounds between users and the shuffler in Fig. 2 by introducing multiple encryption.

After reducing the input domain by filtering, we can efficiently use the LNF protocol to obtain high accuracy.

Based on this, we can consider a *two-round* protocol shown in Fig. 2. In the first round, each user u_i sends her hash value $h(x_i)$ to the shuffler. The shuffler performs augmented shuffling (i.e., sampling, adding dummies to hash values, and shuffling) and sends the shuffled hash values to the data collector. The data collector filters out unpopular items and sends a set $\Lambda \subseteq [d]$ of popular items to the users. In the second round, each user u_i replaces her input value x_i with $\perp$ representing an *unselected item* if $x_i \notin \Lambda$. Then, the parties run the LNF protocol for selected items Λ and $\perp$; i.e., $[d]$ is replaced with $\Lambda \cup \{\perp\}$.

The two-round protocol provides high accuracy for popular items Λ, as it uses the LNF protocol for them. In addition, the second round can be efficiently performed since the domain is restricted to $\Lambda \cup \{\perp\}$. Similar approaches have been taken in [22], [61] – they introduce the first round to find popular items and the second round to calculate statistics for them.

However, the two-round protocol is not desirable for many practical systems, as it significantly requires a great deal of effort from users and synchronization, as described in Section I. We are interested in providing high accuracy and communication/computation efficiency *without* introducing two rounds of interaction between users and the shuffler.

Overview. Our key idea is to remove the two rounds of interaction between users and the shuffler in Fig. 2 by *replacing unselected items with $\perp$ on the data collector side* and *introducing multiple encryption*. Fig. 3 shows the overview of our FME protocol that embodies our key idea.

For ease of explanation, we begin with our FME protocol without encryption. In our protocol, each user u_i sends her

hash value $h(x_i)$ and input value x_i simultaneously. After receiving each user's pair $\langle h(x_i), x_i \rangle$ ($i \in [n]$), the shuffler performs augmented shuffling for the pairs, where dummies are added to hash values (i.e., in the form of $\langle h(i), \perp \rangle$ for $i \in [b]$), and sends the shuffled pairs to the data collector. Then, the data collector selects a set Λ of popular items based on the shuffled hash values and replaces unselected items $x_i \notin \Lambda$ in the shuffled pairs with $\perp$. After that, the data collector sends Λ and the shuffled input values (including $\perp$) back to the shuffler. Note that the shuffled input values are the same as the ones the shuffler receives in the second round in Fig. 2 (except that dummies are added to $\perp$). Thus, as with Fig. 2, the shuffler performs augmented shuffling for Λ and $\perp$ and sends shuffled input values to the data collector.

The above protocol achieves *one round of interaction* between users and the shuffler by replacing unselected items with $\perp$ on the data collector side rather than the user side. Note, however, that we must handle the shuffled input values very carefully to prevent information leakage. For example, if each user u_i encrypts $\langle h(x_i), x_i \rangle$ using a public key of the data collector, then the data collector can obtain $x_1, \ldots, x_n$ by decrypting them. Moreover, the shuffled input values are communicated between the shuffler and the data collector *three times* (shuffler $\rightarrow$ data collector $\rightarrow$ shuffler $\rightarrow$ data collector), and a lot of information can be leaked by comparing them. For example, the shuffler would know whose input values are replaced with $\perp$ by comparing the first and second shuffled data. The data collector would know which values are added as dummies by comparing the second and third shuffled data.

We address this issue by introducing multiple encryption, as shown in Fig. 3. Specifically, each user u_i encrypts her input value x_i *three times* using public keys of the data collector, the shuffler, and the data collector. The data collector and the shuffler decrypt the shuffled data each time they receive them. Then, intuitively, the above information leakage can be prevented for two reasons: (i) the data collector cannot see the contents of the first shuffled data, and (ii) the three shuffled data are completely different from each other. We prove that this approach provides (computational) DP in Section VI-C.

B. Details

Protocol. Algorithm 1 shows an algorithmic description of our FME protocol. Our protocol uses two dummy-count distributions $\mathcal{D}_1$ (mean: μ_1, variance: σ_1^2) and $\mathcal{D}_2$ (mean: μ_2, variance: σ_2^2). $\mathcal{D}_1$ and $\mathcal{D}_2$ are used for adding dummy hash values and dummy input values, respectively. We denote these distributions by $\mathcal{D}^* = (\mathcal{D}_1, \mathcal{D}_2)$ and our FME protocol by $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$. In Appendix C, we show a toy example of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$.

First, each user u_i sends her hash value $E_{\mathsf{pk_d}}[h(x_i)]$ and input value $E_{\mathsf{pk_d}}[E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[x_i]]]$, where $\mathsf{pk_d}$ and $\mathsf{pk_s}$ are public keys of the data collector and the shuffler, respectively (lines 1-3). After receiving them, the shuffler randomly selects each pair with probability β (line 4). Then, for each hash value $i \in [b]$, the shuffler adds a dummy $\langle E_{\mathsf{pk_d}}[i], E_{\mathsf{pk_d}}[E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[\perp]]] \rangle$ for $z_i \sim \mathcal{D}_1$ times (lines 5-7). Let $y_1^H, \ldots, y_{\tilde{n}}^H \in [b]$ (resp. $y_1, \ldots, y_{\tilde{n}} \in [d] \cup \{\perp\}$) be the selected hash (resp. input) values and dummies, where $\tilde{n} \in \mathbb{Z}_{\geq 0}$ is the total number of these values. The shuffler samples a random permutation π over $[\tilde{n}]$ and sends $(E_{\mathsf{pk_d}}[y^H_{\pi(1)}], \ldots, E_{\mathsf{pk_d}}[y^H_{\pi(\tilde{n})}])$ and $(E_{\mathsf{pk_d}}[E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[y_{\pi(1)}]]], \ldots, E_{\mathsf{pk_d}}[E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[y_{\pi(\tilde{n})}]]])$ to the data collector (lines 8-9).

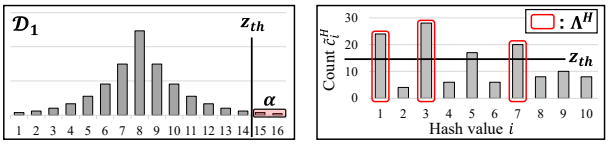

Fig. 4. Example of filtering items. In this example, $\alpha = 0.05$, $z_{th} = 15$, $b = 10$, $l = 3$, and $\Lambda^H = \{1, 3, 7\}$.

Then, the data collector decrypts them and calls the `Count` function, which calculates a count $\tilde{c}_i^H \in \mathbb{Z}_{\geq 0}$ for each hash value $i \in [b]$ from $y^H_{\pi(1)}, \ldots, y^H_{\pi(\tilde{n})}$ (lines 10-11). Based on $\tilde{c}_1^H, \ldots, \tilde{c}_b^H$, the data collector calls the `FilterItems` function, which selects a set $\Lambda^H \subseteq [b]$ of popular hash values and a set $\Lambda \subseteq [d]$ of the corresponding input values (line 12). We explain the `FilterItems` function later in detail. Then, the data collector replaces unselected items $E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[y_{\pi(i)}]]$, whose corresponding hash values are $y^H_{\pi(i)} \notin \Lambda^H$, with $E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[\perp]]$ (lines 13-17). Note that the data collector can do this without seeing $y_{\pi(i)}$, as she knows $y^H_{\pi(i)}$. The data collector sends Λ and shuffled input values $(E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[y_{\pi(1)}]], \ldots, E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[y_{\pi(\tilde{n})}]])$ back to the shuffler (line 18).

The shuffler decrypts $\tilde{n}$ shuffled input values while removing those corresponding to dummy hashes, i.e., $E_{\mathsf{pk_d}}[E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[\perp]]]$ generated by the shuffler (line 19). The shuffler can remove them because she knows the random permutation π. Then, for each selected item $i \in \Lambda$, it adds a dummy $E_{\mathsf{pk_d}}[i]$ for $z_i \sim \mathcal{D}_2$ times (lines 20-22). Let $y_1^*, \ldots, y_{\tilde{n}^*}^* \in [d]$ be $\tilde{n}$ input values and dummies, where $\tilde{n}^* \in \mathbb{Z}_{\geq 0}$ is the total number of these values. The shuffler samples a random permutation ρ over $[\tilde{n}^*]$ and sends $(E_{\mathsf{pk_d}}[y^*_{\rho(1)}], \ldots, E_{\mathsf{pk_d}}[y^*_{\rho(\tilde{n}^*)}])$ to the data collector (lines 23-24). Finally, the data collector calculates an unbiased estimate $\hat{f}_i$ of f_i for $i \in \Lambda$ and outputs Λ and $\{\hat{f}_i | i \in \Lambda\}$ (lines 25-30).

Filtering Items. Below, we explain the details of the `FilterItems` function in Algorithm 1 (line 12). This function takes two parameters as input: a significance level $\alpha \in [0, 1]$ and the maximum number $l \in [b]$ of selected hashes. Specifically, we first calculate a threshold $z_{th} \in \mathbb{Z}_{\geq 0}$ so that a random variable z generated from $\mathcal{D}_1$ is larger than or equal to z_{th} with probability at most α, i.e., $\Pr(z \geq z_{th}) \leq \alpha$. Then, we compare each count $\tilde{c}_i^H$ ($i \in [b]$) to the threshold z_{th} and select the hash value i if $\tilde{c}_i^H \geq z_{th}$. If the number of selected hash values exceeds l, we select l hash values with the largest counts. Finally, we add the selected hash values to Λ^H and the corresponding input values x such that $h(x) \in \Lambda^H$ to Λ. Fig. 4 shows an example of Λ^H.

Suppose that no user has a hash value $i \in [b]$. Then, we incorrectly add i to Λ^H with probability at most α; i.e., the false positive probability is at most the significance level α, which is small. In addition, we add at most l hash values to Λ^H. Thus, we can improve the efficiency by reducing l.

Input: Input values $(x_1, \ldots, x_n) \in [d]^n$, hash function $h : [d] \to [b]$, dummy-count distributions $\mathcal{D}^* = (\mathcal{D}_1, \mathcal{D}_2)$ ($\mathcal{D}_1$ has mean μ_1 and variance σ_1^2; $\mathcal{D}_2$ has mean μ_2 and variance σ_2^2), sampling probability $\beta \in [0, 1]$, significance level $\alpha \in [0, 1]$, maximum number of selected hashes $l \in [b]$.

Output: Selected items $\Lambda \subseteq [d]$, estimates $\{\hat{f}_i | i \in \Lambda\}$.

/* Send hash values and input values (users → shuffler) */

foreach $i \in [n]$ **do**

  [u_i] Send $\langle E_{\mathsf{pk_d}}[h(x_i)], E_{\mathsf{pk_d}}[E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[x_i]]]\rangle$ to the shuffler;

end

/* Random sampling */

[s] Sample $\langle E_{\mathsf{pk_d}}[h(x_i)], E_{\mathsf{pk_d}}[E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[x_i]]]\rangle$ ($i \in [n]$) with probability β;

/* Dummy hash data addition */

foreach $i \in [b]$ **do**

  [s] $z_i \leftarrow \mathcal{D}_1$; Add a dummy $\langle E_{\mathsf{pk_d}}[i], E_{\mathsf{pk_d}}[E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[\perp]]]\rangle$ for z_i times;

end

/* Random shuffling */

[s] Let $y_1^H, \ldots, y_{\tilde{n}}^H \in [b]$ (resp. $y_1, \ldots, y_{\tilde{n}} \in [d] \cup \{\perp\}$) be the selected hash (resp. input) values and dummies. Sample a random permutation π over $[\tilde{n}]$;

/* Send shuffled data (shuffler → data collector) */

[s] Send $(E_{\mathsf{pk_d}}[y^H_{\pi(1)}], \ldots, E_{\mathsf{pk_d}}[y^H_{\pi(\tilde{n})}])$ and $(E_{\mathsf{pk_d}}[E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[y_{\pi(1)}]]], \ldots, E_{\mathsf{pk_d}}[E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[y_{\pi(\tilde{n})}]]])$ to the data collector;

/* Filtering ($\Lambda^H \subseteq [b]$: selected hash values, $\Lambda \subseteq [d]$: selected items) */

[d] Decrypt $y^H_{\pi(1)}, \ldots, y^H_{\pi(\tilde{n})}$ and $E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[y_{\pi(1)}]], \ldots, E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[y_{\pi(\tilde{n})}]]$;

[d] $(\tilde{c}_1^H, \ldots, \tilde{c}_b^H) \leftarrow \mathtt{Count}(y^H_{\pi(1)}, \ldots, y^H_{\pi(\tilde{n})})$;

[d] $\Lambda^H, \Lambda \leftarrow \mathtt{FilterItems}(\tilde{c}_1^H, \ldots, \tilde{c}_b^H, \alpha, \mathcal{D}_1, l)$;

/* Replace unselected items with $\perp$ */

foreach $i \in \tilde{n}$ **do**

  if $y^H_{\pi(i)} \notin \Lambda^H$ **then**

    [d] $E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[y_{\pi(i)}]] \leftarrow E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[\perp]]$;

  end

end

/* Send selected items and shuffled data (data collector → shuffler) */

[d] Send Λ and $(E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[y_{\pi(1)}]], \ldots, E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[y_{\pi(\tilde{n})}]])$ to the shuffler;

/* Dummy input data addition */

[s] Decrypt $E_{\mathsf{pk_d}}[y_{\pi(1)}], \ldots, E_{\mathsf{pk_d}}[y_{\pi(\tilde{n})}]$ while removing those corresponding to dummy hash data;

foreach $i \in \Lambda$ **do**

  [s] $z_i \leftarrow \mathcal{D}_2$; Add a dummy $E_{\mathsf{pk_d}}[i]$ for z_i times;

end

/* Random shuffling */

[s] Let $y_1^*, \ldots, y_{\tilde{n}^*}^* \in [d] \cup \{\perp\}$ be $\tilde{n}$ input values and dummies. Sample a random permutation ρ over $[\tilde{n}^*]$;

/* Send shuffled data (shuffler → data collector) */

[s] Send $(E_{\mathsf{pk_d}}[y^*_{\rho(1)}], \ldots, E_{\mathsf{pk_d}}[y^*_{\rho(\tilde{n}^*)}])$ to the data collector;

/* Compute an unbiased estimate */

[d] Decrypt $y^*_{\rho(1)}, \ldots, y^*_{\rho(\tilde{n}^*)}$;

[d] $(\tilde{c}_1, \ldots, \tilde{c}_d) \leftarrow \mathtt{Count}(y^*_{\rho(1)}, \ldots, y^*_{\rho(\tilde{n}^*)})$;

foreach $i \in \Lambda$ **do**

  [d] $\hat{f}_i \leftarrow \frac{1}{n\beta}(\tilde{c}_i - \mu_2)$;

end

return Λ and $\{\hat{f}_i | i \in \Lambda\}$

Algorithm 1: Our FME protocol $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$. $\mathsf{pk_d}$ (resp. $\mathsf{pk_s}$) represents a public key of the data collector (resp. shuffler).

Parameters. $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ has parameters β, α, and l. We explain how to set l in Section VI-D and β and α in Appendix E-A.

C. Theoretical Properties

Privacy and Robustness. First, we show that $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ provides DP and is robust to both data poisoning and collusion attacks:

Theorem 6. *If the binary input mechanisms $\mathcal{M}_{\mathcal{D}_1,\beta}$ and $\mathcal{M}_{\mathcal{D}_2,1}$ in Definition 5 provide $(\frac{\varepsilon_1}{2}, \frac{\delta_1}{2})$-DP and $(\frac{\varepsilon_2}{2}, \frac{\delta_2}{2})$-DP, respectively, then $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ provides (ε, δ)-CDP, where $(\varepsilon, \delta) = (\varepsilon_1 + \varepsilon_2, \delta_1 + \delta_2)$, and is robust to collusion with users.*

Theorem 7. *Let $\lambda = \frac{n'}{n+n'}$ and $f_{\mathcal{T}} = \sum_{i\in\mathcal{T}} f_i$. $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ provides the following robustness against poisoning attacks:*

$$G_f^{\mathsf{max}} = \lambda(1 - f_{\mathcal{T}}) + \textstyle\sum_{i\in\mathcal{T}} \eta_i f_i, \tag{10}$$

where $\eta_i \in [0, 1]$ is the probability that the i-th item is not selected in the filtering step (i.e., $i \notin \Lambda$) before data poisoning.

Theorem 6 uses the basic composition theorem [1]. In Appendix C, we explain that this is almost tight in our case. $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ spends $(\varepsilon_1, \delta_1)$ and $(\varepsilon_2, \delta_2)$ for hash and input values, respectively. When $\beta = 1$ and the same dummy-count distribution is used for $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}$, $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ needs the total budget (ε, δ) twice as large as the LNF protocol $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$.

In addition, Theorem 6 assumes that the shuffler does not disclose dummies after running the protocol in the same way as [17], [18]. This can be achieved, e.g., by using the TEE. We note, however, that even if the shuffler discloses dummies, we can guarantee DP for the outputs by adding additional noise to the outputs before publishing them. In Appendix E-F, we show the accuracy is hardly affected by this additional noise.

In Theorem 7, the second term in (10) is introduced because a target item $i \in \mathcal{T}$ can be discarded in the filtering step before poisoning and selected after poisoning. This results in a slight increase in the overall gain. We note, however, that the second term in (10) is very small in practice, as η_i is extremely small when f_i is large (e.g., η_i decreases exponentially as f_i increases when $l = b$; see [32]). When the second term is ignored, G_f^{max} of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ is equal to that of $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ in (5).

Efficiency. We next analyze the communication cost of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$:

Theorem 8. *Let $\tau_1, \tau_2, \tau_3 \in \mathbb{R}_{\geq 0}$ be the size of a single ciphertext, double ciphertext, and triple ciphertext, respectively, in $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$. Then, the total communication cost of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ is $C_{tot} = C_{U-S} + C_{S-D}$, where*

$$C_{U-S} = (\tau_1 + \tau_3)n$$
$$C_{S-D} \leq (2\tau_1 + \tau_2 + \tau_3)(\beta n + \mu_1 b) + \tau_1(\mu_2 + 1)\,\mathbb{E}[|\Lambda|],$$

and the expected number $\mathbb{E}[|\Lambda|]$ of selected items is:

$$\mathbb{E}[|\Lambda|] \leq \begin{cases} \frac{(\beta n + \alpha(l - \beta n))d}{b} & \textit{(if } \beta n \leq l \leq b\textit{)} \\ \frac{ld}{b} & \textit{(otherwise).} \end{cases} \tag{11}$$

For example, if we use ECIES with 256-bit security [62], then τ_1, τ_2, and τ_3 are 712, 1392, and 2072 bits, respectively. By optimizing b, C_{tot} of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ can be expressed as $C_{tot} = O(n + \sqrt{ld})$ when $l < \beta n$. See Section VI-D for details. Similarly, the computational cost of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ can be $O(n+\sqrt{ld})$ in this case. See [32] for details.

Accuracy. Finally, we analyze the accuracy of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$:

Theorem 9. *For any item $i \in \Lambda$ selected in the filtering step, $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ outputs an unbiased estimate (i.e., $\mathbb{E}[\hat{f}_i|\Lambda] = f_i$) and achieves the following variance:*

$$\mathbb{V}[\hat{f}_i|\Lambda] = \tfrac{f_i(1-\beta)}{n\beta} + \tfrac{\sigma_2^2}{n^2\beta^2}. \tag{12}$$

$\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ also achieves the following expected squared error:

$$\mathbb{E}[(\hat{f}_i - f_i)^2] = (1 - \eta_i)\,\mathbb{V}[\hat{f}_i|\Lambda] + \eta_i f_i^2, \tag{13}$$

where $\eta_i \in [0, 1]$ is the probability that the i-th item is not selected in the filtering step (i.e., $i \notin \Lambda$).

The variance in (12) is the same as that of $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ in (6). The second term in (13) is introduced because $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ always calculates $\hat{f}_i$ as 0 for an unselected item $i \notin \Lambda$. However, the second term in (13) is very small, as η_i is extremely small for a large f_i, as explained above. When ignoring the second term, the l_2 loss of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ is almost the same as that of $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$.

Summary. Our FME protocol $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ can achieve almost the same accuracy and robustness as the LNF protocol $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ by using (ε, δ) twice as large as $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$. It can also achieve the communication and computational costs of $O(n + \sqrt{ld})$ by optimizing b, as explained below.

D. Optimizing the Range b of the Hash Function

As shown in Theorem 8, the communication cost C_{tot} of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ depends on the hash range b. A larger b results in the increase of dummy values sent from the shuffler to the data collector. In contrast, a smaller b results in the increase of $\mathbb{E}[|\Lambda|]$ in (11). The optimal b can be obtained by calculating b that minimizes the upper bound on C_{tot} in Theorem 8.

For example, if $l = b$, then the optimal value of b is given by $b = \sqrt{\frac{\tau_1(\mu_2+1)\beta(1-\alpha)nd}{(2\tau_1+\tau_2+\tau_3)\mu_1}}$. In this case, C_{tot} of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ can be simplified as $C_{tot} = O(n + \sqrt{nd} + \alpha d)$ by treating τ_1, τ_2, and τ_3 as constants. This is larger than $O(n)$ but much smaller than $O(n + d)$ when $d \gg n$.

If $l < \beta n$, the optimal value of b is $b = \sqrt{\frac{\tau_1(\mu_2+1)ld}{(2\tau_1+\tau_2+\tau_3)\mu_1}}$. In this case, C_{tot} of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ can be written as $C_{tot} = O(n+\sqrt{ld})$.

Setting l. In practice, the frequency distribution $\mathbf{f}$ is sparse in large-domain data, and we are often interested in popular items with large frequencies [23], [34], [63]. Thus, in this work, we propose to set l to $l = \max\{\frac{n^2}{d}, c\}$, where $c \in \mathbb{N}$ is some constant ($c = 50$ in our experiments). In this setting, C_{tot} does not depend on d until d exceeds $\frac{n^2}{c}$. Moreover, this setting guarantees that at least c popular items are selected. In our experiments, we show that our FME protocol with this setting provides high accuracy and efficiency.

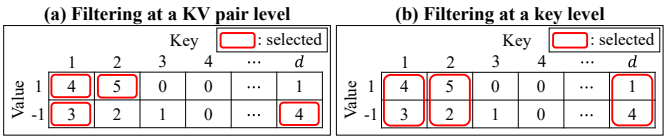

Fig. 5. Example of filtering at a (a) KV pair level or (b) key level. The number in the matrix represents a count of each KV pair in the first shuffled data sent from the shuffler to the data collector.

VII. Application to Key-Value Data

In this section, we propose a protocol for frequency and mean estimation over KV data by applying the FME protocol $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ with an additional technique called *TKV-FK (Transforming KV Pairs and Filtering Keys)*. Section VII-A explains our protocol. Section VII-B shows its theoretical properties.

A. Our Protocol for KV Data

Overview. Our protocol for KV data, denoted by $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$, first uses padding-and-sampling [23], a state-of-the-art sampling technique for a large domain. Then, it applies the FME protocol $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ to KV data with an additional technique called TKV-FK, which transforms KV pairs into one-dimensional data and filters the data at a key level. Finally, it calculates unbiased estimates $\hat{\mathbf{\Phi}}$ and $\hat{\mathbf{\Psi}}$ of frequencies $\mathbf{\Phi}$ and mean values $\mathbf{\Psi}$, respectively. Below, we explain these techniques in detail.

Padding-and-Sampling. We first use a padding-and-sampling technique [23]. This technique samples a single KV pair for each user u_i to avoid splitting the privacy budget ε into multiple KV pairs x_i. Specifically, if the number $|x_i|$ of KV pairs is smaller than a parameter $\kappa \in \mathbb{N}$ called the padding length, user u_i adds dummy KV pairs $\langle d+1, 0\rangle, \ldots, \langle d+\kappa-|x_i|, 0\rangle$ to x_i. Then, u_i samples a single KV pair $\langle k_i, v_i^*\rangle$ from x_i and discretizes v_i^* to $v_i = 1$ with probability $\frac{1+v_i^*}{2}$ and $v_i = -1$ with probability $\frac{1-v_i^*}{2}$. As a result, u_i obtains a single, discretized KV pair $\langle k_i, v_i\rangle \in [d] \times \{-1, 1\}$.

TKV-FK. After padding-and-sampling, we apply the FME protocol $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ to the KV pair $\langle k_i, v_i\rangle$ ($i \in [n]$) with our TKV-FK technique. This technique first transforms the KV pair $\langle k_i, v_i\rangle$ of user u_i into one-dimensional data $s_i \in [2d]$ by $s_i = k_i + \frac{1}{2}(v_i + 1)d$. Then, we can directly apply the FME protocol $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ by treating $(s_1, \ldots, s_n) \in [2d]^n$ as input values. However, this approach does not work well, as filtering is performed *at a key-value pair level*. Fig. 5(a) shows its example. In this example, the KV pair $\langle 2, 1\rangle$ is selected in the filtering step, whereas $\langle 2, -1\rangle$ is not. This causes a large positive bias in the estimate $\hat{\Psi}_2$ of the mean value Ψ_2 for key 2. Thus, mean values $\mathbf{\Psi}$ cannot be accurately estimated.

Our TKV-FK technique addresses this issue by filtering data *at a key level*, as shown in Fig. 5(b). Specifically, it makes the following three changes to the FME protocol $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ (Algorithm 1): (i) each user u_i applies a hash function $h : [d] \to [b]$ to her *key* k_i and sends $E_{\mathsf{pk_d}}[h(k_i)]$ and $E_{\mathsf{pk_d}}[E_{\mathsf{pk_s}}[E_{\mathsf{pk_d}}[s_i]]]$ to the shuffler (line 2), (ii) based on shuffled hash values, the data collector selects a set $\Lambda^H \subseteq [b]$ of popular hash values and a set $\Lambda \subseteq [d]$ of the corresponding *keys* (line 12), and (iii) for each $i \in \Lambda$, the data collector adds $z_{i,1}, z_{i,-1} \sim \mathcal{D}_2$ dummies to KV pairs $\langle i, 1\rangle$ and $\langle i, -1\rangle$, respectively (line 21).

Calculating Unbiased Estimates. Finally, we calculate unbiased estimates $\hat{\mathbf{\Phi}}$ and $\hat{\mathbf{\Psi}}$ as follows. For $i \in \Lambda$, let $\tilde{c}_{i,1}$ (resp. $\tilde{c}_{i,-1}$) $\in \mathbb{Z}_{\geq 0}$ be a count of KV pair $\langle i, 1\rangle$ (resp. $\langle i, -1\rangle$) in the shuffled data the data collector decrypts. For $i \in \Lambda$, the data collector calculates $\hat{\Phi}_i$ and $\hat{\Psi}_i$ as follows[2]:

$$\hat{\Phi}_i = \frac{\kappa}{n\beta}(\tilde{c}_{i,1} + \tilde{c}_{i,-1} - 2\mu_2), \hat{\Psi}_i = \frac{\kappa}{n\beta\hat{\Phi}_i}(\tilde{c}_{i,1} - \tilde{c}_{i,-1}). \quad (14)$$

Thanks to our TKV-FK technique, we can calculate both $\tilde{c}_{i,1}$ and $\tilde{c}_{i,-1}$ for each selected key $i \in \Lambda$. Thus, the estimates in (14) are (almost) unbiased, as shown in Section VII-B.

B. Theoretical Properties

Privacy and Robustness. Our KV protocol $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ also provides DP and is robust to collusion and poisoning attacks:

Theorem 10. *If the binary input mechanisms $\mathcal{M}_{\mathcal{D}_1,\beta}$ and $\mathcal{M}_{\mathcal{D}_2,1}$ in Definition 5 provide $(\frac{\varepsilon_1}{2}, \frac{\delta_1}{2})$-DP and $(\frac{\varepsilon_2}{2}, \frac{\delta_2}{2})$-DP, respectively, then $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ provides (ε, δ)-CDP, where $(\varepsilon, \delta) = (\varepsilon_1 + \varepsilon_2, \delta_1 + \delta_2)$, and is robust to collusion with users.*

Theorem 11. *Let $\lambda = \frac{n'}{n+n'}$, $\Phi_{\mathcal{T}} = \sum_{i\in\mathcal{T}} \Phi_i$, and $\Psi_{\mathcal{T}} = \sum_{i\in\mathcal{T}} \Psi_i$. Let $\mathcal{U}_i$ be the set of users who have key $i \in [d]$. For $u_j \in \mathcal{U}_i$, let $\psi_{j,i} \in [-1, 1]$ be the value of key i held by user u_j. For $j \in [n]$, let $\xi_j = \max\{|x_j|, \kappa\}$. $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ provides the following robustness against poisoning attacks:*

$$G^{max}_{\Phi} = \frac{\kappa}{n+n'}\left(\left(\sum_{i\in\mathcal{T}}\sum_{u_j\in\mathcal{U}_i}\frac{1}{\xi_j}\right) + n'\right) - \Phi_{\mathcal{T}} + \sum_{i\in\mathcal{T}}\eta_i\Phi_i \quad (15)$$

$$G^{max}_{\Psi} \approx \left(\sum_{i\in\mathcal{T}}\frac{\left(\sum_{u_j\in\mathcal{U}_i}\frac{\psi_{j,i}}{\xi_j}\right) + \frac{n'}{|\mathcal{T}|}}{\left(\sum_{u_j\in\mathcal{U}_i}\frac{1}{\xi_j}\right) + \frac{n'}{|\mathcal{T}|}}\right) - \Psi_{\mathcal{T}}, \quad (16)$$

where $\eta_i \in [0, 1]$ is the probability that the i-th item is not selected in the filtering step (i.e., $i \notin \Lambda$) before data poisoning. The approximation in (16) is obtained from a Taylor expansion $\mathbb{E}[\frac{X}{Y}] \approx \frac{\mathbb{E}[X]}{\mathbb{E}[Y]}$ for two random variables X and Y.

Theorem 11 states that G^{max}_{Φ} and G^{max}_{Ψ} of $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ in (15) and (16) do not depend on the privacy budget ε. It is shown in [20] that the existing KV protocols become vulnerable to data poisoning as ε increases or decreases. For example, in PrivKVM [21], G^{max}_{Φ} increases (resp. decreases) as ε decreases when $|\mathcal{T}| = 1$ (resp. $|\mathcal{T}| \geq 3$). In PCKV-GRR and PCKV-UE, G^{max}_{Φ} increases as ε decreases. In contrast, $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ does not suffer from such fluctuation in G^{max}_{Φ} and G^{max}_{Ψ}.

Moreover, G^{max}_{Φ} and G^{max}_{Ψ} of $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ are much smaller than those of the existing KV protocols analyzed in [20]. See [32] for details. In our experiments, we show that $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ is much more robust than the existing protocols.

Efficiency. Our KV protocol $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ achieves the same efficiency as $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$. Specifically, $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ can achieve the communication and computational costs of $C_{tot} = O(n + \sqrt{ld})$.

[2]Note that $\hat{\Psi}_i$ is not defined for an unselected key $i \notin \Lambda$. We set $\hat{\Psi}_i = 1$ for $i \notin \Lambda$ to eliminate the (unnecessary) mean gain for unselected keys.

Accuracy. Finally, we show the accuracy of $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$:

Theorem 12. *If $\kappa \geq |x_j|$ for any $j \in [n]$, then for any key $i \in \Lambda$ selected in the filtering step, $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ outputs almost unbiased estimates. Specifically, $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ achieves:*

$$\mathbb{E}[\hat{\Phi}_i|\Lambda] = \Phi_i \tag{17}$$

$$\mathbb{V}[\hat{\Phi}_i|\Lambda] = \frac{\Phi_i(\kappa-\beta)}{n\beta} + \frac{2\kappa^2\sigma_2^2}{n^2\beta^2} \tag{18}$$

$$\mathbb{E}[\hat{\Psi}_i|\Lambda] \approx \Psi_i \tag{19}$$

$$\mathbb{V}[\hat{\Psi}_i|\Lambda] \lesssim \frac{\kappa^2}{n\beta^2}(2(q_i - q_i^2 + r_i - r_i^2) - \frac{\beta}{\kappa}(1-\frac{\beta}{\kappa})), \tag{20}$$

where $q_i = \frac{\beta(1+\Psi_i)}{2\kappa}$ and $r_i = \frac{\beta(1-\Psi_i)}{2\kappa}$. The approximations in (19) and (20) are obtained from Taylor expansions $\mathbb{E}[\frac{X}{Y}] \approx \frac{\mathbb{E}[X]}{\mathbb{E}[Y]}$ and $\mathbb{V}[\frac{X}{Y}] \approx \frac{\mathbb{V}[X]}{\mathbb{E}[Y]}$ for two random variables X and Y. In addition, $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ achieves the following expected l_2 loss:

$$\mathbb{E}[(\hat{\Phi}_i - \Phi_i)^2] = (1-\eta_i)\,\mathbb{V}[\hat{\Phi}_i|\Lambda] + \eta_i\Phi_i^2$$
$$\mathbb{E}[(\hat{\Psi}_i - \Psi_i)^2] = (1-\eta_i)\,\mathbb{V}[\hat{\Psi}_i|\Lambda] + \eta_i(1-\Psi_i)^2,$$

where $\eta_i \in [0,1]$ is the probability that the i-th key is not selected in the filtering step (i.e., $i \notin \Lambda$).

In our experiments, we show that $\mathcal{S}^{\mathsf{KV}}_{\mathcal{D}^*,\beta}$ provides higher accuracy than the existing KV protocols.

VIII. Experimental Evaluation

A. Experimental Set-up

Datasets. We conducted experiments using four real datasets:

- **Foursquare [64]**: Location dataset with $n = 18201$ users in New York. We divided the city into 1000×1000 regions ($d = 1000000$) at regular intervals.
- **AOL [65]**: Web access dataset. Following [34], we extracted $n = 10000$ users and used the first three characters of each URL as an item ($d = 2^{24} = 16777216$).
- **E-Commerce [66]**: Clothing review dataset with $n = 23486$ users, $d = 1206$ keys (items), and 23486 ratings.
- **Amazon [31]**: Amazon rating dataset with $n = 1210271$ users, $d = 249274$ keys, and 2023070 ratings.

The first two are categorical, and the last two are KV datasets.

Protocols. We compared our FME protocol with the CH protocol and existing shuffle protocols. In our FME protocol, we set $\alpha = 0.05$ and used the asymmetric geometric distribution [18] with $\beta = 1$ as dummy-count distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ (i.e., $(\varepsilon_1, \delta_1) = (\varepsilon_2, \delta_2)$). Then, we optimized the hash range b as described in Section VI-D and set the maximum number l of selected hashes to $l = b$ (denoted by Proposal (large l)) or $\max\{\frac{n^2}{d}, 50\}$ (Proposal (small l)). For KV data, we used our TKV-FK technique. In Appendix E-A, we also evaluated our FME protocol when we changed ε_1 $(= \varepsilon - \varepsilon_2)$, α, and l.

For existing protocols, we evaluated twelve protocols. Specifically, for categorical data, we evaluated four pure shuffle protocols using the GRR [17], RAPPOR [2], OUE [52], and OLH [52]. For these protocols, we used a numerical upper bound in [15] because it is tighter than Theorem 1 (we confirmed that the upper bound is very close to the lower bound in [15]). We also evaluated three multi-message protocols in [34] (for a large domain), [35], [36], denoted by LWY22-Large, BC20, and CM22, respectively. We used their amplification results for these protocols. Since these protocols assume that ε is within a certain range, we evaluated them only within the range. For CM22, we generated 10 dummy values per user in the same way as [36]. We also compared our protocols with the LNF protocol [18] in terms of efficiency. Following [2], [52], we used a significance threshold, which assigns 0 to an estimate below a threshold for each protocol.

For KV data, we evaluated four pure shuffle protocols based on PrivKVM [21], PrivKVM* [22], PCKV-GRR [23], and PCKV-UE [23] using the numerical upper bound in [15]. We did not evaluate the protocol in [67], as it leaks the number of KV pairs held by each user and fails to provide DP. Following [23], we clipped frequency estimates to $[0,1]$ and set the padding length κ to $\kappa = 1$ (resp. 3) in the E-Commerce (resp. Amazon) dataset.

Performance Metrics. Since d is large in our experiments, most items have low or zero frequencies. Thus, we evaluated the accuracy for top-50 items (keys) with the largest frequencies. Specifically, we evaluated the MSE (Mean Squared Error) over the 50 items. Here, we varied ε from 0.1 to 5, as DP with this range of ε provides theoretical privacy guarantees against the inference of input values. See Appendix D for details.

For robustness to collusion attacks, we refer to ε when no (resp. $|\Omega|$) users collude with the data collector as a *target* ε (resp. *actual* ε). We set the target ε to 0.1 and evaluated the actual ε while changing $|\Omega|$. For robustness to poisoning attacks, we evaluated the maximum gains G_f^{max}, G_Φ^{max}, and G_Ψ^{max}. We ran each protocol 10 times and averaged the MSE and the gains. For efficiency, we evaluated C_{tot} and measured the run time using a workstation with Intel Xeon W-2295 (3.00 GHz, 18 Cores) and 256 GB main memory.

User Sampling. In the KV datasets, n is large. Thus, for each protocol (except for the LNF protocol), we introduced user sampling, which randomly samples users with probability 0.05 before running the protocol to improve efficiency. We evaluated the MSE between the estimates and the true frequencies *before* user sampling.

B. Experimental Results

Accuracy. First, we evaluated the relationship between the MSE and ε. Fig. 6 shows the results.

Fig. 6 shows that Proposal (large l) significantly outperforms the existing protocols. This is because our FME protocol with $l = b$ achieves almost the same accuracy as the LNF protocol, which is shown to be very accurate in [18], by doubling (ε, δ). Fig. 6 also shows that Proposal (small l) provides almost the same MSE as Proposal (large l), which means that we can improve the efficiency without affecting accuracy by reducing l as proposed in Section VI-D.

In Appendix E-C, we also show that the CH protocol has poor accuracy for unpopular items due to hash collision and cannot be improved by using user/group-dependent hashes.

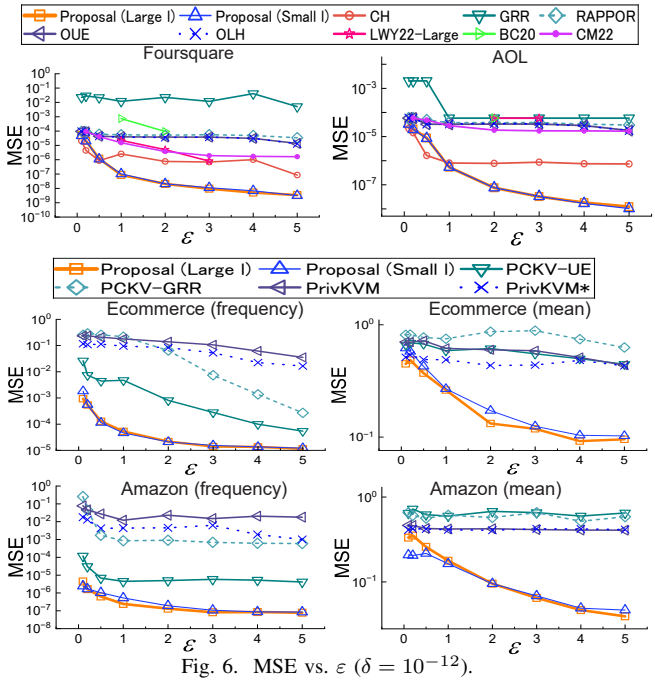

Fig. 6. MSE vs. ε ($\delta = 10^{-12}$).

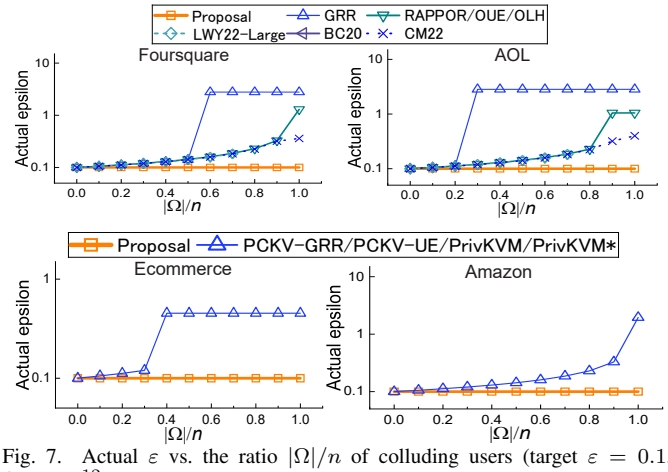

Fig. 7. Actual ε vs. the ratio $|\Omega|/n$ of colluding users (target $\varepsilon = 0.1$, $\delta = 10^{-12}$).

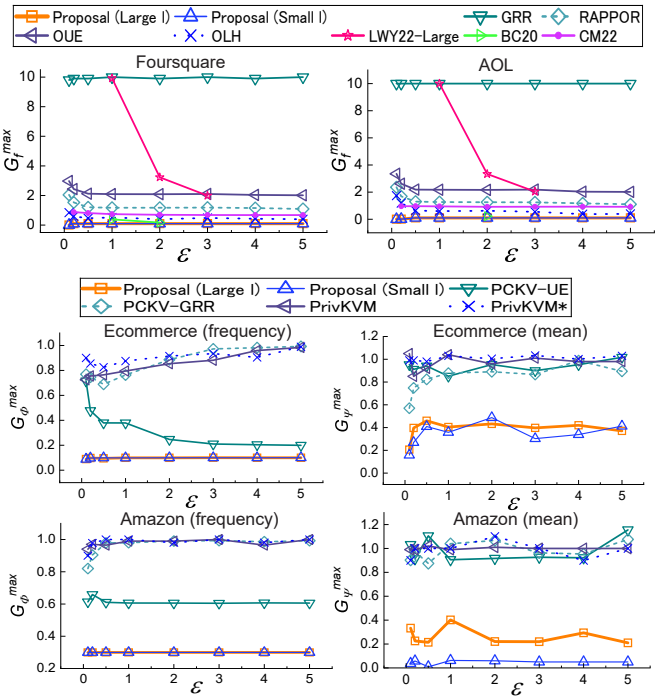

Fig. 8. Maximum gains $G_f^{\max}$, $G_\Phi^{\max}$, and $G_\Psi^{\max}$ vs. ε ($\lambda = 0.1$, $\delta = 10^{-12}$).

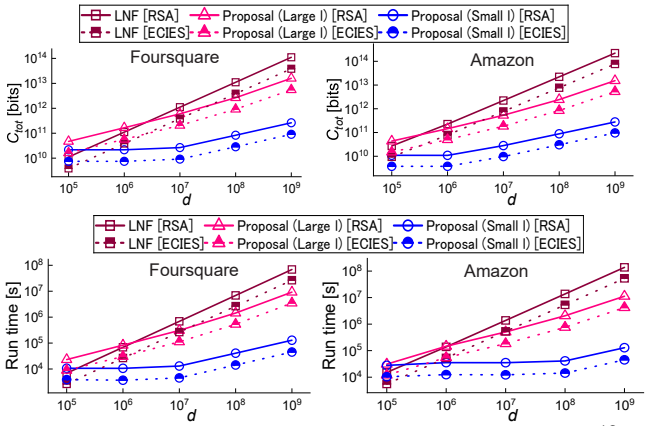

Fig. 9. Computational cost C_{tot} and run time ($\varepsilon = 1$, $\delta = 10^{-12}$).

Moreover, in Appendix E-E, we show the effectiveness of our TKV-FK technique through an ablation study.

Robustness to Collusion Attacks. Next, we evaluated the robustness to collusion with users. Fig. 7 shows the relationship between the actual ε and the ratio $|\Omega|/n$ of colluding users.

We observe that in all the existing protocols, the actual ε rapidly increases with an increase in $|\Omega|$. This vulnerability is inevitable in the existing protocols because they add noise on the user side. In Appendix E-B, we also show that the defense in [17] is insufficient in that the actual ε still increases with an increase in $|\Omega|$. In contrast, the actual ε always coincides with the target ε in our proposals, demonstrating the robustness of our proposals to collusion with users.

In Appendix E-D, we also set $|\Omega|/n = 0.1$ and evaluate the relationship between the actual ε and the target ε.

Robustness to Poisoning Attacks. We also evaluated the robustness to poisoning attacks. Here, we randomly selected $|\mathcal{T}| = 10$ target items for categorical data and $|\mathcal{T}| = 1$ target item for KV data. Then, we set the fraction λ of fake users to $\lambda = 0.1$; we also changed λ in Appendix E-D.

Fig. 8 shows the results.[3] We observe that the existing protocols suffer from large maximum gains. In Appendix E-B, we also show that the defense in [20] has limited effectiveness. In contrast, our proposals always achieve small $G_f^{\max}$, $G_\Phi^{\max}$, and $G_\Psi^{\max}$, and are robust to poisoning attacks.

Efficiency. Finally, we evaluated C_{tot} and the run time. Specifically, we measured C_{tot} and the run time of the LNF protocol, Proposal (large l), and Proposal (small l) in the Foursquare and Amazon datasets. For an encryption scheme, we used the 2048-bit RSA or ECIES with 256-bit security in [62]. Then, we calculated the time to encrypt or decrypt a

[3] In Fig. 8, we omit the gain for the CH protocol because it is very close to the gains for our proposals.

single message using multiple encryption and estimated C_{tot} and the run time when we changed d from 10^5 and 10^9.

Fig. 9 shows the results. We observe that C_{tot} and the run time of the LNF protocol are linear in d and extremely large when d is large; e.g., C_{tot} is about 100 Terabits and the run time is about 3 years when $d = 10^9$ and ECIES is used. Proposal (small l) addresses this issue. Specifically, in Proposal (small l), both C_{tot} and the run time do not depend on d until $d = 10^6$ or 10^7 and then increase in $O(\sqrt{d})$, which is consistent with our theoretical results in Section VI-D.

For example, when $d = 10^9$, Proposal (small l) reduces C_{tot} from about 100 Terabits to 260 Gigabits and the run time from about 3 years to 1 day. Thus, accurate, robust, and efficient data analysis over large-domain categorical and KV data is now possible under DP.

IX. Conclusion

We proposed the FME protocol for large-domain categorical and KV data and showed its effectiveness through theoretical analysis and extensive experiments. LDP protocols for categorical data serve as a basis for many complex tasks, such as frequent itemset mining [41], ranking estimation [68], and range queries [69]. Thus, we believe our FME protocol can also be used as a building block for such tasks. For future work, we would like to generalize our protocol for such tasks.

Acknowledgment

This study was supported in part by JSPS KAKENHI 22H00521, 24H00714, 24K20775, JST NEXUS JPMJNX25C2, JST AIP Acceleration Research JPMJCR22U5, and JST CREST JPMJCR22M1.

References

[1] C. Dwork and A. Roth, *The Algorithmic Foundations of Differential Privacy*. Now Publishers, 2014.

[2] U. Erlingsson, V. Pihur, and A. Korolova, "RAPPOR: Randomized aggregatable privacy-preserving ordinal response," in *Proc. CCS'14*, 2014, pp. 1054–1067.

[3] B. Ding, J. Kulkarni, and S. Yekhanin, "Collecting telemetry data privately," in *Proc. NIPS'17*, 2017, pp. 3574–3583.

[4] A. G. Thakurta, A. H. Vyrros, U. S. Vaishampayan, G. Kapoor, J. Freudiger, V. R. Sridhar, and D. Davidson, learning New Words, US Patent 9,594,741, Mar. 14 2017.

[5] J. Drechsler, "Differential privacy for government agencies—are we there yet?" *J. Am. Stat. Assoc.*, vol. 118, no. 541, pp. 761–773, 2023.

[6] https://www.infosecurity-magazine.com/news/data-breaches-human-error/, 2024.

[7] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, and S. Raskhodnikova, "What can we learn privately?" in *Proc. FOCS'08*, 2008, pp. 531–540.

[8] T. Wang, M. Lopuhaa-Zwakenberg, Z. Li, B. Skoric, and N. Li, "Locally differentially private frequency estimation with consistency," in *Proc. NDSS'20*, 2020, pp. 1–16.

[9] Y. Ye, T. Wang, M. Zhang, and D. Feng, "Revisiting EM-based estimation for locally differentially private protocols," in *Proc. NDSS'25*, 2025, pp. 1–18.

[10] 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 *Proc. SOSP'17*, 2017, pp. 441–459.

[11] U. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, and K. Talwar, "Amplification by shuffling: from local to central differential privacy via anonymity," in *Proc. SODA'19*, 2019, pp. 2468–2479.

[12] A. Cheu, A. Smith, J. Ullman, D. Zeber, and M. Zhilyaev, "Distributed differential privacy via shuffling," in *Proc. EUROCRYPT'19*, 2019, pp. 375–403.

[13] B. Balle, J. Bell, A. Gascon, and K. Nissim, "The privacy blanket of the shuffle model," in *Proc. CRYPTO'19*, 2019, pp. 638–667.

[14] A. M. Girgis, D. Data, S. Diggavi, A. T. Suresh, and P. Kairouz, "On the rényi differential privacy of the shuffle model," in *Proc. CCS'21*, 2021, pp. 2321–2341.

[15] V. Feldman, A. McMillan, and K. Talwar, "Hiding among the clones: A simple and nearly optimal analysis of privacy amplification by shuffling," in *Proc. FOCS'21*, 2021, pp. 954–964.

[16] ——, "Stronger privacy amplification by shuffling for rényi and approximate differential privacy," in *Proc. SODA'23*, 2023, pp. 4966–4981.

[17] T. Wang, B. Ding, M. Xu, Z. Huang, C. Hong, J. Zhou, N. Li, and S. Jha, "Improving utility and security of the shuffler-based differential privacy," *PVLDB*, vol. 13, no. 13, pp. 3545–3558, 2020.

[18] T. Murakami, Y. Sei, and R. Eriguchi, "Augmented shuffle protocols for accurate and robust frequency estimation under differential privacy," in *Proc. S&P'25*, 2025, pp. 3892–3911.

[19] X. Cao, J. Jia, and N. Z. Gong, "Data poisoning attacks to local differential privacy protocols," in *Proc. USENIX Security'21*, 2021, pp. 947–964.

[20] Y. Wu, X. Cao, J. Jia, and N. Z. Gong, "Poisoning attacks to local differential privacy protocols for key-value data," in *Proc. USENIX Security'22*, 2022, pp. 519–536.

[21] Q. Ye, H. Hu, X. Meng, and H. Zheng, "PrivKV: Key-value data collection with local differential privacy," in *Proc. S&P'19*, 2019, pp. 317–331.

[22] Q. Ye, H. Hu, X. Meng, H. Zheng, K. Huang, and C. Fang, "PrivKVM*: Revisiting key-value statistics estimation with local differential privacy," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 17–35, 2023.

[23] X. Gu, M. Li, Y. Cheng, L. Xiong, and Y. Cao, "PCKV: Locally differentially private correlated key-value data collection with optimized utility," in *Proc. USENIX Security'20*, 2020, pp. 967–984.

[24] J. Imola, T. Murakami, and K. Chaudhuri, "Differentially private triangle and 4-cycle counting in the shuffle model," in *Proc. CCS'22*, 2022, pp. 1505–1518.

[25] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*. Wiley, 2015.

[26] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan, "Computational differential privacy," in *Proc. CRYPTO'09*, 2009, pp. 126–142.

[27] R. Eriguchi, A. Ichikawa, N. Kunihiro, and K. Nuida, "Efficient noise generation protocols for differentially private multiparty computation," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 06, pp. 4486–4501, 2023.

[28] P. Scherer, C. Weis, and T. Strufe, "Provable security for the onion routing and mix network packet format sphinx," *PoPETs*, vol. 2024, no. 4, pp. 755–783, 2024.

[29] M. Patrascu and M. Thorup, "On the k-independence required by linear probing and minwise independence," *ACM Trans. Algorithms*, vol. 12, no. 1, pp. 1–27, 2015.

[30] https://amzscout.net/blog/amazon-statistics/, 2024.

[31] https://www.kaggle.com/datasets/skillsmuggler/amazon-ratings, 2018.

[32] T. Murakami, Y. Sei, and R. Eriguchi, "Augmented shuffle differential privacy protocols for large-domain categorical and key-value data," *arXiv:2509.02004*, 2025.

[33] https://doi.org/10.5281/zenodo.17032669, 2025.

[34] Q. Luo, Y. Wang, and K. Yi, "Frequency estimation in the shuffle model with almost a single message," in *Proc. CCS'22*, 2022, pp. 2219–2232.

[35] V. Balcer and A. Cheu, "Separating local & shuffled differential privacy via histograms," in *Proc. ITC'20*, 2020, pp. 1–14.

[36] A. Cheu and M. Zhilyaev, "Differentially private histograms in the shuffle model from fake users," in *Proc. S&P'22*, 2022, pp. 440–457.

[37] A. Beimel, I. Haitner, K. Nissim, and U. Stemmer, "On the round complexity of the shuffle model," in *Proc. TCC'20*, 2020, pp. 683–712.

[38] A. Girgis, D. Data, and S. Diggavi, "Rényi differential privacy of the subsampled shuffle model in distributed learning," in *Proc. NeurIPS'21*, 2021, pp. 29 181–29 192.

[39] A. Cheu, A. Smith, and J. Ullman, "Manipulation attacks in local differential privacy," in *Proc. S&P'21*, 2021, pp. 883–900.

[40] X. Li, N. Li, W. Sun, N. Z. Gong, and H. Li, "Fine-grained poisoning attack to local differential privacy protocols for mean and variance estimation," in *Proc. USENIX Security'23*, 2023, pp. 1739–1756.

[41] W. Tong, H. Chen, J. Niu, and S. Zhong, "Data poisoning attacks to locally differentially private frequent itemset mining protocols," in *Proc. CCS'24*, 2024, pp. 3555–3569.
[42] K. Huang, G. Ouyang, Q. Ye, H. Hu, B. Zheng, X. Zhao, R. Zhang, and X. Zhou, "LDPGuard: Defenses against data poisoning attacks to local differential privacy protocols," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 7, pp. 3195–3209, 2024.
[43] S. Song, L. Xu, and L. Zhu, "Efficient defenses against output poisoning attacks on local differential privacy," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 5506–5521, 2023.
[44] F. Kato, Y. Cao, and M. Yoshikawa, "Preventing manipulation attack in local differential privacy using verifiable randomization mechanism," in *Proc. DBSec'24*, 2021, pp. 43–60.
[45] H. Horigome, H. Kikuchi, and C.-M. Yu, "Local differential privacy protocol for making key–value data robust against poisoning attacks," in *Proc. MDAI'23*, 2023, pp. 241–252.
[46] X. Sun, Q. Ye, H. Hu, J. Duan, T. Wo, J. Xu, and R. Yang, "LDPRecover: Recovering frequencies from poisoning attacks against local differential privacy," in *Proc. ICDE'24*, 2024.
[47] Y. Dodis and J. Katz, "Chosen-ciphertext security of multiple encryption," in *Proc. TCC'05*, 2005, pp. 188–209.
[48] Y. Dai, J. Lee, B. Mennink, and J. Steinberger, "The security of multiple encryption in the ideal cipher model," in *Proc. CRYPTO'14*, 2014, pp. 20–38.
[49] J. Furukawa and K. Sako, "An efficient publicly verifiable mix-net for long inputs," in *Proc. FC'06*, 2006, pp. 111–125.
[50] D. Adams and A.-K. Maier, *Big Seven Study: 7 Open Source Crypto-Messengers to be Compared*. Books on Demand, 2016.
[51] J. Bell, A. Gascon, B. Ghazi, R. Kumar, P. Manurangsi, M. Raykova, and P. Schoppmann, "Distributed, private, sparse histograms in the two-server model," in *Proc. CCS'22*, 2022, pp. 307–321.
[52] T. Wang, J. Blocki, N. Li, and S. Jha, "Locally differentially private protocols for frequency estimation," in *Proc. USENIX Security'17*, 2017, pp. 729–745.
[53] N. Li, M. Lyu, and D. Su, *Differential Privacy: From Theory to Practice*. Morgan & Claypool Publishers, 2016.
[54] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, "Local privacy and statistical minimax rates," in *Proc. FOCS'13*, 2013, pp. 429–438.
[55] P. Kairouz, K. Bonawitz, and D. Ramage, "Discrete distribution estimation under local privacy," in *Proc. ICML'16*, 2016, pp. 2436–2444.
[56] D. Kifer and A. Machanavajjhala, "A rigorous and customizable framework for privacy," in *Proc. PODS'12*, 2012, pp. 77–88.
[57] K. Thomas, D. McCoy, C. Grier, A. Kolcz, and V. Paxson, "Trafficking fraudulent accounts: The role of the underground market in twitter spam and abuse," in *Proc. USENIX Security'13*, 2013, pp. 195–210.
[58] 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 *Proc. NeurIPS'19*, 2019, pp. 13 657–13 664.
[59] S. Xu, Y. Zheng, and Z. Hua, "Camel: Communication-efficient and maliciously secure federated learning in the shuffle model of differential privacy," in *Proc. CCS'24*, 2024, pp. 243–257.
[60] A. Beimel, K. Nissim, and E. Omri, "Distributed private data analysis: Simultaneously solving how and what," in *Proc. CRYPTO'08*, 2008, pp. 451–468.
[61] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, and K. Ren, "Heavy hitter estimation over set-valued data with local differential privacy," in *Proc. CCS'16*, 2016, pp. 192–203.
[62] https://www.bouncycastle.org/, 2025.
[63] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from GPS trajectories," in *Proc. WWW'09*, 2009, pp. 791–800.
[64] D. Yang, D. Zhang, and B. Qu, "Participatory cultural mapping based on collective behavior data in location based social network," *ACM Trans. Intelligent Systems and Technology*, vol. 7, no. 3, pp. 30:1–30:23, 2016.
[65] G. Pass, A. Chowdhury, and C. Torgeson, "A picture of search," in *Proc. InfoScale'06*, 2006, pp. 1–7.
[66] https://www.kaggle.com/datasets/nicapotato/womens-ecommerce-clothing-reviews, 2018.
[67] H. Zhu, X. Tang, L. T. Yang, C. Fu, and S. Peng, "Key-value data collection and statistical analysis with local differential privacy," *Information Sciences*, vol. 640, pp. 1–18, 2023.
[68] P. Zhan, P. Tang, Y. Li, P. Wei, and S. Guo, "Poisoning attacks to local differential privacy for ranking estimation," *arXiv:2506.24033*, 2025.
[69] T.-W. Liao, C.-H. Lin, Y.-L. Tsai, T. Murakami, C.-M. Yu, J. Sakuma, C.-Y. Huang, and H. Kikuchi, "Data poisoning attacks to locally differentially private range query protocols," *arXiv:2503.03454*, 2025.
[70] P. Kairouz, S. Oh, and P. Viswanath, "The composition theorem for differential privacy," in *Proc. ICML'15*, 2015, pp. 1376–1385.
[71] A. Ghosh, T. Roughgarden, and M. Sundararajan, "Universally utility-maximizing privacy mechanisms," *SIAM J. Comput.*, vol. 41, no. 6, pp. 1673–1693, 2012.

APPENDIX A
BASIC NOTATIONS

Table I shows the basic notations in this paper.

APPENDIX B
MORE DETAILS ON THE BASELINES

A. Algorithmic Description of the LNF Protocol

Algorithm 2 shows an algorithmic description of the LNF protocol $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$. The `Count` function (line 11) calculates a count $\tilde{c}_i$ for each item $i \in [d]$ from $y_{\pi(1)}, \ldots, y_{\pi(\tilde{n})}$.

B. DP and Robustness of the CH Protocol

We show DP and the robustness of the CH protocol $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$:

Theorem 13. *If the binary input mechanism $\mathcal{M}_{\mathcal{D},\beta}$ in Definition 5 provides $(\frac{\varepsilon}{2}, \frac{\delta}{2})$-DP, then $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ provides (ε, δ)-CDP and is robust to collusion with users.*

Theorem 14. *Let $\lambda = \frac{n'}{n+n'}$ and $f_{\mathcal{T}} = \sum_{i \in \mathcal{T}} f_i$. $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ provides the following robustness against poisoning attacks:*

$$G_f^{\mathsf{max}} = \lambda(|\mathcal{T}| - f_{\mathcal{T}}). \tag{21}$$

C. Other Baselines

Below, we explain two other baselines than $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ and $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$:

User/Group-Dependent Hash Protocol. We can consider a variant of $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$ that uses a different hash function h_i for each user u_i (or each user group) to avoid the hash collision among users. However, this variant also results in low accuracy, as it needs to add dummy values for each hash function h_i and each hash value in $[b]$ to provide DP. In Appendix E-C, we show that this variant provides *worse* accuracy than $\mathcal{S}^{\mathsf{CH}}_{\mathcal{D},\beta}$.

Protocol in [34]. Yet another baseline is a protocol for large-domain data in [34], denoted by LWY22-Large. Specifically, LWY22-Large applies a hash function h_i different for each user u_i and then repeatedly adds dummy values to a tuple of a hash function and a hash value uniformly chosen from $\mathcal{H} \times [b]$. This protocol also does not work well. This is because it generates dummy values uniformly at random, which is shown to be ineffective in [18]. In Section VIII, we show that LWY22-Large does not provide high accuracy.

APPENDIX C
MORE DETAILS ON THE FME PROTOCOL

Toy Example of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$. Assume that $n = 5$, $d = 8$, $b = 4$, $(x_1, \ldots, x_5) = (2, 8, 4, 8, 2)$, $(h(x_1), \ldots, h(x_5)) = (1, 1, 3, 1, 1)$, $\beta = 1$, and the binomial distribution $B(2, 0.5)$ is used as $\mathcal{D}_1$ and $\mathcal{D}_2$. The shuffler adds $(z_1, z_2, z_3, z_4) = (1, 0, 1, 1)$ dummies for hash values. Then, the shuffled data are, e.g., $(y^H_{\pi(1)}, \ldots, y^H_{\pi(8)}) = (1, 3, 1, 4, 1, 1, 1, 3)$ and

TABLE I
BASIC NOTATIONS.

Symbol	Description
n	Number of users.
d	Number of items.
u_i	i-th user.
x_i	Input value of user u_i.
$\mathcal{X}$	Space of input data.
f_i	Frequency of item i in categorical data. $\mathbf{f} = (f_1, \ldots, f_d)$.
$\hat{f}_i$	Estimate of f_i. $\hat{\mathbf{f}} = (\hat{f}_1, \ldots, \hat{f}_d)$.
Φ_i	Frequency of key i in KV data. $\mathbf{\Phi} = (\Phi_1, \ldots, \Phi_d)$.
$\hat{\Phi}_i$	Estimate of Φ_i. $\hat{\mathbf{\Phi}} = (\hat{\Phi}_1, \ldots, \hat{\Phi}_d)$.
Ψ_i	Mean value of key i in KV data. $\mathbf{\Psi} = (\Psi_1, \ldots, \Psi_d)$.
$\hat{\Psi}_i$	Estimate of Ψ_i. $\hat{\mathbf{\Psi}} = (\hat{\Psi}_1, \ldots, \hat{\Psi}_d)$.
$\mathcal{T}$	Set of target items.
n'	Number of fake users.
λ	Fraction of fake users ($\lambda = \frac{n}{n+n'}$).
$G_f^{\max}$	Maximum gain in categorical data.
$G_\Phi^{\max}$	Maximum frequency gain in KV data.
$G_\Psi^{\max}$	Maximum mean gain in KV data.
C_{tot}	Expected number of bits sent from one party to another.
h	Hash function.
b	Range of hash function h.
$\mathcal{D}^*$	Dummy-count distributions ($\mathcal{D}^* = (\mathcal{D}_1, \mathcal{D}_2)$; $\mathcal{D}_1$ has mean μ_1 and variance σ_1^2; $\mathcal{D}_2$ has mean μ_2 and variance σ_2^2).
β	Sampling probability.
α	Significance level.
l	Maximum number of selected hashes.
Λ^H	Set of selected hash values after filtering.
Λ	Set of selected items after filtering.
κ	Padding length in the padding-and-sampling technique.

Input: Input values $(x_1, \ldots, x_n) \in [d]^n$, #items $d \in \mathbb{N}$, dummy-count distribution $\mathcal{D}$ (mean: μ, variance: σ^2), sampling probability $\beta \in [0, 1]$.
Output: Estimates $\hat{\mathbf{f}} = (\hat{f}_1, \cdots, \hat{f}_d)$.

/* Send input values */
foreach $i \in [n]$ **do**
  $[u_i]$ Send $E_{\mathsf{pk_d}}[x_i]$ to the shuffler;
end
/* Random sampling */
[s] Sample $E_{\mathsf{pk_d}}[x_i]$ $(i \in [n])$ with probability β;
/* Dummy data addition. */
foreach $i \in [d]$ **do**
  [s] $z_i \leftarrow \mathcal{D}$; Add a dummy $E_{\mathsf{pk_d}}[i]$ for z_i times;
end
/* Random shuffling */
[s] Let $y_1, \ldots, y_{\tilde{n}} \in [d]$ the selected input values and dummies. Sample a random permutation π over $[\tilde{n}]$;
/* Send shuffled values */
[s] Send shuffled values $(E_{\mathsf{pk_d}}[y_{\pi(1)}], \ldots, E_{\mathsf{pk_d}}[y_{\pi(\tilde{n})}])$ to the data collector;
/* Compute an unbiased estimate */
[d] Decrypt $y_{\pi(1)}, \ldots, y_{\pi(\tilde{n})}$;
[d] $(\tilde{c}_1, \ldots, \tilde{c}_d) \leftarrow \mathtt{Count}(y_{\pi(1)}, \ldots, y_{\pi(\tilde{n})})$;
foreach $i \in [d]$ **do**
  [d] $\hat{f}_i \leftarrow \frac{1}{n\beta}(\tilde{c}_i - \mu)$;
end
return $\hat{\mathbf{f}} = (\hat{f}_1, \cdots, \hat{f}_d)$

Algorithm 2: LNF protocol $\mathcal{S}^{\mathsf{LNF}}_{\mathcal{D},\beta}$ [18]. $[u_i]$, [s], and [d] represents that the process is run by user u_i, the shuffler, the data collector. $\mathsf{pk_d}$ represents a public key of the data collector.

$(y_{\pi(1)}, \ldots, y_{\pi(8)}) = (8, 4, 2, \perp, 8, \perp, 2, \perp)$. The data collector filters items and selects $\Lambda = \{2, 8\}$ and $\Lambda_H = \{1\}$. In this case, the shuffled input values become $(y_{\pi(1)}, \ldots, y_{\pi(8)}) = (8, \perp, 2, \perp, 8, \perp, 2, \perp)$ (4 is replaced with $\perp$). Finally, the shuffler removes $\perp$ generated by the shuffler and adds $(z_2, z_8) = (1, 2)$ dummies for input values. Then, the shuffled input values are, e.g., $(y^*_{\rho(1)}, \ldots, y^*_{\rho(8)}) = (2, 8, \perp, 8, 2, 8, 2, 8)$, and the estimate $\hat{\mathbf{f}}$ is $\hat{\mathbf{f}} = (0, 0.4, 0, 0, 0, 0, 0, 0.6)$.

On the Composition in $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$. Below, we explain that the basic composition theorem in Theorem 6 is almost tight. To show this, we consider the optimal composition theorem [70]. This theorem provides the tightest bound when each sub-mechanism provides (ε, δ)-DP. However, even this theorem cannot improve the bound in Theorem 6, as our protocol composes *only two* sub-mechanisms – one outputting hash values and the other outputting input values. Specifically, assume that each sub-mechanism provides $(\frac{\varepsilon}{2}, \frac{\delta}{2})$-DP, i.e., $\varepsilon_1 = \varepsilon_2 = \frac{\varepsilon}{2}$ and $\delta_1 = \delta_2 = \frac{\delta}{2}$ in Theorem 6. Then, the theorem in [70] states that the entire protocol provides $(\varepsilon, \delta - (\frac{\delta}{2})^2)$-DP (or $(0, \delta')$-DP with an extremely large δ'), which is almost equivalent to (ε, δ)-DP. Thus, the basic composition theorem is almost tight in our case.

APPENDIX D
ε IN DP AND PRIVACY GUARANTEES

Below, we show the relationship between ε in DP and privacy guarantees against the inference of input values through hypothesis testing interpretations [70]. Specifically, DP considers two neighboring databases, D and D', that differ in the input value of the victim (see Definition 1). Thus, given an output Y of a randomized algorithm $\mathcal{M}$, we can define the following hypotheses: H_0: "Y came from D."; H_1: "Y came from D'." Assume that, given D and D', the attacker who obtains Y guesses which of H_0 or H_1 is correct. The attacker may choose H_1 when H_0 is true (type I error). Conversely, the attacker may choose H_0 when H_1 is true (type II error). Let $p_I, p_{II} \in [0, 1]$ be the probabilities of type I and II errors, respectively. Then, DP is closely related to p_I and p_{II}:

Theorem 15 ([70])**.** *A randomized algorithm $\mathcal{M}$ provides (ε, δ)-DP if and only if the following inequalities holds for any neighboring databases D and D' and any $Y \in \mathrm{Range}(\mathcal{M})$:*

$$p_I + e^{\varepsilon} p_{II} \geq 1 - \delta, \quad e^{\varepsilon} p_I + p_{II} \geq 1 - \delta. \tag{22}$$

Theorem 15 states that (ε, δ)-DP is equivalent to lower bounding type I and II errors by (22). Fig. 10 shows the relationship between ε and the lower bound on the error probability p^* when the type I and II errors are equal, i.e., $p^* = p_I = p_{II}$. Note that p^* is the error probability when the attacker is given two candidates for the victim's input value. The inference of the victim's input value is much more difficult in practice, as there are d ($\gg 2$) candidates for it.

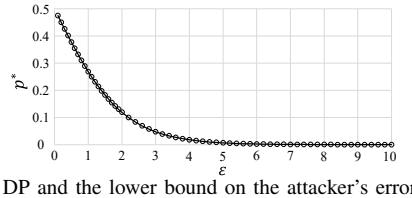

Fig. 10. ε in DP and the lower bound on the attacker's error probability p^* $(= p_I = p_{II})$ obtained from (22) ($\delta = 10^{-12}$).

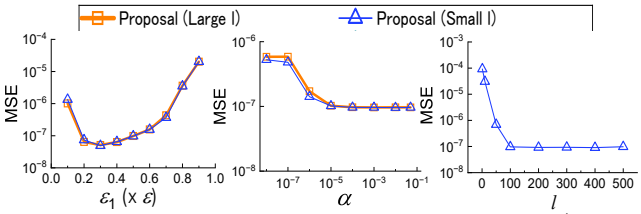

Fig. 11. MSE of our FME protocol when changing parameters ε_1 $(= \varepsilon - \varepsilon_2)$, α, and l in the Foursquare dataset ($\varepsilon = 1$, $\delta = 10^{-12}$).

Fig. 10 shows that when ε is close to 0, p^* is close to 0.5 (the error probability of a random guess). In contrast, when $\varepsilon \geq 5$, p^* is almost zero. This means that DP cannot provide strong privacy guarantees in this case, as claimed in [53]. Taking this into account, we varied ε from 0.1 to 5 in Section VIII.

Appendix E
Additional Experiments

A. Changing Parameters in the FME Protocol

We evaluated the MSE of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ when we changed parameters ε_1 $(= \varepsilon - \varepsilon_2)$, α, and l in the Foursquare dataset. Here, we set $\varepsilon_1 = 0.5\varepsilon$, $\alpha = 0.05$, and $l = \max\{\frac{n^2}{d}, 50\}$ as default values and changed each parameter while fixing the others.

Fig. 11 shows the results. The MSE is roughly the same when ε_1 is between 0.2ε and 0.5ε, which indicates that this range of ε_1 can balance the trade-off between the noise for hash values and the noise for input values. Fig. 11 also shows that the MSE rapidly increases as we decrease α and l from 10^{-6} and 50, respectively. This is because $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ filters out almost all items in this case. Our suggestion is to avoid such extreme settings; e.g., if we are interested in the frequency of top-k items, α and l should be: $\alpha \geq 10^{-5}$ and $l \geq k$.

Note that $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ also has the sampling probability β as a parameter. [18] shows that the communication cost is improved by reducing β. However, it comes at the cost of accuracy. Since we can significantly improve the communication cost by setting l small, our suggestion is to set $\beta = 1$.

B. Existing Defenses against Collusion and Poisoning Attacks

We evaluated the existing defenses against collusion and poisoning attacks not evaluated in Section VIII.

Collusion Attacks. For collusion attacks, we evaluated a defense in [17], which adds dummies uniformly at random from the domain of noisy data. This defense increases the MSE by $(1+\upsilon)^2$ times by adding υn dummies, where $\upsilon \in \mathbb{R}_{\geq 0}$. We set $\upsilon = 0.5$, in which case the MSE is increased by 2.25 times. We applied the defense in [17] to the existing protocols and evaluated the relationship between the actual ε and the ratio $|\Omega|/n$ of colluding users using the Foursquare dataset.

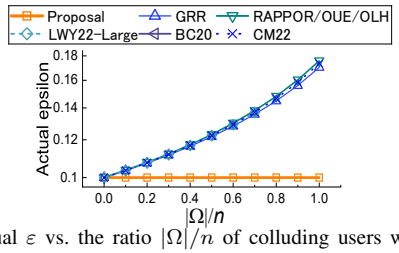

Fig. 12. Actual ε vs. the ratio $|\Omega|/n$ of colluding users when the defense in [8] is applied to the existing protocols in the Foursquare dataset ($\varepsilon = 0.1$, $\delta = 10^{-12}$).

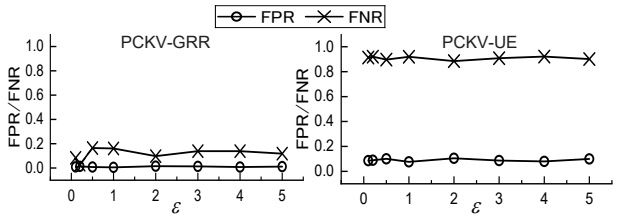

Fig. 13. FPR and FNR of the defense in [20] in the Amazon dataset ($\lambda = 0.1$, $\delta = 10^{-12}$).

Fig. 12 shows the results. The actual ε still increases with an increase in $|\Omega|/n$, which indicates that the defense in [17] is insufficient as a defense against collusion attacks.

Poisoning Attacks. For poisoning attacks, we evaluated a defense in [20]. The defense in [20] detects fake users based on isolation forest in KV statistics estimation. Note that the isolation forest simply divides users into two groups. Since we can set the size of each group in the isolation forest, we assume that the data collector knows the number n' of fake users, divides the users into a group with n' users and another group, and treats the former group as a fake group. Following [20], we used the FPR (False Positive Rate) and FNR (False Negative Rate) as performance measures. The FPR (resp. FNR) is the ratio of genuine users decided as fake (resp. fake users decided as genuine). We evaluated the FPR and FNR of the defense in [20] applied to PCKV-GRR/UE using the Amazon dataset.

Fig. 13 shows the results. Although the FPR and FNR are low for PCKV-GRR, the FNR is very high for PCKV-UE. This is because, for PCKV-UE, the M2GA sets bits corresponding to the target keys to 1 and randomly samples other bits to evade detection [20]. Our result shows that the effectiveness of the defense in [20] is limited for PCKV-GRR. This is consistent with the experimental results in [20].

C. User/Group-Dependent Hash Protocol

We also evaluated the UH (User-Dependent Hash) and GH (Group-Dependent Hash) protocols.

UH/GH Protocol. Below, we explain the GH protocol, as it is more general than the UH protocol. Let $g \in [n]$ be the number of groups. For each $i \in [g]$, the data collector randomly selects a hash function $h_i : [d] \to [b]$ from $\mathcal{H}$. Then, the data collector selects a group ID $r_i \in [g]$ for each user u_i $(i \in [n])$ and sends r_i and h_{r_i} to u_i. User u_i $(i \in [n])$ sends a pair $\langle r_i, h_{r_i}(x_i)\rangle$ of the group ID and the hash value to the shuffler.

The shuffler randomly selects each pair with probability β. For each group ID $j \in [g]$ and each hash value $k \in [b]$,

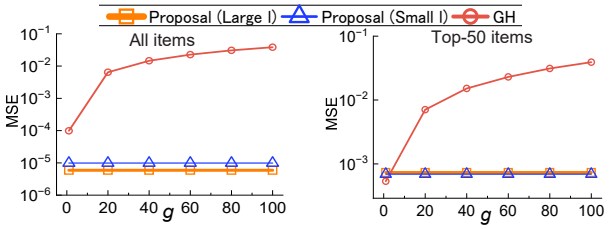

Fig. 14. MSE of our proposal and the CH/GH/UH protocol in the Foursquare dataset (left: MSE over all items, right: MSE over the top-50 items). The CH and UH protocols are the GH protocol with $g = 1$ and 100, respectively.

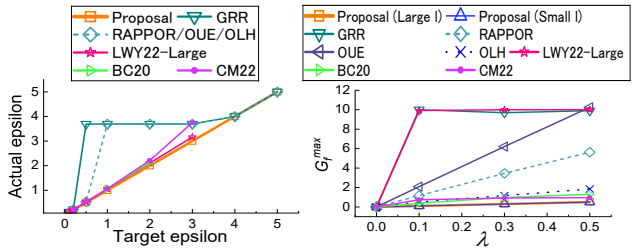

Fig. 15. Robustness against collusion and poisoning attacks when varying the target ε and the fraction λ of fake users in the Foursquare dataset (left: $|\Omega|/n = 0.1$, $\delta = 10^{-12}$; right: $\varepsilon = 1$, $\delta = 10^{-12}$).

the shuffler randomly generates z_{jk} from the dummy-count distribution $\mathcal{D}$ ($z_{jk} \sim \mathcal{D}$) and adds a dummy pair $\langle j, k\rangle$ for z_{jk} times. The shuffler randomly shuffles the selected hash values and dummies to the data collector. Finally, the data collector calculates an unbiased estimate $\hat{f}_i$ ($i \in [d]$) of f_i as: $\hat{f}_i = \frac{b}{n\beta(b-1)}(\sum_{(j,k)\in S_i} \tilde{c}_{jk} - \frac{n\beta}{b} - g\mu)$, where $\tilde{c}_{jk}$ is the number of $\langle j, k\rangle$ in the shuffled data, and $S_i = \{(j, k)|h_j(i) = k\}$, i.e., the set $\langle j, k\rangle$ of pairs that could be produced from item i.

Note that this protocol is equivalent to a protocol that independently applies the CH protocol to each group. Thus, the GH protocol inherits the theoretical properties of the CH protocol, e.g., (ε, δ)-CDP. The UH protocol is a special case of the GH protocol where $g = n$ and $r_i = i$ ($i \in [n]$).

Experimental Results. We compared our proposals with the UH and GH protocols using the Foursquare dataset. Note that the GH protocol is inefficient when g is large. To reduce the run time, we randomly selected $n = 100$ users and divided the city into 100×100 regions ($d = 10000$) at regular intervals. Then, we evaluated the MSE over all items (regions) and the MSE over the top-50 items while changing g in the GH protocol from 1 to 100. Note that the CH and UH protocols are the GH protocol with $g = 1$ and 100, respectively.

Fig. 14 shows the results. In the GH protocol, the MSE increases as g increases. This is because the GH protocol needs to add dummies for each group. In addition, the CH protocol (i.e., GH with $g = 1$) suffers from a large MSE over all items. This is because the CH protocol has poor accuracy for unpopular items due to the hash collision. The poor accuracy cannot be improved by using user/group-dependent hashes.

D. Changing Parameters in Collusion and Poisoning Attacks

We evaluated the robustness against collusion and poisoning attacks when varying the target ε and the fraction λ of fake users in the Foursquare dataset. Fig. 15 shows the results.

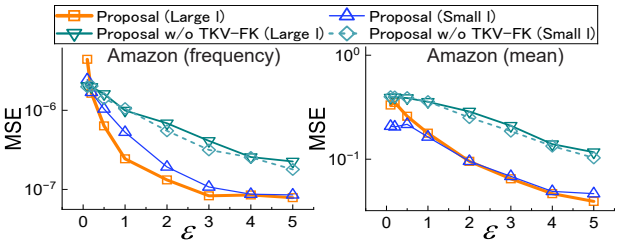

Fig. 16. MSE of our proposals with/without the TKV-FK technique in the Amazon dataset ($\delta = 10^{-12}$).

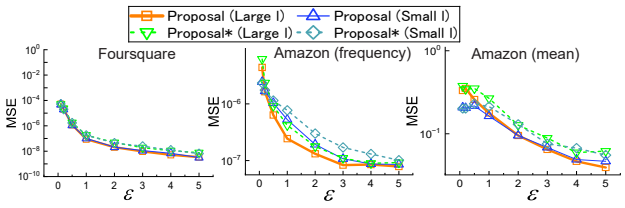

Fig. 17. MSE of our proposals with additional noise in the Foursquare and Amazon datasets ($\delta = 10^{-12}$).

Fig. 15 shows that the actual ε rapidly increases as the target ε increases in the existing protocols. In contrast, the actual ε is always the same as with the target ε in our FME protocol, demonstrating robustness against collusion attacks. Our proposals also achieve the smallest gain $G_f^{\max}$ and is the most robust to poisoning attacks for all values of λ.

E. Effectiveness of TKV-FK

For an ablation study, we evaluated our proposal *with* our TKV-FK technique (Fig. 5(b)) and our proposal *without* our TKV-FK technique (Fig. 5(a)) in the Amazon dataset.

Fig. 16 shows the results. The MSE for means is significantly improved by introducing TKV-FK, demonstrating the effectiveness of TKV-FK for reducing bias. The MSE for frequencies is also significantly reduced by introducing TKV-FK. This is because our proposal with TKV-FK effectively finds popular items by filtering data at a key level.

F. Our Proposals with Additional Noise

Theorem 6 assumes that the shuffler does not disclose dummies after running the protocol. However, even if the shuffler discloses dummies, we can guarantee DP for the outputs of $\mathcal{S}^{\mathsf{FME}}_{\mathcal{D}^*,\beta}$ by adding additional DP noise before publishing them. We denote this modified protocol by Proposal*.

Specifically, Proposal* adds noise randomly generated from the two-sided geometric distribution $\mathsf{Geo}(e^{-\varepsilon/4})$ [71] with parameter $e^{-\varepsilon/4}$ to each of the counts $\tilde{c}_1^H, \ldots, \tilde{c}_b^H$ (cf. Algorithm 1, line 11). Since the sensitivity of the count is 2, adding noise from $\mathsf{Geo}(e^{-\varepsilon/4})$ provides $\frac{\varepsilon}{2}$-DP for Λ. Similarly, Proposal* adds noise from $\mathsf{Geo}(e^{-\varepsilon/4})$ to each of the counts $\tilde{c}_1, \ldots, \tilde{c}_d$ (cf. Algorithm 1, line 26) to provide $\frac{\varepsilon}{2}$-DP for $\mathbf{f}$. Then, publishing Λ and $\mathbf{f}$ provides ε-DP, even if the shuffler discloses dummies after running the protocol.

We evaluated the MSE of this protocol using the Foursquare and Amazon datasets. Fig. 17 shows the results. The accuracy is hardly affected by introducing additional noise.

APPENDIX F
ARTIFACT APPENDIX

Our source code is a Java implementation of the proposed FME (Filtering-with-Multiple-Encryption) protocol. The purpose of this source code is to reproduce our main results (Fig. 6) using the categorical (Foursquare) and key-value (Amazon) datasets.

A. Description & Requirements

1) How to access: Our source code can be downloaded from Zenodo: https://doi.org/10.5281/zenodo.17032669 (DOI: 10.5281/zenodo.17032669). The directory structure of this repository is as follows:

- **data/** – Place raw datasets here (currently empty).
- **dataset/** – Preprocessed datasets (currently empty). The preprocessed files will be automatically stored in the dataset/ directory after running DataPreprocessing.
- **lib/** – External libraries (JAR files) (currently empty).
- **src/** – Java source code.
- **LICENSE.txt** – MIT license.
- **README.md** – README file.

2) Hardware dependencies: None.

3) Software dependencies: Our code needs the following libraries:

- **Apache Commons Math 3.6.1**[4].
- **Bouncy Castle**[5].

4) Benchmarks: Our code uses the following datasets:

- **Global-scale Check-in Dataset**[6].
- **User Profile Dataset**[6].
- **Amazon - Ratings (Beauty Products)**[7].

The first two are the Foursquare dataset, and the last one is the Amazon dataset. The Foursquare dataset contains categorical data, whereas the Amazon dataset contains key-value data.

B. Artifact Installation & Configuration

Install and compile our code as follows:

1) Clone the repository:

```
$ git clone https://github.com/
    FilteringMultipleEncryption/
    FilteringMultipleEncryption.git
$ cd FilteringMultipleEncryption
```

2) Place the following libraries in the lib/ directory:
   - **Apache Commons Math 3.6.1**[4]: Download commons-math3-3.6.1-bin.zip or .tar.gz from the Apache archive and decompress it. The archive contains commons-math3-3.6.1.jar. Place it into the lib/ directory. (NOTE: Commons Math 4.x uses a different package structure and is incompatible with our code.)
   - **Bouncy Castle**[5]: Our code works with bcprov-jdk18on-1.81.jar. Place it into the lib/ directory.
3) Compile our code:

```
$ javac -cp "lib/*" -d bin src/data/*.java
    src/encryption/*.java src/fme/*.java
    src/hash/*.java src/sageo/*.java src/
    util/*.java
```

[4]https://archive.apache.org/dist/commons/math/binaries/
[5]https://www.bouncycastle.org/
[6]https://sites.google.com/site/yangdingqi/home/foursquare-dataset
[7]https://www.kaggle.com/datasets/skillsmuggler/amazon-ratings

TABLE II
ENTRY POINTS.

Data Type	Entry Point	Description
Categorical	fme.CategoricalFME	Evaluate the FME protocol for categorical data.
Key-Value	fme.KeyValueFME	Evaluate the FME protocol for key-value data.

TABLE III
ARGUMENTS IN EACH ENTRY POINT.

Argument	Description
DataConfig	Dataset name.
epsilon	ε in differential privacy.
delta	δ in differential privacy.
alpha	Significance level.
beta	Sampling probability.
topK	Evaluate the MSE of the top-K frequent items.
encryption	Encryption mode (RSA or ECIES).
isLargeL	Use "Proposal (Large l)" if true.
seed	Random seed (optional).

C. Major Claims

- (C1): Our FME protocol "Proposal (Large l)" with $\varepsilon = 1$ achieves the MSE of about 1.6×10^{-7} in the Foursquare dataset. This is proven by the experiment (E1) whose results are shown in Fig. 6 "Foursquare".
- (C2): Our FME protocol "Proposal (Large l)" with $\varepsilon = 1$ achieves the frequency MSE of about 2.8×10^{-7} and the mean MSE of about 0.17 in the Amazon dataset. This is proven by the experiment (E2) whose results are shown in Fig. 6 "Amazon (frequency)" and "Amazon (mean)".

D. Evaluation

Table II shows an entry point in our code for each data type (i.e., categorical or key-value). Each entry point has arguments shown in Table III. Based on them, we explain how to conduct our experiments.

1) Experiment (E1): [Foursquare Dataset] [30 human-minutes + 1 compute-minute]: First, we explain our experiments using the Foursquare dataset. After downloading the datasets, our code can be easily run and does not take time; our estimate is 1 minute even on a laptop. The expected results are: the MSE of about 1.6×10^{-7} for "Proposal (Large l)" with $\varepsilon = 1$.

[How to] Below, we explain how to conduct our experiments using the Foursquare dataset.

[Preparation] Download the following datasets into data/:

- **Global-scale Check-in Dataset** (dataset_TIST2015 .zip)[6].
- **User Profile Dataset** (dataset_UbiComp2016.zip)[6].

Then, preprocess the dataset as follows:
On Windows:

```
$ java -cp "lib/*;bin" util.DataPreprocessing
    foursquare
```

On Linux/MacOS:

```
$ java -cp "lib/*:bin" util.DataPreprocessing
    foursquare
```

[Execution] Run the evaluation code as follows:
On Windows:

```
$ java -cp "lib/*;bin" fme.CategoricalFME
    foursquare 1.0 1E-12 0.05 1.0 50 RSA true
    100
```

On Linux/MacOS:

```
$ java -cp "lib/*:bin" fme.CategoricalFME
    foursquare 1.0 1E-12 0.05 1.0 50 RSA true
    100
```

[Results] Then, the following results will be output to the console:

```
Frequency MSE: 1.5896099937481953E-7
```

This value – about 1.6×10^{-7} – is the MSE of "Proposal (Large l)" with $\varepsilon = 1.0$. We expect that the same results will be output to your console, as the above execution command fixes the random seed (= "100") by setting the 9th argument to "100".

2) Experiment (E2): [Amazon Dataset] [30 human-minutes + 1 compute-minute]: Second, we explain our experiments using the Amazon dataset. Note that we need to log in to the Kaggle to download the Amazon dataset. As with (E1), our code does not take time. The expected results are: the frequency MSE of about 2.8×10^{-7} and the mean MSE of about 0.17 for "Proposal (Large l)" with $\varepsilon = 1$.

[How to] Below, we explain how to conduct our experiments using the Amazon dataset.

[Preparation] Download the following datasets into data/ (please log in to the Kaggle to download the dataset):

- **Amazon - Ratings (Beauty Products)** (ratings _Beauty.csv)[7].

Then, preprocess the dataset as follows:
On Windows:

```
$ java -cp "lib/*;bin" util.DataPreprocessing
    amazon
```

On Linux/MacOS:

```
$ java -cp "lib/*:bin" util.DataPreprocessing
    amazon
```

[Execution] Run the evaluation code as follows:
On Windows:

```
$ java -cp "lib/*;bin" fme.KeyValueFME amazon
    1.0 1E-12 0.05 1.0 50 RSA true 100
```

On Linux/MacOS:

```
$ java -cp "lib/*:bin" fme.KeyValueFME amazon
    1.0 1E-12 0.05 1.0 50 RSA true 100
```

[Results] Then, the following results will be output to the console:

```
Frequency MSE: 2.830363096692403E-7
Mean MSE: 0.1714257324077271
```

These values – about 2.8×10^{-7} and 0.17 – are the frequency MSE and the mean MSE of "Proposal (Large l)" with $\varepsilon = 1.0$. We expect that the same results will be output to your console.

E. Customization

For each data type, we can change the value of ε by changing the 2nd argument (from "1.0" to the desired value). We can evaluate "Proposal (Small l)" by changing the 8th argument from "true" to "false". The 9th argument ("100" in the above examples) is optional and can be omitted. We plotted Fig. 6 "Proposal (Large l)" and "Proposal (Small l)" in the Foursquare and Amazon datasets by changing ε from 0.1 to 5.0 and averaging the MSE over 10 runs.