This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2812146, IEEE Transactions on Information Forensics and Security

LoPub: High-Dimensional Crowdsourced Data Publication with Local Differential Privacy

Xuebin Ren, Chia-Mu Yu, Weiren Yu, Shusen Yang, Xinyu Yang, Julie A. McCann, and Philip S. Yu

***Abstract*—High-dimensional crowdsourced data collected from numerous users produces rich knowledge about our society. However, it also brings unprecedented privacy threats to the participants. Local differential privacy (LDP), a variant of differential privacy, is recently proposed as a state-of-the-art privacy notion. Unfortunately, achieving LDP on high-dimensional crowdsourced data publication raises great challenges in terms of both computational efficiency and data utility. To this end, based on Expectation Maximization (EM) algorithm and Lasso regression, we first propose efficient multi-dimensional joint distribution estimation algorithms with LDP. Then, we develop a Local differentially private high-dimensional data Publication algorithm, LoPub, by taking advantage of our distribution estimation techniques. In particular, correlations among multiple attributes are identified to reduce the dimensionality of crowdsourced data, thus speeding up the distribution learning process and achieving high data utility. Extensive experiments on real-world datasets demonstrate that our multivariate distribution estimation scheme significantly outperforms existing estimation schemes in terms of both communication overhead and estimation speed. Moreover, LoPub can keep, on average, 80% and 60% accuracy over the released datasets in terms of SVM and random forest classification, respectively.**

***Index Terms*—local differential privacy, high-dimensional data, crowdsourced data, data publication, private data release**

I. Introduction

WITH the development of various integrated sensors and crowd sensing systems [26], crowdsourced information from all aspects can be collected and analyzed to produce rich knowledge about the group [22], [46], which can benefit everyone in the crowdsourced system [27]. Particularly, with multi-dimensional crowdsourced data (data records with multiple attributes), a huge amount of potential information and patterns behind the data can be mined or extracted to provide accurate dynamics and reliable prediction for both group and individuals [29]. For example, various genomic data (each gene as one dimension) from a large population of patients can be analyzed to better diagnose and monitor patients' health status [33]. Users' electricity usages in one day, as a typical high dimensional data (each time slot as one dimension), can be aggregated to obtain energy consumption dynamics, thus making better demand response for the smart grid [39].

Despite the usefulness of crowdsourced information, the massive data collection presents serious privacy concerns. End-to-end encryption of data incurs operational limitations on ciphertexts, thus significantly degrading the functionality of crowd sensing systems. Differential privacy (DP) [17] has been a de facto standard for privacy protection. Unfortunately, the participants' privacy can still be easily inferred or identified due to the publication of crowdsourced data [20], [43], especially high-dimensional data, even with the consideration of existing privacy-preserving schemes (e.g., DP). The reasons for privacy leaks are two-fold:

- *Non-Local Privacy*. Most existing solutions for privacy protection focus only on centralized datasets under the assumption that the server is trusted. However, an individual's data may still suffer from privacy leakage before aggregation because of the lack of proper local protection for the data on the user side [13], [24].
- *Curse of High-dimensionality*. With the increase of data dimensions, privacy preservation techniques like DP [14], [17], if naïvely applied to individual attributes with high correlations, will either be weakened [31], [45], thereby increasing the success ratio of many reference attacks like cross-checking, or lead to low-quality data synthesis. Even worse, the privacy guarantee of DP degrades exponentially when multiple correlated queries are processed. From the aspect of utility, conventional DP algorithms can hardly achieve reasonable scalability and desirable data accuracy due to the attribute correlations [45].

In addition to privacy vulnerability, the efficiency and implementation complexity of privacy preservation techniques are also concerns. For example, in IoT applications, the ubiquitous but resource-constrained sensors can only afford lightweight operations. Another example is that privacy-preserving real-time pricing mechanisms require not only effective privacy guarantees for individuals' electricity usage but also fast response to the dynamical changes of demands and supply in the smart grid [30].

Local differential privacy (LDP), a variant of DP, is recently proposed as a state-of-the-art privacy notion. LDP is particularly useful in distributed environment, where each user contributes the single private data record to an untrusted server. Compared to the conventional DP that finds very few large-scale real-world deployments, LDP has found its practical value in collecting user statistics without violating user privacy. For example, RAPPOR [18] is a Google Chrome extension that constantly collects Windows process names and Chrome Homepages from user devices in a LDP manner. Apple announces in WWDC 2016 its implementation of LDP in iOS 10 and MacOS for discovering popular emojis and identifying high energy and memory usage in Safari. Microsoft also deploys an LDP-enabled data collection mechanism in Windows Insiders program to collect application usage statistics. Both users and software companies can benefit from the LDP deployment; users have obvious need of user privacy. For companies, the appreciation of user privacy may gain positive reputation. More importantly, intruders or malicious insiders of the system may be able to retrieve or even steal the user data, violating user privacy. With the LDP deployment, since even the server does not possess the raw data, the server has very limited responsibility for privacy leakage.

All the above LDP implementations in fact only support frequency estimation; i.e., the server can learn the proportion of users with particular property in a population. Nonetheless, in reality, each user is usually associated with multiple attributes and the server is interested in, e.g., learning the correlation between attributes or releasing a privacy-preserving

X. Ren, S. Yang, and X. Yang are with Xi'an Jiaotong University ({xuebinren, shusenyang, yxyphd}@mail.xjtu.edu.cn). C.-M. Yu is with National Chung Hsing University and Taiwan Information Security Center (TWISC@NCHU) (chiamuyu@gmail.com). W. Yu is with Imperial College London and Aston University (weiren.yu@imperial.ac.uk, w.yu3@aston.ac.uk). J. McCann is with Imperial College London (j.mccann@imperial.ac.uk). P. Yu is with University of Illinois at Chicago (psyu@uic.edu)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2812146, IEEE Transactions on Information Forensics and Security

approximate dataset to the third-party for further analysis. Hence, it is desirable to have a design of LDP-enabled data synthesis mechanism to meet various requirements of privacy-preserving data analysis.

Contributions. Our contributions can be summarized as follows.

- Based on EM and Lasso regression, we propose three efficient algorithms for multivariate joint distribution estimation under the circumstance that each user individually reports the data in a local differentially private manner.
- We propose LoPub, a total solution that can generate an approximation of the original crowdsourced data with the guarantee of LDP, by taking advantage of marginal distributions learned from the data after a nontrivial design of efficient dimensionality and sparsity reduction.
- We implemented and evaluated LoPub on real-world datasets. Experimental results demonstrate the efficiency and effectiveness of our proposed distribution estimation and data synthesis mechanisms.

To the best of our knowledge, this is the first work particularly addressing high-dimensional crowdsourced data publication with LDP. We have a comparison among LoPub and three similar solutions in TABLE I. One can see that LoPub reaches lower communication cost, time, and storage complexity. Due to the page limit, some detailed examples, proofs and explanations that are not presented in this paper can be found in our full length technical report [37].

TABLE I: Comparison of LoPub with existing methods

Comparison	LoPub (Our method)	RAPPOR [18]	EM [19]	JTree [10]
LDP	Y	Y	Y	N
High Dimension	Y	N	N	Y
Communication	$O(\sum_j \lvert\Omega_j\rvert)$	$O(\prod_j \lvert\Omega_j\rvert)$	$O(\sum_j \lvert\Omega_j\rvert)$	-
Time Complexity	Low	Large	Large	-
Space Complexity	Low	Large	Large	-

⋆ $|\Omega_j|$ is the domain size of the j-th dimension.

II. Related Work

A. *Differential Privacy in Centralized Setting*

Differential privacy (DP) [14], [17], originally developed for interactive query-response system, forms a mathematical foundation for privacy protection by appropriately randomising the results of statistical queries, using distributions such as the Laplace, Gaussian or Geometric distributions. A special form of DP is *non-interactive* DP, which corresponds to releasing the sanitized dataset, or say, privacy-preserving data publication. For privacy-preserving low-dimensional data publication, to show crowd statistics and to draw the correlations between attributes, both the differentially private histogram (univariate distribution) [3] and contingency table [34] are widely investigated.

However, the techniques for non-interactive DP [15], [16] suffer from the "curse of dimensionality" [10], [45]. In other words, they cannot reach either better utility (SVM and random forest classification accuracy rates as utility metrics in our consideration) or reasonable scalability due to the correlations among attributes.

To deal with the correlations in high-dimensional data, different schemes (e.g., approximations via low dimensional data clusters) have been proposed [10], [11], [25], [28], [41], [45]. For example Chen et al. [10] propose to reduce the dimension by using junction tree algorithm to model the correlations.

B. *Differential Privacy in Distributed Setting*

The schemes mentioned above mainly deal with centralized datasets. Nonetheless, there could be scenarios, where distributed users contribute to the aggregate statistics. Some efforts are also devoted to DP guarantee in distributed environment. For example, Su et al. [40] proposed a multi-party setting to publish synthetic dataset from multiple data owners. However, their multi-party computation can only protect privacy among data owners. Ács and Castellucia [4] and Bindschaedler et al. [8] use noise partitioning technique to mimic the situation that the Laplace-distributed noise sample applies to the aggregate data. Nonetheless, the design of these schemes must involve sophisticated key management or even homomorphic encryption, resulting in the impracticality and inefficiency of these schemes. Despite the privacy protection against difference and inference attacks from aggregate queries, an individual's data may also suffer from privacy leakage before aggregation [17]. Hence, *local differential privacy (LDP)* [9], [13], [23], [24] has been proposed to provide individual privacy guarantees for distributed users. Fig. 1 and Fig. 2 illustrate the difference between conventional differential privatization procedures and local privatization procedures.

The problem that can be solved naturally in a LDP manner is frequency estimation. Randomized response (RR), where the user responds with either faithful or opposite answer depending on coin flipping, is the simplest technique for LDP-enabled frequency estimation. RR is originally designed for binary answer, but can be easily extended to categorical answer with the degrading response usefulness. RAPPOR [18] encodes the data as a Bloom filter and then performs RR on each bit of Bloom filter. The design of RAPPOR enables the server to have an accurate decoding result. While RAPPOR can be seen as collecting users' 1-dimensional data, Fanti et al. [19] propose an association learning scheme, which extends the 1-dimensional RAPPOR to estimate the 2-dimensional joint distribution. However, the sparsity in the multi-dimensional domain and the way it iteratively scans RAPPOR strings mean that it will incur considerable computational complexity. In addition, while the communication cost of each RAPPOR instance is the size of Bloom filter, Bassily and Smith [7] propose an 1-bit protocol for LDP-enabled frequency estimation with the optimal communication efficiency. Wang et al. [42] introduce a framework that can generalize the above protocols by reconciling the RR behaviors in different frequency estimation schemes [7], [18], [19], such that one may derive more accurate statistics. In addition to frequency estimation, there are LDP protocols for the other functionalities. For example, with the consideration of a prefix tree and a user grouping based on the length of the random prefix of data, Bassily et al. [6] develop an efficient way to querying the frequency estimation mechanism, so as to find out heavy hitters. The above all assume that each user is only in possession of single data element. Qin et al. [35] propose a heavy hitter estimation over set-valued data based on two-round user-server interactions. The design of LDP-enabled data collection can also be generalized to the case where the user owns the frequently changing private data [12] and to the case where the user owns a subgraph induced by a specific vertex from a graph [36].

III. System Model

Our system model is depicted in Fig. 3, where a number of users and an honest-but-curious central server constitute a crowdsourcing system. Users generate multi-dimensional data records, and then send these data to the server in a LDP manner. The server aims to release an approximate dataset to third-parties for conducting data analysis. The server is assumed to be able to collude with certain users, attempting to infer others' data. In addition, the server and users share the same public information, such as the privacy-preserving protocols (including the cryptographic hash functions used).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2812146, IEEE Transactions on Information Forensics and Security

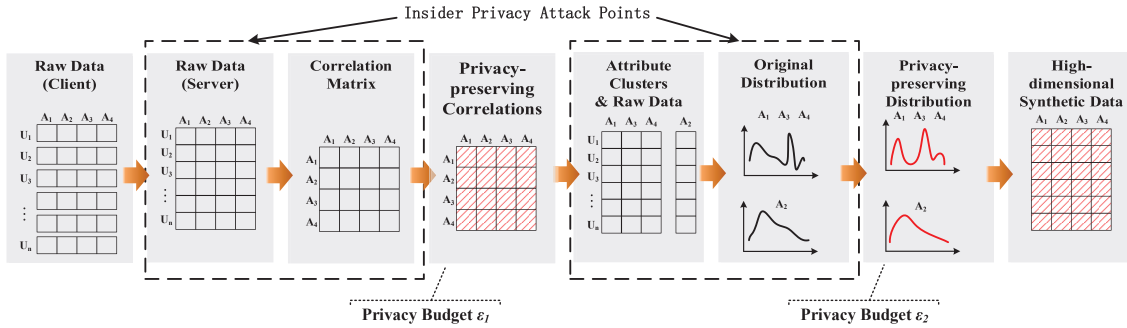


Fig. 1: Main procedures of high-dimensional data publishing with **Non-local** $\epsilon = \epsilon_1 + \epsilon_2$ **DP**

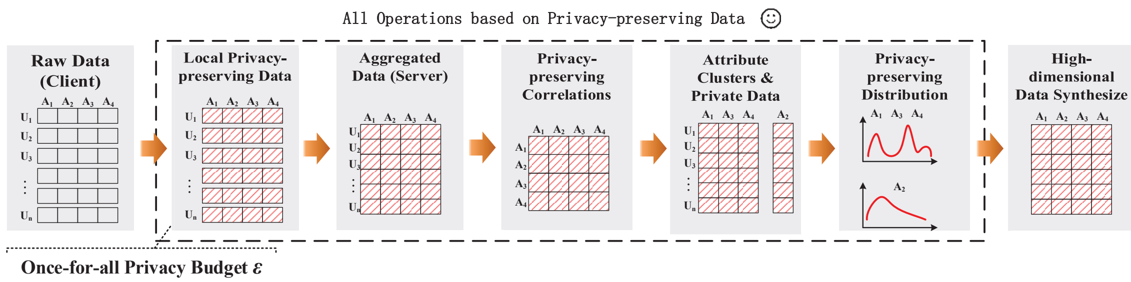


Fig. 2: Main procedures of high-dimensional data publishing with ϵ-**LDP**

In this paper, we mainly focus on data privacy, and thus the detailed network model is omitted.

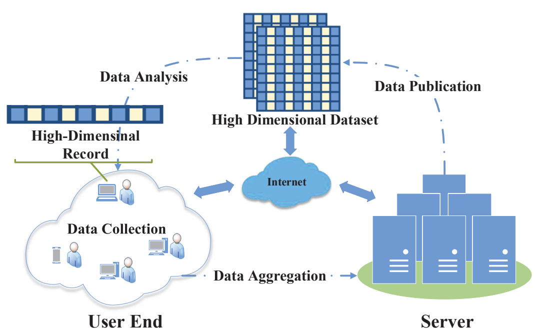


Fig. 3: An architecture of distributed high-dimensional private data collection and publication

Problem Statement. More specifically, given a collection of data records with d attributes from different users, our goal is to enable the server to publish a synthetic dataset that has the approximate joint distribution of d attributes with LDP. Formally, let N be the total number of users (i.e., data records[1]) and sufficiently large. Let $X = \{X^1, X^2, \ldots, X^N\}$ be the crowdsourced dataset, where X^i denotes the data record from the ith user. We assume that there are d attributes $A = \{A_1, A_2, \ldots, A_d\}$ in X. Each data record X^i can be represented as $X^i = [x_1^i, x_2^i, \ldots, x_d^i]$, where x_j^i denotes the jth element of the ith user record. For each attribute A_j $(j = 1, 2, \ldots, d)$, we denote $\Omega_j = \{\omega_j^1, \omega_j^2, \ldots, \omega_j^{|\Omega_j|}\}$ as the domain of A_j, where ω_j^i is the ith possible attribute value of Ω_j and $|\Omega_j|$ is the cardinality of Ω_j.

Alternatively, one can see dataset X of dimension $N \times d$ as a matrix with heterogeneous entries. There is an inherent multivariate data distribution that generates X consisting of N sampled row vectors of size d. Because of the consideration of data distribution, in the following we use terms, *data records*, *row vectors*, and *samples* interchangeably. With the above notations, our problem can be formulated as follows. Given a dataset X of dimension $N \times d$ and each data record is owned by user individually, we aim to release an approximate dataset $X^\star$ of dimension $N \times d$ such that

$$P_{X^\star}(A_1 \ldots A_d) \approx P_X(A_1 \ldots A_d), \tag{1}$$

where $P_X(A_1 \ldots A_d)$ is defined as $P_X(x_1^i = \omega_1, \ldots, x_d^i = \omega_d)$, $i = 1, \ldots, N$, $\omega_1 \in \Omega_1, \ldots, \omega_d \in \Omega_d$ with $P_X(x_1^i = \omega_1, \ldots, x_d^i = \omega_d)$ being defined as the d-dimensional joint distribution on X.

[1]We assume that each user is in possession of a multidimensional data record.

IV. Preliminaries

A. Differential Privacy (DP)

Differential privacy (DP) is the *de facto* standard for providing formal privacy guarantee [14]. It limits the adversary's ability of inferring the participation or absence of any user in a dataset via adding carefully calibrated noise (e.g., Laplace-distributed noise [14]) to query results. The algorithm $\mathcal{M}$ satisfies ϵ-differential privacy (ϵ-DP) if for all neighboring datasets D_1 and D_2 that differ on a single element (e.g., the data of one person), and all subsets S of the image of $\mathcal{M}$,

$$\Pr[\mathcal{M}(D_1) \in S] \leq e^\epsilon \times \Pr[\mathcal{M}(D_2) \in S], \tag{2}$$

where ϵ is called *privacy budget*, which serves as a privacy parameter for the level of privacy protection, with the characteristic that smaller ϵ means better privacy. According to the sequential composition theorem [38], an extra privacy budget will be required when DP mechanisms are applied mutiple times.

B. Local Differential Privacy (LDP)

DP implicitly assumes a trusted server and therefore can hardly apply to the case of privacy-aware crowdsourced systems. Recently, local differential privacy (LDP) is proposed for crowdsourced systems to provide a stringent privacy guarantee that data contributors trust no one but himself/herself [13], [24]. In particular, for any user i, a mechanism $\mathcal{M}$ satisfies

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2812146, IEEE Transactions on Information Forensics and Security

ϵ-local differential privacy (ϵ-LDP) if for any two data records X^i, X^j, and for any possible outputs $\tilde{X} \in Range(\mathcal{M})$,

$$\Pr[\mathcal{M}(X^i) = \tilde{X}] \leq e^{\epsilon} \times \Pr[\mathcal{M}(X^j) = \tilde{X}], \quad (3)$$

where the probability is taken over $\mathcal{M}'s$ randomness and the privacy budget ϵ has a similar impact on privacy as in the ordinary DP in Section IV-A. Intuitively, since $\Pr[\mathcal{M}(X^i) = \tilde{X}]$ is very close to $\Pr[\mathcal{M}(X^j) = \tilde{X}]$, an interpretation of the privacy provided by Equation (3) is that the adversary seeing $\tilde{X}$ cannot determine whether the input is X^i or X^j.

Randomized response (RR) [21], [44] is the simplest technique for achieving LDP and has been widely used in the survey of people's "yes or no" opinions about a sensitive question. In particular, surveyees adopting RR give their true answers with only a certain probability and opposite answers with remaining probability. Due to the randomness, the surveyor cannot determine the individuals' true answers (i.e., LDP is guaranteed) but still can extract the useful statistics from the noisy responses. Recently, RAPPOR has been proposed for statistics aggregation [18] and can be thought of as an extension of RR via either unary encoding or Bloom filter representation of user data.

V. LoPub: High-dimensional Data Publication with LDP

We propose LoPub, a novel solution to achieve high-dimensional crowdsourced data publication with LDP. In this section, we first introduce the basic idea behind LoPub in Section V-A and then elaborate on each component of LoPub in more details in Sections V-B$\sim$V-E.

A. Basic idea

Privacy-preserving high-dimensional crowdsourced data publication aims at releasing an approximate dataset with similar statistical information (i.e., in terms of statistical distribution as defined in Equation (1)) to the source data while guaranteeing the LDP. In the following, we make some observations on the solution to local differentially private data publication.

First, to achieve LDP, some local transformation over data needs to be designed, so as to cloak individuals' original data records. Then, the central server needs to derive the distribution of original data, by which one can generate the synthetic dataset. There are two plausible solutions for learning the original data distribution. One is to obtain the 1-dimensional distribution on each attribute independently. Unfortunately, the lack of consideration of correlations between dimensions will lead to the significant degradation of the utility. Another is to consider all attributes as one and compute an d-dimensional joint distribution. However, the possible domain will increase exponentially with the number of dimensions, thus leading to both low scalability and signal-noise-ratio problems [45]. Therefore, the technical challenge here is to find a solution for reducing the dimensionality while keeping the necessary correlations. Afterwards, with the statistical distribution information on low-dimensional data, one can synthesize an approximate dataset based on the learned distribution information.

To this end, we present LoPub, a Local differentially private data Publication scheme for high-dimensional crowdsourced data. Fig. 4 shows the overview of LoPub, which mainly consists of four mechanisms, local data protection, multi-dimensional distribution estimation, dimensionality reduction, and data synthesis. We describe them in more details as follows.

1) **Local Data Protection.** We first propose a local transformation process (also called *local randomizer*) that adopts RR on encoded data record, ensuring that each user output satisfies LDP. Particularly, we locally transform each attribute value to a random bit string. Then, the sanitized data is sent to and aggregated at the central server. The design of our local randomizer is described in Section V-B.
2) **Multi-dimensional Distribution Estimation.** We then propose three multi-dimensional joint distribution estimation schemes to derive both the joint and marginal probability distributions. Inspired by [19], we first extend the EM-based approach for distribution estimation. However, such a straightforward extension does not consider the sparsity in high-dimensional data, which will lead to high computational complexity for distribution estimation. To speedup the estimation, we present a Lasso-based approach with the cost of slight accuracy degradation. Finally, we propose a hybrid approach striking the balance between the accuracy and efficiency. These distribution estimation schemes are described in Section V-C.
3) **Dimensionality Reduction.** Based on the learned distribution, we develop a technique for dimensionality reduction by identifying correlated attributes and splitting attributes into several compact low-dimensional attribute clusters. More specifically, considering the heterogeneous attributes, we adopt mutual information to measure the correlations, forming an undirected dependency graph. Then, our technique splits the attributes according to the junction tree built from the dependency graph. We also propose a heuristic pruning scheme to further speedup the process of correlation identification. We present the techniques for dimensionality reduction in Section V-D.
4) **Synthesizing the New Dataset.** Finally, we sample each low-dimensional dataset according to the connectivity of attribute clusters and the estimated joint (or conditional) distribution on each attribute cluster, thus synthesizing an approximate dataset.

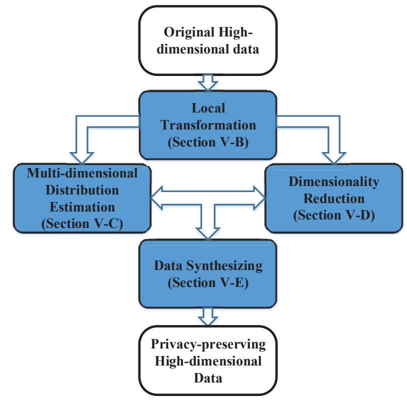


Fig. 4: An overview of LoPub

B. Local Transformation for High-dimensional Data Record

Design Rationale. Local transformation in our design includes two key steps; one is representing the data record as a Bloom filter and another is to introduce uncertainty. Particularly, Bloom filter with multiple hash functions encodes a set of data items into a pre-defined bit string. Thus, the unique bit string is a representative feature of a data record. Then, the individual user performs RR on each individual bit, injecting uncertainty to the Bloom filter-encoded data.

Design of Local Randomizer. Under the above framework, one observation can be made; given Bloom filter as a feature

TABLE II: Notation

N	number of users (data records) in the system
X	entire crowdsourced dataset
X^i	data record from the ith user
x_j^i	jth element of X^i
d	number of attributes in X
A_j :	jth attribute of X
Ω_j	domain of A_j
ω_j	candidate value in Ω_j
$\mathcal{H}_j$	the set of hash functions for A_j that map x into a Bloom filter
$\mathcal{H}_{j,x}(\cdot)$	the xth hash function in $\mathcal{H}_j$
h_j	the number $\lvert\mathcal{H}_j\rvert$ of hash functions in $\mathcal{H}_j$
s_j^i	Bloom filter of x_j^i ($S_j^i = \mathcal{H}_j(x_j^i)$)
$s_j^i[b]$	the bth bit of s_j^i
$\hat{s}_j^i$	randomized Bloom filter of s_j^i
$\hat{s}_j^i[b]$	the bth bit of $\hat{s}_j^i$
m_j	length of s_j^i
f	probability of flipping a bit of a Bloom filter

for a particular attribute value, a concatenation of d Bloom filters can also serve as a feature for d attribute values from a data record. One advantage of using Bloom filters as features for different attributes is that, depending on domain sizes of particular attributes, one can separately optimize the parameters of Bloom filter, such as the length of Bloom filter, so as to minimize the corresponding overhead. In addition, when Bloom filter is seen as feature, existing machine learning techniques like EM and Lasso regression will be effective for further multivariate distribution estimation (described in Section V-C). Some notations used in this paper are listed in Table II.

In essence, our design of local randomizer consists of three steps.

1) The ith user is assumed to have a data record $X^i = [x_1^i, x_2^i, \ldots, x_d^i]$. The ith user encodes each x_j^i as a Bloom filter via the set $\mathcal{H}_j$ of hash functions particularly for A_j; the ith user employs h_j hash functions $\mathcal{H}_{j,1}, \ldots, \mathcal{H}_{j,h_j}$ from $\mathcal{H}_j$ to map x_j^i to a length-m_j bit string s_j^i (called *a Bloom filter*); i.e., x_j^i is inserted to a length-m_j bit Bloom filter with h_j hash functions from $\mathcal{H}_j$. Note that $s_j^i[b]$ denotes the bth bit of the bit string s_j^i. In the rest of this paper, we abuse the notation $\mathcal{H}_j(\omega)$ as the Bloom filter with those bits at positions $\mathcal{H}_{j,1}(\omega), \ldots, \mathcal{H}_{j,h_j}(\omega)$ being set to be 1.
2) Each bit $s_j^i[b]$ ($b = 1, 2, \ldots, m_j$) in s_j^i is randomly flipped into 0 or 1 according to the following RR rule:

$$\hat{s}_j^i[b] = \begin{cases} s_j^i[b], & \text{with probability of } 1-f \\ 1, & \text{with probability of } f/2 \\ 0, & \text{with probability of } f/2 \end{cases} \quad (4)$$

where $f \in [0, 1]$ is a parameter that quantifies the level of randomness for LDP and controls the privacy level. In essence, each user eventually sends out a noisy Bloom filter (or called randomized Bloom filter).
3) After deriving the randomized Bloom filter $\hat{s}_j^i$ ($j = 1, \ldots, d$), the ith user concatenates $\hat{s}_1^i, \ldots, \hat{s}_d^i$ to obtain a ($\sum_{j=1}^d m_j$)-bit vector $\hat{s}_1^i || \ldots || \hat{s}_d^i$ and send it to the server.

Given the false positive probability p and the number $|\Omega_i|$ of elements to be inserted, the parameters of Bloom filter can be easily optimized. In other words, the optimal length m_j of Bloom filter in the case of $N \gg |\Omega_j|$ can be calculated as

$$m_j = \frac{\ln(1/p)}{(\ln 2)^2}|\Omega_j|. \quad (5)$$

Furthermore, the optimal number h_j of hash functions in the Bloom filter is

$$h_j = \frac{m_j}{|\Omega_j|}\ln 2 = \frac{\ln(1/p)}{(\ln 2)}. \quad (6)$$

In the rest of this paper, we assume $h_j = h$ and $m_j = m$ for all j for simplicity.

Communication Overhead. Denote C_X as the communication cost (in terms of number of bits) for scheme X. We have the following theorem for communication cost of our local transformation.

Theorem *1:* C_{LoPub} can be calculated as

$$C_{\mathsf{LoPub}} = \sum_{j=1}^{d} m_j = \frac{\ln(1/p)}{(\ln 2)^2}\sum_{j=1}^{d}|\Omega_j|. \quad (7)$$

When RAPPOR [18] is directly applied to each attribute value of the d-dimensional data, all $\Omega_1 \times \cdots \times \Omega_d$ candidate value will be together regarded as 1-dimensional data[2], then the cost is

$$C_{\mathsf{RAPPOR}} = \frac{\ln(1/p)}{(\ln 2)^2}\prod_{j=1}^{d}|\Omega_j|, \quad (8)$$

where $\prod_{j=1}^{d}|\Omega_j|$ is due to the size of the candidate set $\Omega_1 \times \cdots \times \Omega_d$. The difference between Equation (7) and (8) stems from the fact that given Bloom filter as a feature for a particular attribute value, a concatenation of d Bloom filters can also serve as a feature for d attribute values from a data record.

Privacy Analysis: Because each user runs a local randomizer on data records individually, one can argue the individual privacy by claiming the privacy of local randomizer. According to [18], local transformation of a specific attribute value can achieve ϵ-LDP, where $\epsilon = 2h\ln((2-f)/f)$ with h being the number of hash functions in the Bloom filter and f is the probability of flipping a bit.

According to the sequential composition theorem [32], local transformation of a d-dimensional data record achieves ϵ-LDP, where

$$\epsilon = 2dh\ln((2-f)/f), \quad (9)$$

with d being the number of attributes (dimensions) in original data record X^i. Since the same transformation is done by all users independently, the above ϵ-LDP guarantee holds for all distributed users. It should be noted that, according to Equation (9), the Bloom filter length m_j as well as communication cost C_{LoPub} (or C_{RAPPOR}) is independent of the privacy level achieved.

C. Multivariate Distribution Estimation with LDP

After receiving noisy bit strings, the next step of server is to estimate the joint distribution. We first propose a natural extension of the 2-dimensional distribution estimation [19] for high-dimensional distribution estimation (Section V-C1). However, due to high complexity and overheads, it is only preferable to low dimensions with small domain, which is impractical to many real-world datasets with high dimensions. Therefore, we then propose a Lasso regression-based distribution estimation to prevent the convergence issue, at the cost of utility degradation (Section V-C2). Finally, we present a hybrid algorithm to strike a balance between efficiency and accuracy (Section V-C3).

[2]When the orginal RAPPOR is directly applied, the high-dimensional data will be regarded as one dimensional data. So, each user's data will be transformed into one bit string and randomly flipped to achieve LDP. Now, the number of candidates would be $|\Omega_1| \times \cdots \times |\Omega_d|$. In the case of Bloom filter encoding, the minimal length of Bloom filter strings needs to be $C_{\mathsf{RAPPOR}} = |\Omega_1| \times \cdots \times |\Omega_d| \times \ln(1/p)/((\ln 2)^2)$.

1) EM-based Distribution Estimation: Here, we first extend EM-based estimation [19] for k-dimensional marginal dataset ($2 \leq k \leq d$).

We first introduce the following notations. Without loss of generality, we consider k specified attributes as A_1, A_2, ..., A_k and their index collection $\mathcal{C} = \{1, 2, ..., k\}$. For simplicity, the event $A_j = \omega_j$ or $x_j = \omega_j$ is abbreviated as ω_j. For example, the prior probability $P(x_1 = \omega_1, x_2 = \omega_2, \ldots, x_k = \omega_k)$ can be simplified into $P(\omega_1\omega_2 \ldots \omega_k)$ or $P(\omega_\mathcal{C})$. Algorithm 1 shows our EM-based k-dimensional distribution estimation algorithm, EM_JD. More specifically, it consists of the following five main steps.

Algorithm 1 EM-based k-dimensional Joint Distribution (EM_JD)

Require: $\mathcal{C}$: attribute indexes cluster, i.e., $\mathcal{C} = \{1, 2, ..., k\}$
A_j : k-dimensional attributes $(1 \leq j \leq k)$,
Ω_j : domain of A_j $(1 \leq j \leq k)$,
$\hat{s}_j^i$: observed Bloom filters $(1 \leq i \leq N)$ $(1 \leq j \leq k)$,
f : flipping probability,
δ : convergence accuracy.
Ensure: $P(A_\mathcal{C})$: joint distribution of k attributes specified by $\mathcal{C}$.
1: initialize $P_0(\omega_\mathcal{C}) = 1/(\prod_{j \in \mathcal{C}} |\Omega_j|)$.
2: **for** each $i = 1, \ldots, N$ **do**
3: **for** each $j \in \mathcal{C}$ **do**
4: compute $P(\hat{s}_j^i|\omega_j) = \prod_{b=1}^{m_j} (\frac{f}{2})^{\hat{s}_j^i[b]} (1 - \frac{f}{2})^{1-\hat{s}_j^i[b]}$.
5: **end for**
6: compute $P(\hat{s}_\mathcal{C}^i|\omega_\mathcal{C}) = \prod_{j \in \mathcal{C}} P(\hat{s}_j^i|\omega_j)$.
7: **end for**
8: initialize $t = 0$ /* number of iterations */
9: **repeat**
10: **for** each $i = 1, \ldots, N$ **do**
11: **for** each $(\omega_\mathcal{C}) \in \Omega_1 \times \Omega_2 \times \cdots \times \Omega_k$ **do**
12: compute $P_t(\omega_\mathcal{C}|\hat{s}_\mathcal{C}^i) = \frac{P_t(\omega_\mathcal{C}) \cdot P(\hat{s}_\mathcal{C}^i|\omega_\mathcal{C})}{\sum_{\omega_\mathcal{C}} P_t(\omega_\mathcal{C}) P(\hat{s}_\mathcal{C}^i|\omega_\mathcal{C})}$
13: **end for**
14: **end for**
15: set $P_{t+1}(\omega_\mathcal{C}) = \frac{1}{N} \sum_{i=1}^N P_t(\omega_\mathcal{C}|\hat{s}_\mathcal{C}^i)$
16: update $t = t + 1$
17: **until** $\max_{\omega_\mathcal{C}} P_t(\omega_\mathcal{C}) - \max_{\omega_\mathcal{C}} P_{t-1}(\omega_\mathcal{C}) \leq \delta$.
18: **return** $P(A_\mathcal{C}) = P_t(\omega_\mathcal{C})$

1) At first, we set an uniform distribution $P(\omega_1\omega_2 \ldots \omega_k) = 1/(\prod_{j=1}^{k} |\Omega_j|)$ as the initial prior probability (line 1 of Algorithm 1).
2) In local transformation, each bit $s_j^i[b]$ will be flipped with probability $\frac{f}{2}$. Thus, by comparing the bits $\mathcal{H}_j(\omega_j)$ with the randomized bits, the conditional probability $P(\hat{s}_j^i|\omega_j)$ can be computed (lines 3∼5 of Algorithm 1).
3) The joint conditional probability can be easily calculated by combining individual attributes; i.e., $P(\hat{s}_\mathcal{C}^i|\omega_\mathcal{C}) = \prod_{j \in \mathcal{C}} P(\hat{s}_j^i|\omega_j)$ (line 6 of Algorithm 1).
4) Given all the conditional distributions of one particular combination of bit strings, their corresponding posterior probability can be computed by the Bayes' Theorem,

$$P_t(\omega_\mathcal{C}|\hat{s}_\mathcal{C}^i) = \frac{P_t(\omega_\mathcal{C}) \cdot P(\hat{s}_\mathcal{C}^i|\omega_\mathcal{C})}{\sum_{\omega_\mathcal{C}} P_t(\omega_\mathcal{C}) P(\hat{s}_\mathcal{C}^i|\omega_\mathcal{C})}. \quad (10)$$

where $P_t(\omega_\mathcal{C})$=$P_t(\omega_1\omega_2 \ldots \omega_k)$ is the k-dimensional joint probability at the tth iteration (lines 11∼13 of Algorithm 1).
5) After identifying posterior probability for each user, we calculate the mean of the posterior probability from a large number of users to update the prior probability (lines 15∼16 of Algorithm 1). The prior probability is used in the next iteration to update the posterior probability. The above EM-like procedures are executed iteratively until convergence, i.e., the maximum difference between two estimations is smaller than the specified threshold (line 17 of Algorithm 1).

Complexity. Similar to the other works, we assume that number N of user records is sufficiently large; i.e., $N \gg v^k$, where v denotes the average size of $|\Omega_j|$. Otherwise it is difficult to extract the useful statistics from noisy data (or say, to have an accurate estimation from a sample space with low signal-noise-ratio).

Theorem *2:* Given v as the average size of $|\Omega_j|$, the time complexity of EM_JD is

$$O\big(Nkmv^k + tNv^{2k}\big). \quad (11)$$

Theorem *3:* The space complexity of EM_JD is

$$O\big(Nkm + 2Nv^k\big). \quad (12)$$

EM_JD can converge to a good estimation empirically. However, EM_JD suffers from the following three drawbacks. First, EM_JD might fail when converging to local optimum. Especially when k increases, there will be many local optimums to prevent good convergence because sample space of all combinations in $\Omega_{j_1} \times \Omega_{j_2} \times \cdots \times \Omega_{j_k}$ explodes exponentially. Second, the space overhead could be daunting when either N or k is large. This makes the performance of EM_JD degrade dramatically and not applicable to high dimensional data. Third, in fact, the effectiveness of EM_JD is sensitive to the initial value.

2) Lasso-based Distribution Estimation: To have better efficiency of the distribution estimation, we instead present a Lasso regression-based algorithm Lasso_JD. As mentioned previously, a Bloom filter can be thought of as a representative feature of data. After RR, a large number of noises will be injected to Bloom filter by individual users. More precisely, one may consider that the server receives a large number of samples from a specific distribution, however, with random noise. In this sense, one may estimate the distribution from the noisy samples by taking advantage of linear regression $\mathbf{y} = \mathbf{M}\beta$, where $\mathbf{M}$ is predictor variables and $\mathbf{y}$ is response variable, and β is the regression coefficient vector. Here, our aim is to estimate the distribution on predictor variables $\mathbf{M}$. Instead of the original domain, $\mathbf{M}$ is represented as Bloom filters. The use of Bloom filter can guarantee that the features (predictor variables $\mathbf{M}$) re-extracted at the server are the same as ones extracted by the user. Moreover, response variable $\mathbf{y}$ can be estimated from the randomized bit strings according to the known f[3]. Therefore, the only problem is to find a good solution to the linear regression $\mathbf{y} = \mathbf{M}\beta$. Obviously, k-dimensional data may incur a output domain $\Omega_1 \times ... \times \Omega_k$ with the size of $|\Omega_1| \times ... \times |\Omega_k|$, which increases exponentially with k. With fixed N entries in the dataset X, the frequencies of many combination $\omega_1\omega_2...\omega_k \in \Omega_1 \times ... \times \Omega_k$ are rather small or even zero. So, $\mathbf{M}$ is sparse and only part of the sparse but effective predictor variables need to be chosen. Here, we resort to Lasso regression, effectively solving the sparse linear regression by choosing proper predictor variables.

Our Lasso-based distribution estimation, Lasso_JD, is shown in Algorithm 2 and consists of the following four major steps.

1) After receiving all noisy Bloom filters, for each bit b in each attribute j, the server counts the number of $1's$ as $\hat{y}_j[b] = \sum_{i=1}^N \hat{s}_j^i[b]$ (lines 1∼3 of Algorithm 2).

[3]The estimation of each position in the bit count vector Y can be accomplished via the RR recovery; since $(1 - f)$ original true bits are kept , $f/2$ new 1's are added, and $f/2$ new 0's are added, given the aggregated count $\hat{y}$ of N users, there is $(1-f)*y+N*f/2 = \hat{y}$ and $y = (\hat{y}-N*f/2)/(1-f)$, where y denotes the true bit count.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2812146, IEEE Transactions on Information Forensics and Security

Algorithm 2 Lasso-based k-dimensional Joint Distribution (Lasso_JD)

Require: $\mathcal{C}$: attribute indexes cluster i.e., $\{1, 2, ..., k\}$,
A_j : k-dimensional attributes $(1 \leq j \leq k)$,
Ω_j : domain of A_j $(1 \leq j \leq k)$,
$\hat{s}_j^i$: observed Bloom filters $(1 \leq i \leq N)$ $(1 \leq j \leq k)$,
f : flipping probability.

Ensure:
$P(A_{\mathcal{C}})$: joint distribution of k attributes specified by $\mathcal{C}$.

1: **for** each $j \in \mathcal{C}$ **do**
2: **for** each $b = 1, 2, \ldots, m_j$ **do**
3: compute $\hat{y}_j[b] = \sum_{i=1}^{N} \hat{s}_j^i[b]$
4: compute $y_j[b] = (\hat{y}_j[b] - fN/2)/(1 - f)$
5: **end for**
6: set $\mathcal{H}_j(\Omega_j) = \{\mathcal{H}_j(\omega) \mid \forall\omega \in \Omega_j\}$
7: **end for**
8: set $\mathbf{y} = [y_1[1], \ldots, y_1[m_1] \,|\, y_2[1], \ldots, y_2[m_2] \,|\, \ldots \,|\, y_k[1], \ldots, y_k[m_k]]$
9: set $\mathbf{M} = [\mathcal{H}_1(\Omega_1) \times \mathcal{H}_2(\Omega_2) \times \cdots \times \mathcal{H}_k(\Omega_k)]$
10: compute $\vec{\beta} =$ Lasso_regression$(\mathbf{M}, \mathbf{y})$
11: **return** $P(A_{\mathcal{C}}) = \vec{\beta}/N$

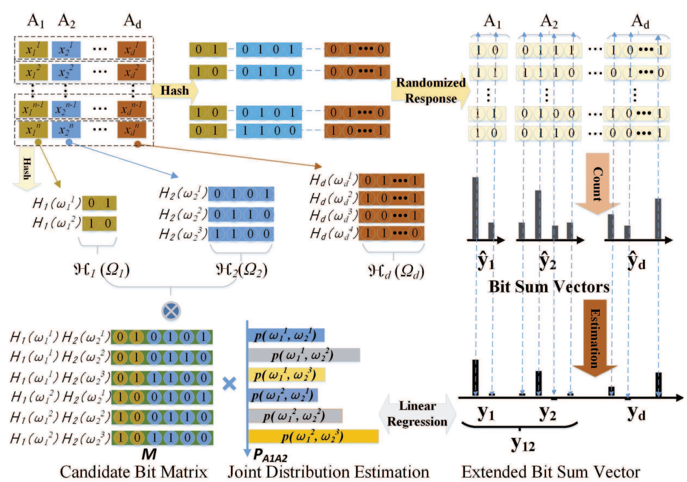


Fig. 5: Illustration of Lasso_JD

2) The true count sum of each bit $y_j[b]$ can be estimated as $y_j[b] = (\hat{y}_j[b] - fN/2)/(1 - f)$ according to the RR applied to the true count. These count sums of all bits form a vector $\mathbf{y}$ with the length of $\sum_{j=1}^{k} m_j$ (lines 4 and 8 of Algorithm 2).
3) The Bloom filters on each dimension A_j is constructed by the server with the same hash functions from $\mathcal{H}_j$. Suppose all distinct Bloom filters on Ω_j are $\mathcal{H}_j(\Omega_j) = \{\mathcal{H}_j(\omega) \mid \forall\omega \in \Omega_j\}$. The candidate set of Bloom filters is then $\mathbf{M} = [\mathcal{H}_1(\Omega_1) \times \mathcal{H}_2(\Omega_2) \times \cdots \times \mathcal{H}_k(\Omega_k)]$(lines 6 and 9 of Algorithm 2).
4) Fit a Lasso regression model to the counter vector $\mathbf{y}$ and the candidate matrix $\mathbf{M}$, and then choose the non-zero coefficients as the corresponding frequencies of each candidate string. By reshaping the coefficient vector into a k-dimensional matrix, we can derive the k-dimensional joint distribution estimation $P(A_1A_2 \ldots A_k)$. For example, in Fig. 5, we fit a linear regression to $\mathbf{y}_{12}$ and the candidate matrix $\mathbf{M}$ to estimate the joint distribution $P_{A_1A_2}$ (lines 10∼11 of Algorithm 2).

The efficiency of Lasso_JD comes from the fact that the N noisy Bloom filters will be scanned to count sums at each position only once and then one-time Lasso regression is performed to estimate the distribution. Furthermore, Lasso regression could extract the most important (i.e., frequent) features with high probability, which fits well with the sparsity of high-dimensional data. The precise computation overhead and memory overhead are shown below.

Complexity. Compared with EM_JD, our Lasso_JD can effectively reduce the time and space complexity.

Theorem *4:* The time complexity of Lasso_JD is

$$O\big(v^{3k} + kmv^{2k} + Nkm\big). \tag{13}$$

As in Section V-C1, we assume $N \gg v^k$. In this case, we can see the time complexity of Lasso_JD (Equation (13)) is much less than that of EM_JD (Equation (11))

Theorem *5:* The space complexity of Lasso_JD is

$$O\big(Nkm + v^k km\big). \tag{14}$$

Generally, the regression operation will lose accuracy only when there are many collisions between Bloom filter strings. We observe that if there is no collision in the bit strings of each single dimension, then there is no collision in concatenated bit strings of different dimensions. In fact, the probability of collision in concatenated bit strings will not increase with the number of dimensions. Moreover, given the false positive rate p, one can derive the optimal number of hash functions used and optimal number of bits for Bloom filter for a specific attribute. Therefore, we only need to choose proper m and h to minimize the collision probability for each dimension and then we are able to reach a proper estimation for multiple dimensions.

3) Hybrid Algorithm: Recall that, with sufficient samples, EM_JD can demonstrate good convergence but also incurs high complexity. On the other hand, Lasso_JD can be very efficient with a slight accuracy degradation compared with the EM-based algorithm.

The high complexity of the EM_JD stems from two facts; first, it iteratively scans users' reports and builds a prior distribution table, which has the size of $O(Nv^k)$. In this sense, for each record, one has to compare $\sum m_j$ bits. However, in the case of high dimensionality, the combination of Ω_j will be very sparse and has lots of zero items. Second, because EM is sensitive to the initial configuration, the initial value of the uniformly random assignment might lead to slow convergence.

To strike a balance between accuracy and efficiency, we propose a hybrid algorithm, Lasso+EM_JD (Algorithm 3), which first eliminates the redundant candidates and estimates the initial value with Lasso_JD and then refines the convergence using EM_JD. Our proposed Lasso+EM_JD possesses two advantages:

1) The sparse candidates will be picked by Lasso_JD very efficiently. So, EM_JD can just compute the conditional probability on those sparse candidates instead of all candidates, leading to the significant reduction of both time and space complexity.
2) Lasso_JD can generate a good initial estimation of the joint distribution. Compared with using initial values with uniformly random assignments, using the initial value generated by Lasso_JD can further speedup the convergence of EM_JD, which is sensitive to the initial value especially when the candidate space is sparse.

The following two theorems show the time and space complexity of Lasso+EM_JD.

Theorem *6:* The time complexity of Lasso+EM_JD is

$$O\big((v^{3k} + kmv^{2k} + Nkm) + (tN(v')^2 + Nkm(v'))\big), \tag{15}$$

where v' is the average size of sparse items in $\Omega_1 \times ... \times \Omega_k$, and $v' < v^k$.

Theorem *7:* The space complexity of Lasso+EM_JD is

$$O\big(Nkm + v^k km + 2Nv'\big). \tag{16}$$

Algorithm 3 Lasso+EM k-dimensional Joint Distribution (Lasso+EM_JD)

Require: A_j : k-dimensional attributes $(1 \le j \le k)$,
Ω_j : domain of A_j $(1 \le j \le k)$,
$\hat{s}^i_j$: observed Bloom filters $(1 \le i \le N)$ $(1 \le j \le k)$,
f : flipping probability.
Ensure: $P(A_1 A_2 \dots A_k)$: k-dimensional joint distribution.
compute $P_0(\omega_1\omega_2 \dots \omega_k) =$ Lasso_JD$(A_j, \Omega_j, \{\hat{s}^i_j\}_{i=1}^N, f)$
set $\mathcal{C}' = \{x | x \in \mathcal{C}, P_0(x) = 0\}$.
for each $i = 1, \dots, N$ **do**
  for each $j = 1, \dots, k$ **do**
    compute $P(\hat{s}^i_j|\omega_j) = \prod_{b=1}^{m_j} (\frac{f}{2})^{\hat{s}^i_j[b]} (1 - \frac{f}{2})^{1-\hat{s}^i_j[b]}$.
  end for
  if $\omega_1\omega_2 \dots \omega_k \in \mathcal{C}'$ **then**
    $P(\hat{s}^i_1 \hat{s}^i_2 \dots \hat{s}^i_k | \omega_1\omega_2 \dots \omega_k) = 0$
  else
    compute $P(\hat{s}^i_1 \hat{s}^i_2 \dots \hat{s}^i_k | \omega_1\omega_2 \dots \omega_k) = \prod_{j=1}^{k} P(\hat{s}^i_j|\omega_j)$.
  end if
end for
initialize $t = 0$ /* number of iterations */
repeat
  ... ...
  /* (similar to Algorithm 1) */
  ... ...
until $P_t(\omega_1\omega_2 \dots \omega_k)$ converges.
return $P(A_1 A_2 \dots A_k) = P_t(\omega_1\omega_2 \dots \omega_k)$

D. Dimensionality Reduction with LDP

Generally, the above mulitivariate joint distribution estimation algorithms can be applied to any k-dimensional data. However, when k further increases, the domain size of the multivariate distribution increases exponentially. With fixed number of users N, the average count for each entry of the domain is $N/(|\Omega|^k)$, which is very small and lacks the statistical significance. This will eventually lead to lower utility for high-dimensional data. Therefore, it is crucial to reduce the dimensionality before data synthesis.

We first present our proposed mutual information-based method for dimensionality reduction in Section V-D1. Afterwards, we present a heuristic to speedup the dimensionality reduction in Section V-D2.

1) Dimensionality Reduction via 2-dimensional Joint Distribution Estimation: The key to reducing dimensionality in a high-dimensional dataset is to find the compact clusters, within which all attributes are tightly correlated to or dependent on each other. Our proposed dimensionality reduction technique based on local differentially private data records, as shown in Algorithm 4, consumes no extra privacy budget and consists of the following three steps:

- **Pairwise Correlation Computation.** We use mutual information to measure pairwise correlations between attributes. The mutual information is calculated as

$$I_{m,n} = \sum_{i \in \Omega_m} \sum_{j \in \Omega_n} p_{ij} \ln \frac{p_{ij}}{p_{i\cdot} p_{\cdot j}}, \quad (17)$$

  where Ω_m and Ω_n are the domains of attributes A_m and A_n, respectively. The notations $p_{i\cdot}$ and $p_{\cdot j}$ represent the marginal probability of the ith value in Ω_m and the probability that A_n is the jth value in Ω_n, respectively. Then, p_{ij} is their joint probability. Particulary, p_{ij} can then be efficiently obtained by using Lasso+EM_JD. As the corresponding marginal distributions, both $p_{i\cdot}$ and $p_{\cdot j}$ then can be learned from p_{ij} or estimated with the 2-dimensional joint distribution of A_i (or A_j) and itself A_i (or A_j) (lines 2∼8 of Algorithm 4).
- **Dependency Graph Construction.** Dependency graph can be used to depict the correlations among attributes. Assume each attribute A_j is a node in the dependency graph and an edge between two nodes A_m and A_n represents that attribute A_m and A_n are correlated. Based on mutual information between two attributes, the dependency graph of attributes can be constructed as follows. First, an adjacent matrix $\mathbf{G}_{d \times d}$ (dependency graph with d attributes as vertices) is initialized with all 0's. Then, all the attribute pairs (A_m, A_n) are chosen to compare their mutual information with an threshold $\tau_{m,n}$, which is defined as

$$\tau_{m,n} = \min(|\Omega_m| - 1, |\Omega_n| - 1) \times \phi^2/2, \quad (18)$$

  where ϕ $(0 \le \phi \le 1)$ is a flexible parameter determining the desired correlation level. Normally ϕ is set to be 0.3. $\mathbf{G}_{m,n}$ and $\mathbf{G}_{n,m}$ are both set to be 1 if and only if $I_{m,n} > \tau_{m,n}$ (lines 9∼15 of Algorithm 4).
- **Compact Clusters Building.** By triangulation, the dependency graph $\mathbf{G}_{d \times d}$ can be transformed to a *junction tree*. Then, based on the junction tree algorithm, several clusters $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_l$ can be derived as the compact clusters of attributes, each of which contains attributes that are correlated. Hence, the whole attributes set can be divided into several compact attribute clusters, which have low correlations between clusters but more correlations within each cluster. Within a cluster, the average number of dimensions (or the number of attributes) will be smaller than the total number of attributes. More importantly, these low-dimensional clusters can then be processed independently due to low correlations. Hence, the high-dimensional data can be split into several low-dimensional data, i.e., dimensionality reduction (lines 16∼17 of Algorithm 4).

Algorithm 4 Dimensionality reduction with LDP

Require: A_j : k-dimensional attributes $(1 \le j \le k)$,
Ω_j : domain of A_j $(1 \le j \le k)$,
$\hat{s}^i_j$: observed Bloom filters $(1 \le i \le N)$ $(1 \le j \le k)$,
f : flipping probability,
ϕ : dependency degree
Ensure: $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_l$: attribute indexes clusters
1: initialize $\mathbf{G}_{d \times d} = \mathbf{0}$.
2: **for** each $j = 1, 2, \dots, d$ **do**
3:   estimate $P(A_j)$ by JD (i.e., Lasso+EM_JD Algorithm 3)
4: **end for**
5: **for** each attribute $m = 1, 2, \dots, d-1$ **do**
6:   **for** each attribute $n = m+1, m+2, \dots, d$ **do**
7:     estimate $P(A_m A_n)$ by JD
8:     compute $I_{m,n} = \sum_{i \in \Omega_m} \sum_{i \in \Omega_n} p_{ij} \ln \frac{p_{ij}}{p_{i\cdot} p_{\cdot j}}$
9:     compute $\tau_{m,n} = \min(|\Omega_m| - 1, |\Omega_n| - 1) * \phi^2/2$
10:     **if** $I(m,n) \ge \tau_{mn}$ **then**
11:       set $\mathbf{G}_{m,n} = \mathbf{G}_{n,m} = 1$
12:     **end if**
13:   **end for**
14: **end for**
15: build dependency graph with $\mathbf{G}_{d \times d}$
16: triangulate the dependency graph into a junction tree
17: split the junction tree into several cliques $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_l$ with elimination algorithm.
18: **return** $\mathfrak{C} = \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_l\}$

Theorem *8:* The time complexity of Algorithm 4 is

$$O(d^2(v^6 + 2mv^4 + 2Nm + tN(v')^2 + 2Nm(v'))). \quad (19)$$

Theorem *9:* The space complexity of Algorithm 4 is

$$O(2Nm + 2v^2m + 2Nv'). \quad (20)$$

2) Entropy based Pruning Scheme: In existing work [25], [41] on homogeneous data, correlations can be simply captured by distance or similarity metrics [47]. With the consideration of heterogeneous attributes (i.e., attributes with different domains) in our paper, mutual information is used to measure general correlations. As the computation of pairwise dependence is necessary for calculating the mutual information of variables X and Y, we propose a pruning-based heuristic to speedup this pairwise correlation learning process.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2812146, IEEE Transactions on Information Forensics and Security

Intuitively, there are two different situations in Algorithm 4:

- When $\phi = 0$ or $\phi = 1$, all attributes will be considered mutually correlated or independent. Thus, there is no need to compute pairwise correlation.
- In the case of ϕ $(0 < \phi < 1)$, less dependencies will be included in the dependency graph; $\mathbf{G}_{d\times d}$ will be sparser. This also means that we may selectively neglect some pairs. Inspired by the relationship between mutual information and information entropy[4], we first heuristically filter out some portion of attributes A_x with least relative information entropy $RH(A_x) = H(A_x)/|\Omega_x|$, and then verify the mutual information among the remaining attributes, thus reducing the pairwise computations.

Furthermore, the adjacent matrix $\mathbf{G}_{d\times d}$ varies with different datasets. For example, the adjacent matrix $\mathbf{G}_{d\times d}$ is rarely sparse in binary datasets but will be very sparse in non-binary datasets. Based on this observation, we can further simplify the calculation by finding the independency in binary datasets or finding the dependency in non-binary datasets. For example, we first set all entries of $\mathbf{G}_{d\times d}$ for a binary datasets as 1's and start from the attributes with least relative information entropy $RH(A_x) = H(A_x)/|\Omega_x|$ to find the uncorrelated attributes. While for non-binary datasets, we first set $\mathbf{G}_{d\times d}$ as 0's and then start from the attributes with largest average entropy to find the correlated attributes. Our entropy-based pruning scheme is shown in Algorithm 5.

Algorithm 5 Entropy-based Pruning Scheme

Require: A_j : k-dimensional attributes $(1 \le j \le k)$,
Ω_j : domain of A_j $(1 \le j \le k)$,
$\hat{s}_j^i$: observed Bloom filters $(1 \le i \le N)$ $(1 \le j \le k)$,
f : flipping probability,
ϕ : dependency degree
Ensure: $\mathbf{G}_{d\times d}$: adjacent matrix $\mathbf{G}_{d\times d}$ of dependency graph of attributes A_j $(j = 1, 2, ..., d)$

initialize $\mathbf{G}_{d\times d} = 0$
for each $j = 1, 2, \ldots, k$ **do**
  compute $P(A_j) = \mathsf{JD}(A_j, \Omega_j, \{\hat{s}_j^i\}_{i=1}^N, f)$
  compute $RH(A_j) = -\frac{1}{|\Omega_j|} \sum_{p \in P(A_j)} p \log p$
end for
sort $list_A = \{A_1, A_2, ..., A_j\}$ according to entropy $H(A_j)$
pick up the previous $\lfloor length(list_A) \times (1-\phi) \rfloor$ items from $list_A$ as a new list $list_{A'}$
compute pairwise mutual information among $list_{A'}$ and set dependency graph $\mathbf{G}_{d\times d}$ as in Algorithm 4.
return $\mathbf{G}_{d\times d}$

E. Synthesizing New Dataset

For brevity, we first define $A_{\mathcal{C}} = \{A_j | j \in \mathcal{C}\}$ and $\hat{X}_{\mathcal{C}} = \{x_j | j \in \mathcal{C}\}$. Then the process of synthesizing a new dataset via sampling is shown in Algorithm 6. We first initialize an empty set $\mathfrak{R}$ to keep the sampled attributes. Then, we randomly choose an attribute cluster $\mathcal{C}$ to estimate the joint distribution and sample new data $\hat{X}_{\mathcal{C}}$ from attributes $A_j \in \mathcal{C}$. Next, given the cluster collection $\mathfrak{C}$ derived in Algorithm 4, we calculate $\mathfrak{C} = \mathfrak{C} \backslash \mathcal{C}$, find the connected component $\mathfrak{D}$ of $\mathcal{C}$, and calculate $\mathfrak{R} = \mathfrak{R} \cup \mathcal{C}$. In the connected component, each cluster D is traversed and sampled as follows. First we estimate the joint distribution on the attributes A_D by our distribution estimations in Section V-C and derive the conditional distribution $P(A_{D\backslash\mathfrak{R}} | A_{D\cap\mathfrak{R}})$. Then, we sample $\hat{X}_{D\backslash\mathfrak{R}}$ according to this conditional distribution and the sampled data $\hat{X}_{D\cap\mathfrak{R}}$. After the traversal of $\mathfrak{D}$, the attributes in the first connected components are sampled. Afterwards, randomly choose a cluster in the remaining $\mathfrak{C}$ to sample the attributes in the second connected components, until $\mathfrak{C}$ is empty. Finally, a new synthetic dataset $\hat{X}$ is generated according to the estimated correlations and distributions in origin dataset X. Algorithm 6 shows the above procedures that synthesize a dataset from the collection of clusters of attributes.

Algorithm 6 New Dataset Synthesizing

Require: $\mathfrak{C}$: a collection of attribute index clusters $\mathcal{C}_1, ...\mathcal{C}_l$,
A_j : k-dimensional attributes $(1 \le j \le k)$,
Ω_j : domain of A_j $(1 \le j \le k)$,
$\hat{s}_j^i$: observed Bloom filters $(1 \le i \le N)$ $(1 \le j \le k)$,
f : flipping probability,
Ensure: $\hat{X}$: Synthetic Dataset of X

initialize $\mathfrak{R} = \varnothing$
repeat
  randomly choose an attribute index cluster $\mathcal{C} \in \mathfrak{C}$
  estimate joint distribution $P(A_{\mathcal{C}})$ by JD
  sample $\hat{X}_C$ according to $P(A_{\mathcal{C}})$
  $\mathfrak{C} = \mathfrak{C} \setminus \mathcal{C}$, $\mathfrak{R} = \mathfrak{R} \cup \mathcal{C}$, $\mathfrak{D} = \{D \in \mathfrak{C} | D \cap \mathfrak{R} \neq \varnothing\}$
  for each $D \in \mathfrak{D}$ **do**
    estimate joint distribution $P(A_D)$ by JD
    obtain conditional distribution $P(A_{D\backslash\mathfrak{R}} | A_{D\cap\mathfrak{R}})$ from $P(A_D)$
    sample $\hat{X}_{D\backslash\mathfrak{R}}$ according to $P(A_{D\backslash\mathfrak{R}} | A_{D\cap\mathfrak{R}})$ and $\hat{X}_{D\cap\mathfrak{R}}$
    $\mathfrak{C} = \mathfrak{C} \setminus D$, $\mathfrak{R} = \mathfrak{R} \cup D$, $\mathfrak{D} = \{D \in \mathfrak{C} | D \cap \mathfrak{R} \neq \varnothing\}$
  end for
until $\mathfrak{C} = \varnothing$
return $\hat{X}$

Theorem *10:* The time complexity of Algorithm 6 is

$$O(l(v^{3k} + kmv^{2k} + Nkm + tN(v')^2 + Nkm(v'))), \quad (21)$$

where l is the number of clusters after dimensionality reduction and k here refers to average number of dimensions in these clusters.

Theorem *11:* The space complexity of Algorithm 6 is

$$O(Nkm + v^k km + 2Nv' + Nd). \quad (22)$$

VI. Evaluation

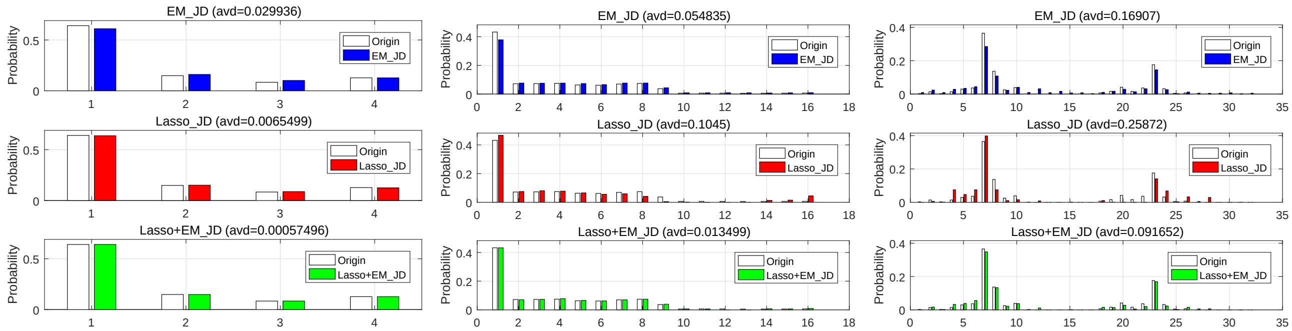
In this section, we conducted extensive experiments on real-world datasets to demonstrate the efficiency of our algorithms in terms of computation time and accuracy. We used three real-world datasets: Retail [1], Adult [5], and TPC-E [2].

Retail is part of a retail market basket dataset, where each record contains distinct items purchased in a shopping visit. Adult is extracted from the 1994 US Census, and contains personal information, such as gender, salary, and education level. TPC-E contains trade records of “Trade type”, “Security”, “Security status” tables in the TPC-E benchmark. We setup a pre-processing phase on the data before running our estimation such that some continuous domains are binned for simplcity.

Datasets	Type	#. Records (N)	#. Attributes (d)	Domain Size
Retail	Binary	27,522	16	2^{16}
Adult	Integer	45,222	15	2^{52}
TPC-E	Mixed	40,000	24	2^{77}

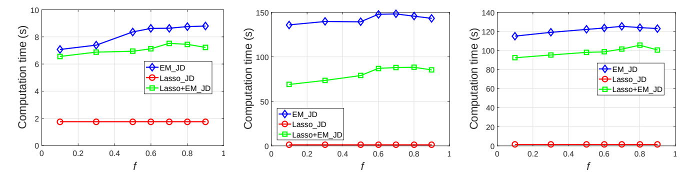
All the experiments were run on a machine with Intel Core i5-5200U CPU 2.20GHz and 8GB RAM, using Windows 7 and Python 2.7. We simulated the crowdsourced environment as follows. First, users read each data record individually and locally transform it into noisy Bloom filters. Then, the crowdsourced bit strings are estimated by the central server for synthesizing and publishing the high-dimensional dataset. We have three strategies, EM_JD, Lasso_JD, and Lasso+EM_JD, in implementing LoPub, which are described below. We run the comparison among EM_JD, Lasso_JD, and Lasso+EM_JD since LoPub adopts a novel LDP paradigm on high-dimensional data. Other competitors are either for

[4]The relationship between mutual information and information entropy can be represented as $I(X;Y) = H(X) + H(Y) - H(X,Y)$, where $H(X)$ and $H(X,Y)$ denote the information entropy of variable X and their joint entropy of X and Y, respectively.



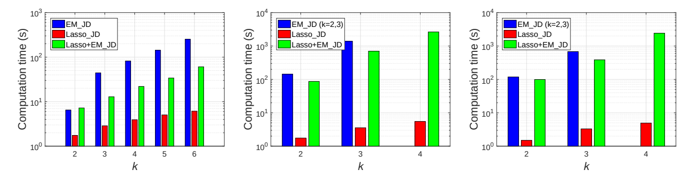
(a) $|\Omega|$=4 (b) $|\Omega|$=16 (c) $|\Omega|$=32

Fig. 6: Histogram ($f = 0.5$)



(a) Retail ($k = 2$) (b) Adult ($k = 2$) (c) TPC-E ($k = 2$)

Fig. 7: Computation time vs. f



(a) Retail ($f = 0.9$) (b) Adult ($f = 0.9$) (c) TPC-E ($f = 0.9$)

Fig. 8: Computation time vs. k

non-LDP [10], [28], [45] or on low-dimension data [18], [19], [23] and therefore not comparable.

For fair comparison, we randomly chose 100 combinations of k attributes from d dimensional data.

The efficiency of LoPub is measured by *computation time* and *accuracy*. The computation time includes CPU time and IO cost. Each set of experiments is run 100 times, and the average running time is reported. To measure accuracy, we used the distance metric AVD (average variant distance), as suggested in [10], to quantify the closeness between the probability distributions $P(\omega)$ and $Q(\omega)$. The AVD is defined as

$$Dist_{\text{AVD}}(P, Q) = \frac{1}{2} \sum_{\omega \in \Omega} |P(\omega) - Q(\omega)|. \tag{23}$$

The default parameters are described as follows. In the binary dataset Retail, the maximum number of bits and the number of hash functions used in the bloom filter are $m = 16$ and $h = 5$, respectively. In the non-binary datasets Adult and TPC-E, the maximum number of bits and the number of hash functions used in Bloom filter are $m = 64$ and $h = 5$, respectively. The convergence gap is set as 0.001 for fast convergence.

A. Multivariate Distribution Estimation

We first demonstrate the running results of our proposed multivariate ($k = 2$) distribution estimation schemes with different domain sizes $|\Omega|$, in Fig. 6. For example, Figs. 6(a), 6(b), and 6(c) show the histograms on a 2 attribute pairs with different domain sizes (i.e., $|\Omega| = 4, 16$, or 32). As we can see, given privacy protection of $f = 0.5$, Lasso_JD can estimate the multivariate distribution effectively but Lasso+EM_JD can have an even better estimation with less complexity. Particularly, in Fig. 6(c), we can see that Lasso_JD can choose those sparse candidates but the items with less probability are assigned to be zero. On the other hand, Lasso+EM_JD can use the estimation result from Lasso_JD as the initial input of EM_JD to speedup the estimation.

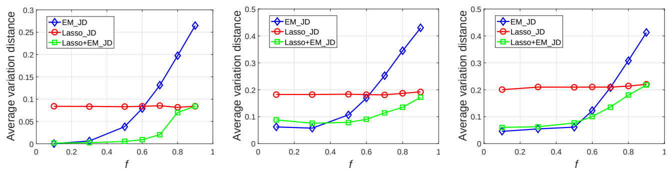
1) Computation Time: We first evaluate the computation time of EM_JD, Lasso_JD, and Lasso+EM_JD for multivariate joint distribution estimation on three real-world datasets with respect to both privacy level f and dimensions k.

Fig. 7 compares the average computation time of $2-$way joint distribution estimation on the three datasets Retail, Adult and TPC-E with the varying privacy levels f. One can see that for different privacy levels f, Lasso_JD is consistently much faster than EM_JD and Lasso+EM_JD, especially when f is large. This is because EM_JD has to repeatedly scan each user's bit string. In other words, the time consumption of EM_JD increases with f because there will be more iterations for the fixed convergence gap. In contrast, Lasso_JD uses the regression to estimate the joint distribution more efficiently. Furthermore, the time complexity of Lasso+EM_JD is also less than EM_JD as the initial estimation of Lasso_JD can effectively reduce both the candidate attribute space and the number of iterations needed. In addition, when f is growing, the computation time of Lasso_JD increases slowly, unlike EM_JD and Lasso+EM_JD. This is because the time complexity of Lasso_JD is mainly subject to the number of users.

Fig. 8 depicts the average computation time with different dimensions k, given a strong privacy protection $f = 0.9$. We should note that, since the domain size of each attribute on dataset Retail is 2, the maximal number of dimension k is chosen as 6 with the domain size of $2^6 = 64$. While the average domain size of Adult and TPC-E after binning is 8, the maximal k is chosen to be 4 with the domain size of 4096. When k is even larger, the maximal domain size will be close to or even exceed the number of records, which will not guarantee the estimation accuracy due to the lack of statistical significance.

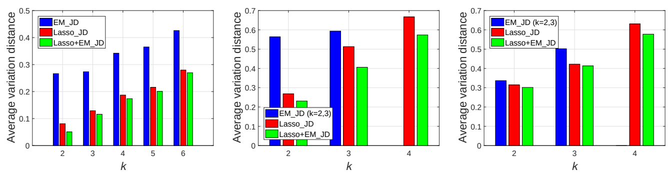
As we can see in Fig. 8(a), EM_JD runs with acceptable time complexity on low dimension $k = 2$. When $k = 3$, the time complexity of EM_JD increases sharply. When k further increases, it does not return any result within an reasonable time in our experiment. However, Lasso_JD can generate the estimation with only a few seconds. This discrepancy is consistent with our complexity analysis, where we envision that the exponential growth of the candidate set will have a significant impact on EM_JD. So, with the initial estimation of Lasso_JD, the combined estimation Lasso+EM_JD can run faster than EM_JD with limited candidate set. The computation time of EM_JD and Lasso_JD on three datasets with

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2812146, IEEE Transactions on Information Forensics and Security



(a) Retail ($k = 2$) (b) Adult ($k = 2$) (c) TPC-E ($k = 2$)

Fig. 9: Accuracy vs. f



(a) Retail ($f = 0.9$) (b) Adult ($f = 0.9$) (c) TPC-E ($f = 0.9$)

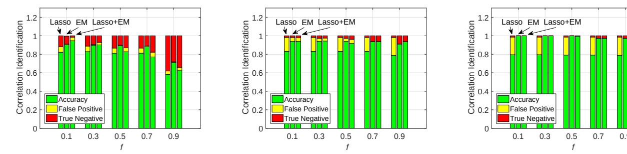
Fig. 10: Accuracy vs. k

different k exhibits a similar tendency, as shown in Figs. 8(b) and 8(c). We omitted the detailed report here due to the space constraint.

2) Accuracy: Next, we compare the estimation accuracy of EM_JD, Lasso_JD, and Lasso+EM_JD.

Fig. 9 reports the average AVD of EM_JD, Lasso_JD, and Lasso+EM_JD on three datasets with different privacy levels f. In particular, when $k = 2$, the AVD of Lasso_JD does not change with f as the aggregated bit sum vector is insensitive for small f. The AVD of EM_JD is very small when f is small, but when f grows, it will sharply increase to as high as 0.28. In contrast, Lasso_JD retains the error around 0.08 even when $f = 0.9$. However, in practice, when f is small, i.e., $f = 0.5$, one can only achieve ϵ-DP with $\epsilon = 10.98$ for each dimension, which is insufficient in general. So, in this sense, when f is large, the AVD of Lasso_JD is comparable to or even better than that of EM_JD. This is because Lasso regression is insensitive to f when estimating the coefficients from the aggregated bit sum vectors. Nonetheless, EM_JD is sensitive to f and prone to some local optimal value because it scans each record of bit strings. In comparison, Lasso+EM_JD achieves a better tradeoff between Lasso_JD and EM_JD. For example, it has less AVD than Lasso_JD when f is small and outperforms EM_JD when f is large. Similar to the conclusion in the binary dataset, when f is large, the trend of Lasso_JD is very close to EM_JD. Besides, Lasso+EM_JD shows very similar performance to EM_JD and incurs relatively small bias.

Fig. 10 also compares the average AVD of EM_JD, Lasso_JD, and Lasso+EM_JD on the three datasets with different k, given sufficient privacy $f = 0.9$ ($\epsilon = 2.0$ for each dimension). We can see that, the AVD of all estimation algorithms increases with k, Particularly, in Fig. 10(a), when k increases from 2 to 6, the estimation error increases gradually. The reason is that the average frequency on k-dimensional attributes is N/v^k and its statistical significance decreases with k exponentially. That is also why dimensionality reduction is necessary for high-dimensional data. When privacy protection is strong, baseline EM_JD is quite sensitive to the initial value and prone to some local optimal due to the scan of each individual's noisy bit string, which leads to great bias. Instead, the AVD of Lasso_JD does not vary with f very much as the aggregated bit sum vector is insensitive to f. However, Lasso+EM_D can further balance between Lasso_JD and EM_JD because the candidate set is much more sparse when k is larger and Lasso+EM_JD can effectively reduce the size of candidate set and iterations. Similar conclusion can be made from the non-binary datasets Adult and TPC-E. As



(a) Retail (b) Adult (c) TPC-E

Fig. 11: Correlation Identification Rate

mentioned before, because of the exhaustive scan on larger candidate domain, the estimation accuracy for $k = 4$ or higher dimensions of EM_JD are not reported on datasets Adult and TPC-E.

B. Correlation Identification

In this section, we present correlations among the multiple attributes that we learn from user data. Particularly, we evaluated loss ratio of dependency relationship of attributes in three datasets. The parameters used in the simulation are set as follows. The dependency threshold is 0.3 for Retail, and is 0.4 for Adult and TPC-E. The number of bits and the number of hash functions in the bloom filter are 16 and 5 for Retail, and 64 and 5 for Adult and TPC-E.

1) Accuracy: Fig. 11 shows both the ratio of correct identification (accuracy), added (false positive) and lost (true negative) correlated pairs after estimation, respectively. From these figures, we can see that all these estimation algorithms can have a relatively accurate identification among the attributes; among which EM_JD and Lasso+EM_JD gain better accuracy than Lasso_JD. One can also see that the accurate rate decreases with f (i.e., privacy level). In Fig. 11(a), the accuracy identified rate is about 85% when the privacy is small (f is less than 0.9). While in Figs. 11(b) and 11(c), the accuracy rate is as high as 95% because the dependency threshold is relatively loose as 0.4. High accurate identification guarantees the correlations among attributes.

In our experiment, the incorrect identification is considered separately with false positive rate and true negative, which reflect the efficiency and effectiveness of dimension reduction. In essence, false positive identification leads to the consideration of the correlations that do not exist; this kind of misidentification only incurs redundant correlations and extra time for learning distribution without imposing leaning errors. On the other hand, true negative identification implies the loss of some correlations among attributes, thus causing information loss in our dimension reduction. For false positive identification, we can see that EM_JD and Lasso+EM_JD are less than Lasso. This comes from the fact that Lasso_JD will choose the sparse probabilities and the mutual information estimated is generally high due to the concentrated probability distribution. Especially in non-binary datasets Adult and TPC-E, the sparsity is much higher, so the estimated probability distribution is more concentrated and the false positive identification rate is high.

The true negative identification in both Adult and TPC-E is small because the true correlations are not very high itself because all attributes have a large domain. Instead, the true correlations in Retail are high and almost any two attributes are dependent. Therefore, the true negative identification is comparatively higher.

2) Effectiveness of Pruning Scheme: We also validated the pruning scheme proposed in Section V-D2 with simulations on the three datasets. We first define the dependency loss ratio as the ratio between the dependency loss after pruning with the original number of dependencies in the adjacent matrix $\mathbf{G}_{d\times d}$ of dependency graph. The complexity reduction ratio is defined as the ratio of reduced pairwise comparisons.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2812146, IEEE Transactions on Information Forensics and Security

TABLE III: Dependency Loss Ratio and Complexity Reduction Ratio (Adult)

ϕ	0.1	0.2	0.3	0.4	0.5
#. Dep (Pruning)	88	38	22	12	6
#. Dep	102	42	24	14	8
Loss Ratio	0.137	0.095	0.083	0.143	0.250
#. Pairs (Pruning)	91	66	55	36	28
#. Pairs	105	105	105	105	105
Reduction Ratio	0.133	0.371	0.476	0.657	0.733

TABLE IV: Dependency Loss Ratio and Complexity Reduction Ratio (TPC-E)

ϕ	0.1	0.2	0.3	0.4	0.5
#. Dep (Pruning)	44	16	16	8	8
#. Dep	46	24	20	10	10
Loss Ratio	0.043	0.333	0.200	0.200	0.200
#. Pairs (Pruning)	231	171	136	66	45
#. Pairs	276	276	276	276	276
Reduction Ratio	0.163	0.380	0.507	0.761	0.837

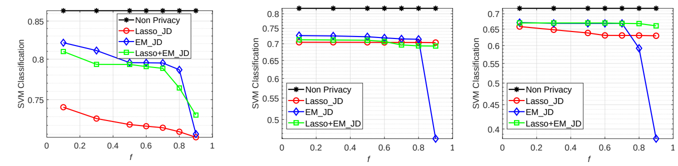
Tables III, IV, and V illustrate the effectiveness of our proposed heuristic pruning scheme. Particularly, as shown in Tables III and IV, with the increase of ϕ, which shows the strength of correlations, the number of original dependencies in dataset Adult decreases dramatically. Also, the dependencies after the heuristic pruning decrease accordingly and their number is quite close to the original dependence. However, when ϕ increases, the number of pairwise comparison becomes less, compared to the full pairwise comparison. So, it shows that the heuristic pruning scheme can effectively reduce the complexity with only small sacrifice of dependency accuracy. Similar conclusion can be found in Table IV on non-binary dataset TPC-E. On the binary dataset Retail, due to the prior knowledge that binary datasets normally have strong mutual dependency, we slightly change the pruning scheme. More specifically, we assume that all the attributes are dependent with each other and our pruning scheme aims at finding the non-dependency from those attributes A_j with less entropy $H(A_j)$. According to Table V, the number of dependencies after pruning decreases slowly and the minus symbol in the dependency loss ratio means that there is no loss of dependencies but there are redundant dependencies that should not exist in original datasets. It should be noted that redundant dependencies cover all the original dependencies. Therefore, the redundancy will not degrade data utility since more correlations are kept. However, efficiency in terms of dimensionality reduction, which should cut off as many unnecessary correlations as possible, is hindered. So, according to Table V, we can also say that the heuristic pruning scheme can achieve up to 50% complexity reduction without loss of dependencies.

C. SVM and Random Forest Classifications

To show the overall performance of LoPub, we evaluated both the SVM and random forest classification error rate in the datasets synthesized by three different implementations, Lasso_JD, EM_JD, and Lasso+EM_JD of LoPub. We first sampled from the three original datasets Retail, Adult, and TPC-E to get both the training sets and test sets. Then, we generated the privacy-preserving synthetic datasets from the training data. Next, we trained three different SVM classifiers and three random forest classifiers on the synthetic datasets. Lastly, we evaluated the classification rate on the original sampled test sets. Particularly, the average random forest classification rate is computed on all the original attributes and the average SVM classification rate is computed on all the original binary-state attributes in each dataset, for example, all attributes in binary dataset Retail, the 10th (gender) and 15th (marital) attribute in Adult, and the 2nd, 10th, 23rd, and 24th attribute in TPC-E. For comparison, we also trained the corresponding SVM and random forest classifiers on each sampled training set and measured their classification rate.

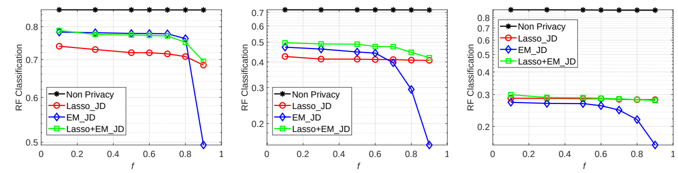
TABLE V: Dependency Loss Ratio and Complexity Reduction Ratio (Retail)

ϕ	0.1	0.15	0.2	0.25	0.3
#. Dep (Pruning)	256	256	256	250	244
#. Dep	240	240	238	220	200
Loss Ratio	-0.067	-0.067	-0.076	-0.136	-0.220
#. Pairs (Pruning)	91	91	78	66	55
#. Pairs	120	120	120	120	120
Reduction Ratio	0.242	0.242	0.350	0.450	0.512



(a) Retail (b) Adult (c) TPC-E

Fig. 12: SVM Classification Rate of LoPub



(a) Retail (b) Adult (c) TPC-E

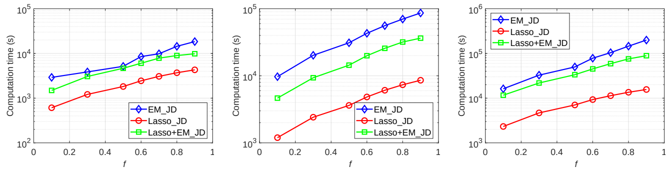
Fig. 13: Random Forest Classification Rate of LoPub

Fig. 12 shows the average accurate SVM classification rate on three datasets Retail, Adult and TPC-E. In all subfigures, the average SVM classification rate decreases with f. Generally, when f is small ($f < 0.9$), the classification rate drops slowly. Nevertheless, when $f = 0.9$, there will be a large gap. This is because the level of different privacy protections varies as shown in Equation (9). For SVM, the classification rate is relatively close to the that of non-private case. This can be attributed to the fact that SVM classification only considers binary-state attributes and the distribution estimation on binary-state attributes can be more accurate than non-binary attributes, which have sparser distribution. In all figures, we can see that Lasso_JD has generally worse classification rate because of its biased estimation. EM_JD generally outperforms others but still shows performance degradation when f is large, while Lasso+EM_JD could find a better balance between other methods.

However, in Fig. 13, due to the high sparsity in the distribution of non-binary attributes, the joint distribution estimation on non-binary attributes may be biased and that is why the random forest classification on our synthetic datasets is not as good as SVM classification. Nonetheless, the synthetic data still keeps sufficient information of original crowdsourced datasets. For example, the worst random forest classification rate of our proposed Lasso_JD and Lasso+EM_JD in the three datasets is 67%, 42%, and 26%, which are much larger than the average random guess rate of 50%, 15%, and 13%, respectively. EM_JD only works well when f is small while Lasso_JD causes larger bias in the random forest classification with small f. However, with the initial estimation of Lasso_JD, Lasso+EM_JD works well and degrades gracefully with f.

The overall computational time for synthesizing new datasets is also presented in Fig. 14. Despite the worst utility, Lasso_JD is the most efficient solution, which achieves approximately ten times faster than the EM_JD. Without the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2812146, IEEE Transactions on Information Forensics and Security



(a) Retail (b) Adult (c) TPC-E

Fig. 14: Overall Time of LoPub

dominant I/O time for data synthesization, the reduction of computation time can be even greater. As mentioned before, that is because Lasso_JD can estimate the joint distribution regardless of the number of bit strings. With the initial estimation of Lasso_JD, Lasso+EM_JD can then be effectively simplified from two aspects: the sparse candidates can be limited and the initial value is well set. Instead, the baseline EM_JD not only needs to build prior probability distribution for all candidates but also begins the convergence with a randomness value.

VII. Conclusion

In this paper, we propose a novel solution, LoPub, to achieve the high-dimensional data publication with LDP in crowdsourced data publication systems. Specifically, LoPub learns from the distributed data records to build the correlations and joint distribution of attributes, synthesizing an approximate dataset for privacy protection. To realize the efficient multi-variate distribution estimation, we propose three distribution estimation schemes, among which the hybrid scheme with the combined use of EM and Lasso regression reaches the best balance between the data utility and privacy. The experimental results using real-world datasets show that LoPub is an efficient and effective mechanism to release a high-dimensional dataset while providing sufficient LDP guarantees for crowdsourced data providers.

References

[1] Frequent itemset mining dataset. http://fimi.ua.ac.be/data/.
[2] Trans. processing performance council. http://www.tpc.org/.
[3] G. Ács, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. In *Proc. of IEEE ICDM*, pages 1–10, 2012.
[4] G. Ács and C. Castelluccia1. I have a dream! (differentially private smart metering). In *Proc. of Information Hiding Workshop*, 2011.
[5] K. Bache and M. Lichman. Uci machine learning repository. https://archive.ics.uci.edu/ml/datasets.html/, 2013.
[6] R. Bassily, K. Nissim, U. Stemmer, and A. Thakurta. Practical locally private heavy hitters. In *Proc. of NIPS*, 2017.
[7] R. Bassily and A. SmithLi. Local, private, efficient protocols for succinct histograms. In *Proc. of ACM STOC*, 2015.
[8] V. Bindschaedler, S. Rane, A. Brito, V. Rao, and E. Uzun. Achieving differential privacy in secure multiparty data aggregation protocols on star networks. In *Proc. of ACM CODASPY*, 2017.
[9] R. Chen, H. Li, A. K. Qin, S. P. Kasiviswanathan, and H. Jin. Private spatial data aggregation in the local setting. In *Proc. IEEE ICDE*, pages 289–300, 2016.
[10] R. Chen, Q. Xiao, Y. Zhang, and J. Xu. Differentially private high-dimensional data publication via sampling-based inference. In *Proc. of ACM KDD*, pages 129–138, 2015.
[11] W. Day and N. Li. Differentially private publishing of high-dimensional data using sensitivity control. In *Proc. of the ASIACCS*, pages 451–462. ACM, 2015.
[12] B. Ding, J. Kulkarni, and S. Yekhanin. Collecting telemetry data privately. In *Proc. of NIPS*, 2017.
[13] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Local privacy and statistical minimax rates. In *Proc. of IEEE FOCS*, pages 429–438, 2013.
[14] C. Dwork. Differential privacy. In *Proc. of ICALP*, pages 1–12. 2006.
[15] C. Dwork. Differential privacy: A survey of results. In *Proc. Springer TAMC*, pages 1–19. 2008.
[16] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. *Proc. of TCC*, pages 265–284, 2006.
[17] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations & Trends®in Theoretical Computer Science*, 9(3), 2013.
[18] U. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proc. of ACM CCS*, 2014.
[19] G. Fanti, V. Pihur, and Ú. Erlingsson. Building a rappor with the unknown: Privacy-preserving learning of associations and data dictionaries. *Proceedings on Privacy Enhancing Technologies*, 2016(3):41–61, 2016.
[20] M. M. Groat, B. Edwards, J. Horey, W. He, and S. Forrest. Enhancing privacy in participatory sensing applications with multidimensional data. In *Proc. of IEEE PerCom*, pages 144–152. IEEE, 2012.
[21] N. Holohan, D. J. Leith, and O. Mason. Optimal differentially private mechanisms for randomised response. *IEEE Trans. Inf. Forensics Security*, 12(11):2726–2735, Nov 2017.
[22] J. Hua, A. Tang, Y. Fang, Z. Shen, and S. Zhong. Privacy-preserving utility verification of the data published by non-interactive differentially private mechanisms. *IEEE Trans. Inf. Forensics Security*, 11(10):2298–2311, Oct 2016.
[23] P. Kairouz, K. Bonawitz, and D. Ramage. Discrete distribution estimation under local privacy. *arXiv preprint: 1602.07387*, 2016.
[24] P. Kairouz, S. Oh, and P. Viswanath. Extremal mechanisms for local differential privacy. In *Proc. of NIPS*, pages 2879–2887, 2014.
[25] G. Kellaris and S. Papadopoulos. Practical differential privacy via grouping and smoothing. *VLDB*, 6(5):301–312, 2013.
[26] W. Z. Khan, Y. Xiang, M. Y. Aalsalem, and Q. Arshad. Mobile phone sensing systems: A survey. *IEEE Commun. Surveys Tuts.*, 15(1):402–427, 2013.
[27] G. Li, J. Wang, Y. Zheng, and M. J. Franklin. Crowdsourced data management: A survey. *IEEE Trans. Knowl. Data Eng.*, 28(9):2296–2319, 2016.
[28] H. Li, L. Xiong, and X. Jiang. Differentially private synthesization of multi-dimensional data using copula functions. In *Proc. EDBT*, pages 475–486, 2014.
[29] L. Li, R. Lu, K. K. R. Choo, A. Datta, and J. Shao. Privacy-preserving-outsourced association rule mining on vertically partitioned databases. *IEEE Trans. Inf. Forensics Security*, 11(8):1847–1861, Aug 2016.
[30] X. Liang, X. Li, R. Lu, X. Lin, and X. Shen. Udp: Usage-based dynamic pricing with privacy preservation for smart grid. *IEEE Trans. Smart Grid*, 4(1):141–150, 2013.
[31] C. Liu, S. Chakraborty, and P. Mittal. Dependence makes you vulnerable: Differential privacy under dependent tuples. In *Proc. of NDSS*, 2016.
[32] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proc. of ACM SIGMOD*, 2009.
[33] M. Naveed, E. Ayday, E. W. Clayton, J. Fellay, C. A. Gunter, J. Hubaux, B. A. Malin, and X. Wang. Privacy in the genomic era. *ACM Comput. Surv.*, 48(1):6, 2015.
[34] W. Qardaji, W. Yang, and N. Li. Priview: practical differentially private release of marginal contingency tables. In *Proc. of ACM SIGMOD*, pages 1435–1446, 2014.
[35] 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. of ACM CCS*, 2016.
[36] Z. Qin, T. Yu, Y. Yang, I. Khalil, X. Xiao, and K. Ren. Generating synthetic decentralized social graphs with local differential privacy. In *Proc. of ACM CCS*, 2017.
[37] X. Ren, C.-M. Yu, W. Yu, S. Yang, X. Yang, J. McCann, and P. Yu. Lopub: High-dimensional crowdsourced data publication with local differential privacy. *arXiv preprint arXiv:1612.04350*, 2016.
[38] R. Sarathy and K. Muralidhar. Evaluating laplace noise addition to satisfy differential privacy for numeric data. *Transactions on Data Privacy*, 4(1):1–17, 2011.
[39] H. Shen, M. Zhang, and J. Shen. Efficient privacy-preserving cube-data aggregation scheme for smart grids. *IEEE Trans. Inf. Forensics Security*, 12(6):1369–1381, 2017.
[40] S. Su, P. Tang, X. Cheng, R. Chen, and Z. Wu. Differentially private multi-party high-dimensional data publishing. In *Proc. of IEEE ICDE*, pages 205–216, 2016.
[41] Q. Wang, Y. Zhang, X. Lu, Z. Wang, Z. Qin, and K. Ren. Rescuedp: Real-time spatio-temporal crowd-sourced data publishing with differential privacy. In *Proc. of IEEE INFOCOM*, pages 1–9, 2016.
[42] T. Wang, J. Blocki, N. Li, and S. Jha. Locally differentially private protocols for frequency estimation. In *USENIX Security Symposium*, 2017.
[43] W. Wang and Q. Zhang. Privacy-preserving collaborative spectrum sensing with multiple service providers. *IEEE Trans. Wireless Commun.*, 14(2):1011–1019, 2015.
[44] S. L. Warner. Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias. *J. Am. Stat. Assoc.*, 60:63–69, 1965.
[45] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: Private data release via bayesian networks. In *Proc. of ACM SIGMOD*, pages 1423–1434, 2014.
[46] Y. Zhang, Q. Chen, and S. Zhong. Privacy-preserving data aggregation in mobile phone sensing. *IEEE Trans. Inf. Forensics Security*, 11(5):980–992, May 2016.
[47] T. Zhu, P. Xiong, G. Li, and W. Zhou. Correlated differential privacy: Hiding information in non-iid data set. *IEEE Trans. Inf. Forensics Security*, 10(2):229–242, 2015.