Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Topology-theoretic approach to address attribute linkage attacks in differential privacy

Jincheng Wang[a,*], Zhuohua Li[a], John C.S. Lui[a,*], Mingshen Sun[b]

[a] Department of Computer Science and Engineering, The Chinese University of Hong Kong,Hong Kong, China
[b] Baidu Security, Beijing, China

ARTICLE INFO

Article history:
Received 15 May 2021
Revised 4 November 2021
Accepted 15 November 2021
Available online 18 November 2021

Keywords:
Differential privacy
Attribute linkage attack
Set-valued dataset
Topology theory
APLKiller

ABSTRACT

Differential Privacy (DP) is well-known for its strong privacy guarantee. Briefly speaking, DP algorithms guarantee that the statistical information of the data is roughly preserved, and at the same time, individual privacy is protected with guarantees. However, when there are correlations among the attribute in the dataset, only relying on DP is not sufficient to defend against the attribute linkage attack, which is a well-known privacy attack aiming at deducing individuals' private information. In the attribute linkage attack, the adversary can leverage prior knowledge about the victim, combined with accessing the published dataset, to infer sensitive information about a victim. In this paper, we study the attribute linkage attack in DP settings, and argue that enhancing DP can give users a higher level of privacy guarantees. Our contributions are ① we show that the attribute linkage attack can be initiated with high probability under the protection of DP, ② we propose a variant of DP called *APL-Free ϵ-DP* to provide a higher level of privacy guarantees, ③ we design an algorithm *APLKiller* which satisfies the APL-Free ϵ-DP. Finally, experiments show that our algorithm not only eliminates the attribute linkage attack, and at the same time, it has a better ability to extract useful information from the data.

1. Introduction

In the current digital era, personal information has become valuable. Using these data, companies can provide personalized recommendations by understanding users' behavior and devise better advertising strategies to improve recommendation models. These personal data range from names, shopping preferences, health records, etc., and these data are continuously being collected by companies or internet service providers via various channels. In the meantime, because data collection is becoming ubiquitous, privacy becomes a serious concern. For instance, there have been severe privacy breaches in recent years (Cadwalladr, 2018; MyFitnessPal, 2018) which compromised user privacy.

Industry and academia have put in significant effort on how to protect personal privacy. One method is to anonymize the data before publishing them to the public (Machanavajjhala et al., 2006; Sweeney, 2002). Unfortunately, individuals' private information could still be leaked (Barbaro et al., 2006; Narayanan and Shmatikov, 2006). One of the most damaging privacy attacks is the *attribute linkage attack* (Chen et al., 2013). In this attack, the attacker can leverage part of attribute information to deduce more information about the victim.

Recently, differential privacy (DP) was proposed (Dwork et al., 2014), and researchers have proposed various DP algorithms. Briefly speaking, a DP algorithm adds random noise to the dataset to preserve user privacy. Currently, companies like Google (Erlingsson et al., 2014) and Uber (Near, 2018) are using DP algorithms to enhance their data services and to protect user privacy. DP algorithms can be categorized in two different settings: *non-interactive DP* and *interactive DP*. The former is for publishing datasets to the public, while the latter is for responding to users' queries to a dataset, e.g., a query can be: "how many participants in the dataset are male?".

Despite its strong privacy guarantees, DP is not without any pitfalls. For example, how to determine the privacy budget ϵ is a non-trivial issue that requires rich experience from data publishers (Lee and Clifton, 2011). A high value of ϵ will result in a low degree of privacy but a high degree of data utility, that is, the ability to extract useful information from the data. So how to set a proper value for ϵ is known as the *utility-privacy dilemma*. Also, correlations among records, e.g., social relationships, can degrade the privacy guarantee of DP (Chen et al., 2014; Kifer and Machanavajjhala, 2011; 2012; Liu et al., 2016). In this paper, we show that DP-processed datasets are prone to the attribute linkage

* Corresponding authors.
E-mail addresses: jcwang@cse.cuhk.edu.hk (J. Wang), zhli@cse.cuhk.edu.hk (Z. Li), cslui@cse.cuhk.edu.hk (J.C.S. Lui), sunmingshen@baidu.com (M. Sun).

https://doi.org/10.1016/j.cose.2021.102552
0167-4048/© 2021 Elsevier Ltd. All rights reserved.

Table 1
Examples of a retail dataset D and its perturbed version D_p using differential privacy.

ID	Itemset	Occurrences	ID	Itemset	Occurrences
1	beer, toothpaste, scissor, hanger	20	1	beer, toothpaste, scissor, hanger	23
2	beer, toothpaste	4	2	beer, toothpaste	2
3	beer	1	3	beer, toothpaste, scissor	8
4	beer, toothpaste, scissor	12 4	scissor, hanger	5	
5	scissor, hanger	3	5	toothpaste	2

(a) An example of retail dataset D (b) A possible DP processed dataset D_p

attack when attributes in the original dataset are correlated. Such attribute correlations will introduce attribute linkage vulnerability with high probability.

Table 1 is an example to illustrate the attribute linkage attack. Specifically, Table 1(a) is a retail dataset D in which each record contains a unique itemset with the number of its occurrence, i.e., the number of customers who bought the corresponding itemset. Note that in Table 1(a), there exists correlations among items. For example, statistically most customers who bought {*toothpaste, scissor, hanger*} would like to buy *beer*. The consequence of having such a correlation is that all itemsets which contain {*toothpaste, scissor, hanger*} have zero occurrence, except for the itemset {*beer, toothpaste, scissor, hanger*} with 20 occurrence. Given the above correlation in the dataset, adversaries can initiate the attribute linkage attack. For simplicity, we will use “Eve” to represent the adversary and “Alice” to represent the victim. If Eve knows that Alice bought the itemset {*toothpaste, scissor, hanger*} in advance and accesses the dataset D, she can search for all records which contain these three items, and finally uniquely identify the first record in Table 1(a). The consequence is that Eve can deduce that Alice also bought *beer*, which is private information for Alice. Since Eve can use the itemset {*toothpaste, scissor,hanger*} as prior knowledge to uniquely identify a record, we call this itemset an *attribute privacy leakage* (APL), and a formal definition will be given in Section 4. Specifically, Eve can initiate the attribute linkage attack by linking the APL with a unique record in D, which eventually causes Alice’s private information leakage.

To provide privacy protection, the data publisher may use DP algorithms to perturb the dataset D and output the DP-processed dataset D_p. For example, a classical DP algorithm for histogram data is presented (Dwork et al., 2014): For each itemset in the itemset universe, a random positive/negative Laplace noise is added to the number of its occurrence to compute the perturbed number of its occurrence. Any itemset with a non-positive perturbed number of occurrences will not be added to D_p. As a result, for an itemset S, if the number of its actual occurrence is low, the probability for S to be added to D_p will be low. Table 1(b) shows a possible DP-processed dataset D_p. Because of the noise introduced by the DP algorithm, the number of occurrences of itemsets has been modified. For example, the itemset {*beer*} exists in D, but it does not exist in D_p because of the negative Laplace noise.

Unfortunately, in our example, one can check that the APL is not eliminated in the DP-processed dataset D_p. Specifically, the itemset {*toothpaste, scissor, hanger*} is not only an APL in D, but also an APL in D_p. The consequence is that Eve can uniquely identify Alice’s record in D_p, and deduce that Alice bought *beer*. In this work, we highlight that there is a high probability that the existence of an APL in D will be well preserved in D_p, which allows the attribute linkage attack in DP settings. The root cause is that the existence of an APL in the dataset depends on the underlying item correlation, which DP algorithms try their best to preserve for data utility. As a result, DP algorithms preserve the correlation for the data utility, and at the same time, preserve the existence of APLs. In our example, the underlying correlation in D between {*toothpaste, scissor, hanger*} and *beer* guarantees that the occurrence of {*beer, toothpaste, scissor, hanger*} is large, while any other itemset which contains {*toothpaste, scissor, hanger*} is zero. Such a correlation is well preserved in D_p as well as the existence of the APL {*toothpaste, scissor, hanger*}.

In order to eliminate the APLs and defend against the attribute linkage attack in DP settings, we argue that enhancing DP can strengthen users’ privacy, and propose the *APL-Free ϵ-DP*, which addresses the APL issue in DP settings. Furthermore, we design an algorithm, *APLKiller*, which aims to publish datasets and, at the same time, satisfies the APL-Free ϵ-DP. Experimental results show that our algorithm APLKiller guarantees user privacy and also provides high data utility. In summary, our contributions are:

- We show that applying traditional DP algorithms to real-world datasets can create possibilities of the attribute linkage attack on the DP-processed dataset. Specifically, an adversary can deduce the private information of victims with a high probability in DP settings.
- To eliminate APLs and defend against the attribute linkage attack, we propose the APL-Free ϵ-DP. We show that an algorithm which satisfies APL-Free ϵ-DP can guarantee that no attribute linkage attack will be initiated on the processed dataset.
- We design a novel algorithm, APLKiller, which is based on a topology-theoretic approach (Dowker, 1952) to defend against the attribute linkage attack in DP settings. It has $O(mn)$ time complexity, where m is the number of records in the dataset, and n is the number of items in the item universe. Furthermore, our algorithm preserves the data utility when processing the dataset. Evaluation results show that the privacy guarantee of APLKiller is better than that of the traditional DP algorithm. Moreover, evaluation results show that the data utility of APLKiller is higher than that of the traditional DP algorithm.

The rest of this paper is organized as follows: In Section 2, we review correlation issues in DP, the attribute linkage attack and current work of privacy protection using topology theory. In Section 3, we present some preliminaries. In Section 4, we give an illustration and analysis of the attribute linkage attack in DP settings. In order to defend against the attribute linkage attack, we propose the APL-Free ϵ-DP and the algorithm APLKiller in Section 5.2, with experiment evaluations in Section 6. Finally in Section 7, we conclude our work.

Table 2
A table of definitions for commonly used symbols.

Symbol	Descriptions
$\mathcal{I}$	The item universe. Specifically, let n be the size of the universe, and $I_k \in \mathcal{I}$ ($k \in \{1, \ldots, n\}$) is the kth item in $\mathcal{I}$.
S	An itemset which consists of several items, i.e., $S \subseteq \mathcal{I}$. Specifically, $\|S.items\|$ is the number of items in S, and $\|S\|$ is the number of occurrences of S in the dataset.
B_S	The boundary set for the itemset S (See Definition 1). Specifically, each element in B_S is called a "boundary itemset".
Q	An APL in a dataset (See Definition 4).
D	A set-valued dataset which is in the form of $\{(S, \|S\|)\}$. Specifically, D^I is the set of itemsets in D.
D_p	The DP-processed dataset of the original dataset D.
D_i	A subset of the dataset D which contains all i-itemset records in D, i.e., the set of records which contain itemsets of size i for $i > 0$.
ϵ	The privacy budget parameter used for DP algorithms.

2. Background

In this section, we first provide the background information of DP. Then we give an overview of the attribute linkage attack. Finally, we discuss how topology theory (Erdmann, 2017) helps to defend against the attribute linkage attack. Note that we here list descriptions of commonly used symbols throughout this paper for reference, as shown in Table 2.

2.1. Correlation issues in differential privacy

The main idea of standard DP is that by adding or deleting one record from the dataset, it will have a negligible impact on the query results. Therefore, it can hide evidence of an individual in the dataset. DP was initially used for interactive settings in which the user submits a query on the dataset, for example, "how many people visit google.com in the web log?", and the user will get the corresponding answer with some added noise to increase privacy. However, it is reported that only a limited number of queries can be answered and the flexibility of performing personalized data analysis tasks is constrained under this setting (Leoni, 2012; Mohammed et al., 2011). Non-interactive DP, which aims to sanitize the publication of the whole datasets, becomes an alternative. It is essential in many situations like publishing medical or census data. Specialized non-interactive DP algorithms were designed for low dimensional data publishing (Acs et al., 2012; Hardt et al., 2012; Li et al., 2014; Zhang et al., 2014), as well as for high-dimensional data release (Chen et al., 2011; 2015; Wang et al., 2016; Zhang et al., 2017).

Although standard DP is elegant, there are still usability and privacy leakage issues. For example, how to set the privacy parameter ϵ is not easy to decide (Lee and Clifton, 2011). Attacks targeting DP systems have also been reported (Haeberlen et al., 2011). One of the most serious issues is that correlations among records can decrease privacy guarantees of DP (Chen et al., 2014; Kifer and Machanavajjhala, 2011; 2012; Liu et al., 2016). Specifically, the victim's private information can be encoded in the social correlation, which is formed by a specific group of participants, e.g., friends or families. Once such underlying social correlations are discovered, the victim's private information is under leakage.

Our paper points out that the victim's private information can also be encoded in the item correlation, which is formed by all participants. The item correlation can create APLs and allows the attacker to initiate the attribute linkage attack in the original dataset. Even worse, DP algorithms cannot correctly handle these APLs, and these APLs are preserved in the DP-processed dataset with high probability. The consequence is that even though DP algorithms are applied to increase privacy, the attacker can still initiate the attribute linkage attack and cause privacy leakage. Compared with the social correlation issue in DP, we highlight that the item correlation issue is more general and severe for the following reasons.

1. It is easier for attackers to discover the item correlation in data than the social correlation.
2. By leveraging item correlations, the attacker can easily construct the attribute linkage attack to leak the private information.
3. It is harder for data publishers to defend against the attribute linkage attack because the underlying item correlation is formed by all participants instead of a small group.

Note that we focus on the non-interactive DP setting in most parts of this paper because of its broad applicability and flexibility. Also, it is much more intuitive and clear to use the non-interactive DP setting to show the attribute linkage attack. In addition, many papers focus on counting queries since counting is a fundamental task in data mining. So we also focus on counting queries and their derivatives.

2.2. Attribute linkage attack

The attribute linkage attack is one of the most damaging privacy leakage attacks (Xu et al., 2014). The attacking philosophy is to use the combination of non-private attributes, e.g., zip-code and birthday, to uniquely identify the victim's record and deduce values of private attributes, e.g., the medical information of an individual. This attack can happen because an adversary can take advantage of specific attribute correlations in the dataset, which create APLs. Once the adversary identifies these APLs, by accessing the dataset, the adversary can deduce the victim's private information, e.g., the victim's medical information.

In a set-valued dataset, each item in the dataset can be viewed as a binary attribute. In Section 1, we have provided an example to give an intuition of the attribute linkage attack on a set-valued dataset. Previous investigations (Barbaro et al., 2006; Kifer and Machanavajjhala, 2012; Narayanan and Shmatikov, 2006; 2008) have shown some real-world cases of the attribute linkage attack on set-valued datasets. One can check that the attribute linkage attack can be easily initiated in the real world. For example, in Narayanan and Shmatikov (2008), authors show that one can uniquely identify a movie subscriber's record in the IMDb dataset by only using part of his/her movie rating histories. Specifically, only two rating histories with dates can uniquely identify 68% of subscribers' records. Once the identification is successful, deducing more ratings of a subscriber is then possible. Note that it is usually challenging to distinguish beforehand which items are private and which are non-private. Therefore, treating each item as equally private is necessary to provide a strong privacy guarantee, while it increases the complexity of designing privacy-preserving algorithms.

Various anonymization standards have been proposed for publishing datasets while defending against the attribute linkage attack. They are usually based on the generalization techniques (He and Naughton, 2009; Terrovitis et al., 2008). The main criticisms of such techniques are that they are easy to attack, and data generalization sharply decreases data utility (Dwork et al., 2014; 2017). Although DP provides a much stronger privacy guarantee than those anonymization standards, little research has focused on the attribute linkage attack in DP settings. In this paper, we argue that DP is still not immune to the attribute linkage attack. The attribute correlation will enlarge the possibility of the attribute linkage attack in DP settings. In order to defend against the attribute linkage attack, we here introduce a topology-theoretic approach.

Table 3
Relation R for the dataset D_p in Table 1>(b).

ID/Item	beer	toothpaste	scissor	hanger
1	•	•	•	•
2	•	•		
3	•	•	•	
4			•	•
5		•		

Table 4
Relation R_q on {beer, toothpaste, scissor, hanger}.

ID / Item	beer	toothpaste	scissor	hanger
1	•	•		
2	•	•	•	
3			•	•
4		•		

Table 5
Perturbed D_p' to defend against the attribute linkage attack.

ID	Itemset	Occurrences
1	beer, toothpaste, scissor, hanger	23
2	beer, toothpaste	2
3	beer, toothpaste, scissor	8
4	scissor, hanger	5
5	toothpaste	2
6	toothpaste, scissor, hanger	2
7	beer,scissor, hanger	3
8	beer,toothpaste, hanger	1

2.3. Topology of privacy

Recently security researchers presented a formalism to study privacy using topology theory (Erdmann, 2017). Specifically, they first model a dataset D as a relation $R : X \times Y$, in which X represents a set of records in D, and Y represents the item universe, e.g., all kinds of commodities in a shop. For example, one can transform the DP-processed dataset D_p in Table 1(b) into a relation R as shown in Table 3.

Suppose one wants to determine whether an attribute linkage attack can be initiated on a record r which contains the itemset $S =$ {*beer, toothpaste, scissor, hanger*}, the first step is to project R in Table 3 onto a sub-relation $R_q : X' \times S$, in which X' is the set of records containing at least one item in S. Table 4 is an example. Then we can use topology theory to prove that there will be a possibility of an attribute linkage attack targeting the itemset S if and only if any element in the *boundary set* B_S is missing in R_q.

Definition 1 (Boundary Set)**.** The boundary set B_S of an itemset S is generated by removing each item from S. That is,

$$B_S = \{S' \subset S : |S.items| - |S'.items| = 1\},$$

where $|S.items|$ is the size of S, and each element $S' \in B_S$ is called a *boundary itemset*.

Note that given $|S.items| = n$, the number of boundary itemsets in B_S is also n. For example, the boundary set B_S of the itemset $S =$ {*beer, toothpaste, scissor, hanger*} contains the following four boundary itemsets: {*toothpaste, scissor, hanger*}, {*beer, scissor, hanger*}, {*beer, toothpaste, hanger*}, and {*beer, toothpaste, scissor*}. The above result implies that any missing boundary itemset S' is an APL, and can be used as the prior knowledge to uniquely identify the target's itemset S. Moreover, $S - S'$ is the leaked information. Referencing to our example, since the boundary itemset {*toothpaste, scissor, hanger*} is missing in Table 4, Eve can deduce that Alice also bought *beer*.

Let D^I be the set of itemsets in D. We can further use topology theory to prove that if there is no missing boundary itemset for each *maximal itemset* in D^I, then there will be no attribute linkage attack targeting records in D.

Definition 2 (Maximal Itemset)**.** A maximal itemset of D^I is an itemset $S \in D^I$ that is not a subset of any other itemset in D^I.

For example, in Table 1, the only maximal itemset is {*beer, toothpaste, scissor, hanger*}. The above result implies a powerful defense methodology: *For each maximal itemset $S \in D^I$, if one can artificially generate records for all missing boundary itemsets, no attribute linkage attack can happen in the generated dataset.* We will use this idea to derive our defense methodology in Section 5.2. Going back to our example, Table 5 is an example dataset wherein every itemset is free from the attribute linkage attack. Records 6–8 are the artificial records which are added for the only maximal itemset {*beer, toothpaste, scissor, hanger*}. Specifically, each artificial record contains a missing boundary itemset in the original dataset. One can check that with the artificial record which contains the itemset {*toothpaste, scissor, hanger*}, Eve cannot uniquely identify Alice's record, and deduce that Alice bought *beer* anymore. We will discuss the detail of our proposed methodology in Section 5.2, which defends against the attribute linkage attack in DP settings.

3. Preliminaries

In this section, we first give a formal definition of the set-valued dataset, then we introduce various terms to formalize differential privacy. Note that in this paper, we will use the term "item" and "attribute" interchangeably because each item can be viewed as a binary attribute, i.e., an attribute with range {0, 1}.

3.1. Set-valued dataset

Set-valued data are commonly used to represent data, e.g., shopping lists, visited web-pages and click streams. Let $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ be the item universe with size n, and an itemset $S \subseteq \mathcal{I}$ is a subset of $\mathcal{I}$, with $|S.items|$ being the number of items in S and $|S|$ being the number of occurrences of S in the dataset. For example, in Table 1(a), each item is a commodity in a supermarket and $\mathcal{I} = \{beer, toothpaste, scissor, hanger\}$. An itemset S could be {*beer, toothpaste, scissor, hanger*} with $|S.items| = 4$ and $|S| = 20$. A set-valued dataset D can be represented using a histogram, in which each record r stores a unique itemset S with its number of occurrences $|S|$ in the dataset, i.e., $D = \{(S, |S|)\}$. We use D^I to represent the set of itemsets in D, that is, $D^I = \{S : (S, |S|) \in D\}$.

3.2. Differential privacy

DP (Dwork, 2011) is a mathematical framework designed to protect users' privacy. The main goal of DP is to guarantee that whether or not a person participated in the dataset will not dramatically increase the risk of individual information being leaked. In DP, the parameter ϵ, which is the privacy budget, is determined in advance to decide the level of privacy and noise introduced to the dataset D. Note that in the set-valued dataset, one's participation will influence the number of occurrences of a specific itemset. For example, removing Alice's participation from Table 1(a) will reduce the number of occurrences of the itemset {*beer, toothpaste,* *scissor, hanger*} by 1. Specifically, the formal definition of DP is as follows.

Definition 3 (**ϵ-differential privacy**)**.** A randomized algorithm $\mathcal{A}$ provides ϵ-differential privacy if for any two neighboring set-valued datasets D_1 and D_2 which only differ in one occurrence of an itemset, and for any output $D_p \subseteq Range(\mathcal{A})$,

$$\frac{\Pr(\mathcal{A}(D_1) = D_p)}{\Pr(\mathcal{A}(D_2) = D_p)} \le e^{\epsilon},$$

where the probability is taken over the randomness of $\mathcal{A}$.

Generally speaking, two popular noise additive mechanisms are widely used by DP algorithms: the Laplace mechanism and the Exponential mechanism (Dwork et al., 2014). Specifically, the key idea of the Laplace mechanism, which is used in our paper, is to generate a noisy answer $x = Lap(\mu, b)$ from the Laplace distribution, where b is the scaling parameter, and μ is the true query answer. In particular, $b = \Delta f/\epsilon$, in which Δf is the *global sensitivity* measuring the impact of changing at most one occurrence for one record in the original dataset. In this paper, since we focus on counting queries, the global sensitivity is 1.

4. Attribute linkage attack on DP-processed datasets

In this section, we first give a formal definition of the APL, which adversaries use to initiate the attribute linkage attack. Then we give an analysis to show that the attack can be initiated with a high probability when the traditional DP algorithm is applied. Finally, we use two popular DP algorithms with real-world datasets to give a case study, which confirms the feasibility of the attack in the real world.

4.1. Attribute privacy leakage

We here formally define the *attribute privacy leakage* (APL), which is used by adversaries to initiate the attribute linkage attack. Roughly speaking, an APL Q is an itemset that the adversary can use to uniquely identify a specific itemset S in the dataset.

Definition 4 (APL)**.** Given a dataset D, we say that $Q \subset \mathcal{I}$ is an APL in D if

$$|\{S \mid S \in D^I \text{ and } Q \subset S\}| = 1,$$

where $\mathcal{I}$ is the item universe and D^I is the set of itemsets in D (see Table 2).

By using the APL Q, an adversary can initiate the attribute linkage attack and uniquely identify an itemset S in D^I. Specifically, $S - Q$ is the leaked information that the adversary could obtain. In our previous example, {*toothpaste, scissor, hanger*} is an APL which helps the adversary to uniquely identify the itemset {*beer, toothpaste, scissor, hanger*}, and *beer* is the leaked information. Note that for a real-world dataset, the size n of the item universe (See Table 2) is usually large. As a consequence, there are $2^n - 1$ itemsets which can be APLs, and so it is computationally expensive to locate all APLs exhaustively.

However, the topology-theoretic approach in Section 2.3 shows that one only needs to check whether boundary itemsets (See Definition 1) for all maximal itemsets (See Definition 2) in a dataset are APLs or not so to defend against the attribute linkage attack. Therefore, in the rest of the paper, unless we state otherwise, *all itemsets considered are maximal itemsets, and all APLs considered are boundary itemsets which can be used to uniquely identify those maximal itemsets.* Such an assertion also shows that we consider a highly damaging threat model: The adversary can have the most prior knowledge, which is the boundary itemset of the target maximal itemset. The following section will formally analyze why the attribute linkage attack is highly probable in DP settings when there are correlations among items.

4.2. Attack analysis

We first state the attack methodology, i.e., how the adversary leverages the APL to initiate the attribute linkage attack. Then, a discussion about the insufficient protection of standard DP is given, when there are correlations among items. Finally, we will analyze the probability of the attribute linkage attack in DP settings. In this paper, we assume the adversary has the following prior information in advance.

1. The victim's itemset S is in D^I.
2. The adversary knows an APL $Q \subset S$. Specifically, Q is a boundary itemset in B_S, and the adversary can use it to uniquely identify S in D^I.

However, the adversary cannot access the original dataset D and cannot identify the victim's itemset S in D^I. Instead, she can only access the DP-processed dataset D_p and wants to check whether Q can be used to uniquely identify the victim's itemset S in D_p^I, which is the set of itemsets in D_p. If that is the case, it means that Q is an APL in D_p, and the adversary can successfully deduce the victim's private information. Specifically, the attack methodology is described as follows.

Attack methodology: After accessing the DP-processed dataset D_p, the adversary first links her prior knowledge Q with records in D_p. As as result, she locates a *candidate set* $G = \{S' | S' \in D_p^I$ and $Q \subseteq S'\}$. Then the attribute linkage attack is successfully initiated if and only if $G = \{S\}$, which means that the victim's itemset S is uniquely identified. Going back to our example, if Eve knows in advance that Q = {*toothpaste, scissor, hanger*} and accesses D_p shown in Table 1b, Eve can locate the candidate set G = {{*beer, toothpaste, scissor, hanger*}}, which only contains Alice's itemset. In this case, the attribute linkage attack is successfully initiated. On contrary, if Eve accesses the dataset D'_p shown in Table 5, she will locate the candidate set G = {{*beer, toothpaste, scissor, hanger*}, {*toothpaste, scissor, hanger*}}. In this case, Eve cannot determine Alice's itemset, and the attribute linkage attack cannot be successfully initiated.

Note that the attribute linkage attack is also feasible in the context of the interactive DP setting. Specifically, the interactive setting and the non-interactive setting only differ in formats of representing data. At the core, they apply the same noise additive mechanism, e.g., the Laplace mechanism, to achieve the same level of privacy guarantees. Here we also state the attack methodology in the interactive setting. Specifically, by constructing specific combinations of queries, the adversary can leverage her prior knowledge to deduce private information. Using our previous example, because the number of itemsets in the itemset universe which contain {*toothpaste, scissor, hanger*} is two, i.e., {*toothpaste, scissor, hanger*} and {*beer, toothpaste, scissor, hanger*}, the adversary can propose two queries:

- Q_1: How many people purchased the itemset {*toothpaste, scissor, hanger*} exactly?
- Q_2: How many people purchased the itemset {*beer, toothpaste, scissor, hanger*}?

Let the true answer of Q_1 be x and Q_2 be y in D. Given a randomized DP algorithm $\mathcal{A}$ which outputs the perturbed query result, one can derive that

$$\begin{aligned}\Pr(\text{Eve deduces beer}) &= \Pr(\mathcal{A}(Q_1) = 0 \text{ and } \mathcal{A}(Q_2) = r) \\ &= \exp(-\epsilon|0 - x|) \cdot exp(-\epsilon|r - y|) \\ &= \exp(-\epsilon(x + y - r)),\end{aligned}$$

where $\mathcal{A}(Q_1)$ and $\mathcal{A}(Q_2)$ are two noisy answers which provide ϵ-differential privacy for Q_1 and Q_2. $\mathcal{A}(Q_1) = 0$ combined with

$\mathcal{A}(Q_2) = r > 0$ denotes that all customers who bought the itemset {*toothpaste, scissor, hanger*} also bought *beer*, which implies that Eve can successfully launch the attribute linkage attack.

Insufficient protections under correlations: Continuing the discussion in the interactive setting, we want to emphasize that correlations among items can sharply decrease the privacy guarantee of DP, and it is not feasible to rely on DP to defend against the attribute linkage attack. Given two neighboring datasets D_1 and D_2, which only differ in one occurrence of an itemset, according to Definition 3, one can derive that

$$\max_{D_1,D_2}\frac{\Pr(\text{Eve deduces beer in } D_1)}{\Pr(\text{Eve deduces beer in } D_2)} \le \frac{\exp(-\epsilon(x+y-r))}{\exp(-\epsilon(x+y-1-r)))} = \exp(\epsilon),$$

However, suppose there are correlations among the items, for example,

$$\Pr(\text{beer}|\text{toothpaste, scissor, hanger}) = 0.9,$$

In this case, one can simply derive that $y = 9x$, and

$$\max_{D_1,D_2}\frac{\Pr(\text{Eve deduces beer in } D_1)}{\Pr(\text{Eve deduces beer in } D_2)} \le \frac{\exp(\epsilon(10x-r))}{\exp(\epsilon(10(x-1)-r))} = \exp(10\epsilon).$$

Such an amplification shows the sharp decrease of DP's privacy guarantee, and adding or deleting one record can have significantly impacts on the probability of the attribute linkage attack. The result implies insufficient protection against the attribute linkage attack using standard DP. Similar dependency issues for standard DP have also been proposed (Liu et al., 2016). However, the attribute linkage attack proposed in our paper is more severe, and the reason is the following.

- It is not difficult to find out that the larger the correlation is, the lower the privacy guarantee DP will provide. Specifically, when the conditional probability equals to $\gamma \in (0, 1)$, the privacy guarantee reduces to $\exp(\frac{\epsilon}{1-\gamma})$, which is unacceptable.
- Such item correlations are common in the real world, e.g., shopping preferences and medical information. Moreover, the number of possible correlations increases exponentially as the number of items in the dataset increases, and it is difficult to use some DP variants, e.g., dependent differential privacy (Liu et al., 2016), to eliminate the attack.

Deriving the probability of attribute linkage attack: Remember that the first step for the adversary is to use the prior knowledge Q to locate a candidate set G. We say Q is an APL, and the attribute linkage attack is successfully initiated if and only if the candidate set $G = \{S\}$, where S is the victim's itemset. Therefore, the probability of the successful attribute linkage attack equals the probability that the condition "$G = \{S\}$" holds in the DP-processed dataset D_p. To derive the probability that the condition "$G = \{S\}$" holds, we find that the condition can be further dissect into two components, i.e., $\mathcal{C}_1 : S \in D_p^I$ and $\mathcal{C}_2 : G' \cap D_p^I = \emptyset$ where $G' = \{S' | Q \subseteq S' \subseteq \mathcal{I}$ and $S' \neq S\}$. Specifically, $\mathcal{C}_1$ requires that the victim's itemset S should exist in the DP-processed dataset D_p, such that the adversary can identify it using the prior knowledge Q and make sure $S \in G$. $\mathcal{C}_2$ requires that for other itemsets which can also be identified by the prior knowledge Q, they should not exist in D_p to make sure that the victim's itemset S is uniquely identified. As a result, we have

$$\Pr(\text{Successful Attribute Linkage Attack}) = \Pr(G = \{S\}) = \Pr(\mathcal{C}_1\mathcal{C}_2).$$

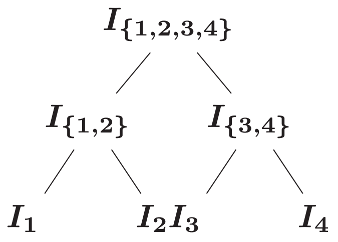


Fig. 1. Taxonomy tree T with $f = 2$.

Computing the above probability is non-trivial for different DP algorithms. However, because DP algorithms need to maintain a high data utility, the probability of a specific itemset being added to D_p is positively related to its frequency in D. For example, the basic idea of the partitioning-based DP algorithms (Chen et al., 2011) is first to partition the whole itemset space into many subregions, then keep those regions with large numbers of occurrences (Leoni, 2012). Sampling algorithms in DP (Chen et al., 2015; Zhang et al., 2017) learn the item correlations from the original data distribution first, then generate D_p from noisy joint distribution. Based on this observation, one can assert that

$$\Pr(\text{Successful Attribute Linkage Attack}) = \Pr(\mathcal{C}_1\mathcal{C}_2) \propto f_D(S) \cdot \prod_{S' \in G'} (1 - f_D(S')),$$

where S is the victim's itemset, $f_D(S)$ is the frequency of S in the dataset D, and $G' = \{S' | Q \subseteq S' \subseteq \mathcal{I}$ and $S' \neq S\}$.

In our attack, since Q is an APL for S in the original dataset D, no other itemsets contain Q, and a strong correlation exists between Q and $S - Q$. Using our previous example, most customers choose to buy *beer* after they purchase the itemset {*toothpaste, scissor, hanger*}. Such a correlation increases $f_D(S)$ where S is the itemset {*beer, toothpaste, scissor, hanger*}, while at the same time keeps $f_D(S')$ as low as possible. The consequence is that in D_p, there is a high probability for the attacker to uniquely identify S. Another way to understand the influence of correlations on the probability of attribute linkage attack is that, since the item correlation is an essential statistical property, which DP algorithms try to preserve, the item correlation is likely to be kept in D_p. However, the truth is that the existence of the APL depends on the correlation of underlying items. As long as the item correlation is well preserved, the existence of APLs will also be well preserved. The consequence is that when the adversary observes D_p, there is a high chance that the attack can be successfully initiated. Next, we will use experiments to demonstrate the high probability of the attack in the DP setting.

4.3. Case study

In this section, we use real-world datasets and two DP algorithms, *DiffPart* (Chen et al., 2011) and *PrivBayes* (Zhang et al., 2017), to demonstrate the attribute linkage attack on DP-processed datasets. The reason to choose these two algorithms is that they are popular and representative: one is partitioning-based, and another is sampling-based. We first give an overview of these two algorithms, then we show the probability of the attack using our designed experiments.

DiffPart: On startup, DiffPart requires a context-free taxonomy tree T to instruct the partitioning procedure, and Fig. 1 is an example. Specifically, the tree is constructed with four leaf nodes, each denoting a specific item $I_k, k \ge 1$. We map each item in the item universe {*beer, toothpaste, scissor, hanger*} to I_1, I_2, I_3 and I_4 respectively. Besides those leaf nodes, internal nodes of the tree are a set of their leaves. For example, the internal node $I_{\{1,2,3,4\}} =$

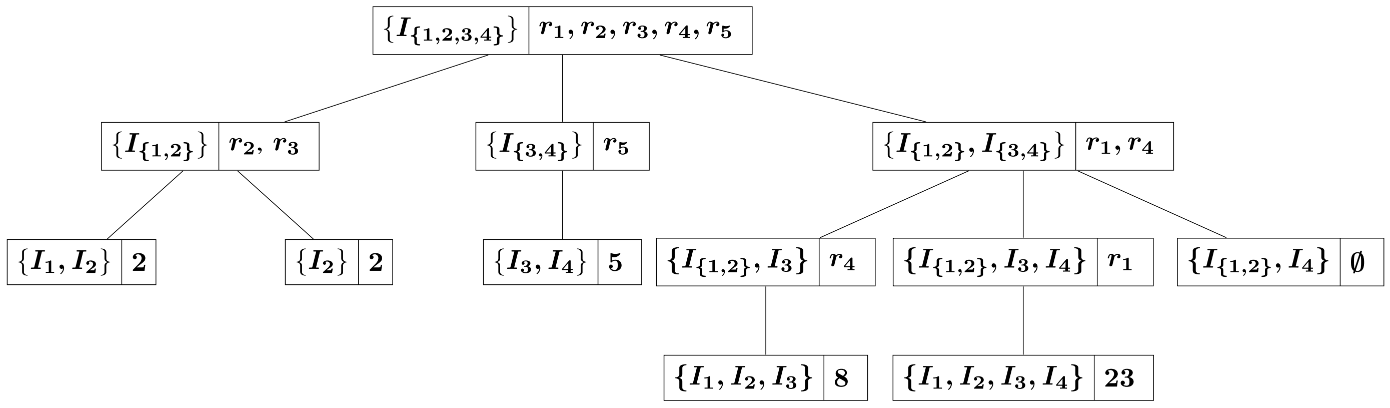

$\{I_{\{1,2,3,4\}}\}$ r_1, r_2, r_3, r_4, r_5
$\{I_{\{1,2\}}\}$ r_2, r_3
$\{I_{\{3,4\}}\}$ r_5
$\{I_{\{1,2\}}, I_{\{3,4\}}\}$ r_1, r_4
$\{I_1, I_2\}$ 2
$\{I_2\}$ 2
$\{I_3, I_4\}$ 5
$\{I_{\{1,2\}}, I_3\}$ r_4
$\{I_{\{1,2\}}, I_3, I_4\}$ r_1
$\{I_{\{1,2\}}, I_4\}$ $\emptyset$
$\{I_1, I_2, I_3\}$ 8
$\{I_1, I_2, I_3, I_4\}$ 23

Fig. 2. A possible partitioning process using *DiffPart*.

$\{I_1, I_2, I_3, I_4\}$. The parameter f controls the maximum degree of nodes. In Fig. 1, we let $f = 2$.

Note that a set of taxonomy tree nodes can generalize records in a dataset. Specifically, for a record $r : (S, |S|)$ in a dataset D (See Table 2) and a set of tree nodes $\mathcal{N}$, $\mathcal{N}$ generalizes the record r if (1) $\forall I_k \in S, \exists N \in \mathcal{N}, I_k \in N$, and (2) $\forall N \in \mathcal{N}, N \cap S \neq \emptyset$. For example, the set of tree nodes $\{I_{\{1,2\}}, I_{\{3,4\}}\}$ can generalize a record which contains the itemset $\{I_1, I_2, I_3\}$. Given the idea of generalization, DiffPart uses sets of tree nodes to create disjoint partitions of the dataset D. Fig. 2 is an example which processes the dataset shown in Table 1a. In particular, each partition p is a rectangle area, which contains a set of tree nodes *p.cut* (the left part of the rectangle) and a set of records *p.records* which are generalized by *p.cut* (the right part of the rectangle).

Specifically, DiffPart initiates a top-down partitioning procedure. It starts from creating an initial partition p in which the set of nodes *p.cut* contains the single root node of the taxonomy tree, i.e., $I_{\{1,2,3,4\}}$. Because the root node is the set of all items in the item universe, it can generalize all five records from r_1 to r_5, and *p.records* stores all these records. After that, DiffPart creates sub-partitions by (1) expanding the root node ($I_{\{1,2,3,4\}}$) with its child nodes ($\{I_{\{1,2\}}\}$ and $\{I_{\{3,4\}}\}$) in the tree, and (2) further generalizing records in the initial partition. Such sub-partition generation procedures will not stop until the set of tree nodes in newly generated sub-partitions cannot be expanded. That is, in the end, each generated sub-partition p will contain a set of nodes *p.cut*, in which each node is a leaf node. In this case, the set of nodes *p.cut* represents a specific itemset. One can check in Fig. 2 that each leaf partition contains a specific itemset, e.g., $\{I_1, I_2, I_3\}$.

Every time a partition p is generated, DiffPart first computes the sum of occurrences for all records in *p.records*. After that, a Laplace noise is added to the sum, and if the noisy sum is larger than a threshold, the partition p will be kept for further processing. Otherwise, the partition p will not be processed anymore. Specifically, the threshold is controlled by a user-specified parameter c_1: the larger c_1 is, the larger the threshold will be. In other words, the parameter c_1 controls the degree to which partitions with small numbers of occurrences of records will be filtered out. Note that in Fig. 2, we only draw the sub-partition which are kept during the procedure. We did not show those sub-partitions which are filtered out because of small numbers of occurrences. Finally, when a leaf partition p is generated, the set of nodes *p.cut* represents a specific itemset. DiffPart further computes its noisy number of occurrences, and adds the itemset *p.cut* with its noisy number of occurrences to the output dataset D_p.

PrivBayes: The idea of PrivBayes is different from DiffPart. It aims to first derive an approximate distribution of items in the dataset, then apply sampling methods to generate a synthetic dataset D_p. The algorithm runs in three phases:

1. Construct a k-degree Bayesian network $\mathcal{B}$ over the items in the dataset using ϵ_1-DP methods.
2. Generate a set of conditional distributions of the original dataset D using ϵ_2-DP methods.
3. Compute the approximate joint distribution over the original dataset D. Combined with the network $\mathcal{B}$ and conditional distributions derived from the second step, the algorithm samples itemsets from the derived distribution to generate a synthetic dataset D_p.

Specifically, the choice of the parameter k is affected by another parameter θ, which measures the usefulness of the noisy distribution. One can check that both DiffPart and PrivBayes are consistent with our analysis in Section 4.2 that the frequency of itemsets will influence the probability of the attack. Specifically, in DiffPart, the number of occurrences of a specific itemset S will influence the noisy sum in partitions where S locates, and further influence the probability of these partitions generating sub-partitions. For PrivBayes, the number of occurrences will directly influence the conditional distribution derived in the second step and further influence the joint distribution. When a sampling method is applied, the itemset with a more significant number of occurrences has a larger probability of being sampled. Further, we will use experiments to demonstrate how significant the probability of the attack can be.

Experiment results: We perform experiments to show the attribute linkage attack on DP-processed datasets. The real-world datasets are *MSNBC*[1] and *NLTCS*[2]. MSNBC is a public dataset on the UCI machine learning repository, which contains 989,818 records and 17 items, while NLTCS contains 17,721 records and 16 items. In NLTCS, there is only one maximal itemset S_N, which contains all 16 items. The same property applies for MSNBC, and there is only one maximal itemset S_M which contains all 17 items. Moreover, the only maximal itemsets in both datasets have APLs.

We set S_N and S_M as the target, and in each dataset, we choose an arbitrary APL for the target to investigate whether the APL still exists in D_p. If that is the case, the attack succeeds. Specifically, we are interested in the probability that the DP-processed dataset D_p has the APL for the target. For each DP algorithm and each parameter setting, we generate 1000 DP-processed datasets to compute the average probability of having APLs for the target. Note that it

1 https://archive.ics.uci.edu/ml/datasets/msnbc.com+anonymous+web+data.
2 https://www.icpsr.umich.edu/web/NACDA/studies/9681/publications.

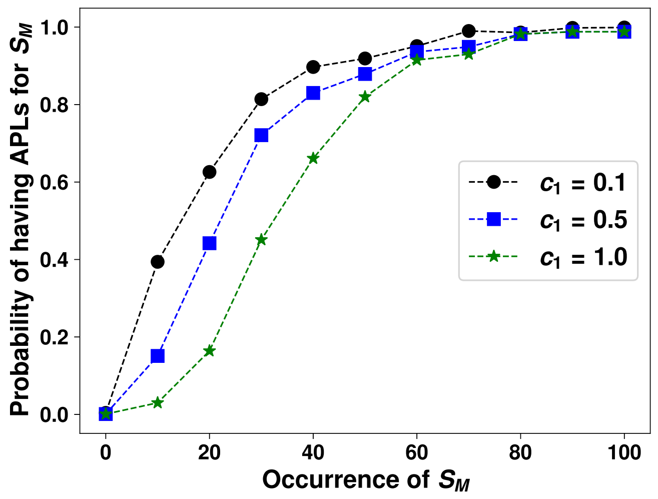


Fig. 3. Probability of having APLs for S_M with different c_1 and the number of occurrences.

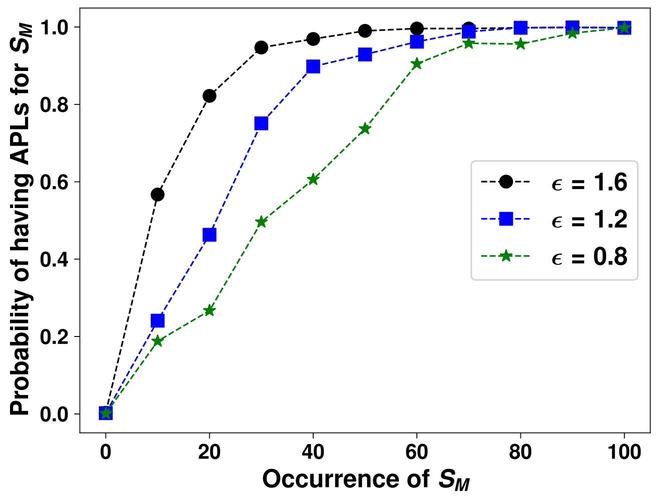


Fig. 4. Probability of having APLs for S_M with different ϵ and the number of occurrences.

is time-consuming for PrivBayes to process large datasets (over 24 hours). Instead, we use DiffPart to process the MSNBC dataset and PrivBayes to process the NLTCS dataset separately.

Figs. 3 and 4 show that the probability of having the APL for S_M increases quickly, as we increase the number of occurrences. The occurrence of 80 is sufficient for the attacker to launch the attribute linkage attack on S_M with a probability of 0.92. Note that in the original dataset, the number of occurrences of S is 13, which means that in the real world, Eve can successfully launch the attribute linkage attack on S_M with a probability of at least 0.4. In Fig. 3, one can observe that a larger c_1, which is a user-defined parameter in DiffPart, results in a more significant probability of being attacked. We vary c_1 from 0.1 to 1 in this experiment.

Fig. 4 shows the influence of privacy budget ϵ given $c_1 = 1$. As the privacy budget increases, the variance of added noise becomes smaller, such that the utility improves at the cost of weakening the privacy guarantee. One can check that the probability of having the APL for S_M increases as a larger value of privacy budget is assigned. For example, if one publishes the dataset with $\epsilon = 1.6$, the target's information can be leaked with probability of at least 0.6 in the real world, which is an unacceptable privacy leakage risk.

We further use PrivBayes and NLTCS to show the probability of having APLs for S_N. For the PrivBayes algorithm, the parameter θ

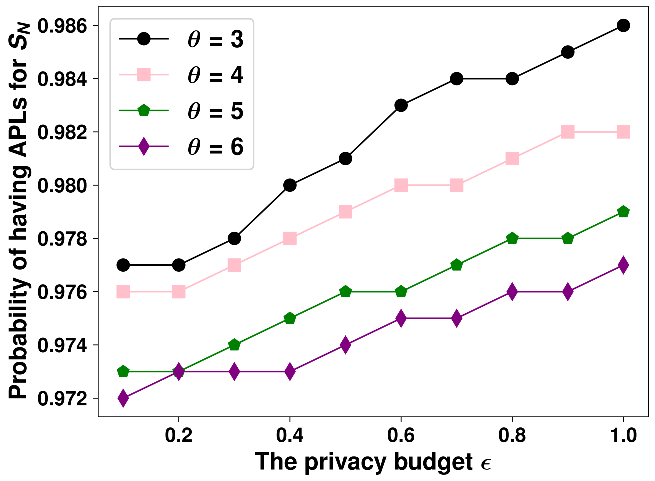


Fig. 5. Probability of having APLs for S_M with different θ and ϵ.

controls the degree in the constructed Bayesian network $\mathcal{N}$, and ϵ is the privacy budget. Fig. 5 shows the experimental results as we vary ϵ from 0.1 to 1. One can check that as ϵ increases, the generated noisy distribution is closer to the original distribution in D, which means that the existence of the APL is more likely to be preserved. Moreover, a lower θ will also incur a higher probability of the attack. In summary, there are two important conclusions for the attribute linkage attack in DP settings.

1. For traditional DP algorithms, users need to carefully select the parameter to reduce the probability of being attacked.
2. There is a *privacy-utility dilemma*: A larger ϵ means a lower scale of noise, which can bring a better data utility. However, in this section, one observes that a larger ϵ also brings a higher probability of being attacked.

In the following section, we will propose our defense methodology and solution.

5. Defense methodology

In this section, we propose a variant of DP to address the attribute linkage attack, which is called APL-Free ϵ-DP. Further, we design an algorithm, APLKiller, which satisfies the APL-Free ϵ-DP and is used to publish datasets securely. Experiment results show that our algorithm provides a stronger privacy guarantee and better data utility than traditional DP algorithms.

5.1. APL-Free ϵ-DP

First, let us provide the definition of APL-Free ϵ-DP.

Definition 5. A randomized algorithm $\mathcal{A}$ satisfies APL-Free ϵ-DP if $\mathcal{A}$ satisfies the following requirements:

1. For any $D_p \in Range(\mathcal{A})$, there is no APL in D_p.
2. For any two neighboring datasets D_1 and D_2 which only differ in one occurrence of an itemset, and for any possible output $D_p \subseteq Range(\mathcal{A})$,

$$\frac{Pr(\mathcal{A}(D_1) = D_p)}{Pr(\mathcal{A}(D_2) = D_p)} \leq exp(\epsilon).$$

Compared with Definition 3, APL-Free ϵ-DP adds an additional constraint: there should be no APLs in the output dataset. Besides inheriting the privacy guarantee of standard DP, the additional constraint also guarantees that the attacker cannot launch the attribute linkage attack in the output dataset D_p. Note that any sequence of computations that each provides APL-Free ϵ-DP in isolation also provides APL-Free ϵ-DP. Specifically, the following sequential composition theorem holds.

Theorem 1 (Sequential Composition Theorem)**.** *Let $\mathcal{A}_i$ each be a randomized algorithm that satisfies APL-Free ϵ_i-DP. A sequence of $\mathcal{A}_i(D)$ over the dataset D provides APL-Free $\sum_i(\epsilon_i)$-DP.*

In some cases where a sequence of computations are conducted on disjoint datasets, the privacy budget ϵ only depends on the worst guarantee of all computations. Specifically, the following parallel composition theorem holds.

Theorem 2 (Parallel Composition Theorem)**.** *Let $\mathcal{A}_i$ each be a randomized algorithm that satisfies APL-Free ϵ_i-DP. A sequence of $\mathcal{A}_i(D_i)$ over a set of disjoint datasets D_i provides APL-Free $(max(\epsilon_i))$-DP.*

We defer the detailed proof in Appendix A.1 and Appendix A.2 for further reference.

5.2. APLKiller

In this section, we propose the algorithm APLKiller, which is used for publishing datasets efficiently and accurately. Most importantly, we prove that our algorithm satisfies the APL-Free ϵ-DP, which provides a higher privacy guarantee.

Framework: One of the advantages of our algorithm APLKiller over traditional DP algorithms is the elimination of APLs. To achieve this goal, we use the topology-theoretic approach (See Section 2.3) to instruct the design of our algorithm. Specifically, topology theory shows that for each maximal itemset in a dataset, all boundary itemsets should also exist in the dataset to defend against the attribute linkage attack. The above result implies that *whenever a maximal itemset S is added to D_p, it is critical to make sure the existence of its boundary set B_S in D_p*. Based on the above understanding, we designed the APLKiller, which contains two procedures.

- *Level Partitioning*: The goal of this partitioning procedure is to generate records that contain itemsets of given sizes.
- *Boundary Adding*: The goal of this adding procedure is to add artificial records. These records contain boundary itemsets for itemsets added by the partitioning procedure.

APLKiller generates the output dataset D_p in several rounds. Specifically, a size $i \in \{1, \dots, |\mathcal{I}|\}$ is given in each round, and APLKiller first initiates the level partitioning procedure. The partitioning procedure will generate records that contain itemsets of the given size i (the number of items in the itemset). After that, for each generated itemset S, APLKiller initiates the boundary adding procedure to ensure the existence of the boundary set B_S in D_p. Such iteration ends when the size i reaches 0. Since the boundary set for each itemset generated by the partitioning procedure is well checked, according to topology theory, there will be no APLs in D_p, and the attribute linkage attack cannot be launched. Also, all of these operations should be done in a differentially private manner to satisfy APL-Free ϵ-DP.

Let *i-itemset* be an itemset which contains i items, and D_i be the part of a dataset D containing all i-itemset records (See Table 2). Algorithm 1 shows the pseudocode of the overall framework of APLKiller. The core components are *LevelPart* (Line) and the *Boundary Adding* (Line to Line): They correspond to the level partitioning procedure and the boundary adding procedure, respectively. In each round, LevelPart first initiates the level partitioning procedure and takes the size i as well as D_i as the input. As a result, it returns D'_i, which is the perturbed dataset containing i-itemset records. After that, APLKiller adds records in D'_i to the output dataset D_p, and initiates the boundary adding procedure. Specifically, APLKiller first

Algorithm 1 APLKiller.

```
Require: $D$, parameter vectors $F$, $C_1$, privacy budget $\epsilon$
Ensure: perturbed dataset $D_p$
1: $i \leftarrow |\mathcal{I}|$
2: Initialize an empty set $D_p$ and a vector of empty sets $Q$
3: Partition $D$ into $\bigcup_{i=1}^{n} D_i$ {$D_i$ contains all $i$-itemsets}
4: while $i \geq 1$ do
5:     $D'_i \leftarrow LevelPart(i, D_i, F_i, C_1^i, \epsilon)$ {Generate $D'_i$}
6:     $D_p = D_p \cup D_i^{l'}$
7:     for $S_j^i \in D_i^{l'}$ do
8:         $Q_i = Q_i \cup B_{S_j^i}$ {Aggregate boundary itemsets for $D'_i$}
9:     end for
10:    for $S_k^i \in Q_i$ do
11:        $N_k^i = 0$
12:        {Determine the noisy number of occurrences of boundary itemsets}
13:        while $N_k^i \leq 0$ do
14:            $N_k^i = NoisyCount(|S_k^i|, \epsilon)$
15:        end while
16:        Add record $(S_k^i, N_k^i)$ to $D_p$.
17:    end for
18:    {Remove influences of $Q_i$ on generating $D'_{i-1}$}
19:    Remove records which contains itemset $S_k^i \in Q_i$ from $D_{i-1}$
20:    $i \leftarrow i - 1$
21: end while
22: return $D_p$
```

aggregates boundary itemsets for each i-itemset in D'_i. Then it determines the noisy number of occurrences of each boundary itemset to generate artificial records. Finally, these artificial records are also added to the output dataset D_p.

Note that the iteration follows the decreasing order of the size i (Line), and the iteration ends when the size $i = 0$. The output dataset D_p is the union of D'_i for $i \in \{1, \dots, |\mathcal{I}|\}$, and the set of records which contain those boundary itemsets. For example, if the size of the item universe $|\mathcal{I}| = 10$, APLKiller will first call LevelPart to generate D'_{10}, which contains 10-itemset records. Then APLKiller aggregates boundary itemsets for those 10-itemsets in D'_{10}, determines their noisy number of occurrences, and adds them to D_p with those records in D'_{10}. The iteration stops when $i = 0$, and APLKiller returns D_p to users.

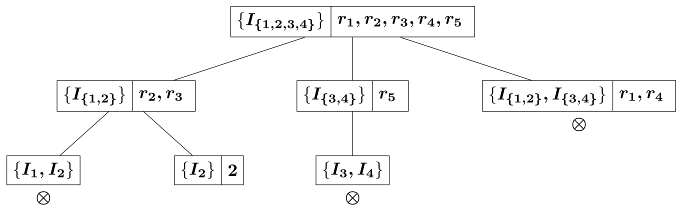
In the remaining part of this section, we first introduce the framework of LevelPart, which is responsible for the partitioning procedure. Then we introduce our boundary adding procedure. Finally, we introduce the privacy budget allocation scheme and give an analysis of APLKiller.

Level partitioning procedure. The algorithm LevelPart is responsible for the level partitioning procedure, and its pseudocode is shown in Algorithm 2. Specifically, it takes the size l, the dataset D_l, the privacy budget ϵ and several algorithm-specific parameters as inputs. As a result, it returns a perturbed dataset D'_l, which contains l-itemset records with noisy numbers of occurrences. LevelPart shares a similar top-down partitioning procedure with that of DiffPart (See Section 4.3). However, LevelPart applies additional size constraints to prune “illegal” partitions, which we will introduce later. The advantage of the size-restricted partitioning procedure is to generate records that contain itemsets of the given size.

LevelPart starts the partitioning procedure by initializing a taxonomy tree and creating an initial partition p, in which the set of nodes *p.cut* contains the single root node of the taxonomy tree

Algorithm 2 LevelPart.

```
Require: size $l$, dataset $D_l$, fan-out $F_l$, constant $C_1^l$, and privacy budget $\epsilon$
Ensure: Perturbed dataset $D_l'$
1: Initialize $D_l'$ and construct taxonomy tree $T$ with $F_l$
2: Create Partition $p$ which includes all records and store root of $T$ into $p.cut$. Let $p.B = \frac{\epsilon}{2}$ and $p.\alpha = \frac{p.B}{Par(p,l)}$
3: Add $p$ to an empty queue $Q$
4: while $Q \neq \emptyset$ do
5:     Dequeue $p'$ from $Q$
6:     $P \leftarrow$ LevelSGP($p', T, l, C_1^l$) {Generate subpartitions of $p'$}
7:     for each $p_i \in P$ do
8:         if $p_i$ is leaf partition then
9:             {Determine the noisy occurrence of the itemset}
10:            $N_{p_i} = NoisyCount(|p_i|, \frac{\epsilon}{2} + p_i.B)$
11:            if $N_{p_i} \geq \sqrt{2}\frac{C_1^l}{\epsilon/2+p_i.B}$ then
12:                Add $N_{p_i}$ copies of $p_i.cut$ to $D_l'$
13:            end if
14:        else
15:            Add $p_i$ to $Q$ {Continue to generate subpations of $p_i$}
16:        end if
17:    end for
18: end while
19: return $D_l'$
```



$\{I_{\{1,2,3,4\}}\}$ r_1, r_2, r_3, r_4, r_5
$\{I_{\{1,2\}}\}$ r_2, r_3
$\{I_{\{3,4\}}\}$ r_5
$\{I_{\{1,2\}}, I_{\{3,4\}}\}$ r_1, r_4
$\otimes$
$\{I_1, I_2\}$
$\otimes$
$\{I_2\}$ 2
$\{I_3, I_4\}$
$\otimes$

Fig. 6. The generation of 1-itemsets using LevelPart.

(Line 1 and Line 2). Note that the related definitions, e.g., the taxonomy tree, the partition and the generalization operation, are the same as those used in DiffPart. One can refer to Section 4.3 for further information. Fig. 6 is an example: The topmost partition contains a single root node $I_{\{1,2,3,4\}}$, which generalizes all records $r1$ - $r5$.

In each round, LevelPart picks a partition p' (Line), and calls LevelSGP to generate a set of sub-partitions P (Line). Specifically, if the generated sub-partition $p_i \in P$ is a leaf partition ($p_i.cut$ represents a specific itemset), LevelPart will first compute its noisy number of occurrences by applying the Laplace mechanism (Line to Line). After that, LevelPart uses a threshold to determine whether the noisy number of occurrences is large enough (Line). If that is the case, the itemset represented by $p_i.cut$ with the noisy number of occurrences will be added to D_l' (Line). Note that the threshold is controlled by a user-specified parameter C_1^l: The threshold equals the multiplication of the standard deviation of the Laplace noise and the parameter C_1^l. In order to compute the noisy number of occurrences and the standard deviation, LevelPart needs to determine the privacy budget. We will introduce the privacy budget allocation scheme later in this section. On the other hand, if the generated sub-partition p_i is not a leaf partition, it will be preserved for further partitioning (Line). An example is given in Fig. 6. For the initial partition p with $p.cut = \{I_{\{1,2,3,4\}}\}$, it generates 3 sub-partitions. One of its sub-partitions with cut $I_{\{1,2\}}$ further generates two leaf partitions, which present itemsets $\{I_1, I_2\}$ and $\{I_2\}$ respectively.

It is important to introduce LevelSGP, which is responsible for the sub-partition generation given the size constraint. The pseudocode is shown in Algorithm 3. Specifically, LevelSGP first creates

Algorithm 3 LevelSGP.

```
Require: Partition $p$, taxonomy tree $T$, length $l$, constant $C_1^l$
Ensure: Vector of sub-partitions $V$
1: Initialize empty vector $V$
2: if $|p.cut| > l$ or $|p.items| < l$ then
3:     return $V$
4: end if
5: Randomly select $u \in p.cut$, expand and generate the set of non-empty sub-partitions $S$ in which $|s_i.cut| \leq l$ and $|s_i.items| \geq l$ for $s_i \in S$.
6: Generalize records in $p$ to sub-partitions in $S$
7: for $s_i \in S$ do
8:     $N_{s_i} = NoisyCount(|s_i|, p.\alpha)$
9:     if $N_{s_i} \geq \sqrt{2}\frac{C_1^l}{p.\alpha}$ then
10:        $s_i.B = p.B - p.\alpha$, $s_i.\alpha = \frac{s_i.B}{Par(s_i,l)}$
11:        Add $s_i$ to $V$ {The subpartition can be further processed}
12:    end if
13: end for
14: $j = 1$
15: {Select some empty subpartitions by random sampling}
16: while $j \leq 2^{|p.items|}$ do
17:    if $NoisyCount(0, p.\alpha) \geq \sqrt{2}\frac{C_1^l}{p.\alpha}$ then
18:        Randomly generate an empty sub-partition $s_j'$
19:        if $|s_j'.cut| \leq l$ and $|s_j'.items| \geq l$ then
20:            $s_j'.B = p.B - p.\alpha$, $s_j'.\alpha = \frac{s_j'.B}{Par(s_i,l)}$
21:            Add $s_j'$ to $V$
22:        end if
23:    end if
24: end while
25: return $V$
```

sub-partitions for the current partition p by (1) randomly selecting one node from $p.cut$ and expanding it, and (2) generalizing records in $p.records$ (Line). After that, created sub-partitions with non-zero sums of occurrences and those with zero sums are processed differently. For sub-partitions with non-zero sums of occurrences, a Laplace noise is added to the sum and the noisy sum is compared with a user-defined threshold (Line to Line). If the noisy sum is larger than the threshold, the sub-partition will be kept for further processing. Otherwise, the sub-partition with records in it will be discarded. For sub-partitions with zero sums of occurrences, a sampling method is applied to guarantee the randomness of the algorithm. Specifically, Every time a Laplace noise is generated and is larger than the user-defined threshold, LevelSGP will randomly select an empty partition and keep it for further processing (Line to Line).

Note that the goal of LevelSGP is to generate sub-partitions given the size constraint, which is not supported in DiffPart. Consider Fig. 2 as an example. DiffPart expands the partition with cut $\{I_{3,4}\}$ to create sub-partitions with cuts $\{I_3\}$, $\{I_4\}$, and $\{I_3, I_4\}$. However, if 1-itemsets are required, one should remove the partition with cut $\{I_3, I_4\}$, because it only contains 2-itemsets. In order to make LevelSGP generate sub-partitions that meet the size constraint, in this paper, we propose a novel pruning technique.

Specifically, given the size l, we find two situations in which a partition should be pruned.

1. $|p.cut| > l$.
2. $|p.items| < l$.

In the first situation, the number of nodes in *p.cut* is computed and is used to compare with the size l. Since each node in *p.cut* contributes to at least one item, if the total number of nodes is more significant than l, the partition p will generate at least $(l+1)$-itemsets, and the partition should be filtered out. For example, in Fig. 6, the partition with the cut $\{I_{\{1,2\}}, I_{\{3,4\}}\}$ is pruned, because it will generate itemsets of length at least 2. For the second situation, $|p.items|$ is the sum of sizes for all nodes in *p.cut*. For example, for the partition p with the cut $\{I_{\{1,2\}}, I_{\{3,4\}}\}$, $|p.items| = 4$. If it is less than l, the partition p can generate at most $(l-1)$-itemsets, which should also be pruned. By enforcing the size checking (Line and Line), LevelSGP is guaranteed to only generate sub-partitions containing l-itemsets. Finally, when a leaf partition is generated, it will only contain one l-itemset.

Boundary adding procedure. The boundary adding procedure is initiated by the algorithm APLKiller. When LevelPart returns D_i' (Line), APLKiller will first add records in D_i' to the output dataset D_p (Line). Then the boundary itemset for each i-itemset in D_i' is derived deterministically, and APLKiller collects those boundary itemsets in Line, where B_{S_j} is the boundary set for the itemset S_j. For each collected boundary itemset, APLKiller repeatedly generates a number of occurrences with Laplace noise until it is positive (Line). After that, the boundary itemset with the noisy number of occurrences is added to D_p to eliminate possible APLs in D_p. Before APLKiller starts the next round to generate $(i-1)$-itemsets, it first removes the existence of each boundary itemset S_k^i from D_{i-1} (Line). This is because they have been processed, and they should not cause any influence in generating $(i-1)$-itemsets.

Privacy budget allocation and analysis. When initiating the level partitioning procedure and the boundary adding procedure, APLKiller allocates the privacy budget to calculate the noisy sum for a specific sub-partition or calculate the noisy number of occurrences of an itemset. Therefore, it is necessary to show how the privacy budget is allocated.

First, we show the budget allocation scheme in the level partitioning procedure. APLKiller calls LevelPart iteratively to do level partitioning using D_i as inputs. As every D_i is disjoint from each other, according to Theorem 2, one can safely allocate the whole privacy budget ϵ each time LevelPart is called. LevelPart first creates an initial partition p, and assigns the whole budget ϵ to it (Line). Specifically, *p.B* is the privacy budget for the sub-partition generation, which is initially $\frac{\epsilon}{2}$. The remaining $\frac{\epsilon}{2}$ budget will be finally used to determine the noisy number of occurrences of the generated itemset. Every time sub-partitions of a partition p are generated, *p.B* will be reduced, and the residue will be inherited by these generated sub-partitions. Note that $p.\alpha$ represents the computed budget cost for the coming sub-partition generation.

Specifically, every time LevelSGP is called to generate sub-partitions of a partition p, it will use up $p.\alpha$, and assign the remaining $(p.B - p.\alpha)$ budgets to all generated sub-partitions for further partitioning (Line in LevelSGP). Since these sub-partitions represent disjoint subsets of the original dataset D, such a privacy budget allocation scheme also follows the Theorem 2. Moreover, in Line of LevelSGP, for all generated sub-partitions s_i, $s_i.\alpha$ is computed which is the budget cost for partition s_i to launch the next sub-partition generation. Finally, in LevelPart, if LevelSGP returns a leaf partition p_i, the remaining budget for sub-partition generation $p_i.B$, plus the preserved half of the total privacy budget $\frac{\epsilon}{2}$, will be used for determining the noisy count of the itemset in D_p (Line in LevelPart).

In order to compute the privacy budget cost $p.\alpha$ for the sub-partition generation, the idea is to first compute the maximum number of partitioning operations for the partition p to generate the leaf partition. After that, the algorithm allocates a fraction of the unused partitioning budget according to the maximum number of partitioning operations. For example, if $p.B = \frac{\epsilon}{3}$ and the maximum number is 4, then the budget cost for the next sub-partition generation $p.\alpha$ is $\frac{\epsilon}{12}$. Note that DiffPart also applies similar allocation scheme, and it has been proved that the total budget cost for partitioning operations will not exceed $\frac{\epsilon}{2}$, which is the initially allocated budget for the sub-partition generation (Chen et al., 2011).

Therefore, calculating $Par(p,l)$, which represents the maximum number of partitioning operations for the partition p to generate leaf partitions and l-itemsets, is necessary for deriving the allocated budget $p.\alpha$. Once $Par(p,l)$ is computed, $p.\alpha$ can be simply computed as $\frac{p.B}{Par(p,l)}$. We first compute $Par(p,l)$ for partition p which contains single internal node in *p.cut*, e.g., $p.cut = \{I_{\{1,2,3,4\}}\}$. Based on the result, we provide a solution to deal with more general cases, where a partition may contain multiple leaf nodes and internal nodes, e.g., $p.cut = \{I_{\{1,2\}}, I_{\{3,4\}}\}$.

Theorem 3. *For a partition p, if p.cut contains single internal node u and* $|u.items| = n$, *then given the fan-out parameter f, for* $f^k \le \frac{n}{l} \le f^{k+1}$ *and* $l > 0$, $k \ge 0$,

$$Par(p,l) = Par(u,l) = \begin{cases} 0, & l = 0 \\ \frac{n-1}{f-1} + \sum_{i=1}^{k} (l - \lceil \frac{n}{f^i} \rceil), & otherwise \end{cases}$$

The detailed proof is deferred to AppendixA.4 for further reference. We extend the above result to more general cases: Compute $Par(s,l)$ for a partition s which contains multiple leaf nodes and internal nodes. Specifically, let p be its parent partition, the number of leaf nodes in s be n_s^l and the number of internal nodes in s be n_s^i. Here we give the computation of $Par(s,l)$.

$$Par(s,l) = \begin{cases} Par(p,l) - 1, & n_s^i > 1 \\ Par(u, l - n_s^l), & n_s^i = 1, \end{cases} \qquad (1)$$

where u is the only internal node in partition s. Since for the parent partition p, the generation of s takes one round, when there are multiple internal nodes in the partition s, we simply let $Par(s,l)$ equal to $Par(p,l)$ minus 1. However, when there is only one internal node in s, because leaf nodes will contribute to one item for constructing l itemsets, and it will not increase the number of rounds of partitions, we let $Par(s,l)$ be $Par(u, l-n_s^l)$, which means that the maximum number of partitioning operations only depends on how we select $(l - n_s^l)$ items from the internal node u. To better explain the mechanism, the following example is given. Suppose $p.cut = \{I_{\{1,2,3,4\}}\}$ and $s.cut = \{I_{\{1,2\}}, I_{\{3,4\}}\}$. In this case there are two internal nodes in s, and we let $Par(s,l) = Par(p,l) - 1$. Suppose $s'.cut = \{I_{\{1,2\}}, I_3\}$, in this case, there is one internal node and one leaf node in s', so we let $Par(s,l) = Par(I_{\{1,2\}}, l-1)$.

For any partition s, one can first leverage Theorem 3 and Eq. 1 to compute $Par(s,l)$. Then one can derive the budget cost for the next partitioning operation, i.e., $s.\alpha = \frac{s.B}{Par(s,l)}$. Also, it has been shown that the total budget cost for the level partitioning procedure will not exceed ϵ. We further introduce the budget allocation scheme for the boundary adding procedure. Specifically, for each boundary itemset, we use the total budget ϵ to generate its noisy number of occurrences (Line in APLKiller). Note that such an allocation scheme also follows the Theorem 2. The reason is that (1) each boundary itemset is disjoint from each other, and (2) because APLKiller removes each boundary itemset from the dataset (Line), the boundary itemset is also disjoint from those itemsets in D_i, which are processed by the level partitioning procedure. There-

Table 6
Description of experimental datasets.

Dataset	$\|D\|$	$\|\mathcal{I}.items\|$	$max(r)$
MSNBC	989,818	17	17
Checkin-Foursquare	266,909	77	31
NLTCS	17,721	16	16

fore, it is appropriate to allocate the total budget ϵ to each boundary itemset.

After introducing the algorithm design as well as the allocation scheme for the privacy budget used in partitioning operations, we state that Lemma 1 holds.

Lemma 1. *LevelPart satisfies ϵ-DP.*

Next, we will analyze APLKiller regarding its privacy guarantees, data utility and time complexity.

5.3. Algorithm analysis

In this section, we give the analysis of APLKiller. We first prove that APLKiller satisfies APL-Free ϵ-DP, then we give an analysis of the time complexity of our algorithm.

Theorem 4. *APLKiller satisfies APL-Free ϵ-DP.*

The detailed proof is in the Appendix A.3. Moreover, for the time complexity analysis, we have the following result, and the proof is shown in Appendix A.5.

Lemma 2. *The time complexity of DiffPart is $O(mn)$, where m is the number of records in the dataset D, and n is the number of items in the item universe.*

The time complexity of APLKiller is attractive, compared with other algorithms which have exponential time complexity, e.g., PrivBayes. In the next section, we will use the experiment to show that our algorithm also enjoys better data utility than DiffPart and PrivBayes.

6. Experimental evaluation

In this section, we evaluate our algorithm by comparing with traditional DP algorithms in terms of the execution time, privacy level and data utility.

Three datasets are used, and detailed information about these datasets is shown in Table 6, where $|D|$ is the number of records in the dataset, $|\mathcal{I}.items|$ is the size of item universe, and $max(r)$ is the maximum number of items in one record. In the rest of this section, we (1) compare the execution time of our algorithm with that of DiffPart and PrivBayes algorithms, (2) perform the privacy analysis to show the privacy guarantee which our algorithm can provide, and (3) show that our algorithm provides better data utility. All experiments were conducted on an Intel Core i5 2.4 GHz PC with 8GM RAM.

6.1. Efficiency analysis

In this subsection, we report the time cost of publishing datasets using APLKiller, DiffPart, and PrivBayes. The result is shown in Fig. 7. For each dataset and each algorithm, we run the algorithm 1000 times to publish the corresponding dataset and compute the average of the execution time. Note that the algorithm PrivBayes cannot handle large datasets efficiently. Specifically, for Checkin and MSNBC datasets, the execution time for the PrivBayes algorithm is over 24 hours. Therefore, the efficiency

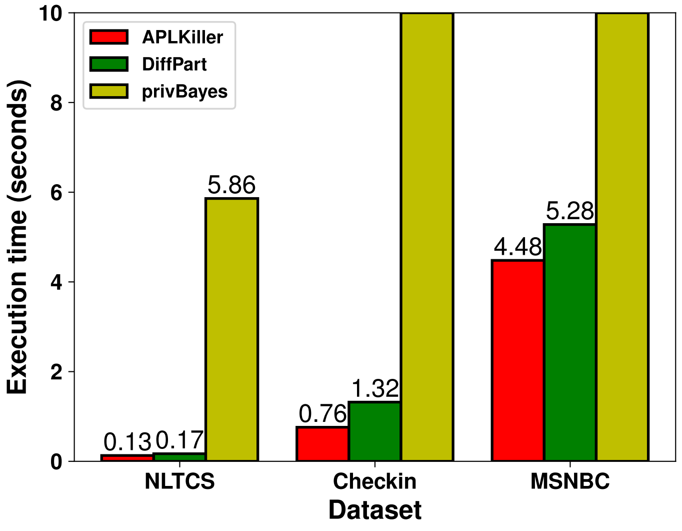


Fig. 7. Execution time of APLKiller, DiffPart and PrivBayes.

of the algorithm PrivBayes is the worst. For the privacy analysis (Section 6.2) and the utility analysis (Section 6.3), we will not apply PrivBayes on the Checkin dataset and the MSNBC dataset because of its low efficiency.

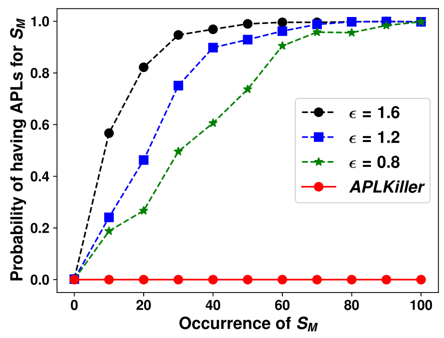
One can check that our algorithm APLKiller outperforms DiffPart in efficiency. Specifically, the execution time of APLKiller is reduced by 27.04% on average, compared with that of DiffPart. For some datasets, e.g., the Checkin dataset, the execution time of APLKiller can even be reduced by nearly 50%. The main reason for the speedup is that APLKiller prunes unnecessary partitions, which speeds up the generation of sub-partitions (See Section 5.2). As a result, users can use our algorithm to publish the dataset efficiently.

6.2. Privacy guarantee analysis

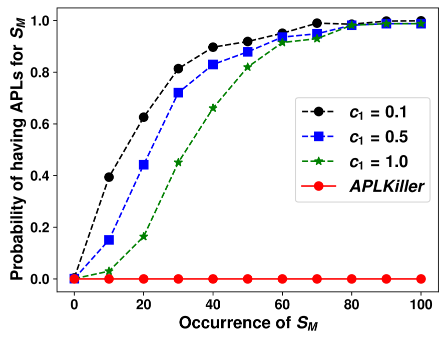
In Section 4, we have used DiffPart and PrivBayes to show the possible attribute linkage attack. In this section, we extend our experiments and add APLKiller to make the comparison. Specifically, the only maximal itemset in the NLTCS dataset is denoted as S_N, and the only maximal itemset in the MSNBC dataset is denoted as S_M. We set S_N and S_M as the target and check the probability of having APLs for these two targets in the generated dataset.

Fig. 8 shows the comparison result between DiffPart and APLKiller. We used the MSNBC dataset to do the experiment. The value of each point is derived from the average of the analysis result among 1000 generated datasets. In Fig. 8(a), we change the privacy budget ϵ, and we further change the parameter c_1 in Fig. 8(b. For the record, although APLKiller allows one to set different parameters at different levels, in the following analysis, unless we state otherwise, the same parameter for each level is used to compare with DiffPart. For example, $c_1 = 1$ means that we set $C_1^i = 1$ for each size i in APLKiller. By using our algorithm APLKiller, one can check that for each parameter setting, there is no single APL in the generated dataset, which means that the target itemset is guaranteed to be protected from the attribute linkage attack.

We also compare APLKiller with PrivBayes using the NLTCS dataset, and the result is shown in Fig. 9. Note that by using PrivBayes, the probability of having APLs for the target itemset varies little as one sets different ϵ (97.2% - 98.6%). Therefore, lines drawn from PrivBayes overlap heavily. However, by using APLKiller, no matter how the parameter is set, there is no APL in the generated dataset, and so the probability of having APLs is always 0.



(a) Probability of having APLs for S_M with different ϵ and the number of occurrences



(b) Probability of having APLs for S_M with different c_1 and the number of occurrences

Fig. 8. Privacy comparison using DiffPart and MSNBC.

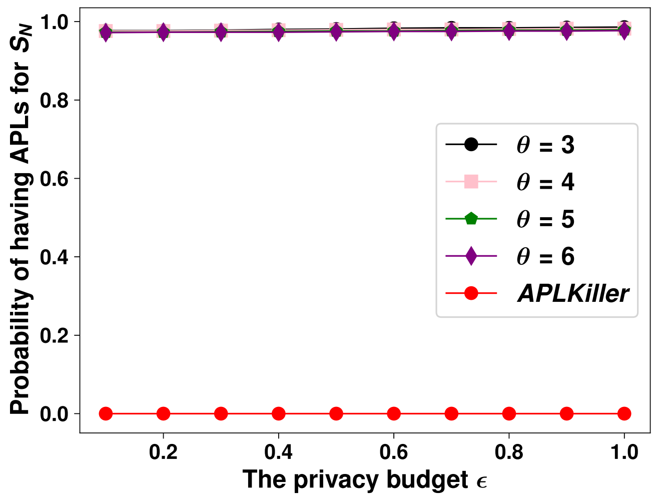
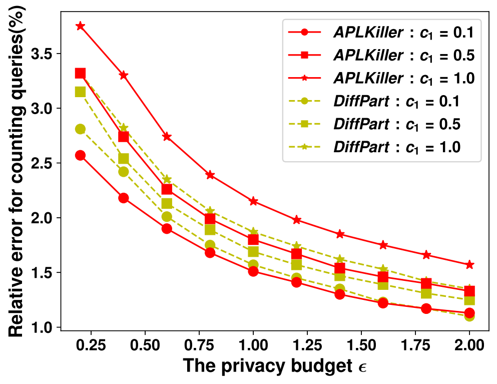


Fig. 9. Privacy comparison using PrivBayes and NLTCS.

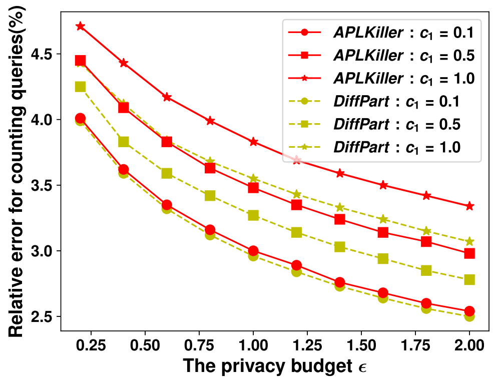
From the above analysis, one can see that our algorithm provides a stronger guarantee. We will further analyze the data utility and show that our algorithm preserves a good data utility.

6.3. Data utility analysis

First, it is important to introduce the utility metric. Since we focus on counting queries, for each experiment, 50,000 counting queries are generated. Moreover, the 50,000 queries are divided into two groups: half for querying existing itemsets in the original dataset and half for querying random itemsets. By querying existing itemsets, we examine the real data utility, and by querying random itemsets, we simulate the real-world use cases. Given



(a) Relative error on MSNBC dataset with different ϵ and c_1



(b) Relative error on Checkin dataset with different ϵ and c_1

Fig. 10. Utility comparison using MSNBC and Checkin datasets.

a query Q, for example, "How many people bought the beer?", the relative error (Chen et al., 2011) for Q is computed as $\frac{|Q(D')-Q(D)|}{max(Q(D),s)}$, where $Q(D')$ is the query result on the generated dataset, $Q(D)$ is the query result on the original dataset, and s is the sanity bound in order to weaken the influence of queries with extremely small counting answers (Chen et al., 2011). All three datasets mentioned in Table 6 are used to do the utility analysis, and we set the sanity bound to 0.01% of the size of the original dataset.

Fig. 10 shows the comparison result on MSNBC and Checkin datasets. For these two datasets, we only use them to compare APLKiller with DiffPart. We do not consider PrivBayes because MSNBC and Checkin datasets are two relatively large datasets, and it is time-consuming for PrivBayes to generate perturbed datasets when the input dataset is large. Experiment results show that it will take more than 24 hours for PrivBayes to generate a perturbed dataset, which is infeasible for experiments and real-world use case. In order to compare APLKiller with PrivBayes, we use NLTCS, which is a small dataset, to do the experiment.

Fig. 10 a shows the utility result on MSNBC dataset, with $c_2 = 1$, $f = 2$ and ϵ ranging from 0.25 to 2. We also set $c_1 \in [0.1, 0.5, 1]$. Each point is the average computed by generating 50,000 queries in terms of 1000 rounds. As c_1 is decreased from 1.0 to 0.1, one can see that the data utility is increased, and the data utility of APLKiller is better than that of DiffPart when $c_1 = 0.1$. Experiment results show that the relative error for APLKiller is reduced by 3.6% on average, as ϵ varies. Fig. 10b shows similar results when the Checkin dataset is used, and the relative error is only 2.5% to 4.9%.

Moreover, one can observe the utility-privacy dilemma: Although a smaller ϵ can decrease the probability of having the APL in Fig. 8, it brings a more significant relative query error shown in

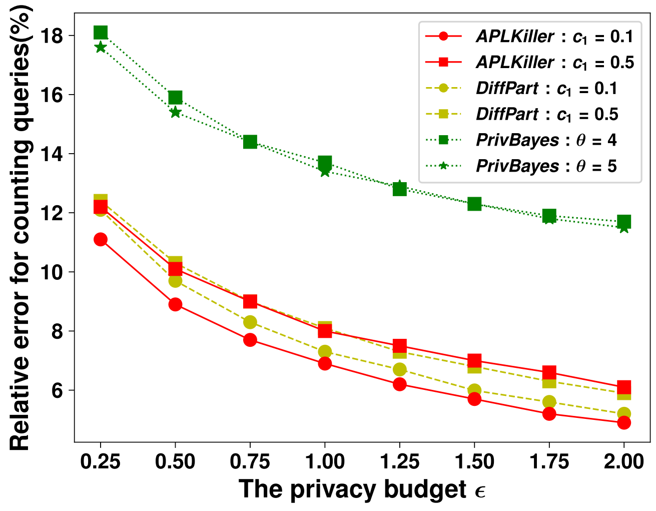


Fig. 11. Utility comparison using the NLTCS dataset.

Fig. 10, which implies that the data utility becomes worse. However, APLKiller eliminates this dilemma: No matter how the privacy parameter ϵ is set, the probability of having the APL is guaranteed to be always zero. Therefore, our algorithm lets publishers publish the dataset with good data utility while comprehensively defending against the attribute linkage attack.

In Fig. 11, we further show the utility result and consider PrivBayes. We use the NLTCS dataset to do the experiment. For DiffPart and APLKiller, we set c_1 to 0.1 and 0.5; For PrivBayes, we follow the experiment evaluation instruction (Zhang et al., 2017) and set θ to 4 and 5 separately. One can see that the data utility of APLKiller is better than that of DiffPart and PrivBayes. In detail, for different ϵ settings, our algorithm APLKiller reduces the relative error by 6.8% compared with that of DiffPart, and 49.1% compared with that of PrivBayes.

7. Conclusions

In this paper, we show that the attribute linkage attack is a severe problem under DP settings. In order to eliminate this attack, we enhance standard DP and propose the APL-Free ϵ-DP. We further propose a top-down partitioning algorithm APLKiller, which is based on a topology-theoretic approach to defend against the attack in DP settings. Compared with traditional DP algorithms, our algorithm has a lower execution time, which is efficient for publishing the dataset. Moreover, our algorithm eliminates the issue of the attribute linkage attack and achieves a higher level of privacy guarantees. Finally, better data utility is achieved.

How to set the parameter for our algorithm APLKiller to publish the dataset is our ongoing research. Specifically, APLKiller supports customized parameters for generating itemsets of different sizes, and exploring the appropriate parameter setting given a specific dataset can help users to get better data utility. Moreover, extending the scope of attacks and proposing more stringent DP variants are also exciting directions to build a more general privacy framework based on APL-Free ϵ-DP and APLKiller. For example, investigating how to address the *probabilistic attribute linkage attack* is a challenging while valuable topic. The work of John C.S. Lui was supported in part by the RGC's RIF R4032-18.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

In the appendix, we list the proof for theorems and lemmas which were proposed in previous sections.

A1. Proof for sequential sequential composition theorem

Proof. For any sequence of computations $\mathcal{A}_i(D) = D_i$, $i \in \{1, \ldots, k\}$, the probability of getting the output is $\prod_{i=1}^{k} \Pr[\mathcal{M}_i(D) = D_i]$. By applying the definition of APL-Free ϵ-DP, and for any two neighboring datasets D and D', we have

$$\prod_{i=1}^{k} \Pr[\mathcal{M}_i(D) = D_i]$$

$$\leq (\prod_{i=1}^{k} \Pr[\mathcal{M}_i(D') = D_i] \cdot \exp(\epsilon_i |D - D'|))$$

$$= \prod_{i=1}^{k} \Pr[\mathcal{M}_i(D') = D_i] \cdot \exp(\sum_{i=1}^{k} \epsilon_i)$$

Further, according to the definition of APL-Free ϵ-DP and the result derived from the topology theory, there will be no APLs in any output dataset D_i. Proof completes. □

A2. Proof for parallel sequential composition theorem

Proof. Consider two neighboring datasets D, D', and a general partitioning procedure which partitions D into $\bigcup_{i=1}^{k} D_i$, and partitions D' into $\bigcup_{i=1}^{k} D'_i$. According to the definition of the neighboring dataset, one can derive that there exists a single $j \in \{1, \ldots, k\}$, such that $|D_j - D'_j| = 1$. Then for any sequence of computations $\mathcal{A}_i(D_i)$, each of which outputs $D_i^{out} \in Range(\mathcal{A}_i)$, the probability of getting the sequence of outputs is $\prod_{i=1}^{k} \Pr[\mathcal{A}_i(D_i) = D_i^{out}]$. By applying the definition of APL-Free ϵ-DP, we have

$$\prod_{i=1}^{k} \Pr[\mathcal{A}_i(D_i) = D_i^{out}]$$

$$\leq (\prod_{i=1}^{k} \Pr[\mathcal{A}_i(D'_i) = D_i^{out'}] \cdot \exp(\epsilon_i \times |D_i - D'_i|))$$

$$\leq \prod_{i=1}^{k} \Pr[\mathcal{A}_i(D'_i) = D_i^{out'}] \cdot \exp(\max_i \epsilon_i)$$

Further, according to the definition of APL-Free ϵ-DP and the result derived from the topology theory, there will be no APLs in any output D_i^{out}. Proof completes. □

A3. Proof for APLKiller satisfying APL-Free ϵ-DP

Proof. To prove that APLKiller satisfies APL-Free ϵ-DP, we need to prove that it satisfies those two constraints in Definition 5. Because our handling of boundary itemsets is inspired by the topology-theoretic approach, which has been proved that there will be no APLs in the generated dataset, the first constraint is satisfied.

The only thing we need to prove is that APLKiller satisfies the second constraint. It can be checked that only two components of APLKiller use the original dataset D. The first component is LevelPart, which uses D_i to generate D'_i in Line. The second component is adding boundary itemsets in Line. Note that Q_i and D_{i-1} are generated in a deterministic way once D'_i is generated, so it can be derived that

$$\Pr[D_p] = \Pr[D'_1 D'_2 \ldots D'_n Q_1 \ldots Q_n]$$

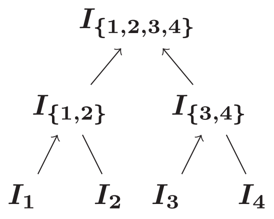


Fig. 12. Counting the number of partitioning operations needed for generating the itemset {1,3} in T. First, from the level of leaf nodes, locate I_1 and I_3, then aggregate them to nodes in the upper level. The final number of partitioning operations equals the number of internal nodes traversed during the aggregation, which is 3.

$$= \prod_{i=n}^{1}(\Pr[\mathcal{L}(D_i) = D'_i]\Pr[D_{i-1}, Q_i|D'_i] \prod_{S^i_k \in Q_i}\Pr[|S^i_k| = N^i_k])$$

$$= \prod_{i=n}^{1}(\Pr[\mathcal{L}(D_i) = D'_i] \cdot \prod_{S^i_k \in Q_i} \Pr[|S^i_k| = N^i_k]),$$

where $\mathcal{L}$ represents our algorithm APLKiller, $|S^i_k|$ is the true occurrence of kth boundary itemset in Q_i, and N^i_k is the noisy occurrence.

Because all D_i and S^i_k are disjoint, according to Theorem 2, the total privacy budget ϵ can be assigned to the generation of each D'_i and S^i_k. Since D'_i is generated by LevelPart, according to Lemma 1, it satisfies the second constraint. For each S^i_k, APLKiller directly generates the positive Laplace noise using privacy budget ϵ. Therefore, the APLKiller satisfies APL-Free ϵ-DP, and the proof is completed. □

A4. Proof for Theorem 3

Proof. In Chen et al. (2011), it has been shown that if there is no pruning operation, the maximum number of partitioning operations to reach the leaf partition is $\frac{n-1}{f-1}$, which is the number of internal nodes in the taxonomy tree T rooted at u. Now given pruning operations, one can simply derive that $Par(u, l) \le \frac{n-1}{f-1}$. It is not difficult to show that the number of partitioning operations equals the number of internal nodes visited during the aggregation of items from the bottom of the taxonomy tree to the top. To illustrate this fact, we give an example shown in Fig. 12.

Let the level of leaf nodes in T be 0. There are two cases to consider. In level $i > 0$,

1. for a l-itemset, if $l \le \lceil \frac{n}{f^i} \rceil$, which is maximum number of nodes in level i, it will visit at most l nodes and leave $\lceil \frac{n}{f^j} \rceil - l$ number of nodes unvisited at each level $j < i$.
2. for a l-itemset, if $l \ge \lceil \frac{n}{f^i} \rceil$, it will visit at most $\lceil \frac{n}{f^j} \rceil$ nodes at each level $j \ge i$.

Consider $l \in [\frac{n}{f^{k+1}}, \frac{n}{f^k})$. For level $j \ge k+1$, all nodes can be visited. For level $j \le k$, it will leave at least $\lceil \frac{n}{f^j} \rceil - l$ nodes unvisited. So

$$Par(u, l) \le \frac{n-1}{f-1} - \sum_{j=1}^{k}(\lceil \frac{n}{f^j} \rceil - l),$$

and the proof is completed. □

A5. Proof for the time complexity

Proof. In each round of APLKiller, The main computational cost comes from Line, where LevelPart is called to generate D'_i, given the length i.

Now let us prove the time complexity of LevelPart. Similar to the proof for DiffPart: For each partitioning operation, the main computational cost comes from the distribution of records from a partition to its sub-partitions, and the time complexity is $O(|D_i|)$. Since the maximum number of partitioning operations, according to Theorem 3, is bound by $\frac{n-1}{f-1}$, the time complexity of LevelPart for generating i-itemset is $O(n|D_i|)$.

Finally, for generating itemsets for all lengths, the time complexity is $O(n\sum_{i=1}^{n}|D_i|)$, which is $O(mn)$, and the proof is completed. □

CRediT authorship contribution statement

Jincheng Wang: Conceptualization, Methodology, Software, Investigation, Writing – original draft. **Zhuohua Li:** Conceptualization, Validation, Writing – review & editing. **John C.S. Lui:** Conceptualization, Supervision, Project administration, Resources, Writing – review & editing. **Mingshen Sun:** Conceptualization, Validation, Resources, Writing – review & editing.

References

Acs, G., Castelluccia, C., Chen, R., 2012. Differentially private histogram publishing through lossy compression. In: 2012 IEEE 12th International Conference on Data Mining. IEEE, pp. 1–10.

Barbaro, M., Zeller, T., Hansell, S., 2006. A face is exposed for aol searcher no. 4417749. N.Y. Times 9 (2008), 8.

Cadwalladr, C., 2018. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election.

Chen, R., Fung, B.C., Mohammed, N., Desai, B.C., Wang, K., 2013. Privacy-preserving trajectory data publishing by local suppression. Inf. Sci. 231, 83–97.

Chen, R., Fung, B.C., Yu, P.S., Desai, B.C., 2014. Correlated network data publication via differential privacy. VLDB J. 23 (4), 653–676.

Chen, R., Mohammed, N., Fung, B.C., Desai, B.C., Xiong, L., 2011. Publishing set-valued data via differential privacy. Proc. VLDB Endow. 4 (11), 1087–1098.

Chen, R., Xiao, Q., Zhang, Y., Xu, J., 2015. Differentially private high-dimensional data publication via sampling-based inference. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 129–138.

Dowker, C.H., 1952. Homology groups of relations. Ann Math 84–95.

Dwork, C., 2011. Differential privacy. Encyclopedia of Cryptography and Security 338–340.

Dwork, C., Roth, A., et al., 2014. The algorithmic foundations of differential privacy. Found. Trends® Theor. Comput. Sci. 9 (3–4), 211–407.

Dwork, C., Smith, A., Steinke, T., Ullman, J., 2017. Exposed! a survey of attacks on private data. Annu. Rev. Stat. Appl. 4, 61–84.

Erdmann, M., 2017. Topology of privacy: lattice structures and information bubbles for inference and obfuscation. arXiv preprint arXiv:1712.04130.

Erlingsson, Ú., Pihur, V., Korolova, A., 2014. Rappor: Randomized aggregatable privacy-preserving ordinal response. In: Proceedings of the 2014 ACM SIGSAC conference on computer and communications security. ACM, pp. 1054–1067.

Haeberlen, A., Pierce, B.C., Narayan, A., 2011. Differential privacy under fire. USENIX Security Symposium.

Hardt, M., Ligett, K., McSherry, F., 2012. A simple and practical algorithm for differentially private data release. In: Advances in Neural Information Processing Systems, pp. 2339–2347.

He, Y., Naughton, J.F., 2009. Anonymization of set-valued data via top-down, local generalization. Proc. VLDB Endow. 2 (1), 934–945.

Kifer, D., Machanavajjhala, A., 2011. No free lunch in data privacy. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. ACM, pp. 193–204.

Kifer, D., Machanavajjhala, A., 2012. A rigorous and customizable framework for privacy. In: Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems. ACM, pp. 77–88.

Lee, J., Clifton, C., 2011. How much is enough? choosing ε for differential privacy. In: International Conference on Information Security. Springer, pp. 325–340.

Leoni, D., 2012. Non-interactive differential privacy: a survey. In: Proceedings of the First International Workshop on Open Data. ACM, pp. 40–52.

Li, C., Hay, M., Miklau, G., Wang, Y., 2014. A data-and workload-aware algorithm for range queries under differential privacy. Proc. VLDB Endow. 7 (5), 341–352.

Liu, C., Chakraborty, S., Mittal, P., 2016. Dependence makes you vulnerable: Differential privacy under dependent tuples. In: NDSS, Vol. 16, pp. 21–24.

Machanavajjhala, A., Gehrke, J., Kifer, D., Venkitasubramaniam, M., 2006. l-diversity: privacy beyond k-anonymity. In: 22nd International Conference on Data Engineering (ICDE'06). IEEE. 24–24

Mohammed, N., Chen, R., Fung, B., Yu, P.S., 2011. Differentially private data release for data mining. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp. 493–501.

MyFitnessPal, I., 2018. Data breach hack: Myfitness pal. https://content.myfitnesspal.com/security-information/FAQ.html.

Narayanan, A., Shmatikov, V., 2006. How to break anonymity of the netflix prize dataset. arXiv preprint cs/0610105.

Narayanan, A., Shmatikov, V., 2008. Robust de-anonymization of large datasets (how to break anonymity of the netflix prize dataset). University of Texas at Austin.

Near, J., 2018. Differential privacy at scale: Uber and berkeley collaboration. Enigma 2018 (Enigma 2018).

Sweeney, L., 2002. K-anonymity: a model for protecting privacy. Int. J. Uncertainty Fuzziness Knowledge Based Syst. 10 (05), 557–570.

Terrovitis, M., Mamoulis, N., Kalnis, P., 2008. Privacy-preserving anonymization of set-valued data. Proc. VLDB Endow. 1 (1), 115–125.

Wang, Q., Zhang, Y., Lu, X., Wang, Z., Qin, Z., Ren, K., 2016. Rescuedp: Real-time spatio-temporal crowd-sourced data publishing with differential privacy. In: IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications. IEEE, pp. 1–9.

Xu, Y., Ma, T., Tang, M., Tian, W., 2014. A survey of privacy preserving data publishing using generalization and suppression. Appl. Math. Inform. Sci. 8 (3), 1103.

Zhang, J., Cormode, G., Procopiuc, C.M., Srivastava, D., Xiao, X., 2017. Privbayes: private data release via bayesian networks. ACM Trans. Database Syst. 42 (4), 25.

Zhang, X., Chen, R., Xu, J., Meng, X., Xie, Y., 2014. Towards accurate histogram publication under differential privacy. In: Proceedings of the 2014 SIAM International Conference on Data Mining. SIAM, pp. 587–595.

Jincheng Wang received the B.Eng. degree from the Huazhong University of Science and Technology. He is currently a PhD student in the Chinese University of Hong Kong, supervised by Professor John C. S. Lui. He is a member of Advanced Networking and System Research Laboratory (ANSRLab) in CUHK. His research interests include privacy and system security.

Zhuohua Li is a Ph.D. student under the supervision of Prof. John C.S. Lui in the Department of Computer Science and Engineering at The Chinese University of Hong Kong. He is a member of Advanced Networking and System Research Laboratory (ANSRLab) at CUHK. He received his B.E. degree in Computer Science and Technology at the University of Science and Technology of China in 2017. His research interests are system security and program analysis.

John C. S. Lui was born in Hong Kong. He received the Ph.D. degree in computer science from the University of California at Los Angeles, Los Angeles, CA, USA, in 1992. He was the Chairman of the Department from 2005 to 2011. He is currently a Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong (CUHK), Hong Kong. His current research interests are in communication networks, network/system security (e.g., cloud security and mobile security), network economics, network sciences, large-scale distributed systems, and performance evaluation theory. He is a Fellow of the Association for Computing Machinery (ACM) and IEEE, a Croucher Senior Research Fellow, and an elected member of the IFIP WG 7.3. He received various departmental teaching awards and the CUHK Vice-Chancellor's Exemplary Teaching Award.

Mingshen Sun is a security researcher of Baidu. He received his Ph.D. degree from The Chinese University of Hong Kong. He was a member of Advanced Networking and System Research Laboratory (ANSRLab) in CUHK. He is the Apache Teaclave (incubating) PMC member. Mingshen also leads, maintains and actively contributes to several open source projects. His research interests are system security, mobile security, Trusted Execution Environment (TEE), and memory-safe programming language.