IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) | 978-1-6654-0443-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/INFOCOMWKSHPS51825.2021.9484499

Topology-Theoretic Approach To Address Attribute Linkage Attacks In Differential Privacy

Jincheng Wang, Zhuohua Li, John C.S. Lui
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Hong Kong, China
{jcwang, zhli, cslui}@cse.cuhk.edu.hk

Mingshen Sun
Baidu Security
sunmingshen@baidu.com

***Abstract*—Differential Privacy (DP) is well-known for its strong privacy guarantee. In this paper, we show that when there are correlations among attributes 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 participant's attribute information. Our contributions are ① we show that the attribute linkage attack can be initiated with high probability even when data are protected under DP, ② we propose an enhanced DP standard called "APL-Free ϵ-DP", ③ by leveraging on topology theory, we design an algorithm "APLKiller" which satisfies this standard. Finally, experiments show that our algorithm not only eliminates the attribute linkage attack, but also achieves better data utility.**

***Index Terms*—differential privacy, attribute linkage attack, topology theory.**

I. Introduction

In the current digital era, personal information is extremely valuable. Since data collection is quite common, privacy becomes a major concern. Recently, differential privacy (DP) was proposed [1]. Companies like Google are using DP algorithms to protect user privacy [2]. However, DP is not without any pitfall [3]. In this paper, we show that even datasets processed by DP are still prone to "*attribute linkage attack*" [4] when attributes are correlated. Under such attack, attackers can leverage part of attribute information to deduce more information of a victim.

Table I is an example that illustrates the attribute linkage attack. Table I(a) is a retail dataset D in which each record is a unique itemset with its occurrence. Before demonstrating the attack, some terms should be formally defined. Let $\mathcal{I} = \{I_1, I_2, ..., I_n\}$ be the item universe with size n. In the example, we will represent *beer*, *toothpaste*, *scissor* and *hanger* as I_1 to I_4 respectively. For an itemset $S \subseteq \mathcal{I}$, $|S.items|$ is the number of items in S and $|S|$ is S's occurrence. Thus, $D = \{r : (S, |S|)\}$. For example, in Table I(a), an itemset S can be $\{I_1, I_2\}$, with $|S.items| = 2$ and $|S| = 4$.

Note that in Table I(a), correlations exist among items. For example, most customers who bought $\{I_2, I_3, I_4\}$ would like to buy I_1, which is *beer*. The consequence of having such correlation is that all itemsets which contain $\{I_2, I_3, I_4\}$ have zero occurrence, except for itemset $\{I_1, I_2, I_3, I_4\}$ with 20 occurrence. To provide better privacy protection, the data publisher may choose to use DP algorithms to perturb the dataset D. For example, a traditional algorithm [1] to achieve DP is designed. Given the DP algorithm, the lower occurrence an itemset S has, the lower probability S will be added to the perturbed dataset. Table I(b) shows a possible output D_p.

Unfortunately, once D_p is published, the attribute linkage attack can be initiated. For simplicity, we use "Eve" to represent the adversary and "Alice" to represent the victim. If Eve knows that Alice went to this shop last week and bought the itemset $\{I_2, I_3, I_4\}$ (this is the attacker's prior knowledge), she can search for all records in D_p which contain these three items, and finally uniquely identify the first record in Table I(b). The consequence is that Eve can now infer that Alice also bought *beer*, which causes the leakage of Alice's privacy.

The root cause is that even though DP algorithms try to add random noise to increase the privacy level, the private information of individuals is embedded in and leaked through the underlying item correlations which DP algorithms need to preserve. As a result, most DP algorithms maintain the correlation to preserve the data utility, but this also exposes the attribute linkage vulnerability. In our example, the underlying correlation makes sure that in D, $\{I_1, I_2, I_3, I_4\}$'s occurrence is large, while any other itemset's occurrence which contains $\{I_2, I_3, I_4\}$ is zero. Such correlation is well preserved by the DP algorithm, and as a result, increases the possibility for Eve to deduce that "Alice drinks beer".

In summary, our contributions are as followings. We first show that correlations among items introduce the attribute linkage attack in DP settings. Then we propose the "*APL-Free ϵ-DP*" standard which guarantees no attribute linkage attack is possible in the published dataset. Finally, we design a novel algorithm, "*APLKiller*", which leverages the topology theory [5] to defend against attribute linkage attack, while preserves the data utility for published datasets.

II. Background

In this section, we provide the background of DP, then discuss how topology theory [6] can help to tackle the attribute linkage attack.

A. Correlation Issues in Differential Privacy

A formal definition of DP is given as followings.

978-1-6654-0443-3/21/$31.00 ©2021 IEEE

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

(a) An example of retail dataset D.

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

(b) A possible published dataset D_p using DP.

TABLE I: Examples of a retail dataset D and its perturbed version D_p using differential privacy.

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

TABLE II: Relation R for the dataset D_p in Table I(b).

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

TABLE III: Perturbed D'_p without attribute linkage vulnerabilities.

Definition *1 (ϵ-differential privacy):* A randomized mechanism $\mathcal{M}$ provides ϵ-differential privacy if for any two neighboring datasets D_1 and D_2, for which only differ in one occurrence for a record, and for any output $D_p \subseteq Range(\mathcal{M})$,

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

where the probability is taken over the randomness of $\mathcal{M}$, and ϵ is called "*privacy budget*". The smaller ϵ is, the better privacy guarantee a DP algorithm has.

Although DP provides a strong privacy guarantee, correlation issues has been reported in several researches [7]–[9]. They assume that the victim's privacy is encoded in a social correlation which is formed by a special group of participants, e.g, friends or families. Once such underlying social correlations are discovered, the victim's privacy is under leakage.

Few researches analyze the severity of the attribute linkage attack in DP settings. In our paper, we find out that the victim's privacy can also be encoded in the attribute correlation which is formed by all participants, and such correlation allows the attacker to launch the attribute linkage attack. Our proposed correlation issue is more general and severe with the following reasons. First, it is easier for attackers to discover the attribute correlation in data, compared with the record correlation. Second, by leveraging the attribute correlation, the attacker can construct the attribute linkage attack to leak the private information. Third, it is harder for data publishers to defend against the attribute linkage attack, because the underlying correlation is formed by all participants instead of a small group. Previously proposed methods cannot solve our issues well. In order to eliminate the attribute linkage attack, we introduce a methodology built upon topology theory.

B. Topology of Privacy

A formal methodology [6] is presented to study privacy using topology theory. The dataset D is modeled as a relation R, and an example for D_p in Table Ib is shown in Table II. To eliminate the attribute linkage attack targeting on an itemset S, a topology theory is proposed to prove that the "*boundary set*" B_S should exist in R.

Definition *2 (Boundary Set):* The boundary set B_S of an itemset S can be generated by removing each item from S. That is,

$$B_S = \{S' \subset S \mid |S - S'| = 1\}.$$

For any boundary itemset $S' \in B_S$, if S' does not exist in R, then the adversary can use S' as a prior knowledge to deduce target's itemset S, and $S - S'$ contains the leaked item information. In our example, if Eve knows $S' = \{I_2, I_3, I_4\}$, she can deduce that Alice also bought I_1, which is *beer*. The author further presents a proof that only checking the maximal itemsets is sufficient to protect D against the attribute linkage attack.

Definition *3 (Maximal Itemset):* Let D^I be the set of itemsets in the dataset D. An itemset $S \in D^I$ is a "maximal itemset" if it is not a subset of any other itemset in D^I.

The above result suggests a defense methodology: For each maximal itemset S, if one can artificially generate records for all missing boundary itemsets of S, then no attribute linkage attack can happen in D_p. We will use this idea to improve traditional DP. In our example, Table III is a generated dataset wherein every itemset is free from attribute linkage vulnerabilities. Records 6-8 are the artificially generated records to satisfy the boundary-presence requirement for the only maximal itemset $\{I_1, I_2, I_3, I_4\}$. With the existence of itemset $\{I_2, I_3, I_4\}$, Eve cannot deduce that Alice bought *beer*.

III. Attribute linkage attack on DP-processed datasets

In this section, we show that an attribute linkage attack can be initiated in DP-processed datasets.

A. Attribute Privacy Leakage

We formally define the attribute linkage attack. We use "Attribute Privacy Leakage" (APL) to represent an itemset which an adversary can use to capture user's private information.

Definition *4 (APL):* Given a dataset D, we say D has an APL if there is an itemset $Q \subset \mathcal{I}$, where $\mathcal{I}$ is the item universe, such that

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

where D^I is the set of itemsets in D. By using the itemset Q, the adversary can uniquely identify S in D^I and $S - Q$ is the leaked information that the adversary could obtain. In our previous example, $Q = \{I_2, I_3, I_4\}$, and $S = \{I_1, I_2, I_3, I_4\}$. By using this APL, "Alice bought *beer*" is the leaked information.

In the rest of the paper, unless we state otherwise, all itemsets mentioned are maximal itemsets and all APLs mentioned are boundary itemsets which can be used to uniquely identify maximal itemsets. Such an assertion implies that we allow a highly powerful threat model: The adversary is allowed to have the most prior information.

B. *Attack Analysis*

Let us first state the attack method.

Attack method: In this paper, we assume the adversary has the following prior information in advance.

1) The victim's itemset S is in D.
2) The adversary knows a boundary itemset $Q \subset S$, where $Q \neq \emptyset$ and Q can uniquely identify S in D.

By accessing DP processed dataset D_p, the adversary will find a set $G = \{S' | S' \in D_p^I \text{ and } Q \subseteq S'\}$, which is the set of itemsets containing Q in D_p. Then Q is an APL in D_p and the attack is successfully launched if and only if $G = \{S\}$.

Deriving probability: Note that the probability of successful attribute linkage attack depends on the set G. Once there is an itemset in G and it is not S, the attack will fail. Formally, let $P = \{S' \mid Q \subsetneq S' \subsetneq \mathcal{I} \text{ and } S' \neq S\}$. Let $\mathcal{C}_1 : S \in D_p^I$ and $\mathcal{C}_2 : P \cap D_p^I = \emptyset$, we have

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

Computing the above probability is non-trivial for different DP algorithms. However, since DP algorithms need to maintain a high data utility, the probability for a specific itemset S' to be added to D_p has a strong correlation with the occurrence for S' in D. Based on this observation, one can assert that

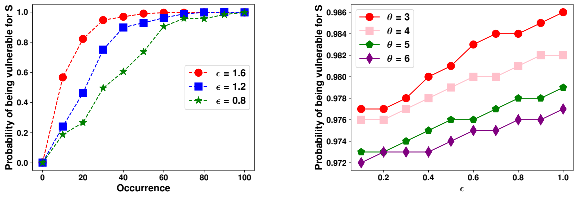
$$\Pr(\mathcal{C}_1\mathcal{C}_2) \propto f_D(S) \cdot \prod_{S' \in P} (1 - f_D(S')),$$

where $f_D(S)$ is the frequency of itemset S in D.

In our attack, since Q is an APL for S in D, which means that there is no other itemset containing Q, a strong correlation exists between Q and $S - Q$. The correlation guarantees that $f_D(S')$ is close to 0, and such property is well preserved by DP algorithms. The consequence is that in D_p, there is high probability for the attacker to uniquely identify S. In the next section, we will use experiments to demonstrate the high probability of the attack in the DP setting.

C. *Case Study*

We use DiffPart [10] and PrivBayes [11] to demonstrate the attribute linkage attack. These two algorithms are popular and representative: one is partitioning-based and another is sampling-based. Also, both algorithms aim to preserve the item correlation to increase the data utility.



(a) MSNBC dataset.

(b) NLTCS dataset.

Fig. 1: Probability of having S vulnerable in MSNBC.

Experiment results: *MSNBC*[1] and *NLTCS*[2] are used to demonstrate the feasibility of the attack in the real world. A detailed description of datasets is in Section V. Note that there is only one maximal itemset in both datasets, and it has APLs. Therefore, we set the only maximal itemset in both datasets as the target, and choose an arbitrary APL in the original dataset to investigate whether the target is vulnerable in D_p. For each parameter setting, we generate 1,000 datasets to compute the average attack probability.

In Figure 1a, we use DiffPart algorithm, and increase the occurrence of S to change $f_D(S)$ and to observe the impact. The result shows that the probability of being vulnerable for S increases quickly. We further use PrivBayes to show the attack probability in Figure 1b. One can observe that in Figure 1, as ϵ increases, the property of being vulnerable for S is more likely to be preserved. In summary, there are two important conclusions.

1) Users need to carefully select parameters to reduce the attack probability for traditional DP algorithms.
2) A larger ϵ can bring a better data utility, but also increases the attack probability.

IV. Defense Methodology

In this section, we propose an enhanced DP standard "*APL-Free ϵ-DP*" and our algorithm "*APLKiller*".

A. *APL-Free ϵ-DP*

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

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

1) For any $D_p \in Range(\mathcal{M})$, there is no APL in D_p.
2) For any two neighboring datasets D_1 and D_2, and for any possible output $D_p \subseteq Range(\mathcal{M})$,
$$\frac{Pr(\mathcal{M}(D_1) = D_p)}{Pr(\mathcal{M}(D_2) = D_p)} \leq exp(\epsilon).$$

Besides providing the privacy guarantee of traditional DP, APL-Free ϵ-DP requires that there should be no APLs in D_p to defend against the attribute linkage attack. It is important to state that APL-Free ϵ-DP follows the parallel composition theorem.

Theorem *1 (Parallel Composition Theorem):* Let $\mathcal{M}_i$ be an APL-Free ϵ_i-DP algorithm, and let $\bigcup_t^k D_t = D$ be a

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

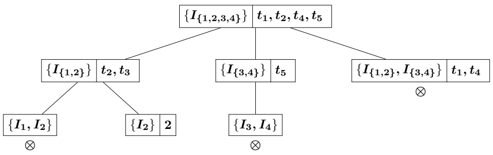

$\{I_{\{1,2,3,4\}}\}$ | t_1, t_2, t_4, t_5
$\{I_{\{1,2\}}\}$ | t_2, t_3
$\{I_{\{3,4\}}\}$ | t_5
$\{I_{\{1,2\}}, I_{\{3,4\}}\}$ | t_1, t_4
⊗
$\{I_1, I_2\}$
⊗
$\{I_2\}$ | 2
$\{I_3, I_4\}$
⊗

Fig. 2: The generation of 1-itemsets using LevelPart

set of arbitrary disjoint subsets of D. Then $\mathcal{M}_{[k]}(D) = (\mathcal{M}_1(D_1), ..., \mathcal{M}_k(D_k))$ provides APL-Free $\max_t \epsilon_t$-DP.

In the following section, we present our proposed APL-Free ϵ-DP algorithm, which is called "*APLKiller*".

B. APLKiller

Framework: The aim of APLKiller is to eliminate APLs. The methodology is to make sure that *whenever a maximal itemset S is added to D_p, the boundary set B_S should also be added to D_p*. Let *l-itemset* be the itemset which contains l items, and let D_l be the part of records in D which contain the l-itemset. Algorithm 1 shows the pseudocode of APLKiller. APLKiller first generates l-itemsets in the decreasing order of the size by calling "*LevelPart*" (Line 5). The benefit is that in each round, all generated itemsets are guaranteed to be maximal itemsets. Then all boundary itemsets are determined and added to D_p such that there will be no APLs in D_p (Line 7 to Line 16).

Algorithm 1 APLKiller

```
Input: $D$, parameter vectors $F$, $C_1$ and $C_2$, privacy budget $\epsilon$
Output: perturbed dataset $D_p$
 1: $l \leftarrow |\mathcal{I}|$
 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 $l \geq 1$ do
 5:     $D'_l \leftarrow LevelPart(l, D_l, F_l, C_1^l, C_2^l, \epsilon)$ //Generate $D'_l$
 6:     $D_p = D_p \cup D'_l$
 7:     for $S_j^l \in D'_l$ do
 8:         $Q_l = Q_l \cup B_{S_j^l}$ //Aggregate boundary itemsets for $D'_l$
 9:     end for
10:     for $S_k^l \in Q_l$ do
11:         $N_k^l = 0$
12:         //Determine the noisy occurrence for boundary itemsets
13:         while $N_k^l \leq 0$ do
14:             $N_k^l = NoisyCount(|S_k^l|, \epsilon)$
15:         end while
16:         Add $S_k^l$ to $D_p$ with $|S_k^l| = N_k^l$
17:     end for
18:     //Remove influences of $Q_i$ on generating $D'_{l-1}$
19:     Remove each $S_k^l \in Q_l$ from $D_{l-1}$
20:     $l \leftarrow l - 1$
21: end while
22: return $D_p$
```

Generation of l-itemset. The pseudocode of LevelPart is in Algorithm 2, and it applies similar partitioning procedure with DiffPart [10]. Figure 2 is an example to show the partitioning procedure. Each rectangle represents a partition p, which stores an hierarchy cut $p.cut$ and a set of records in D. $p.cut$ is a set of nodes in the taxonomy tree T, and it determine which records can be stored in p. An example of T is in Figure 3. When p needs to generate its sub-partitions, a random node $u \in p.cut$ will be picked and expanded to its child nodes in T.

Algorithm 2 LevelPart

```
Input: length $l$, dataset $D_l$, fan-out $F_l$, constant $C_1^l$ and $C_2^l$
Output: Perturbed dataset $D'_l$
 1: Initialize $D'_l$
 2: Construct taxonomy tree $T$ with $F_l$
 3: Create Partition $p$ which includes all records and store root of $T$
    into $p.cut$. Let $p.B = \frac{\epsilon}{2}$, $p.r = Par(p, l)$ and $p.\alpha = \frac{p.B}{p.r}$
 4: Add $p$ to an empty queue $Q$
 5: while $Q \neq \emptyset$ do
 6:     Dequeue $p'$ from $Q$
 7:     $P \leftarrow$ LevelSGP($p', T, l, C_1^l$) //Generate subpartitions of $p'$
 8:     for each $p_i \in P$ do
 9:         if $p_i$ is leaf partition then
10:             //Determine the noisy occurrence of the itemset
11:             $N_{p_i} = NoisyCount(|p_i|, \frac{\epsilon}{2} + p_i.B)$
12:             if $N_{p_i} \geq \sqrt{2}\frac{C_2^l}{\epsilon/2 + p_i.B}$ then
13:                 Add $N_{p_i}$ copies of $p_i.cut$ to $D'_l$
14:             end if
15:         else
16:             Add $p_i$ to $Q$ //Continue to generate subpatitions of $p_i$
17:         end if
18:     end for
19: end while
20: return $D'_l$
```

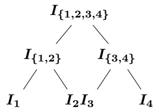


Fig. 3: Taxonomy tree T with $f = 2$ to control T's degree

In Line 7, "*LevelSGP*" is called to generate the sub-partitions for p. Such process will be repeated until each partition in the queue Q is a leaf partition, which represents a specific itemset S. After that, the noisy occurrence for S is computed in Line 11, and is further compared with a bound controlled by user-defined parameter C_2^l in Line 12.

We now present our LevelSGP, and the pseudocode is in Algorithm 3. It is used to generate sub-partitions for current partition p, and it guarantees that all generated sub-partitions contain l-itemsets. LevelSGP first generates sub-partitions in Line 5, then uses the similar framework with DiffPart [10]: For each non-empty sub-partition s_i, a noisy sum of occurrence for all records in s_i is computed to determine whether current sub-partition deserves to be preserved (Line 8 to Line 14). For empty sub-partitions, a sampling mechanism is applied to randomly sample empty sub-partitions (Line 17 to Line 22).

In order to generate sub-partitions which contain l-itemsets, we design a novel procedure. Suppose we want to generate p's sub-partitions. First a random node $u \in p.cut$ is selected, and is expanded by its child nodes in T. For example, in Figure 2, the partition with cut $\{I_{3,4}\}$ can generate sub-

Algorithm 3 LevelSGP

```
Input: Partition $p$, taxonomy tree $T$, length $l$, constant $C_1^l$
Output: 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: Allocate records in $p$ to sub-partitions in $S$
 7: Bound $d = \sqrt{2}C_1^l \times height(p.cut)/p.\alpha$ //Compute the bound
 8: for $s_i \in S$ do
 9:     $N_{s_i} = NoisyCount(|s_i|, p.\alpha)$
10:     if $N_{s_i} \geq d$ then
11:         $s_i.B = p.B - p.\alpha$, $s_i.r = p.r - 1$, $s_i.\alpha = \frac{s_i.B}{s_i.r}$
12:         Add $s_i$ to $V$ //The subpartition can be further processed
13:     end if
14: end for
15: $j = 1$
16: //Select some empty subpartitions by random sampling
17: while $j \leq 2^{|u.items|} - |S|$ do
18:     if $NoisyCount(0, p.\alpha) \geq d$ then
19:         Randomly generate an empty sub-partition $s'_j$
20:         if $|s'_j.cut| \leq l$ and $|s'_j.items| \geq l$ then
21:             $s'_j.B = p.B - p.\alpha$, $s'_j.r = p.r - 1$, $s'_j.\alpha = \frac{s'_j.B}{s'_j.r}$
22:             Add $s'_j$ to $V$
23:         end if
24:     end if
25: end while
26: return $V$
```

partitions with cuts $\{I_3\}$, $\{I_4\}$, and $\{I_3, I_4\}$. However, given the length constraint, not all sub-partitions are needed. For example, the partition with cut $\{I_3, I_4\}$ should be pruned, as it only contains a 2-itemset $\{I_3, I_4\}$.

To filter out those pruned partitions, there are two situations in which a partition should be pruned: (1) $|p.cut| > l$; (2) $|p.items| < l$. In the first situation, since each node in $p.cut$ provides at least one item to generate itemsets, if the total number of nodes is larger than l, the partition p will generate at least $(l+1)$-itemsets and should be filtered out. In the second situation, $|p.items|$ is the sum of items for all nodes in $p.cut$. If it is less than l, the partition p can generate at most $(l-1)$-itemsets, which should also be pruned. By adding the length check in Line 5 and Line 20, LevelSGP only generates sub-partitions containing l-itemsets. Next we will introduce the privacy budget allocation scheme.

Privacy budget allocation. Initially, in Line 3 of LevelPart, a startup partition p is created. $p.B$ records the remaining budget for further sub-partition generation, and initially we set it to be $\frac{\epsilon}{2}$. The remaining $\frac{\epsilon}{2}$ budget will be finally used to determine the noisy occurrence for the generated itemset. $p.\alpha$ represents the computed budget cost for the incoming sub-partition generation, and $p.r$ is used to compute $p.\alpha$.

In Line 9 and Line 18 of LevelSGP, $p.\alpha$ is consumed to compute the noisy sum of occurrences N_{s_i} for the generated sub-partition s_i. Then in Line 11 and Line 21, we update $s_i.B$, $s_i.r$ and $s_i.\alpha$, and preserve s_i for further partitioning operations. Note that since generated sub-partitions are disjoint from each other, according to Theorem 1, $p.B - p.\alpha$ can be allocated to each sub-partition. Finally, in Line 11 of LevelPart, a leaf partition p is generated, and the remaining partitioning budget $p.B$ plus preserved $\frac{\epsilon}{2}$ are used for determining the noisy occurrence for the itemset in $p.cut$.

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

TABLE IV: Description of experimental datasets

In order to generate l-itemsets from p and to determine $p.\alpha$, we first compute the maximum number of rounds for p to reach the leaf partition, which is $p.r$. Then we let $p.\alpha = \frac{p.B}{p.r}$. Such privacy allocation scheme is proved to cost less than $\epsilon/2$ budget [10]. Note that every time a sub-partition s_i is generated from p, we have $s_i.r \leq p.r - 1$. Based on this finding, we first use the following theorem to compute $p.r$ for the initial partition in Line 3 of LevelPart. Then for any generated sub-partition s_i from its parent partition p, we let $s_i.r = p.r - 1$.

Theorem *2:* Suppose p is an hierarchy cut which contains single internal node u and $|u.items| = n$. Given the fan-out parameter f, for $f^k \leq \frac{n}{l} \leq f^{k+1}$ and $l > 0$, $k \geq 0$, we have

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

The detailed proof is in our technical report [12].

Handling of boundary itemsets. When LevelPart returns D'_l, APLKiller first adds itemsets in D'_l to the published dataset D_p (Line 6). Then the boundary set $B_{S_j^l}$ for each l-itemset S_j^l is derived. APLKiller collects those boundary itemsets in Line 8. Then for each boundary itemset S_k^l, APLKiller repeatedly generates a Laplace noisy occurrence for S_k^l until it is positive (Line 13). After that, S_k^l is added to D_p. Note that in Line 19, the record of each boundary itemset S_k^l is removed from D_{i-1}. They have been processed as the boundary itemset, and they should not cause any influence in generating $(l-1)$-itemsets.

C. Algorithm Analysis

We first give the privacy analysis of our algorithm.

Theorem *3:* APLKiller satisfies APL-Free ϵ-DP.

Moreover, for the time complexity analysis, we have the following result.

Lemma *1:* The time complexity of APLKiller 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.

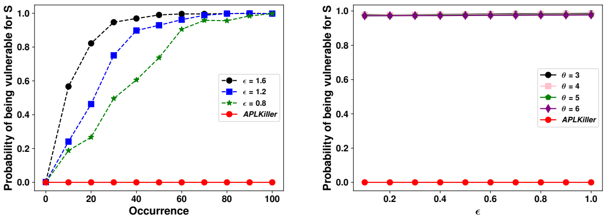
The detailed proof is given in our technical report [12].

V. Evaluation

In this section, we analyze the privacy guarantee and data utility of our algorithm. Detailed information about datasets is shown in Table IV, 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.

A. *Privacy Guarantee Analysis*

Figure 4 shows the experiment result, and the experiment design is similar with that in Section III-B. One can observe that by using APLKiller, there is no single APL in the generated dataset, which shows a high privacy guarantee.



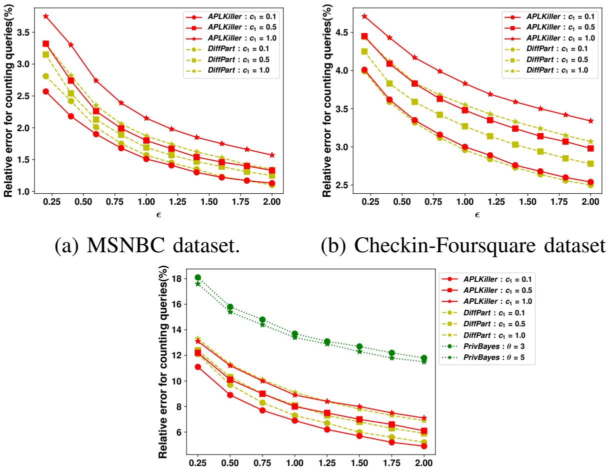
(a) MSNBC dataset.

(b) NLTCS dataset.

Fig. 4: Privacy comparison using real-world datasets.

B. *Data Utility Analysis*

Since counting query is the most fundamental operator in data mining today, we focus on it. For each parameter setting, 50,000 random counting queries are generated. Given a query Q, the relative error [10] 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. we set the sanity bound to 0.01% of the size of the original dataset.



(a) MSNBC dataset.

(b) Checkin-Foursquare dataset.

(c) NLTCS dataset.

Fig. 5: Relative error for APLKiller, DiffPart and PrivBayes.

In Figure 5, each point is the average computed by generating 50,000 queries in terms of 1,000 rounds. For DiffPart and APLKiller, we change the parameter ϵ and c_1, which is used to control the partitioning bound. In APLKiller, we allow users to set C_1^l for generating l-itemsets, and we set $C_1^l = c_1$ for any length l. Note that it is time-consuming for PrivBayes to process large datasets (over 24 hours), so PrivBayes is not used for MSNBC and Checkin datasets. In Figure 5a and Figure 5b, experiment results show that the relative error for APLKiller is reduced by 3.6% in average, as ϵ varies. In Figure 5c, one can check that APLKiller reduces the relative error by 6.8% compared with that of DiffPart, and 49.1% compared with that of PrivBayes. These show APLKiller has a higher data utility.

For traditional DP algorithms, although a smaller ϵ can decrease the probability of being attacked, the data utility becomes worse. However, APLKiller eliminates this dilemma: No matter how the privacy parameter ϵ is set, the probability of being attacked is guaranteed to be zero. Therefore, our algorithm lets publishers to publish the dataset with good data utility, while defending against the attribute linkage attack comprehensively.

VI. Conclusions

In this paper, we first show that the attribute linkage attack is a severe problem when using DP. In order to eliminate this attack, we improve DP and propose APL-Free ϵ-DP. We further design an algorithm, APLKiller, which leverages the topology-theoretic approach to defend against the attribute linkage attack. However, in our paper, we did not consider the probabilistic attribute linkage attack, which is a more advanced attack. Also, we did not give a clear instruction on how to choose APLKiller's parameters to get better data utility. These are potential directions for future research.

The work of John C.S. Lui was supported in part by the RIF R4032-18.

References

[1] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[2] J. Near, "Differential privacy at scale: Uber and berkeley collaboration," in *Enigma 2018 (Enigma 2018)*, 2018.

[3] J. Lee and C. Clifton, "How much is enough? choosing ε for differential privacy," in *International Conference on Information Security*. Springer, 2011, pp. 325–340.

[4] R. Chen, B. C. Fung, N. Mohammed, B. C. Desai, and K. Wang, "Privacy-preserving trajectory data publishing by local suppression," *Information Sciences*, vol. 231, pp. 83–97, 2013.

[5] C. H. Dowker, "Homology groups of relations," *Annals of mathematics*, pp. 84–95, 1952.

[6] M. Erdmann, "Topology of privacy: Lattice structures and information bubbles for inference and obfuscation," *arXiv preprint arXiv:1712.04130*, 2017.

[7] D. Kifer and A. Machanavajjhala, "No free lunch in data privacy," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 193–204.

[8] R. Chen, B. C. Fung, P. S. Yu, and B. C. Desai, "Correlated network data publication via differential privacy," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 23, no. 4, pp. 653–676, 2014.

[9] C. Liu, S. Chakraborty, and P. Mittal, "Dependence makes you vulnberable: Differential privacy under dependent tuples." in *NDSS*, vol. 16, 2016, pp. 21–24.

[10] R. Chen, N. Mohammed, B. C. Fung, B. C. Desai, and L. Xiong, "Publishing set-valued data via differential privacy," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 1087–1098, 2011.

[11] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao, "Privbayes: Private data release via bayesian networks," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 4, p. 25, 2017.

[12] Technical report: https://github.com/wang70880/infocom2021workshop/blob/main/topoPrivacy.pdf.