PrivSuper: a Superset-First Approach to Frequent Itemset Mining under Differential Privacy

Ning Wang[1], Xiaokui Xiao[2], Yin Yang[3], Zhenjie Zhang[4], Yu Gu[5], Ge Yu[5]

[1,5]*School of Computer Science and Engineering, Northeastern University, China*
[2]*School of Computer Science and Engineering, Nanyang Technological University , Singapore*
[3]*College of Science and Engineering, Hamad Bin Khalifa University, Qatar*
[4]*Advanced Digital Sciences Center, Illinois at Singapore Pte., Singapore*
[1]wning1217@gmail.com,[2]xkxiao@ntu.edu.sg,[3]yyang@qf.org.qa,
[4]zhenjie@adsc.com.sg,[5]{guyu,yuge}@cse.neu.edu.cn

***Abstract*—Differential privacy, which has been applied in Google Chrome and Apple iOS, provides strong privacy assurance to users while retaining the capability to discover statistical patterns from sensitive data. We focus on top-k frequent itemset mining on sensitive data, with the goal of obtaining high result utility while satisfying differential privacy. There are two basic methodologies to design a high-utility solution: one uses generic differential privacy mechanisms as building blocks, and minimizes result error through algorithm design. Most existing work follows this approach. The other methodology is to devise a new building block customized for frequent itemset mining. This is much more challenging: to our knowledge, only one recent work, *NoisyCut*, attempts to do so; unfortunately, *Noisycut* has been found to violate differential privacy.**

This paper proposes a novel solution *PrivSuper*, which contains both a new algorithm and a new differential privacy mechanism. Unlike most existing methods that follow the Apriori framework, which starts from single items and iteratively forms larger itemsets, *PrivSuper* directly searches for maximal frequent itemsets, and subsequently adds their sub-itemsets to the results without additional privacy budget consumption. During the search, *PrivSuper* applies a customized mechanism to extend the current itemset with one more item, which we call the *sequence exponential mechanism* (SEM). Notably, SEM does not consume any privacy budget at all, if it turns out that the current itemset cannot be extended. Extensive experiments using several real datasets demonstrate that *PrivSuper* achieves significantly higher result utility compared to previous solutions.

I. INTRODUCTION

Differential privacy [1] is a strong and rigorous standard for privacy protection that has been applied to well-known software systems such as Google Chrome [2] and Apple iOS [3]. It enables data analysts to discover statistical patterns from sensitive data, and release a perturbed version of the these patterns with the guarantee that no third party can infer individual-level information with high confidence based on the released results. The level of privacy protection is tunable through a parameter ε, which is called the *privacy budget*. This paper focuses on mining top-k frequent itemsets [4], [5] over sensitive data under differential privacy, which has numerous applications, ranging from classic market basket analysis [6] to more recent ones such as physician visits analysis [7]. In particular, each record in the dataset, called a *transaction*, consists of a set of *items*, e.g., a transaction at a supermarket contains multiple purchased commodities. The result consists of the k most frequent itemsets (e.g., {beer, diaper} [8]) among all transactions. Our goal is to report k itemsets that provide the highest utility to the user under ε-differential privacy constraints. Note that the utility of an estimated top-k itemsets involves multiple aspects, which we elaborate in Section II-B.

In general, there are two main methodologies for obtaining a high-utility, differentially private technique. One is to use generic mechanisms for enforcing differential privacy as basic building blocks, and combine them in a well-engineered way to maximize result utility. In other words, the focus is on *algorithm design*. As outlined in Section II-B, the majority of existing solutions for differentially private frequent itemset mining follow this approach, which includes the current state of the art, *PrivBasis* [4]. In particular, as described in Section II-A, two most commonly used building blocks are the Laplace mechanism (LM) [9] and the exponential mechanism (EM) [10], which can be combined through the sequential composition and parallel composition properties of differential privacy [11]. Each usage of either LM or EM requires a portion of the total privacy budget ε. Therefore, a goal in the differentially private algorithm design is to minimize privacy budget consumption.

The other methodology, which is much more challenging, is to create a new, custom-built differentially private mechanism for the target application, i.e., frequent itemset mining. To our knowledge, only one recent work, namely *NoisyCut* [5], attempts to do so. Specifically, as explained in Section II-B, *NoisyCut* includes a new differentially private mechanism for publishing all frequent itemsets with constant privacy budget consumption [5]. Unfortunately, this mechanism (and, thus, *NoisyCut* itself) violates differential privacy [12], [13], with no known fix.

This paper proposes *PrivSuper*, a novel solution for differentially private top-k frequent itemset mining, which contains both a new algorithm and a new basic differential privacy mechanism. Specifically, unlike most existing solutions that follow the Apriori framework [14] which starts from frequent singleton itemsets and iteratively combines them to form larger ones, *PrivSuper* employs a superset-first approach that directly searches for maximal frequent itemsets (MFIs). The main benefit for this approach is that once we identify an MFI, we can simply add all its sub-itemsets to the results without further privacy budget consumption, since these subsets always

have higher frequencies than that of the MFI [1]. The challenge, however, is that we must repeatedly test whether a given itemset is maximal, i.e., whether it can be extended to a larger set that is still within the top-k most frequent ones. In a simple implementation, each such test has to be performed with an invocation of EM, leading to high privacy budget usage.

PrivSuper addresses the above problem through a novel *sequence exponential mechanism (SEM)* that combines the ideas of both LM and EM. Specifically, SEM is applied to extend a given itemset S by adding one more item x to S that maximizes the frequency of resulting itemset $S' = S \cup \{x\}$. Perhaps surprisingly, SEM consumes zero privacy budget if the resulting itemset S' is no longer frequent; otherwise, SEM consumes the same amount of budget as the plain EM. In other words, with SEM we perform maximal checks for free. The use of SEM leads to significant privacy budget savings, and, thus, significant utility improvements. Extensive experiments using multiple real datasets confirm that SEM achieves considerably higher result utility compared to *PrivBasis*, the current state of the art.

The rest of this paper is organized as follows. Section II introduces necessary background on differential privacy, and overviews existing solutions for differentially private top-k frequent itemset mining. Section III presents the general framework of *PrivSuper*. Section IV describes the sequence exponential mechanism. Section V provides algorithmic details and further optimizations of *PrivSuper*. Section VI contains an extensive experimental evaluation. Section VII concludes the paper with directions for future work.

II. Background

Section II-A provides necessary background on differential privacy. Section II-B overviews existing solutions for differentially private top-k itemset mining.

A. Differential Privacy

Differential privacy is based on the concept of *neighboring databases*, as follows.

Definition 1: (Neighboring Databases [15]). Two databases D_1 and D_2 are neighboring databases, if and only if we can obtain one from the other by adding or removing one record, i.e., there exists record t such that either $D_1 = D_2 \cup \{t\}$ or $D_2 = D_1 \cup \{t\}$.

One popular definition of differential privacy is ε-differential privacy, as follows.

Definition 2: (ε-Differential Privacy [15]). Let $\mathcal{A}$ be a randomized algorithm. $\mathcal{A}$ satisfies ε-differential privacy, if and only if for any pair of neighboring databases D_1 and D_2, and any output O of $\mathcal{A}$, we have:

$$Pr\left[\mathcal{A}(D_1) = O\right] \leq e^{\varepsilon} \cdot Pr\left[\mathcal{A}(D_2) = O\right].$$

[1]Note that although it is possible for *PrivSuper* to only report maximal frequent itemsets, it is difficult to measure the utility of a perturbed MFI, e.g., a noisy output that is slightly longer or shorter than a true MFI. For this reason, in this paper we follow the popular problem definition of top-k frequent itemset mining.

In the above definition, ε is called the *privacy budget*, which controls the strength of privacy protection. A smaller ε leads to stricter privacy protection, and vice versa.

Two fundamental mechanisms for enforcing ε-differential privacy are the *Laplace Mechanism* (LM) [1] and the *exponential mechanism* (EM) [16]. Both mechanisms involve the concept of *sensitivity*, defined as follows.

Definition 3: (Sensitivity [15]). Given a query Q with numerical outputs, the sensitivity of Q is $\Delta_Q = \max_{D_1,D_2} ||Q(D_1) - Q(D_2)||_1$, where D_1 and D_2 are two arbitrary neighboring databases and $||Q(D_1) - Q(D_2)||_1$ denotes the $\mathcal{L}_1$ distance between $Q(D_1)$ and $Q(D_2)$.

Given a numerical-valued query Q, LM obtains a differentially private version of Q's results, by injecting random noise into each of Q's output, where the noise follows zero-mean Laplace distribution with scale $\frac{\Delta_Q}{\varepsilon}$. EM, on the other hand, is often used to enforce ε-differential privacy on a query with a categorical output. Specifically, given the output domain Ω, EM requires a user-specified *quality function* q, such that given database D, for each value $\omega \in \Omega$, q outputs a real-valued score $q(\omega, D)$ that measures how desirable ω is to the user (larger scores are preferred). To ensure ε-differential privacy, EM samples ω from Ω with probability:

$$Pr\left[w\right] \propto \exp\left(\frac{\varepsilon}{2\Delta_q} q(\omega, D)\right)$$

where Δ_q denotes the sensitivity of the quality function q.

Note that if D_1 is transformed into its neighboring database D_2, the quality function values of all outcomes in Ω change only in one direction, i.e., all the values from D_1 keep no more (less) than the ones from D_2. In this case, we can remove "2" from the above formula.

An important property of differential privacy is *composition*, which means that several mechanisms can be combined in order to answer a complex query. There are two composition rules, *sequential composition* and *parallel composition*[11]:

Lemma 1: (Sequential Composition). Given a randomized algorithm $\mathcal{A}$ consisting of a sequence of procedures $\{\mathcal{A}_1, ..., \mathcal{A}_i, ..., \mathcal{A}_m\}$, if every procedure $\mathcal{A}_i$ ($1 \leq i \leq m$) satisfies ε_i-differential privacy, then $\mathcal{A}$ satisfies ε-differential privacy for $\varepsilon = \sum_i \varepsilon_i$.

Lemma 2: (Parallel Composition). Given a randomized algorithm $\mathcal{A}$ consisting of procedures $\{\mathcal{A}_1, ..., \mathcal{A}_i, ..., \mathcal{A}_m\}$, if every procedure $\mathcal{A}_i$ ($1 \leq i \leq m$) applies to a disjoint subset of records of the input database and at the same time satisfies ε_i-differential privacy, then $\mathcal{A}$ satisfies ε-differential privacy for $\varepsilon = \max_i \varepsilon_i$.

B. Top-k Frequent Itemset Mining under Differential Privacy

Frequent itemset mining aims to identify sets of items that appear in a large number of records (called *transactions*) in a given database D. Specifically, each record $t \in D$ consists of a set of items. Given an itemset s, we consider that s appears in t, if and only if s is a subset of t, i.e., $s \subseteq t$. The number

of transactions which s appears in is called the *support* of s. Formally, let θ_s denote the support of s, we have $\theta_s = |\{t|t \in D, s \subseteq t\}|$, where $|\cdot|$ denotes the cardinality of a set. Accordingly, the exact result of top-k frequent itemset mining consists of the k itemsets with the highest supports.

Objectives. Our goal is to release a randomized version of the top-k frequent itemsets with high utility to the user, under ε-differential privacy constraints. Here utility includes multiple aspects. Let S^* be the exact top-k frequent itemsets, and S be a randomized version of the result containing k itemsets. One aspect of utility is how similar S and S^* are, e.g., in terms of number of common itemsets. Another aspect is the quality of itemsets in S, e.g., in terms of their supports. For example, consider an extreme case that k = 1, $S = \{s\}$, $S^* = \{s^*\}$, and $s \neq s^*$, i.e., the released itemset s is not the most frequent one s^*. Here, S and S^* have no common element; yet, the support of s can be very close to that of s^*, which might be useful in certain applications. Finally, some solutions (e.g., [4]) also report perturbed support values for itemsets in S. In this situation, a third aspect of result utility is the accuracy of the published noisy supports, e.g., in terms of relative error.

Earlier solutions. Bhaskar et al. [17] propose two solutions that tackle the problem directly, one using LM and the other using EM (described in Section II-A). Specifically, their first solution applies LM to compute noisy supports of all possible itemsets, and then publishes the top-k itemsets with the highest noisy supports. The second solution assigns to each possible itemset a quality score which is equal to its support, and applies EM k times to sample k itemsets without replacement. The problem with these methods is that there are an exponential number of possible itemsets. To alleviate this issue, Bhaskar et al. [17] limit themselves to top-k size-i itemsets (called i-itemsets), where i is a system parameter. Still, since the search space grows exponentially to i, these solutions suffer from poor utility for large i.

Several solutions are based on the classic Apriori algorithm [14], which exploits the monotonicity of support: specifically, given an itemset s and one of its subsets $s' \subseteq s$, their (exact) supports θ_s and $\theta_{s'}$ satisfy that $\theta_s \leq \theta_{s'}$. Accordingly, Apriori starts by enumerating 1-itemsets, i.e., singleton sets containing one item each, and then combines them to form 2-itemsets, and then 3-itemsets, etc. If an itemset s is not frequent (i.e., its support is lower than the current k-th highest support), then all supersets of s are pruned.

Zeng et al. [18] applies Apriori directly to differentially private top-k itemset mining, with an optimization called *transaction truncating*. In particular, in the original problem setting, each transaction in the database can contain an arbitrary number of items. Consequently, in the worst case, adding or removing a transaction affects the supports of all itemsets, leading to high sensitivity. The method in [18] truncates the transactions so that after the truncations, each transaction contains no more than l items, thus limiting the sensitivity. Clearly, such truncation causes information loss. In particular, a small value of l causes high information loss, whereas a large l leads to high sensitivity, and, thus, a high noise level required to satisfy differential privacy. The best value of l, however, depends on the data and contains sensitive information. Thus, Zeng et al. [18] also propose a heuristic technique to set l.

States of the Art. To our knowledge, the state of the art for top-k frequent itemset mining remains to be *PrivBasis* [4]. In a nutshell, *PrivBasis* uses EM to perform the first two iterations of Apriori under differential privacy, which obtains a number of frequent 1-itemsets and 2-itemsets, respectively. Then, it uses these to generate a set of candidate itemsets, applies LM to compute a noisy support for each candidate, and reports the top-k candidates with the highest noisy supports.

Specifically, in the first (resp., second) iteration of Apriori, similar to [17], *PrivBasis* applies EM to sample without replacement k_1 most frequent items (resp., k_2 most frequent 2-itemsets). The values of k_1 and k_2 are discussed shortly. Then, *PrivBasis* constructs an undirected graph with k_1 nodes (each corresponding to a frequent item found in the first step) and k_2 edges (each corresponding to a frequent 2-itemset, connecting the two items therein). Based on the graph, the method computes maximal cliques in this graph. After that, *PrivBasis* adds all possible combinations of items in the clique to the candidate set. Finally, *PrivBasis* applies LM to obtain a noisy support for each candidate, and reports the top-k candidates.

It remains to clarify the values of k_1 and k_2. To obtain k_1, *PrivBasis* first computes the exact top-k frequent itemsets, as well as the exact support of the k-th itemset. Let τ denote this support value. Then, *PrivBasis* sorts all items in decreasing order of their supports, and applies EM to select k_1 with the objective of minimizing the difference between τ and the support of the k_1-th frequent item. Regarding k_2, *PrivBasis* resorts to a heuristic formula.

Observe that in *PrivBasis*, a clique of items does not necessarily correspond to a frequent itemset. In the worst case, only the items and item-pairs in a clique are frequent, and all larger itemsets combining these items have low support. Hence, *PrivBasis* often generates a far larger number of candidates than k, leading to high privacy budget consumption in the subsequent filtering step.

Other solutions. Xu et al. [19] study frequent sequence mining under differential privacy, and propose an Apriori-based solution. Note that this problem is different from frequent itemset mining, since items in a sequence are ordered. Nevertheless, the solution in [19] can be applied to our problem, and we include it in our experiments. Specifically, to identify frequent sequences of length i, they utilize a sample database to estimate the supports of candidate sequences, and prune infrequent ones. After that, the method uses the original database to filter the remaining candidate sequences. The use of a sample set reduces sensitivity, and, thus, the amount of injected noise. However, the sample data can be rather different from the full database, leading to erroneous filtering. As our experiments demonstrate, the utility of this solution is usually lower than that of *PrivBasis* for our problem.

Finally, Lee et al. propose *NoisyCut* [5], which contains a novel generic mechanism for enforcing differential privacy, which can be considered as a variant of the sparse vector technique [20]. Ref. [5] claims that this new mechanism consumes only $O(1)$ privacy budget for identifying an arbitrary number of itemsets whose supports are larger than a threshold, leading to low privacy budget usage for *NoisyCut* and, thus, high result utility. Unfortunately, this mechanism has been shown to

TABLE I: List of frequent notations

Notation	Meaning
D, t	the input database and a record in D
s, θ_s	an itemset and its exact support
k	the number of frequent itemsets to find
τ	the exact support of the true k-th most frequent itemset
$\tilde{\tau}$	a noisy version of τ obtained with LM
l	*PrivSuper*'s limit on the max. cardinality of a transaction
m	the max. cardinality of an itemset in *PrivSuper*'s results
F, λ	the set of frequent items in *PrivSuper* and its cardinliaty
B	the set of candidate maximal frequent itemsets in *PrivSuper*
C	the set of candidate frequent itemset sets in *PrivSuper*
Ω	max. number of times that SEM consumes privacy budget

violate differential privacy in [12], [13], and it is unclear how (and whether it is possible) to devise a replacement mechanism that has the same properties as the one in [5].

III. PrivSuper Framework

This section describes the general framework of the proposed solution *PrivSuper*, highlighting its novel algorithmic designs and leaving out several important details that are covered later in Sections IV and V. *PrivSuper* involves three major steps, namely *preprocessing*, *MFI search*, and *candidate filtering*. Given a total privacy budget ε, *PrivSuper* allocates to the three steps privacy budgets ε_1, ε_2 and ε_3 respectively, such that $\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_3$. According to Lemma 1, the algorithm as a whole satisfies ε-differential privacy. The choice of parameters ε_1-ε_3 are discussed further in Section V. In the following, Sections III-A, III-B and III-C describe these three steps, respectively. Section III-C additionally summarizes the complete *PrivSuper* algorithm, and establishes its privacy guarantees. Table I summarizes the frequent notations used in the paper.

A. Preprocessing

During preprocessing, *PrivSuper* computes the true top-k most frequent itemsets, which is used in multiple places of algorithms. Meanwhile, the preprocessing step also involves three operations that are performed under differential privacy; hence, each of them requires a portion of the privacy budget. For simplicity (and without a better idea), we allocate to each of them 1/3 of the total budget ε_1 for preprocessing. First, similar to Ref. [18] described in Section II-B *PrivSuper* truncates the transactions in the input data D to limit the maximum number of items in each transaction. In particular, given a maximum length l, *PrivSuper* scans the database; for each transaction $t \in D$ containing more than l items, the algorithm samples l items uniformly at random from t, and discards all remaining items in t. Similar to [18], *PrivSuper* determines an appropriate value for l based on the distribution of transaction lengths in D, under $\frac{\varepsilon_1}{3}$-differential privacy. Since this is not our main contribution, we defer the details until Section V-A.

Second, *PrivSuper* chooses an upper limit m, so that each itemset returned by *PrivSuper* contains no more than m items. Note that the maximum possible value for m is $\lfloor \log_2(k+1) \rfloor$. The reason is that an itemset s containing more than $\lfloor \log_2(k+1) \rfloor$ items has more than k subsets, whose supports are no smaller than that of s, due to the monotonicity of supports as explained in Section II-B. Assuming that there are more than k items in the database, the minimum possible value of m is 1, which corresponds to the extreme case that every frequent itemset contains exactly one item. *PrivSuper* employs EM to determine an appropriate value for m, using privacy budget $\frac{\varepsilon_1}{3}$. The main idea is that m should be close to the cardinality of the longest itemset among the exact top-k frequent results. Again, we leave the details in Section V-A.

Lastly, *PrivSuper* computes an estimate λ for the number of items involved in the exact top-k frequent itemsets, using privacy budget $\frac{\varepsilon_1}{3}$. Put it another way, λ estimates the cardinality of the union of the exact top-k frequent itemsets. We call items in this union set *frequent items*. The computation of m can be done using EM, similarly as in [4]. Details are provided in Section V-A.

Although we compute l, m and λ with small parts of privacy budget respectively, the derived values are still close to the true ones, due to small sensitivities of these computing algorithms. After preprocessing, we have truncated each transaction to no more than l items, and determined the maximum length m of a result itemset as well as the number of frequent items λ. We are now ready to search for maximal frequent itemsets, described in the next subsection.

B. Searching for Maximal Frequent Itemsets

To search for maximal frequent itemsets (MFIs), *PrivSuper* first determines the set of items that appear in MFIs, i.e., the set of λ frequent items described in Section III-A. Let F denote this set. After obtaining F, *PrivSuper* continues to search for a set of MFIs, denoted as B. The computations of both F and B require privacy budget. For simplicity, we allocate to each of them $\frac{\varepsilon_2}{2}$ for MFI search.

We first explain the computation of F. Note that this step also commonly appears in other solutions such as *PrivBasis* [4], and there has been existing solutions. Our specific implementation of this step is a new hybrid approach: in particular, *PrivSuper* compares the number of frequent items λ with the truncation length limit l. If $l < \lambda$, *PrivSuper* computes F with LM; otherwise, it does so with EM. We leave the details to Section V-B.

Next we focus on the computation of the MFI set B using F. *PrivSuper* searches for MFIs in a depth-first manner. One main difference between *PrivSuper* and standard, non-private depth-first search lies in the order of branches to follow at each node, which will become clear later as we explain the algorithm. We outline the main ideas of the search using the example shown in Figure 1, which contains 4 frequent items $F = \{1, 2, 3, 4\}$. A support threshold τ is also shown in the figure (as a curve), whose value is equal to the exact support of the true k-th most frequent itemset. Itemsets above this curve are frequent (i.e., with support no less than τ), and vice versa.

Suppose that the maximum number of items in an output itemset is $m = 4$. The search maintains a current itemset s, initialized to $s = \emptyset$. *PrivSuper* first selects one item in F to add to s, and updates s to a 1-itemset, e.g., $s = \{1\}$. Since each item in F is considered frequent, the corresponding 1-itemset is also frequent. So, the search goes on to add another

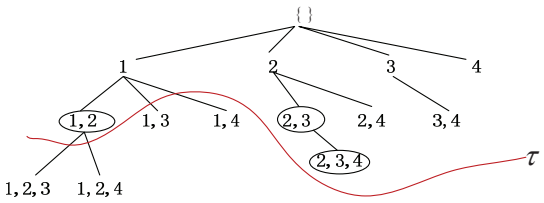


Fig. 1: Example of MFI search

item to s, say, item 2, leading to $s = \{1, 2\}$. At this point, the algorithm needs to determine whether the new s is frequent, i.e., whether its support θ_s is no less than the threshold τ. If so, the search continues to add more items to s; otherwise, this branch is pruned, due to monotonicity of supports (i.e., no superset of s can have a higher support). Note that this check must be done under differential privacy, since both θ_s and τ contain private information.

One way to check whether s is frequent is to apply LM to compute a perturbed version $\tilde{\theta}_s$ and $\tilde{\tau}$ of θ_s and τ, respectively, and compare $\tilde{\theta}_s$ against $\tilde{\tau}$. This method needs to compute a noisy support for every branch. In our example, it calls LM once for each of $\{1, 2\}$, $\{1, 3\}$ and $\{1, 4\}$. In general, when there are a large number of items, this approach incurs high privacy budget consumption, since each call of LM consumes a share of the privacy budget.

Another way is to apply EM to repeatedly select the branch with the highest support without replacement. To do this, we assign to each branch a quality score that is equal to the support of the resulting itemset. In our example, there are three branches $\{1, 2\}$, $\{1, 3\}$ and $\{1, 4\}$, each assigned its corresponding support as its quality score. Suppose that EM yields branch $\{1, 2\}$. *PrivSuper* then considers that $\{1, 2\}$ has a higher support than its siblings, i.e, $\{1, 3\}$ and $\{1, 4\}$. To check whether $\{1, 2\}$ is frequent, we can apply LM to obtain a noisy version of its support, and check if it is no smaller than the noisy threshold $\tilde{\tau}$. In our example, suppose that item $\{1, 2\}$ passes this test, and *PrivSuper* continues the search with this itemset by adding more items. Once it finishes this branch and backtracks to node $\{1\}$, *PrivSuper* again applies EM to select a branch among the remaining two, corresponding to $\{1, 3\}$ and $\{1, 4\}$, respectively. Suppose that this time EM returns itemset $\{1, 3\}$, whose noisy support obtained with LM is lower than the threshold $\tilde{\tau}$. Then, *PrivSuper* immediately backtracks to the root, without considering the remaining itemset $\{1, 4\}$. The reason is that the itemset selected by EM is considered the one with the highest support. Thus, any itemset not selected is considered to have a lower support, and, thus, is not expected to pass the threshold.

The above approach does not need to test all branches against the threshold. On the other hand, for each branch tested it needs to call first EM and then LM, consuming two shares of the privacy budget. We observe that it is actually sufficient to use one invocation of EM to both select the itemset with the highest support, and test its support against τ. The trick to do so is that when selecting the highest-support itemset with EM, we also consider a *dummy itemset* $\perp$, whose quality score is fixed to τ. If EM returns a non-dummy itemset, then its support is expected to pass the threshold. Otherwise, i.e., EM returns the dummy, then all other itemsets are expected to have lower support values than τ; thus, we can discard them and backtrack. *PrivSuper* follows this idea. However, instead of using EM, *PrivSuper* employs a customized mechanism SEM, which does not consume any privacy budget at all when it turns out that the selected item is the dummy. In Figure 1, SEM only consumes privacy budget at the circled itemsets. We elaborate on SEM in Section IV.

Besides employing SEM, another trick that *PrivSuper* uses to reduce privacy budget consumption is to exploit the monotonicity of supports. Continuing the example, after the search backtracks to the root, *PrivSuper* then searches along the path $\{\} \rightarrow \{2\} \rightarrow \{2, 3\} \rightarrow \{2, 3, 4\}$. The last itemset $\{2, 3, 4\}$ is an MFI, since there is no other item to add to it [2]. Therefore, we do not need to test itemsets $\{2, 4\}$ and $\{3, 4\}$, since they are both subsets of MFI $\{2, 3, 4\}$, and, thus, have higher supports than the threshold τ. To enable this optimization, during the search we maintain a set of frequent itemsets C, which we call the *candidate set*. For each MFI identified, *PrivSuper* adds all its subsets to C (if they are not already present in C). Then, when encountering a branch corresponding to an itemset in C (e.g., $\{2, 4\}$ in our example), *PrivSuper* directly descends the search tree along this branch, without calling SEM.

An important parameter to estimate in the MFI search is the number of times Ω that SEM consumes privacy budget, which is needed to determine the share of the privacy budget allocated to each invocation of SEM, explained further in Section IV. Let Ω denote this value. Clearly, $\Omega < k$, since each time SEM consumes privacy budget, at least one new frequent itemset is found. On the other hand, setting Ω to k is overly conservative, since one MFI often leads to multiple frequent itemsets. The estimation of Ω is clarified later in Section V-B. Finally, there is a special case where $m = 2$, i.e., each result itemset contains no more than two items. When this happens, *PrivSuper* does not perform depth-first search, and uses a different method instead, described in Section V-B.

Algorithm 1 summarizes the MFI search procedure of *PrivSuper*, which we call *PrivSuperDFS*. Specifically, given the current itemset s and a set F' of available items that can be used to extend s, *PrivSuperDFS* identifies an item $x \in F'$ such that $s \cup \{x\}$ is frequent. Specifically, the algorithm finds x when (i) there exists such an x satisfying that $s \cup \{x\} \in C$, i.e., it is a subset of a previously found MFI or (ii) SEM returns such an item x. Then, *PrivSuperDFS* recursively calls itself to continue the search with $s \cup \{x\}$. Otherwise, the algorithm backtracks.

Note that after the MFI search finishes, the candidate set C often contains more than k itemsets. Hence, *PrivSuper* applies an additional candidate filtering step, explained later.

C. Candidate Filtering and Complete Algorithm

Once *PrivSuper* finishes the MFI search and obtains the set of candidate itemsets C, it checks whether C contains more than k itemsets. If so, the algorithm continues to select the k itemsets from C; otherwise, it simply outputs all itemsets in C. In the former case, *PrivSuper* computes a randomized version of the support for each candidate under differential privacy,

[2]Note that item 1 cannot be added to this set, since all itemsets containing item 1 have already been explored in another branch.

Algorithm 1: *PrivSuperDFS* algorithm for MFI search

Input: Current itemset s, database D, set of available items F', MFI set B, candidate set C, maximum length m for s, remaining privacy budget ε_r

```
1  while F' ≠ ∅ do
2      if there exists x ∈ F' such that s ∪ {x} ∈ C then
3          Set x to any such item;
4      else
5          x = SEM (s, D, F', ε_r);
6          // Note that SEM internally updates ε_r;
7      if x is not NULL then
8          Add to C all subsets of s ∪ {x} that are not in C;
9          Remove x from F';
10         if s ∪ {x} contains m items then
11             Add s ∪ {x} to B;
12         else
13             Call PrivSuperDFS(s ∪ {x}, D, F', B, C, m,
                ε_r);
14     else
15         Set F' to ∅;
16 if no superset of s is in B then
17     Add s to B;
```

Algorithm 2: *PrivSuper* framework

Input : Database D, privacy budget ε, k
Output: Top-k frequent itemsets

```
1  Split the privacy budget ε into three portions ε_1, ε_2 and ε_3;
2  Compute the true top-k frequent itemsets, and the exact
     support τ for the k-th most frequent itemset;
3  Choose an appropriate value for l using privacy budget ε_1/3;
4  Truncate each transaction t ∈ D to up to l items;
5  Choose an appropriate value for m using privacy budget ε_1/3;
6  Choose an appropriate value for λ using privacy budget ε_1/3;
7  Compute the set F of frequent items using privacy budget
     ε_2/2;
8  Initialize MFI set B and candidate set C to ∅;
9  Add to C all 1-itemsets, each obtained with an item in F;
10 Initialize itemset s to ∅;
11 Estimate (with no privacy budget consumption) the number of
     times Ω that SEM consumes privacy budget;
12 Initialize remaining privacy budget ε_r for MFI search to ε_2/2;
13 Initialize SEM with ε_r, τ and Ω; // explained in Section IV;
14 Call PrivSuperDFS(s, D, F, B, C, m, ε_r);
15 Compute perturbed support for each candidate in C using
     privacy budget ε_3;
16 Return the top-k itemsets in C with the highest supports;
```

and reports top-k candidates with the highest noisy supports. Note that such filtering also exists in other solutions, e.g., in [4]. Our implementation is an optimized version of the method in [4], and we present its details in Section V-C.

Algorithm 2 summarizes the complete *PrivSuper* framework. Lines 2-6 perform preprocessing, Lines 7-14 perform MFI search, and Line 15 performs candidate filtering. The algorithm calls the *PrivSuperDFS* procedure for MFI search, which is listed in Algorithm 1. Note that the *PrivSuper* framework leaves out several details for parameter selection (e.g., for ε_1-ε_3 l, m, λ, Ω), which are implemented in Section V, as well as details specific to SEM (e.g., its initialization and privacy budget maintenance), which are presented in Section IV. Lastly, we observe that if MFI search does not use up its allocated privacy budget, i.e., $\varepsilon_r > 0$ after the search completes, then the remaining budget ε_r can be merged to the budget ε_3 for candidate filtering.

The following lemma establishes the correctness of *PrivSuper*. The proof follows directly from the composition rules presented in Section II-A, and are omitted for brevity.

Lemma 3 (Correctness of PrivSuper): Suppose that all components of *PrivSuper*, i.e., the computations of l, m, λ, F, as well as MFI search and candidate filtering, satisfy differential privacy with their respectively privacy budgets. Then, *PrivSuper* (Algorithm 2) satisfies ε-differential privacy.

Next we present SEM, the core component of *PrivSuper*.

IV. Sequence Exponential Mechanism

As described in Section III-B, given a number of itemsets and a dummy itemset (denoted as $\perp$) whose support is equal to the threshold τ, SEM aims to select the itemset with the highest support. Meanwhile, when SEM returns $\perp$, it does not consume any privacy budget; otherwise, it consumes the same amount of privacy budget as EM. The key idea of SEM is simple: instead of assigning the exact τ to the dummy item $\perp$, SEM assigns to it a noisy version $\tilde{\tau}$ of τ, which is obtained using LM in an initialization step. Note that although $\tilde{\tau}$ is computed with LM under differential privacy, it is important that the value of $\tilde{\tau}$ is never released, in order to satisfy the property that when SEM outputs $\perp$, it does not require any privacy budget. This is because the randomness in unpublished $\tilde{\tau}$ is used to protect the privacy information involved in the processes of SEM outputting $\perp$s.

Algorithms 3 and 4 summarize SEM. Note that the initialization step (Algorithm 3) also requires a privacy budget. We thus split the total budget for MFI search into two portions, one for *Init_SEM* and the other for all invocations of *SEM*. In our implementation of *PrivSuper*, the total budget for MFI search is $\frac{\varepsilon_2}{2}$. We reserve 1/4 of this budget to *Init_SEM* (i.e., $\frac{\varepsilon_2}{8}$), and the rest (i.e., $\frac{3\varepsilon_2}{8}$) to *SEM*. Since the number of times that SEM consumes privacy is Ω, we allocate privacy budget $\varepsilon^* = \frac{3\varepsilon_2}{8\Omega}$ to each invocation of *SEM*. In Algorithm 4, SEM firstly checks if the privacy budget ε_r is used up. If so, i.e., $\varepsilon_r = 0$, SEM returns $\perp$, since it cannot pay privacy budget to choose an item to extend the current itemset s. Otherwise, with privacy budget ε^*, SEM utilizes EM to chooses one item from the set consisting of s's extensible items and the dummy item $\perp$. Note that in SEM, privacy budget ε^* is consumed only when EM returns one non-dummy item. As a result, SEM outputs Ω non-dummy items at most.

Unlike the *NoisyCut* method [5], which derives the frequent itemsets by comparing one one noisy version $\tilde{\tau}$ with noisy version supports of itemsets, SEM uses exponential mechanism coupled with one noisy version $\tilde{\tau}$ to choose the frequent itemset with high support.

Next we establish the correctness of SEM in the following theorem. The proof of the theorem is highly non-trivial, which resembles that of the sparse vector technique [20].

Algorithm 3: Algorithm *Init_SEM* for initializing SEM

Input: total privacy budget ε for *Init_SEM* and all invocations of *SEM*, support threshold τ

Choose positive ε_{init} and ε_{SEM} with $\varepsilon_{init} + \varepsilon_{SEM} = \varepsilon$;
Compute $\tilde{\tau}$ with LM using privacy budget ε_{init};
Set $\varepsilon^* = \frac{\varepsilon_{SEM}}{\Omega}$;
Set the remaing privacy budget ε_r with ε_{SEM};

Theorem 1 (Correctness of SEM): SEM satisfies ε-differential privacy. Here, SEM involves one invocation of Algorithm 3 and multiple times invocations of Algorithm 4, while Algorithm 4 outputs a non-dummy item Ω times at most.

Proof: Consider any two neighboring datasets D and D'. Suppose that given D (resp. D'), Algorithm 3 returns a noisy threshold $\tilde{\tau}$ (resp. $\tilde{\tau}'$). Assume that Algorithm 4 is invoked n times. Obviously, there are at most Ω times that Algorithm 4 outputs one non-dummy item, due to the framework of it. And the input to the i-th invocation ($i \in [1, n]$) of Algorithm 4 contains an itemset s_i, and a set of items F_i'. Further assume that the output of the i-th invocation is o_i. For convenience, we define functions $f_i, f_i' : F' \cup \{\perp\} \to \mathbb{Z}$ as

$$f_i(\alpha) = \begin{cases} \text{the support of } s_i \cup \{\alpha\} \text{ in } D, & \text{if } \alpha \in F' \\ \tilde{\tau}, & \text{otherwise} \end{cases}$$

$$f_i'(\alpha) = \begin{cases} \text{the support of } s_i \cup \{\alpha\} \text{ in } D', & \text{if } \alpha \in F' \\ \tilde{\tau}', & \text{otherwise} \end{cases}$$

Let $p(o_i \mid D, \tilde{\tau})$ be the probability that the i-th invocation of Algorithm 4 returns o_i given D and $\tilde{\tau}$. Then, the total privacy cost of SEM is:

$$\max_{D,D'} \log \left(\frac{\int_{-\infty}^{+\infty} \Pr[\tilde{\tau} = x] \prod_i p(o_i \mid D, \tilde{\tau}) \mathbf{d}x}{\int_{-\infty}^{+\infty} \Pr[\tilde{\tau}' = x] \prod_i p(o_i \mid D', \tilde{\tau}') \mathbf{d}x} \right). \quad (1)$$

Meanwhile, since the output of Algorithm 4 is either an item or $\perp$, we have:

$$\frac{\int_{-\infty}^{+\infty} \Pr[\tilde{\tau} = x] \prod_i p(o_i \mid D, \tilde{\tau}) \mathbf{d}x}{\int_{-\infty}^{+\infty} \Pr[\tilde{\tau}' = x] \prod_i p(o_i \mid D', \tilde{\tau}') \mathbf{d}x} = \frac{\int_{-\infty}^{+\infty} \Pr[\tilde{\tau} = x] \prod\limits_{i:o_i \neq \perp} p(o_i \mid D, x) \prod\limits_{i:o_i = \perp} p(o_i \mid D, x) \mathbf{d}x}{\int_{-\infty}^{+\infty} \Pr[\tilde{\tau}' = x] \prod\limits_{i:o_i \neq \perp} p(o_i \mid D', x) \prod\limits_{i:o_i = \perp} p(o_i \mid D', x) \mathbf{d}x} \quad (2)$$

To prove the theorem, it suffices to show that the r.h.s. of Equation 2 is no more than $\exp(\varepsilon)$.

Let $\varepsilon^* = \varepsilon_{SEM}/\Omega$. In what follows, we focus the case when $|D'| = |D| + 1$, where $f_i'(a) = f_i(\alpha) + 1$ or $f_i'(\alpha) = f_i(\alpha)$ for any $\alpha \in F'$; the case when $|D| = |D'| + 1$ can be proved in a similar manner. Firstly, considering the invocations of Algorithm 4 which return non dummy items, we have:

Algorithm 4: Algorithm for executing SEM

Input: the remaining privacy budget ε_r for invocations of SEM, set of items F', current itemset s, database D, noisy support threshold $\tilde{\tau}$, privacy budget ε^* for each invocation of SEM

if $\varepsilon_r = 0$ **then**
  Return $\perp$;
Set quality score of dummy item $\perp$ to $\tilde{\tau}$;
Set quality score of each item $x \in F'$ to the support of $s \cup \{x\}$;
Use EM to select an item from $F' \cup \{\perp\}$ with ε^*;
if *EM returns a non-dummy item* **then**
  Deduct the total budget ε_r by ε^*;
Return the output of EM;

$$\begin{aligned} \prod_{i:o_i \neq \perp} p(o_i \mid D, x) &= \prod_{i:o_i \neq \perp} \frac{\exp(\varepsilon^* f_i(o_i))}{\sum_{\alpha \in F_i' \cup \{\perp\}} \exp(\varepsilon^* f_i(\alpha))} \\ &= \prod_{i:o_i \neq \perp} \frac{\exp(\varepsilon^* f_i(o_i))}{\exp(\varepsilon^* x) + \sum_{\alpha \in F_i'} \exp(\varepsilon^* f_i(\alpha))} \\ &= \prod_{i:o_i \neq \perp} \frac{\exp(\varepsilon^* (f_i(o_i) + 1))}{\exp(\varepsilon^* (x+1)) + \sum_{\alpha \in F_i'} \exp(\varepsilon^* (f_i(\alpha) + 1))} \end{aligned} \quad (3)$$

Since $f_i'(\alpha) \leq f_i(\alpha) + 1$, the loose upper bound of $\prod_{i:o_i \neq \perp} p(o_i \mid D, x)$ is shown as follows:

$$\begin{aligned} \prod_{i:o_i \neq \perp} p(o_i \mid D, x) &\leq \prod_{i:o_i \neq \perp} \frac{\exp(\varepsilon^*) \exp(\varepsilon^* f_i'(o_i))}{\exp(\varepsilon^*(x+1)) + \sum_{\alpha \in F_i'} \exp(\varepsilon^* f_i'(\alpha))} \\ &= \prod_{i:o_i \neq \perp} \exp(\varepsilon^*) p'(o_i \mid D, x+1). \end{aligned} \quad (4)$$

In addition, the loose upper bound of $\prod_{i:o_i = \perp} p(o_i \mid D, x)$ can be derived in the similar way as that of $\prod_{i:o_i \neq \perp} p(o_i \mid D, x)$:

$$\begin{aligned} \prod_{i:o_i = \perp} p(o_i \mid D, x) &= \prod_{i:o_i = \perp} \frac{\exp(\varepsilon^* f_i(o_i))}{\sum_{\alpha \in F_i' \cup \{\perp\}} \exp(\varepsilon^* f_i(\alpha))} \\ &= \prod_{i:o_i = \perp} \frac{\exp(\varepsilon^* x)}{\exp(\varepsilon^* x) + \sum_{\alpha \in F_i'} \exp(\varepsilon^* f_i(\alpha))} \\ &= \prod_{i:o_i = \perp} \frac{\exp(\varepsilon^* (x+1))}{\exp(\varepsilon^* (x+1)) + \sum_{\alpha \in F_i'} \exp(\varepsilon^* (f_i(\alpha) + 1))} \\ &\leq \prod_{i:o_i = \perp} \frac{\exp(\varepsilon^* (x+1))}{\exp(\varepsilon^* (x+1)) + \sum_{\alpha \in F_i'} \exp(\varepsilon^* f_i'(\alpha))} \\ &= \prod_{i:o_i = \perp} p'(o_i \mid D, x+1) \end{aligned} \quad (5)$$

Furthermore, due to the property of LM, for any x, we have

$$\frac{\Pr[\tilde{\tau} = x]}{\Pr[\tilde{\tau}' = x+1]} \leq \exp(\varepsilon_{init}). \quad (6)$$

Combining Equations 4, 5, and 6, we have

r.h.s. of Eqn. 2

$$= \frac{\int_{-\infty}^{+\infty} \Pr[\tilde{\tau} = x] \prod_{i:o_i \neq \perp} p(o_i \mid D, x) \prod_{i:o_i = \perp} p(o_i \mid D, x) \mathbf{d}x}{\int_{-\infty}^{+\infty} \Pr[\tilde{\tau}' = x+1] \prod_{i:o_i \neq \perp} p(o_i \mid D', \tilde{\tau}') \prod_{i:o_i = \perp} p(o_i \mid D', \tilde{\tau}') \mathbf{d}x}$$

$$\leq \exp(\varepsilon_{init}) \cdot \prod_{i:o_i \neq \perp} \exp(\varepsilon^*)$$

$$\leq \exp(\varepsilon_{init}) + \exp(\Omega \cdot \varepsilon^*)$$

$$= \exp(\varepsilon_{init} + \varepsilon_{SEM}) = \exp(\varepsilon).$$

Thus, the theorem is proved. ■

V. Implementation of PrivSuper

This section clarifies important implementation details in *PrivSuper*. First we clarify the privacy budget allocation for the three main steps: preprocessing (ε_1), MFI search (ε_2) and candidate filtering (ε_3). In particular, we choose $\varepsilon_1 = 0.15\varepsilon$, $\varepsilon_2 = 0.5\varepsilon$ and $\varepsilon_3 = 0.35\varepsilon$. The idea is to allocate half of the budget to candidate generation (similar to *PrivBasis* [4]), and the other half to preprocessing and filtering. Since the filtering step is much more complex than preprocessing, the former is assigned a larger portion of the budget.

A. Implementation of Preprocessing

Computation of l. Let $\{z_1, ..., z_{|I|}\}$ be a length statistics vector where I is the set of all items in the database D, and z_i denotes the number of transactions with length i in the input database D. The sensitivity of this vector as a whole is 1, since adding or removing a transaction can only affect one of $\{z_1, ..., z_{|I|}\}$ by at most 1. Thus, similar to [18], we inject random noise following the geometric distribution with scale $\frac{3}{\varepsilon_1}$ to each of them, which satisfies $\frac{\varepsilon_1}{3}$-differential privacy [21]. Then, the truncating length l is set to the value such that the percentage of the transactions with length no greater than l is at least 85%.

Computation of m. Let vector $\mathbf{y}(D)$ be $\{y_1(D), ..., y_{\lfloor \log_2(k+1) \rfloor}(D)\}$, where $y_i(D)$ denotes the maximum support of i-itemsets. To satisfy $\frac{\varepsilon_1}{3}$-DP, we apply EM to sample i from $[1, \lfloor \log_2(k+1) \rfloor]$ with probability proportional to $e^{-\frac{\varepsilon_1 |y_i(D) - \tau|}{3 \times 2}}$. The result is used as m.

Computation of λ. To satisfy $\frac{\varepsilon_1}{3}$-DP, we apply EM to sample i from $[1, |I|]$ with probability $\propto e^{\frac{-\varepsilon_1 |x(D,i) - \tau|}{3 \times 2}}$, where $x(D, i)$ denotes the support of the ith most frequent item. (Note that the sensitivity of $x(D, i)$ is 1.) The result is used as λ.

B. Implementation of MFI Search

Computation of F. First we compare the truncation length l with the number of frequent items λ. If $l < \lambda$, the supports of the truncated database have relatively low sensitivity, i.e., l. Hence, *PrivSuper* applies LM and adds Laplace noise with scale $\frac{2l}{\varepsilon_2}$ to the support of each item, and then selects the top-λ items based on their noisy supports. Otherwise, i.e., $l \geq \lambda$, our algorithm applies EM to sample λ times without replacement from the set of all items I. Specifically, each item x is sampled with probability proportional to $e^{\frac{\varepsilon_2 \theta_x}{2\lambda}}$, where θ_x denotes the support of x in database D. Note that the support of an item has sensitivity 1. In both cases the computation of F satisfies $\frac{\varepsilon_2}{2}$-differential privacy, due to the use of LM / EM.

Computation of Ω. Ω is the estimated total number of times that SEM outputs non-dummy items throughout the MFI search. Our implementation chooses Ω according to Proposition 1, as follows.

Proposition 1: The number of times that SEM outputs non-dummy items is estimated by $\frac{u_1 + u_2 + u_3}{3}$, where u_1, u_2 and u_3 are as follows: $u_1 = m + \left\lceil \frac{k - [2^m - 1]}{2^{m-1} - 1} \right\rceil$, $u_2 = m + k - (2^m - 1)$, $u_3 = m + \left\lceil \frac{k - [2^m - 1]}{2^m - 1} \right\rceil m$.

Here, u_1, u_2 and u_3 are values of Ω used in three representative cases. We use the mean of them to estimate Ω. Intuitively, we expect in the results at least one MFI with cardinality m. This MFI can generate $2^m - 1$ frequent itemsets and m non-dummy items. We arrange the remaining $k - (2^m - 1)$ frequent itemsets in three different ways:

u_1: Adding one non-dummy item can generate at most $2^{m-1} - 1$ new frequent itemsets . This happens in the scenario that adding the non-dummy item generates another MFI which shares in common $(m-1)$ items with the existing MFIs. In this scenario, $k - (2^m - 1)$ frequent itemsets can generate additional $\frac{k - (2^m - 1)}{2^{m-1} - 1}$ non-dummy items.

u_2: If every itemset in the remaining frequent itemsets is associated with one non-dummy item, we can get extra $k - (2^m - 1)$ non-dummy items.

u_3: All the $k - (2^m - 1)$ frequent itemsets can be derived from MFIs with cardinality m, and there are no crossovers among these MFIs. Then, for $k - (2^m - 1)$ frequent itemsets, there are at most $\left\lceil \frac{k - (2^m - 1)}{2^m - 1} \right\rceil$ MFIs and each one generates m non-dummy items. So this case generates additional $\left\lceil \frac{k - (2^m - 1)}{2^m - 1} \right\rceil m$ non-dummy items.

Special Case with $m = 2$. If m is 2, then searching for MFIs does not have the benefit that subsets of an MFI can be directly added to the candidates. This is because the only possible subsets are 1-itemsets, which are known to be frequent as we only consider frequent items in the search. Hence, in this situation *PrivSuper* uses a different algorithm. Specifically, in this scenario a truncated database with truncation length l has sensitivity C_l^2 (i.e., the number of combinations for choosing 2 out of l items), if $C_l^2 < k - \lambda$. Hence, we add Laplace noise to the supports of all candidate 2-itemsets consisting of frequent items in F, and then choose the top-$(k - \lambda)$ frequent 2-itemsets. Otherwise, we apply EM $(k - \lambda)$ times to sample $(k - \lambda)$ frequent 2-itemsets without replacement from all the candidate 2-itemsets in D.

C. Implementation of Candidate Filtering

To our knowledge, the current state-of-the art technique for obtaining a noisy support of each candidate itemset is the bottom-up approach used in *PrivBasis*[4]. In a nutshell, *PrivBasis* first adds noise to statistics with low sensitivity, and then computes the noisy supports, in order to improve the accuracy. A key component before running the bottom-up

approach is combining cliques (i.e., MFIs) for further error reduction. Here, we follow the main idea in *PrivBasis* to obtain noisy supports, with modifications to the algorithm for combining itemsets. Our modifications are motivated by the fact that *PrivBasis* only considers the errors of candidate frequent 1-itemsets and 2-itemsets, whereas in *PrivSuper*, candidates are generated from all sub-itemsets (i.e., itemset with any length) of MFIs. In the following, we first outline the algorithm in *PrivBasis*, and then describe our novel itemset combining module.

Let L be a set of maximal cliques in the graph constructed from candidate frequent 1-teimsets and 2-itemsets (see Section II). For any element $L_i \in L$, *PrivBasis* scatters transactions in the database into $2^{|L_i|} - 1$ disjoint bins, each of which corresponds to one sub-itemset X of L_i. The bin associated with X includes all transactions that contain items in X, and no item in $L_i \backslash X$. *PrivBasis* then applies LM and injects Laplace noise of scale $\frac{|L|}{\varepsilon_3}$ to the count of transactions in each bin, and outputs these noisy counts. This satisfies ε_3-differential privacy, since the bins are disjoint, meaning that the overall sensitivity of the counts of transactions in all bins is 1. Then the noisy support of X is computed by summing up the counts of the bins which correspond to the super-itemsets of X in L_i, in a bottom-up manner. The number of bins corresponding to the super-itemsets of X is then $2^{|L_i|-|X|}$. It follows that the variance of X's noisy support $V_i(L, X)$ is $\frac{2|L|^2}{\varepsilon_3^2} 2^{(|L_i|-|X|)}$. Clearly, if itemset X consists of the common items between L_i and L_j, there are two noisy supports of X. In this situation, *PrivBasis* uses their weighted average as the noisy support, i.e., $\sum_{i=0 \wedge X \in L_i}^{|L|} \frac{cf(L,X)}{V_i(L,X)} f_i(L, X)$, where $f_i(L, X)$ denotes the noisy support from clique L_i and $cf(L, X) = \frac{1}{\sum_{i=0 \wedge X \in L_i}^{|L|} \frac{1}{V_i(L,X)}}$. The error variance of the generated support is then $cf(L, X) \frac{2|L|^2}{(\varepsilon_3)^2}$.

Before performing the above bottom-up algorithm for every clique, *PrivBasis* applies a heuristic to combine some cliques to minimize the total error of all candidate frequent 1-itemsets C_1 and 2-itemsets C_2, i.e., $\frac{2|L|^2}{\varepsilon_3^2} \sum_{X \in C_1 \cup C_2} cf(L, X)$. Specifically, this heuristic iteratively chooses one pair of L_i and L_j, and then replaces them in L with their combination L_k. A pair of L_i and L_j is chosen only if their combination can decrease the error more than any other pairs. The iterative computation is terminated when the error cannot be further decreased.

Next we clarify our modification to the itemset combining module. In our scenario, we regard each MFI in the MFI set B as one clique and follow the bottom-up method of *PrivBasis* to publish the noisy supports. In our strategy for combining maximal frequent itemsets, we aim to minimize the total error of noisy supports of all candidate frequent itemsets in C, instead of candidate frequent 1-itemsets C_1 and 2-itemsets C_2 only. In particular, our objective is to minimize $\frac{2|B|^2}{\varepsilon_3^2} \sum_{X \in C} cf(B, X)$. Meanwhile, in the heuristic method of *PrivBasis*, it is inefficient to re-compute $cf(B, X)$ for each itemset X in C under different combination strategies B, since the size of C is much larger than that of $C_1 \cup C_2$. Fortunately, $cf(B, X)$ satisfies the decomposable property shown in Lemma 4, and we utilize this property to simplify the computation of $cf(B, X)$.

TABLE II: Datasets

dataset	$\|D\|$	$\|I\|$	$\max_{t \in D} \|t\|$	$\frac{1}{\|D\|} \sum_{t \in D} \|t\|$
Mushroom [22]	8,124	119	23	23.0
Pumsb [22]	49,046	2,088	63	50.5
BMS-POS [23]	515,597	1,657	164	6.5
Retail [22]	88,162	16,470	76	10.3
Tafeng [24]	32,266	2,012	405	17.7

Lemma 4: Let M_1 and M_2 be two subsets of B. If $M_1 \cap M_2 = \varnothing$ and $M_1 \cup M_2 = B$, then $cf(B, X) = \frac{cf(M_1,X) cf(M_2,X)}{cf(M_1,X)+cf(M_2,X)}$.

According to Lemma 4, if MFIs B_i and B_j from the original MFI set B are combined into B_k to get a new MFI set $B^{'}$, then $cf(B^{'}, X)$ of itemset X can be updated incrementally, i.e., $cf(B^{'}, X) = (\frac{1}{cf(B,X)} + \frac{1}{cf(\{B_k\},X)} - \frac{1}{cf(\{B_i,B_j\},X)})^{-1}$.

Note that the above combination procedure only involves the MFIs in B and the candidate frequent itemsets in C. In other words, it does not involve any private information.

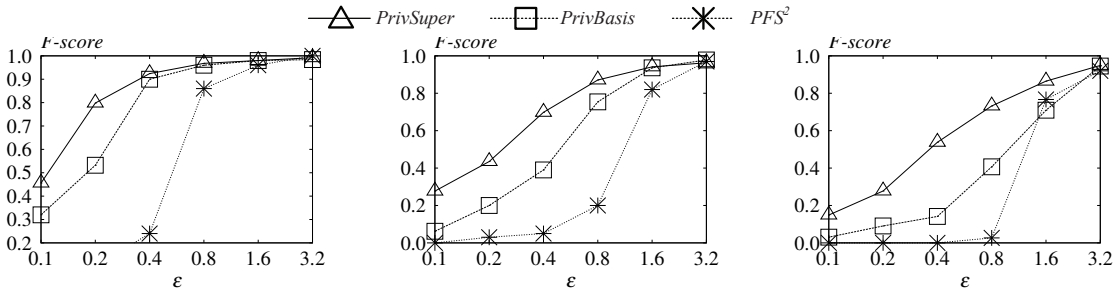
VI. EXPERIMENTS

The section evaluates *PrivSuper* against two state-of-the-art techniques for frequent itemset mining under differential privacy, *PrivBasis* [4] and *PFS*^{2} [19]. (We do not consider other existing solutions [17], [18] since they are shown in [4], [19] to be inferior to either *PrivBasis* or *PFS*^{2}).

We use five real datasets, as shown in Table II. We apply all techniques to generate top-k frequent itemsets from each dataset and, following previous work [18], [19], we assess the performance of each technique based on its *F-score* and *average relative error*. In particular, the *F-score* is defined as $\frac{1}{k}|R \cap \hat{R}|$, where $\hat{R}$ (resp. R) denotes the published (resp. actual) top-k frequent itemsets. A larger *F-score* indicates a more accurate set of top-k results. Meanwhile, the *average relative error* is defined as $\frac{1}{k} \sum_{x \in \hat{R}} \frac{|\hat{f}(x) - f(x)|}{\max\{f(x), 0.5\% \cdot |D|\}}$, where $\hat{f}(x)$ ($f(x)$) denotes the noisy (actual) support of itemset x. A smaller *average relative error* indicates that the noisy supports returned by a technique are closer to the actual supports.

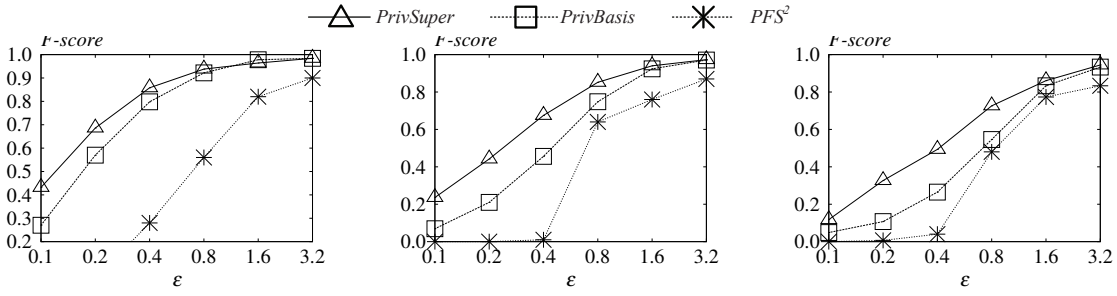
A. F-Scores of Top-k Frequent Itemsets

In the first set of experiments, we evaluate the *F-score* of the top-k frequent itemsets published by each technique. Figures 2a, 2b, and 2c show the results on the Retail dataset when k equals 50, 100, and 150, respectively. Observe that *PrivSuper* considerably outperforms *PrivBasis* and *PFS*^{2}, especially when ε is small or when k is large. In particular, when $k = 150$ and $\varepsilon = 0.4$, the *F-score* of *PrivSuper* is larger than those of *PrivBasis* and *PFS*^{2} by more than 0.4. The superior performance of *PrivSuper* is due to its superset-first search strategy and its usage of SEM, both of which reduce the privacy cost of searching for frequent itemsets. On the other hand, *PFS*^{2} yields a much lower *F-score* than *PrivBasis*, which indicates that its sample-based filtering approach is not as effective as the heuristics adopted in *PrivBasis* for identifying frequent itemsets. When ε is fixed, the *F-score*s of all methods decrease with the increase of k, since a larger k and a constant ε allow a smaller privacy budget on each frequent itemset,



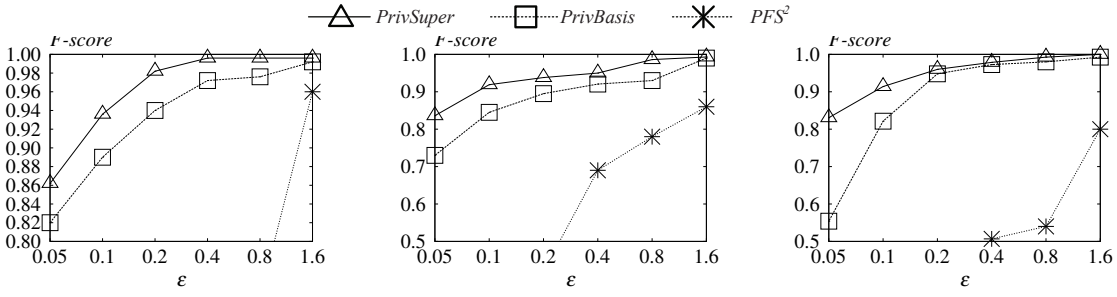
(a) $k = 50$ (b) $k = 100$ (c) $k = 150$

Fig. 2: F-scores on the Retail dataset.



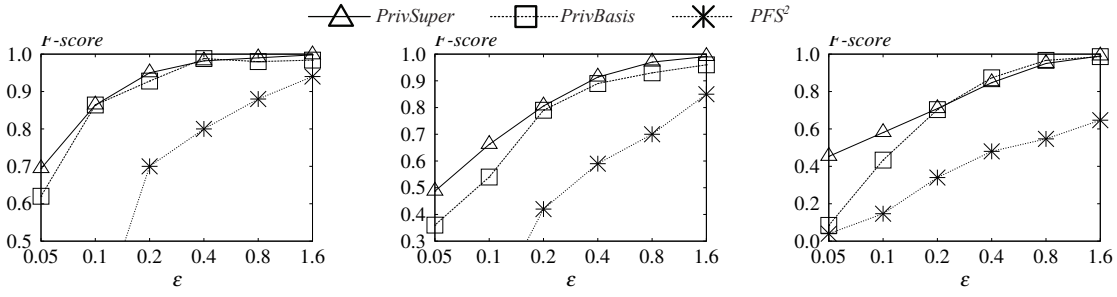
(a) $k = 50$ (b) $k = 100$ (c) $k = 150$

Fig. 3: F-scores on the Tafeng dataset.



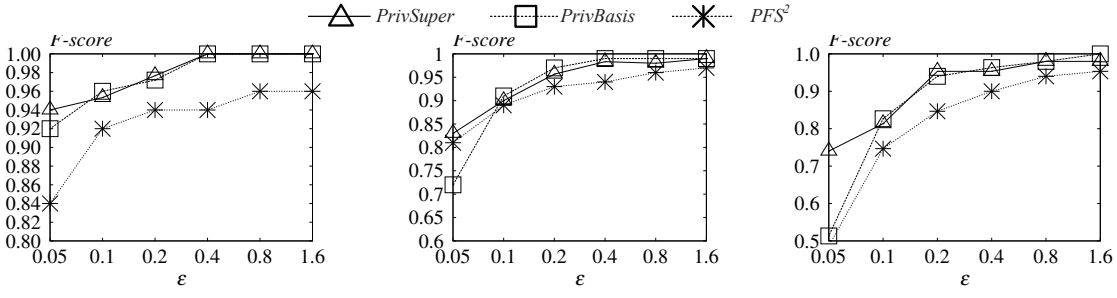
(a) $k = 50$ (b) $k = 100$ (c) $k = 150$

Fig. 4: F-scores on the Pumsb dataset.



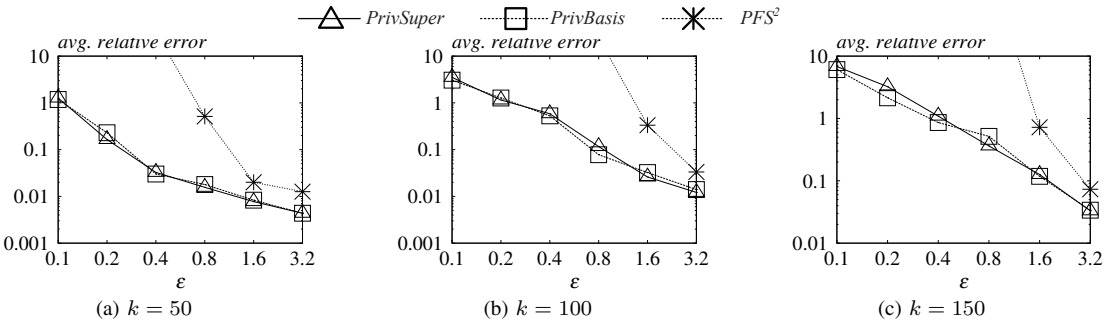
(a) $k = 50$ (b) $k = 100$ (c) $k = 150$

Fig. 5: F-scores on the Mushroom dataset.



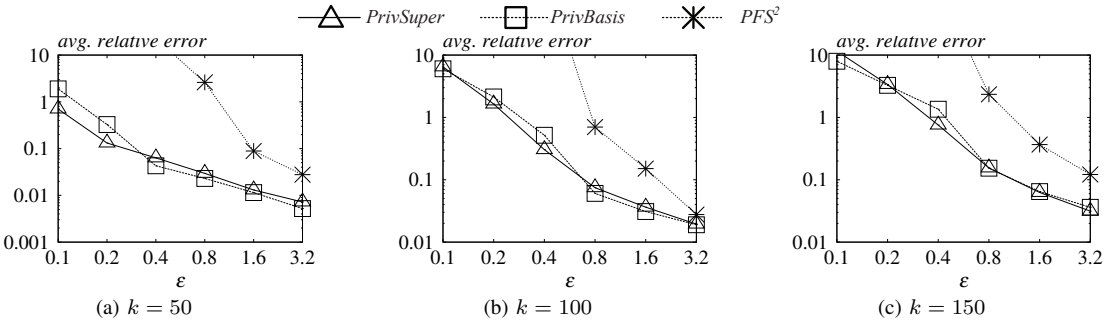
(a) $k = 50$ (b) $k = 100$ (c) $k = 150$

Fig. 6: F-scores on the BMS-POS dataset.



(a) $k = 50$ (b) $k = 100$ (c) $k = 150$

Fig. 7: Average relative errors on the Retail dataset.



(a) $k = 50$ (b) $k = 100$ (c) $k = 150$

Fig. 8: Average relative errors on the Tafeng dataset.

which leads to larger amount of noise, and hence, less accurate results.

Figures 3-6 show the *F-score* of each method on the other four datasets. The results on Tafeng and Pumsb are qualitatively similar to those in Figure 2, as they show that *PrivSuper* yields more accurate frequent itemsets than *PrivBasis* and *PFS*^{2} in most cases. However, on Mushroom (resp. BMS-POS), *PrivSuper* outperforms *PrivBasis* only when $\varepsilon \leq 0.1$ (resp. when $\varepsilon = 0.05$). To explain this, we first note that *PrivBasis* has a particular heuristic: if it infers that the number of frequent items is smaller than 12, it would skip the generation of frequent 2-itemsets and produce candidate itemsets directly based on the frequent items [4]. The Mushroom dataset happens to have a small number of frequent items, and the above heuristic of *PrivBasis* tends to yield accurate results on Mushroom, which reduces the performance gap between *PrivSuper* and *PrivBasis*. Meanwhile, on the BMS-POS dataset, it happens that the supports of top-k frequent itemsets are significantly larger than almost all of the remaining itemsets. In that case, it is relatively easy to identify top-k frequent itemsets under differential privacy, which explains why *PrivSuper* and *PrivBasis* have comparable performance on BMS-POS. Nevertheless, when $\varepsilon = 0.05$, the amount of noise required for differential privacy makes it much more challenging to distinguish top-k frequent itemset from the others on BMS-POS. In that scenario, *PrivSuper* is more effective than *PrivBasis* due to the adoptions of superset-first approach and SEM.

B. Average Relative Errors of Published Supports

In the second set of experiments, we evaluate all methods on the *average relative error*s of the supports of top-k frequent itemsets that they return. Figures 7-11 show the results. *PFS*^{2} incurs significant errors in all datasets but BMS-POS, since it directly injects Laplace noise into the supports of all candidate itemsets, which leads to rather noisy results when the number of candidate itemsets is large. In contrast, the errors of *PrivSuper* and *PrivBasis* are considerably smaller than those of *PFS*^{2}, due to their more advanced mechanisms for releasing itemset supports. In addition, the support accuracy of *PrivSuper* is noticeably better than that of *PrivBasis* in a number of cases (e.g., $k = 50$ on Pumsb, and $k = 100$ on Mushroom with $\varepsilon \geq 0.8$); in all other cases, the two methods have comparable errors, although *PrivSuper* uses 0.35ε to publish the noisy supports instead of 0.5ε in *PrivBasis*. This indicates the effectiveness of *PrivSuper*'s method for releasing supports of frequent itemsets.

VII. Conclusion

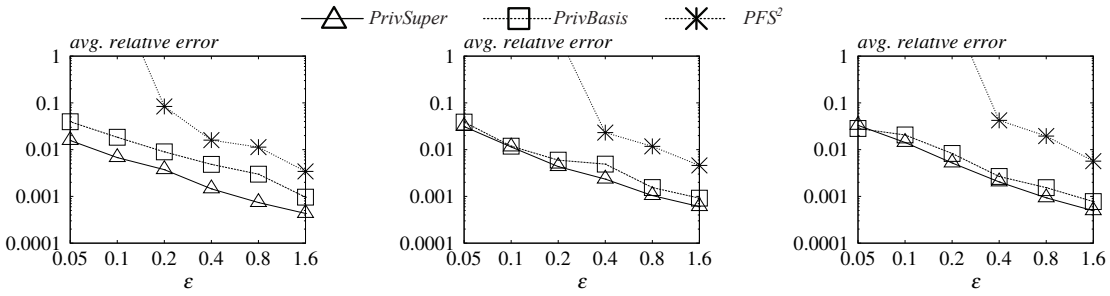
This paper investigates the problem of publishing top-k frequent itemsets under ε-DP. Different from existing solutions that directly use generic DP mechanisms, we employ the concept of MFIs, rather than Apriori, to detect the final results, so that corresponding subsets are added into results without privacy budget consumption. Furthermore, the SEM technique is designed to extend the current itemset with one more item in privacy-free when there is not one frequent super-itemset of the itemset any more. Extensive experiments on real-world datasets validate the effectiveness of our proposals.

Acknowledgment

This work was supported by National Natural Science Foundation of China (61433008, 61472071), by grant ARC19/14 from MOE (Singapore), and by gifts from MSRA and AT&T. Ning Wang was a visiting student at Nanyang Technological University, supported by China Scholarship Council, when this work was performed.

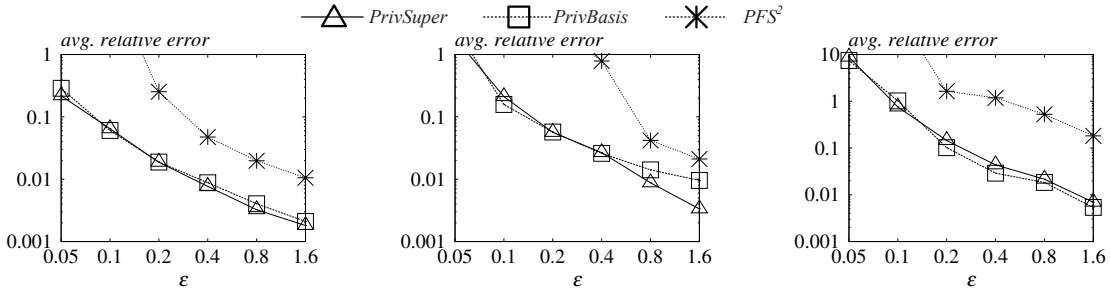
References

[1] C. Dwork, "Differential privacy," in *ICALP*, 2006, pp. 1–12.



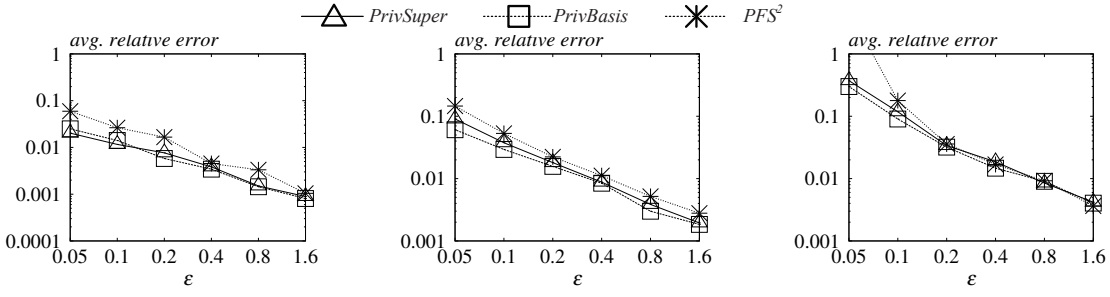
(a) $k = 50$ (b) $k = 100$ (c) $k = 150$

Fig. 9: Average relative errors on the Pumsb dataset.



(a) $k = 50$ (b) $k = 100$ (c) $k = 150$

Fig. 10: Average relative errors on the Mushroom dataset.



(a) $k = 50$ (b) $k = 100$ (c) $k = 150$

Fig. 11: Average relative errors on the BMS-POS dataset.

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

[3] https://www.wired.com/2016/06/apples-differential-privacy-collecting-data/.

[4] N. Li, W. H. Qardaji, D. Su, and J. Cao, "Privbasis: Frequent itemset mining with differential privacy," *PVLDB*, vol. 5, no. 11, pp. 1340–1351, 2012.

[5] J. Lee and C. W. Clifton, "Top-k frequent itemsets via differentially private fp-trees," in *KDD*, 2014, pp. 931–940.

[6] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket data," in *ACM SIGMOD Record*, vol. 26, no. 2, 1997, pp. 255–264.

[7] C.-Y. Tu, T.-J. Chen, and L.-F. Chou, "Application of frequent itemsets mining to analyze patterns of one-stop visits in taiwan," *PloS One*, vol. 6, no. 7, p. e14824, 2011.

[8] H. Lu, J. Han, and L. Feng, "Stock movement prediction and n-dimensional inter-transaction association rules," in *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1998, p. 12.

[9] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006, pp. 265–284.

[10] F. McSherry and K. Talwar, "Mechanism design via differential privacy," in *FOCS*, 2007, pp. 94–103.

[11] F. McSherry, "Privacy integrated queries: an extensible platform for privacy-preserving data analysis," in *SIGMOD*, 2009, pp. 19–30.

[12] Y. Chen and A. Machanavajjhala, "On the privacy properties of variants on the sparse vector technique," *CoRR*, vol. abs/1508.07306, 2015.

[13] J. Zhang, X. Xiao, and X. Xie, "Privtree: A differentially private algorithm for hierarchical decompositions," in *SIGMOD*, 2016, pp. 155–170.

[14] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB*, 1994, pp. 487–499.

[15] C. Dwork, F. Mcsherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006, pp. 265–284.

[16] F. McSherry and K. Talwar, "Mechanism design via differential privacy," in *FOCS*, 2007, pp. 94–103.

[17] R. Bhaskar, S. Laxman, A. Smith, and A. Thakurta, "Discovering frequent patterns in sensitive data," in *KDD*, 2010, pp. 503–512.

[18] C. Zeng, J. F. Naughton, and J. Cai, "On differentially private frequent itemset mining," *PVLDB*, vol. 6, no. 1, pp. 25–36, 2012.

[19] S. Xu, S. Su, X. Cheng, Z. Li, and L. Xiong, "Differentially private frequent sequence mining via sampling-based candidate pruning," in *ICDE*, 2015, pp. 1035–1046.

[20] 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.

[21] A. Ghosh, T. Roughgarden, and M. Sundararajan, "Universally utility-maximizing privacy mechanisms," *SIAM Journal on Computing*, vol. 41, no. 6, pp. 1673–1693, 2012.

[22] Http://fimi.ua.ac.be/data/.

[23] Http://www.kdd.org/kdd-cup/view/kdd-cup-2000.

[24] Http://stackoverflow.com/questions/25014904/download-link-for-ta-feng-grocery-dataset.