Volume 5, Issue 5, May 2015 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Outsourcing Transaction Databases in PPDM of Association Rules

Supriya S. Borhade[*], **Prof.S.V.Gumaste**
Computer Engineering Department, SPCOE & SPPU
Maharashtra, India

Abstract—Now a days there has been more interest in the paradigm of data mining-as-a-service. External service provider is beneficial for outsourcing the data mining task. Organization or data owner are unwilling to do so due to the concern of loss of privacy .This paper will provide an enhancing technique for preserving privacy of association rules and private data of individuals or organizational in an outsourced business transaction database. Generally organizations (data owner) are not expertise to outsource its mining needs. However, private data and the association rules of the outsourced database are considered private property of the organizations (data owner). To protect privacy of the corporate data, the data owner transforms its data and ships it to the server, when mining queries received by the server, and recovers the true patterns from the extracted patterns received from the server. In this paper studied the problem of outsourcing the association rule mining task within a organizational privacy-preserving framework. Here, proposing an attack model based on background knowledge and invent a scheme for privacy preserving outsourced mining. This scheme will ensures that each transformed item is indistinguishable with respect to the attacker's background knowledge, from at least k−1 other transformed items. These comprehensive experiments on a very large and real transaction database demonstrate that these techniques are efficient, useful, scalable, and guard privacy.

Keywords —K-privacy, privacy preserving outsourcing, Association rule mining.

I. INTRODUCTION

To protect customer privacy when outsourcing data mining tasks, it should control the direct access to the original data. In other words, the third party should not be allowed to access the original transactional data while performing mining tasks, and should not be able to interpret the mining results. Ling Qiu · Yingjiu Li · Xintao Wu presented a Bloom filter based approach which provides an algorithm for privacy preserving association rule mining with computation efficiency and predictable (controllable) analysis precision[1]. The Bloom filter [10] is a stream (or a vector) of binary bits. It is a computationally efficient and irreversible coding scheme that can represent a set of objects while preserving privacy of the objects. The model of mining and management of data as service will doubtless grow as popularity of cloud computing grows [2]. With this data mining-as-a-service paradigm, aimed at enabling organizations with limited computational resources and/or data mining expertise to outsource their data mining needs to a third party service provider [1].

It is advantageous to achieve sophisticated analysis on tremendous volumes of data in a cost-effective way. There is several severe security issues exist with the data-mining as-a-service model. One of the main security issues is that while accessing valuable data of the data owner it might learn sensitive information from it. For example, when server accesses valuable data from the owner, the server (or an intruder who gains access to the server) can learn which items are always co-purchased. However, both the transactions and the mined frequent patterns are the property of the data owner and should remain safe from the server. The problem of securing important data of organization is referred to as corporate privacy[6].In this paper studied the problem of outsourcing the association rule mining task within a corporate privacy preserving framework. A considerable body of work has been done on privacy-preserving data mining (PPDM) in a variety of contexts. In previous studied model observed a common characteristics is that the patterns mined from data are intended to be shared with parties except data owner (which may be distorted, encrypted, anonymized or transformed).

The key distinction between such bodies of work and problem is that, in the latter, both the original data and the mined results are not intended for sharing and must remain private to the data owner.

In this, adopting a conventional frequency-based attack model in which the server knows the accurate set of items in the owner's data and additionally, it also knows the correct support of every item in the original data.

Wong et al. [3] was one of the initial work which protecting against the frequency-based attack in the data mining outsourcing scenario. They use fake items idea to defend against the frequency based attack, but they lack to provide guarantee of privacy of a formal theoretical analysis and has been shown to be inconsistent very recently in [4], where a method for breaking the proposed encryption is given. Therefore, in the previous and preliminary work [7], It is proposed to solve this problem by using k-privacy, i.e., each item in the outsourced dataset should be indistinguishable from at least k − 1 items regarding their support. In this paper, the goal is to formulate an encryption scheme which enables formal privacy guarantees to be proved, and to validate this model over large-scale real-life transaction databases (TDB).

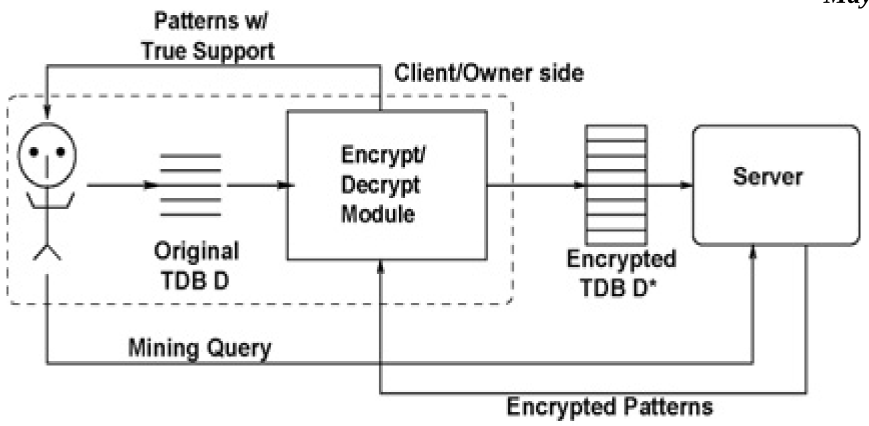


Fig. 1. Architecture of mining-as-service paradigm

The architecture behind this model is shown in Fig. 1. Encrypt/decrypt (E/D) module is used for encrypting clients / owners data, which can be basically treated as a black box from its point of view. Section V has all the details of this module, how it works or transforms the input data into an encrypted database. The server performs data mining and sends the (encrypted) patterns to the owner. This encryption scheme has the property that the returned supports are not true supports. The E/D module recovers the true identity of the returned patterns as well their true supports. It is trivial to show that if the data are encrypted using 1–1 substitution ciphers (without using fake transactions), many ciphers and hence the transactions and patterns can be broken by the server with a high probability by launching the frequency-based attack. Thus, the key focus of this paper is to formulate encryption schemes such that formal privacy guarantees can be proven against attacks conducted by the server using background knowledge, while keeping the resource necessities under control.

1st, Formally define an attack model for the opponent and make the background knowledge the opponent may possess particular. The belief of privacy needs that, for each cipher text item, there are at least k−1 distinct cipher items that are indistinguishable from the item regarding their supports.

2nd, Develop an encryption scheme, called RobFrugal, that the E/D module can employ to transform client data before it is shipped to the server.

3rd, To allow the E/D module to recover the true patterns and their correct support. This proposes that it will create and keep a dense structure, called synopsis. This will also provide the E/D module with an efficient strategy for incrementally maintaining the synopsis against updates in the form of appends.

4th, By conducting a formal analysis based on this attack model and prove that the probability that an individual item, a transaction, or a pattern can be broken by the server can always be controlled to be below a threshold chosen by the owner, by setting the anonymity threshold k. This result holds unconditionally for the RobFrugal scheme.

5th, Last but not the least, This can conduct experimental analysis of the model by using a large real dataset from the Cooperate store chain in India . The Results of this model shows that, this encryption schema is effective, scalable, and achieve the desired level of privacy.

Next section will describe the related work. Section III will give quick review of the background on frequent pattern mining. This privacy-preserving outsourcing model and the associated problem study is given in Section IV. Section V formulates the encryption/decryption scheme used for this model. Section VI focus on the theoretical results which concern the complexity and privacy guarantees. Section VII discusses the results of a comprehensive set of experiments conducted using real and synthetic datasets. Finally, concluded this paper and discuss directions for future research in Section VIII.

II. RELATED WORK

PPDM has received considerable attention in research. Here, in this model that data is collected from a various sources by a collector for the purpose of consolidating the data and conducting mining. The collector is not the trusted one with protecting the privacy. Data are subjected to a random perturbation as it is collected. Techniques have been developed for perturbing the data so as to preserve privacy while ensuring the mined patterns or other analytical properties are sufficiently close to the patterns mined from original data. This module of work was initiated by [7] and has been followed up by several papers since [8]. This approach is not suited for corporate privacy, in that some analytical properties are revealed. Another most important associated issue is secure multiparty mining over distributed datasets.

Data on which mining is to be performed is might be horizontally partitioned or vertically partitioned, and distributed among a number of parties. In such case the partitioned data cannot be shared and must remain private but the results of mining on the union of the data are shared among the participants, by means of multiparty secure protocols [9]–[11]. They do not believe third parties. This approach partially implements corporate privacy, as local databases are kept private, but it is too weak for this outsourcing problem, as the resulting patterns are revealed to multiple parties.

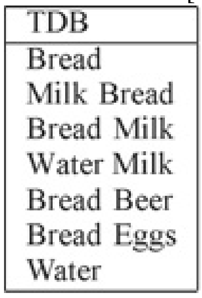
The problem attended in this paper is outsourcing of pattern mining within a corporate privacy-preserving model. A key distinction between this problem and the aforementioned PPDM problems is that, this models setting is projected for sharing underlying data and results while data owner's data remain private. when server holds background knowledge and conducts attacks on that basis, it should not be able to guess the correct candidate item or item set corresponding to a given cipher item or item set with a probability above a given threshold. The models that are most similar to this model are [3] and [12]. They also studied and discussed background attack and assume that the opponent has prior knowledge of the frequency of items or item sets, which can be used to try to recognize/re-identify the encrypted items. The model [3] utilizes a 1-to-n item mapping together with non-deterministic addition of cipher items to protect the identification of individual items. In paper [4] has proven that the encoding model in [3] can be broken without using context-specific information. The success of the attacks is [4] mainly relies on the existence of unique, common, and fake items, defined in [3]; this model does not create any such items, and the attacks in [4] are not applicable to this model. It is assumed the attacker knows accurate frequency of single items, similarly to this model[11]. They use a similar privacy model as this, which requires that each real item must have the same frequency count as k−1 other items in the outsourced dataset. They show that their outsourced dataset satisfies k-support anonymity. However, they do not offer any theoretical analysis of anonymity of item sets. Instead they confine themselves to an empirical analysis. Compared with these two works, one can analyze that this model can always achieve demonstrable privacy guarantee with respect to the background knowledge of the attacker and the notion of privacy.

In general, it is too expensive to achieve perfect secrecy of outsourced frequent itemset mining [4]. With this model less strict privacy models, one can achieve practical privacy preserving methods that provide reasonable privacy assured. This experimental study also shows that in practice, due to specific characteristics of the real transaction datasets (e.g., the power-law distribution of items), even the privacy-preserving methods for less-strict privacy models can enjoy a relatively high level of privacy in practice.

Besides this, an important issue in association rule mining (or frequent item set mining) outsourcing is the ability to deal with updates. Neither of the schemes above addresses this concern. Here, proposed an incremental method for updating the compact synopsis maintained by the owner against updates to the database.

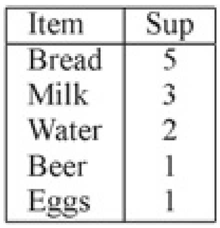
III. PATTERN MINING TASK

Here, considering that the reader is familiar with the basics of association rule mining. let I = i1, ..., in be the set of items and D = t1, ..., tm a TDB of transactions, each of which is a set of items. Denoted the support of an itemsetS ⊆ I as suppD(S) and the frequency by freqD(S). Recall that freqD(S) = suppD(S)/|D|. For each item i, suppD(i) and freqD(i) denote, respectively, the individual support and frequency of i. The function suppD(.), projected over items, is also called the item support table of D represented in tabular form [support table in Fig. 2(b)]



TDB
Bread
Milk Bread
Bread Milk
Water Milk
Bread Beer
Bread Eggs
Water

(a)



Item	Sup
Bread	5
Milk	3
Water	2
Beer	1
Eggs	1

(b)

The well-known frequent pattern mining problem [13] is: given a TDB D and a support threshold σ find all itemsets whose support in D is at least σ.

IV. MODEL PRIVACY

Consider D denote the original TDB that the owner has. to protect individual identification of items, when the owner applies an encryption function to D and transforms it to D* ,the encrypted database. Refer items in D as plain items and items in D* as cipher items. The term item shall mean plain item by default. The notions of plain item sets, plain transactions, plain patterns, and their cipher counterparts are defined in the obvious way. use I to denote the set of plain items and E to refer to the set of cipher items.

a)Adversary Knowledge :The server or an opponent might gains access to it may possess some prior or background knowledge using which opponent can conduct attacks on the encrypted database D *. Generically refer to any of these agents as an attacker.

Here adopting a conventional model and assuming that the attacker knows exactly the set of (plain) items I in the original TDB D and their true supports in D, i.e., suppD(i), ∀ i ∈ I. The attacker may have access to similar data from a competing organization, may read published reports, etc. In reality, the opponent may be having approximate knowledge of the supports or may know the exact/approximate supports of a subset of items in D. However, to make the analysis robust, one can adopt the conventional assumption that he knows the exact support of every item. Remember, that as the opponent has access to the encrypted database D*, he also knows the supports supp D*(e), e ∈ E, where E is the set of cipher items in the encrypted database D*.

The encryption schema used in this paper is based on:

1) Replacing each plain item in D by a 1–1 substitution cipher and
2) Adding fake transactions to the database.

Consider, that the opponent knows this and thus he knows that |E| = |I|. Essentially, compared to [3], adversary knowledge model corresponds to a (100%, 0%) knowledge model, confined to single items. Assume that the opponent neither has the knowledge of plaintext transactions nor the frequency of item sets and the distribution of transaction lengths in the original database.

*b)Attack Model :*By assuming that the service provider (who can be an opponent) is semi-honest in the sense that although he does not know the details of encryption algorithm, opponent can be curious and he can use his prior knowledge to make assumption on the encrypted transactions. Also considered, that the opponent always returns (encrypted) item sets together with their exact support. The data owner considers the true identity of:

1) Every cipher item;
2) Every cipher transaction;
3) Every cipher frequent pattern;

As intellectual property which should be protected. The following attack model will considers all these.

1) Item-based attack: $\forall$ cipher item $e \in E$, the attacker constructs a set of candidate plain items $Cand(e) \subset I$. The probability that the cipher item e can be broken $pb(e) = 1/|Cand(e)|$.

2) Set-based attack: Given a cipher itemset E, the attacker constructs a set of candidate plain itemsets Cand(E), where $\forall X \in Cand(E)$, $X \subset I$, and $|X| = |E|$. The probability that the cipher itemset E can be brokenpb(E) = $1/|Cand(E)|$.

Refer to pb(e) and pb(E) as crack probabilities. From the point of view of the owner, minimizing the probabilities of crack is desirable. Intuitively, Cand(e) and Cand(E) should be as large as possible. Ideally, Cand(e) should be the whole set of plaintext items. This can be achieved if one can bring each cipher item to the same level of support, e.g., to the support of the most frequent item in D. Unfortunately, this option is impractical, as it will lead to a large size of the fake transactions, which in turn leads to a dramatic explosion of the frequent patterns and making pattern mining at the server side computationally prohibitive. This motivates us of relaxing the equal-support constraint and introducing item k-anonymity as a compromise.

Definition 1: Let D be a TDB and D* its encrypted version. D* satisfies the property of item k-privacy provided for every cipher item $e \in E$, if there are at least k−1 other distinct cipher items $e1, ..., ek-1 \in E$ such that $suppD^* (e) = suppD^* (ei), 1 \leq i \leq k - 1$.

The concept of item k-anonymity is similar to the k-support anonymity [11] (based on the well-known k-anonymity) as also require that for each cipher text item e, there are at least k−1 distinct cipher items that are indistinguishable from e regarding their supports.

*c) Problem Statement:*To compute the privacy guarantees of an encrypted database, Here define the following notion.

Definition 2: Given a database D and its encrypted version D*, say D* is k-private if:

1) for each cipher item $e \in D^*$,$pb(e) \leq 1/k$;
2) for each cipher itemset E with support $suppD^* (E) > 0$, $pb(E) \leq 1/k$.

The problem studied here is as follows.

Problem studied: Given a plain database D, construct a k-private cipher database D* by using substitution ciphers and adding fake transactions such that from the set of frequent cipher patterns and their support in D* sent to the owner by the server, the owner can reconstruct the true frequent patterns of D and their exact support. Additionally, like to minimize the space and time incurred by the owner in the process and the mining overhead incurred by the server.

V. ENCRYPTION/DECRYPTION SCHEME

A. Encryption

In this section, introducing the encryption scheme, called RobFrugal, which transforms a TDB D into its encrypted version D*. This scheme is parametric with respect to k > 0 and consists of three main steps: 1) using 1–1 substitution ciphers for each plain item; 2) using a specific item k-grouping method; and 3) using a method for adding new fake transactions for achieving k-privacy. The constructed fake transactions are added to D (once items are replaced by cipher items) to form D *, and transmitted to the server. A record of the fake transactions, i.e., DF = D * \D, is stored by the E/D module in the form of a compact synopsis, as discussed in Sections V-C and V-D.

B. Decryption

When the client requests the execution of a pattern mining query to the server, specifying a minimum support threshold σ, the server returns the computed frequent patterns from D*.

Clearly, for every itemset S and its corresponding cipher itemset E, have that $suppD(S) \leq suppD^* (E)$. For each cipher pattern E returned by the server together with suppD* (E), the E/D module recovers the corresponding plain pattern S. It needs to reconstruct the exact support of S in D and decide on this basis if S is a frequent pattern. To achieve this goal, the E/D module adjusts the support of E by removing the effect of the fake transactions.

$$suppD(S)= suppD^* (E)-suppD^*\backslash D(E).$$

This follows from the fact that support of an itemset is additive over a disjoint union of transaction sets. Finally, the pattern S with adjusted support is kept in the output if $suppD(S) \geq \sigma$. The calculation of suppD*\D(E) is performed by the E/D module using the synopsis of the fake transactions in D * \D.

The proposed encryption/decryption scheme is a viable solution for privacy-preserving pattern mining over outsourced TDB, provided that a correct and efficient implementation exists. On the efficiency side, it is not practical to store the support suppD*\D(E) for every cipher pattern. In order to realize the encryption scheme efficiently, need to address the following technical issues.

1) How do cluster items into groups of k?
2) How do create the needed fake transactions?
3) How is the synopsis represented and stored?

C. Grouping Items for k-Privacy

Given the items support table, several strategies can be adopted to cluster the items into groups of size k. Start from a simple grouping method called Frugal. By assuming the item support table is sorted in descending order of support and refer to cipher items in this order as e1, e2, etc.

Definition 3: The Frugal method consists of grouping together cipher items into groups of k adjacent items in the item support table in decreasing order of support, starting from the most frequent item e1. Assume e1, e2, . . . , en is the list of cipher items in descending order of support (with respect to D), the groups created by Frugal are {e1, . . . , ek}, {ek+1, . . . , e2k}, and so on. The last group, is less than k in size, is merged with its previous group. By denoted the grouping obtained using the above definition as Gfrug. For example, consider the example TDB and its associated (cipher) item support shown in Fig. 2. For k = 2, Gfrug has two groups: {e2, e4} and {e5, e1, e3}. This corresponds to the partitioning groups shown in Table I(a). Thus, in D *,the support of e4 will be brought to that of e2; and the support of e1 and e3 brought to that of e5. Given the fact that the support of the items strictly decreases monotonically, Frugal grouping is optimal among all the groupings with the item support table sorted in descending order of support. This means, it minimizes ||G||, the size of the fake transactions added, and hence the size ||D*||. But is Frugal a robust grouping, i.e., will it guarantee that itemsets (or transactions) cannot be cracked with a probability higher than k ? The answer is no, in general. To see this point, consider the item support table in Table I: the first group created by Frugal for k = 2, {e2, e4} [see Table I(a)] is supported in D, because e2, e4 occur together in a transaction of D. Therefore, there only exists one itemset candidate of {e2, e4}, i.e., the privacy guarantee is 1-privacy.To fix the privacy vulnerabilities of Frugal, introduce the RobFrugal grouping method, which modifies Frugal by requiring that no group is a supported itemset in D.

Definition 4: Given a TDB D and its Frugal grouping Gfrug = (G1, ...,Gm), the grouping method RobFrugal consists in modifying the groups of Gfrug by repeating the following operations, until no group of items is supported in D: 1) select the smallest j ≥ 1 such that suppD(Gj) > 0; 2) find the most frequent item i ∈ Gj such that, for the least frequent item i of Gj have: suppD(Gj \ {i} ∪ {i}) = 0; and 3) swap i with i in the grouping.

For example, given the item support table in Fig. 2, the grouping illustrated in Table I(b), obtained by exchanging e4 and e5 in the two groups of Frugal, is now robust: none of the two groups, considered as itemsets, is supported by any transaction in D. The aim of Step 2 in Definition 4 is to obtain a robust grouping while maintaining as small as possible the number of fake transactions that are added to achieve k-privacy. In particular, It will show the information about fake transactions can be maintained by the data owner using a compact synopsis. This step is used to ensure the synopsis is as small as possible.

The key property of RobFrugal is that, by construction, it is a robust grouping for any input TDB D. It is immediate to note that if the support in D of each group Gi of the initial grouping Gfrug is 0, then RobFrugal produces a robust and optimal grouping, where optimal means that it minimizes the number of the fake transactions that are created by this encryption approach. On the other hand, it should be noted that a grouping according to RobFrugal may not exist, depending on the extent of density/sparsity in the TDB. For example, in a TDB where each pair of items occurs at least once together, RobFrugal will not find a grouping for k = 2. In this case, a simple solution is to keep increasing the value of k until a RobFrugal grouping scheme exists. The intuition is that as k gets larger it is less likely that there is a real transaction containing all items in a group. However, with a large k, the number of fake transactions increases. This affects storage and processing at the server side although the data owner can always maintain information about fake transactions using a compact synopsis of size O(n), n being the number of items. In practice, It has been found that even for small values of k = 10 to 50, a RobFrugal grouping scheme does exist. This was the case in all these experiments with real transaction data. In the RobFrugal encryption scheme, the output of grouping can be represented as the noise table. It extends the item support table with an extra column "Noise" indicating, for each cipher item e, the difference among the support of the most frequent cipher item in e's group and the support of e itself, as reported in the item support table. Denoted the noise of a cipher item e as N(e). Continuing the example, the noise table obtained with RobFrugal is reported in Table II(a). The noise table represents the tool for generating the fake transactions to be added to D to obtain D*.

C. Constructing Fake Transactions:

Given a noise table specifying the noise N(e) needed for each cipher item e, It generate the fake transactions as follows. First, by dropping the rows with zero noise, corresponding to the most frequent items of each group or to other items with support equal to the maximum support of a group. Second, by sorting the remaining rows in descending order of noise. Let e 1. . . em be the obtained ordering of (remaining) cipher items, with associated noise N(e1), . . . ,N(em). The following fake transactions are generated:

1) N(e1) − N(e2) instances of the transaction {e1};
2) N(e2) − N(e3) instances of the transaction {e1, e2};
3) . . . ;
4) N(em−1) − N(em) instances of the transaction {e1, . . . ,em−1};

5) N(em) instances of the transaction {e1, . . . , em}.

Continuing the example, consider cipher items of nonzero noise in Table II(a). The following two fake transactions are generated: two instances of the transaction {e5, e3, e1} and one instance of the transaction {e5}. Note that even though the attacker may know the details of the construction method, he/she is not able to distinguish these fake transactions from the true ones, since the attacker does not have any background knowledge of frequency of item sets or of original transaction length distribution. It can be shown that this method yields a minimum number of different types of fake transactions that equal the number of cipher items with distinct noise. This observation yields a compact synopsis for the client of the introduced fake transactions. The purpose of using a compact synopsis is to reduce the storage overhead at the side of the data owner who may not be equipped with sufficient computational resources and storage, which is common in the outsourcing data model.

In order to implement the synopsis efficiently, here uses a hash table generated with a minimal perfect hash function. Minimal perfect hash functions are widely used for memory efficient storage and fast retrieval of items from static sets. A minimal perfect hash function is a perfect hash function that maps n keys to n consecutive integers, usually [0 . . . n − 1]. Hence, h is a minimal perfect hash function over a set S if and only if ∀ i, j ∈ S, h(j) = h(i) implies j = i, and there exists an integer p such that the range of h is p, . . . , p + |S| − 1. A minimal perfect hash function h is order-preserving if for any keys j and i, j < i implies h(j) < h(i). In this scheme, the items of the noise table ei with N(ei) > 0 are the keys of the minimal perfect hash function. Given ei, function h computes an integer in [0 . . . n − 1], denoting the position of the hash table storing the triple of values _ei, times i, occi, where times i represents the number of times that the fake transaction {e1, e2, . . . , ei} occurs in the set of fake transactions, and occi is the number of times that ei occurs altogether in the future fake transactions after the transaction {e1, e2, . . . , ei}.Given a noise table with m items with non null noise, this approach generates hash tables for the group of items. In general, the ith entry of a hash table HT containing the item ei has times i = N(ei) − N(ei+1), occi = _g j=i+1 N(ej), where g is the number of items in the current group. Note that each hash table HT represents concisely the fake transactions involving all and only the items in a group of g ≤ l max items. The hash tables for the items of nonzero noise in Table II(a) are shown in Table II(b). Finally, used a (second-level) ordinary hash function H to map each item e to the hash table HT containing e. Note that after the data owner outsources the encrypted database (including the fake transactions), he/she does not need to maintain the fake transactions in its own storage. Instead the data owner only has to maintain a compact synopsis, which stores all the information needed on the fake transactions, for later recovery of real supports of item sets. The size of the synopsis is linear in the number of items and is much smaller than that of the fake transactions.

With the above data structure, one can define the function RS that allows an efficient computation of the real support of a pattern E = {e1, e2, . . . , en} with fake support s as follows: RS(E) = s − (HT[h(emax)].times + HT[h(emax)].occ), where: i) emax is the item in E such that for 1 ≤ j ≤ n, have h(ej) ≤ h(emax), and ii) HT = H(ei) is the hash table associated by H to any item ei of E. For example, in Table I(b), for E1 = {e5}, RS(E1) = s1 − (1 + 2), whereas for E2 = {e5, e3}, RS(E2) = s2 − (2 + 0), where si is the fake support of Ei. This is exactly right since e5 is fakely added three times while e3 is fakely added two times.

D. Incremental Maintenance

Now consider incremental maintenance of the encrypted TDB. The E/D module is responsible for this. Once focus on batches of appends, which are very natural in data warehouses. Let D be an initial TDB and _D be a set of transactions that are appended. Let D * be the original encrypted TDB. The E/D module stores D as a prefix tree T . Let syn(D,D *) denote the compact synopsis stored by the E/D module for encoding the generation of fake transactions in D *. The server and client have the item support tables IST of D and IST* of D*.Next, the new TDB _D arrives, together with its item support table IST_. The following steps can be applied to obtain an incremental version of the E/D module according to the RobFrugal scheme.

1) The new transactions in _D are inserted into the prefix tree T , obtaining a cumulative representation of D∪_D. Also, a cumulative item support table IST is constructed by adding the support of each item in IST* and IST. In particular, for each item ei ∈ IST* the support of ei is added to the support of ei ∈ IST_. Clearly, IST_ could both: a) not contain some item belonging to IST, and b) contain some new items. In case a, the support of these items in the cumulative item support table IST is equal to the support of them in IST; while in case b the support of these items in IST is equal to their support in IST_. Note that when the cumulative item support table IST is constructed the method keeps the order of the items in the IST*. Thus, if an item belonging to IST* is in the position i, then in the cumulative item support table IST its position is i. When an item only belongs to the IST, then this item is appended to the list. Clearly, the balance of support in each group is now generally destroyed by the new item supports, and it is needed to add new fake transactions to restore the balance. The old grouping is checked for robustness with respect to the overall prefix-tree T and the existing synopsis, which is equivalent to checking against to D * ∪ F*. If the check for robustness fails, then a new grouping is tried out with swapping, until a robust grouping is found. Then, the new synopsis for the new fake transactions is constructed as usual; notice that the new grouping is robust with respect to the new fake transactions by construction, as the most frequent item of each group does not occur in any fake transaction. The E/D module uses both old and new synopses to reconstruct the exact support of a pattern from the server.
This method extends to the case when simultaneously, a new batch is appended and old batch is dropped; the method also works in the case when new items arrive or old items are dropped. Details can be found in .

VI. ANALYSIS

Providing the main technical results on the robustness and the effectiveness of this encryption/decryption schema.

A. Complexity

The complexity analysis shows that the encryption method requires O(n) storage and O(n2) time, where n is the number of distinct items. These complexity figures essentially concern the space and time needed for the creation and maintenance of the synopsis representing the fake transactions. Concerning decryption, finding here that the procedure to recover the true support of a pattern by using the synopsis requires O(m) time, where m is the size of the patterns.

B. Privacy

Against the item-based attack: Recall that here assumed the attack does not know the details of this encryption algorithms. However, when the attacker tries to map plaintext and ciphertext items based on their frequencies, he may observe that for every ciphertext item e, there always exist a plaintext item whose frequency is no larger than that of e. Then he may guess that fake transactions are inserted into the original dataset. Assume that he does so. Then, he can infer that for every cipher item e, suppD(i) ≤ suppD* (e), where i is the true plain item corresponding to e. For each cipher item e, the attacker tries to infer the true plain item i corresponding to e. Recall that the attacker knows suppD* (e) and suppD(i), and that suppD(i) ≤ suppD* (e). Based on this, for each cipher item e, he can construct a set of candidate items which could have been transformed by the owner to e. It is tempting to think that all items i

such that suppD(i) ≤ suppD* (e) are candidates for e. However, this can be narrowed down substantially as follows. Let en be any cipher item with the smallest support in D *. Consider the set of all cipher items E that have the same support in D* as en and let S = {I | suppD(i) ≤ suppD* (en)}. By the grouping established using RobFrugal, have |S| = |E| and ∀ e ∈ E, the set of candidate items must be S. Now, consider any cipher item e with support suppD* (e) > suppD* (en). It is easy to see that any item i ∈ S corresponding to en cannot be in the candidate set of e, since mapping e (back) to i would make it infeasible to map all cipher items consistently to an item,while respecting the support constraints. Using this notion,the attacker can prune the set of candidate sets of items as follows. Let ICand(e) = {i | suppD(i) ≤ suppD* (e)} be the initial candidate set ∀ e ∈ E. The attacker can sort the cipher items in nondecreasing order of their frequency in D*. Let S = {e1, ..., em} be the set of cipher items with the smallest support in D*. He can infer ICand(e1) = · · · = ICand(em) and |ICand(ei)| = m(1 ≤ i ≤ m). Clearly, every cipher item ei ∈ Smust be mapped to a plain item in ICand(ei), and no cipher item in E − S can be mapped to a plain item in ICand(ei), since doing so makes it impossible to map all cipher items consistently back to some plain item. Thus, the attacker can remove both S and ICand(e) from further consideration. This has the effect of pruning ICand(e), for every cipher item e ∈ E − S. The attack can repeat the procedure on the remaining list of E −S until he prunes the initial candidate set of every cipher item. Denote the set of candidates for a cipher item e as Cand(e)∀ e ∈ E. Define a mapping ι : E → I to be consistent provided suppD(ι(e)) ≤ suppD* (e). An assignment e _→ i of an item to a cipher item is feasible if there is a consistent mapping ι : E → I such that ι(e) = i. A candidate set for a cipher item is minimal provided assigning any item from the candidate set to the cipher item is a feasible assignment.

VII. EXPERIMENTS

In this section, reported here experimental evaluation to assess the encryption/decryption overhead and the overhead at the server side incurred by the proposed schema.

A. Datasets

By experimenting on a large real-world database. The real world database is donated to us by Coop, a cooperative of consumers that is today the largest supermarket chain in Italy. Selected here the transactions occurring during four periods of time in a subset of Coop stores, creating in this way four different databases with varying number of transactions: from 100k to 300k transactions. In all the datasets the transactions involve 15 713 different products grouped into 366 marketing categories. Transactions are itemsets, i.e., no product occurs twice in the same transaction.

Consider two distinct kinds of TDBs:

1) product-level Coop TDBs, denoted by Coop- Prod, where items correspond to products, and 2) category level Coop TDBs, that denote by CoopCat, where items correspond to the category of the products in the original transactions. In these datasets, lmax = 188 for CoopProd, while lmax = 90 for CoopCat. Sparsity/density of the two TDBs has a dramatic effect on pattern mining: the number of frequent patterns found in CoopCat tends to explode for higher support thresholds, compared to CoopProd. By experimenting with these algorithms for both CoopProd and CoopCat.

B. Experimental Evaluation

This implemented the RobFrugal encryption scheme, as well as the decryption scheme, as described in Section V, in Java. All experiments were performed on an intel Core2 Duo processor with a 2.66 GHz CPU and 6GB RAM over a Linux platform (ubuntu 8.10). This adopted the a priori implementation by Christian Borgelt,2 written in C and one of the most highly optimized implementations.

1) Encryption Overhead: First, assessed the total time needed by the ED module to encrypt the database (grouping,synopsis construction, creation of fake transactions): timing share reported for CoopProd and CoopCat, for different values of k and different number of transactions. The results show that the encryption time is always small; it is under 1 s for the biggest CoopProd TDB, and below 0.8 s for the biggest CoopCat TDB. Therefore, when there are multiple mining queries, which is always the case for the outsourcing system, the encryption overhead of this scheme is negligible compared with the cost of mining. It is worth noting that these experiments provide empirical evidence that the theoretical complexity upper bound of O(n2) is to be sure highly negative. To see this point, counted the number of queries (to check that each group is unsupported) performed by the ED module (RobFrugal), over the two TDBs for the different values of k, and discovered that such number always coincides with nk , except for CoopCat TDBs in the cases k = 10 and k = 20: for example, for k = 10 and number of transactions 400K (the biggest TDB), an additional 3790 item swaps are needed to find a robust grouping and only 10 for k = 20. This is a strong empirical evidence that in real life databases RobFrugal reaches a solution very fast, with complexity far below the O(n2) worst case: e.g., for CoopCat with k = 10 and 400 transactions, RobFrugal only needs to check a total of 3826 queries, while 3662 = 133, 956! Secondly , it is assessed the size of fake transactions added to the databases after encryption. It reports the sizes of fake transactions for different values of k in CoopProd* and CoopCat* with 300k transactions. It is observed that the size of fake transactions increases linearly with k. Also, Observe that density affects the generation of fake transactions: e.g., that CoopProd*, for k = 30, is only 8% larger than CoopProd while, for the same k, CoopCat* is 80% larger than CoopCat. It is also assessed the size of the fake transactions on synthetic databases. Finally, it is assessed the overhead of incremental encryption, which occurs when a new TDB is appended; to this end, split CoopProd with 500k transactions into two halves CoopProd1 and CoopProd2, and treat CoopProd1 as the original TDB and CoopProd2 as the appended one. Consider the non-incremental method, which is to encrypt CoopProd1∪CoopProd2 from scratch, and compare its encryption time with that of the incremental approach. By simply ignoring the time for transmitting TDBs between the client and server as assume that the TDB streams into the ED module and the client can send the data that has been encrypted to the server while encrypting the remaining data. Furthermore, thanks to the incremental procedure, the client avoids to send different encrypted versions of the same set of transactions to the server. This reduces the cost for data retransmission and makes this approach more robust against the possible attack based on the comparison of multiple versions of the encrypted TDB.

2) Mining Overhead: Studied the overhead at the server side for the pattern mining task over CoopProd* with respect to CoopProd with 300K transactions. Instead of measuring performance in run time, measure the increase in the number of frequent patterns obtained from mining the encrypted TDB, considering different support thresholds. Results are plotted for different values of k; notice that k = 1 means that the original and encrypted TDB are the same. The x-axis shows the relative support threshold in the mining query, wrt the total number of original transactions (300k); the number of frequent patterns obtained is reported on the y-axis. It is observe that the number of frequent patterns, at a given support threshold, increases with k, as expected. However, mining over CoopProd* exhibits a small overhead even for very small support thresholds, e.g., a support threshold of about 1% for k = 10 and 1.5% for k = 20. Mining over CoopCat with 300k transactions and CoopCat* is more demanding, given the far higher density, but having similar observation, although at higher support thresholds. In either case, found that, for reasonably small values of the support threshold, the incurred overhead at server side is kept under control; clearly, a tradeoff exists between the level of privacy, which increases with k, and the minimum affordable support threshold for mining, which also increases with k. Note that, the client for extracting pattens from CoopProd* has to consider the number of fake transactions when he specifies the minimum support threshold in his query. Indeed, the increasing of the number of transactions in CoopProd* requires to use a smaller support threshold to have the same patterns that one could have from the original data. For example, for k = 10 CoopProd* has 306k transactions so, to have the patterns obtained from the original data (300k trans.) with a support of 2% the client has to use the support threshold equal to 1.9%, obtained by the computation 2 × 300k/306k. The need to use a smaller support could make harder the discovery of frequent patterns. But in these experiments, given real TDBs, found that the number of fake transactions does not change the support threshold too much, making the problem still tractable.

3) Decryption Overhead: by the ED Module: now consider the feasibility of the proposed outsourcing model.

The ED module encrypts the TDB once which is sent to the server. Mining is conducted repeatedly at the server side and decrypted every time by the ED module. Thus, need to compare the decryption time with the time of directly executing a priori over the original database. This comparison is particularly challenging, as one has to select one of the most optimized versions of a priori (written in C), while this decryption method is written in Java without particular optimizations, except for the use of hash tables for the synopsis. Nevertheless, the decryption time is about one order of magnitude smaller than the mining time; for higher support threshold, the gap increases to about two orders of magnitude.The situation is similar in CoopCat.

4) Crack Probability: also analyzed the crack probability for transactions and patterns over the Coop TDBs.

Discovered that in both CoopCat and CoopProd TDBs encrypted by RobFrugal, around 90% of the transactions can be broken with probability strictly less than 1 k . For example, considering the encrypted version of CoopProd with 300K transactions, Also discovered from experiments the following facts, even for small k. For instance, for k = 10, every transaction E has at least 10 plain itemset candidates, i.e., prob(E) ≤ 110 . Around 5% of transactions have exactly a crack probability 110 , while 95% have a probability strictly smaller than 110 . Around 90% have a probability strictly smaller than 1100 . No single transaction contains any pattern consisting exactly of the items in a group created by RobFrugal

VIII. CONCLUSION AND FUTURE WORK

In this paper, representing the problem of corporate or commercial privacy-preserving mining of frequent patterns from which association rules can easily be computed on an encrypted outsourced TDB. One can assume that a conventional model where the opponent knows the domain of items and their exact frequency and can use this knowledge to identify cipher items and cipher itemsets. In this paper, proposed an encryption scheme, called RobFrugal, that is based on 1–1 substitution ciphers for items and adding fake transactions to make each cipher item share the same frequency as ≥ k−1 others. It makes use of a compact synopsis of the fake transactions from which the true support of mined patterns from the server can be efficiently recovered. Also proposed a approach for incremental maintenance of the outline against updates consisting of added and dropping of old transaction batches. Unlike previous works, such as [2] and [11],it has been formally proved that this method is robust against an adversarial attack based on the original items and their accurate support. These experiments based on both large real and synthetic datasets yield strong evidence in favor of the practical applicability of approach. Currently, this privacy analysis is based on the assumption of equal probability of candidates. It would be worthy to enhance the framework and the analysis by appealing to cryptographic notions such as ideal secrecy [13]. Moreover, this model considers the cipher text-only attack model, in which the opponent has access only to the encrypted items. It could be remarkable to consider other attack models where the attacker knows some pairs of items and their cipher values. For example, one can study the privacy guarantees of this method in case of known-plaintext attacks where the opponent knows some item, cipher item pairs, chosen-plaintext attacks (where the opponent knows some item and cipher pairs for selected items), and chosen-ciphertext attacks. Another worthy direction is to relax these assumptions about the attacker by allowing him to know the details of encryption algorithms and/or the frequency of item sets and the distribution of transaction lengths. This framework assumes that the attacker does not hold such knowledge. Moreover, one can explore how to improve the RobFrugal algorithm to minimize the number of spurious patterns.

REFERENCES

[1] L. Qiu, Y. Li, and X. Wu, "*Protecting business intelligence and customer privacy while outsourcing data mining tasks,*" Knowledge Inform. Syst.,vol. 17, no. 1, pp. 99–120, 2008.

[2] R. Buyya, C. S. Yeo, and S. Venugopal, "*Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities,"* in Proc. IEEE Conf. High Performance Comput. Commun., Sep. 2008, pp. 5–13.

[3] W. K. Wong, D. W. Cheung, E. Hung, B. Kao, and N. Mamoulis,"*Security in outsourcing of association rule mining,*" in Proc. Int. Conf.Very Large Data Bases, 2007, pp. 111–122.

[4] I. Molloy, N. Li, and T. Li, "*On the (in)security and (im)practicality of outsourcing precise association rule mining,*" in Proc. IEEE Int. Conf.Data Mining, Dec. 2009, pp. 872–877.

[5] P. K. Prasad and C. P. Rangan, "*Privacy preserving birch algorithm for clustering over arbitrarily partitioned databases*," in Proc. Adv. Data Mining Appl., 2007, pp. 146–157.

[6] C. Clifton, M. Kantarcioglu, and J. Vaidya, "*Defining privacy for data mining*," in Proc. Nat. Sci. Found. Workshop Next Generation Data Mining, 2002, pp. 126–133.

[7] F. Giannotti, L. V. Lakshmanan, A. Monreale, D. Pedreschi, and H. Wang, "*Privacy-preserving data mining from outsourced databases,"*in Proc. SPCC2010 Conjunction with CPDP, 2010, pp. 411–426.

[8] S. J. Rizvi and J. R. Haritsa, "*Maintaining data privacy in association rule mining*," in Proc. Int. Conf. Very Large Data Bases, 2002, pp. 682– 693.

[9] M. Kantarcioglu and C. Clifton, "*Privacy-preserving distributed mining of association rules on horizontally partitioned data*," IEEE Trans. Knowledge Data Eng., vol. 16, no. 9, pp. 1026–1037, Sep. 2004.

[10] Bloom B (1970) Space time tradeoffs in hash coding with allowable errors. Commune ACM 7(13):422– 426.

[11] C. Tai, P. S. Yu, and M. Chen, "*K-support anonymity based on pseudo taxonomy for outsourcing of frequent itemset mining*," in Proc. Int.Knowledge Discovery Data Mining, 2010, pp. 473–482.

[12] Kantarcio glu M, Jin J, Clifton C (2004) When do data mining results violate privacy? In: Proceedings of the 10th ACM SIGKDD KDD 2004, pp 599–604.

[13] C. E. Shannon, "*Communication theory of secrecy systems*," Bell Syst. Tech. J., vol. 28, pp. 656–715, 1948.