An Improved Rob Frugal Scheme for Privacy Preserving Data Mining From Outsourced Transaction Databases

Shobana S\textsuperscript{1} Anitha K M\textsuperscript{2} Geetha V\textsuperscript{3} Aileen chirs P\textsuperscript{4}

\textsuperscript{1, 2, 3}Final year ME (CSE) Student, Karpagam University, Coimbatore.
\textsuperscript{4}Asst.Professor, Department of Computer science engineering, Karpagam University, Coimbatore.

Abstract— Privacy preserving data mining (PPDM) has obtained more attraction now days. The important part is that private data is collected from many number of source by a collector for the purpose of consolidating the gathered data. A data vendor needs expertise or computational resources can outsource its mining needs to a third party service provider or server. However, both the items and the organization rules of the outsourced database are considered private property of the corporation (data owner). The data vendor is not trusted with protecting the privacy, so data are subjected to a random agitation as it is collected. Methods have been created for agitating 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.

Index Terms— data privacy, privacy preserving mining, association rule mining, privacy guarantees, FP-growth algorithm, ARM

I. INTRODUCTION

Data privacy is a major distress that intimidates the widespread deployment of data in many domains. We propose an approach for discover the knowledge like, By way of a mining model and from a data, while ensuring that the data is cryptographically safe. It explains some common cryptographic tools [1] and constructs used in several Privacy-preserving data mining techniques. This is made possible by an innovative, yet natural generalization for the accepted trusted third party model and a new privacy-preserving data mining algorithm. Privacy-preserving data mining is to replace each message exchange in an ordinary data mining algorithm with a cryptographic primitive that provides the same information without disclosing the data of the participants: for example, replacing a sum reduction by a cryptographically secured sum reduction in which the participants learn only the final sum and not each other’s partial sums. When practiced well, this approach guarantees the privacy.

One of the main security issues is that the server has access to valuable data of the owner and may learn sensitive information from it. For example, by looking at the transactions, the server (or an intruder who gains access to the server) can learn which items is always co purchased. However, both the transactions and the mined patterns are the property of the data owner and should remain safe from the server. This problem of protecting important private information of organizations/companies is referred to as corporate privacy [2]. There are two approaches that can protect sensitive information. The first is to apply an encryption function that transforms the original data to a new format [3]. The second is to apply data perturbation, which modifies the original raw data randomly. In order for an encryption to be appropriate for the problem, the following conditions should be satisfied.

\begin{center}
\hspace{1cm}
\begin{tikzpicture}
  \node [draw] at (0,0) (D) {D};
  \node [draw] at (2,0) (Encryption) {Encryption function};
  \node [draw] at (4,0) (D*) {D*};
  \draw[->] (D) -- (Encryption);
  \draw[->] (Encryption) -- (D*);
\end{tikzpicture}
\end{center}

Fig. 1. Encryption of Transaction database D

i. First, there should be perfect and well developed decryption method that transforms the association rules found in the encrypted database to the true association rules in the original database.

ii. Second, the encryption and decryption processes must be fast; otherwise, owners may choose to apply association rules mining locally. Here cost is the only concern.

iii. Third, the encryption method must be perfect and secure enough to prevent the attacker from retrieving the original transactions and

iv. Finally, it must be secure enough to prevent the attacker from recovering the true association rules among the actual items by processing the encrypted data.
Our goal is to plan an encryption scheme which enables formal privacy guarantees to be proved, and to validate this model over large-scale real-life transaction databases (TDB). The architecture behind our model is illustrated in Fig. 2. The client/owner encrypts its data using an encrypt/decrypt (E/D)) module, which can be essentially treated as a black box from its perspective. We adopt the idea of substitution ciphers [4,5] in transaction encryption. The encryption schema proposed in this paper is based on, replacing each plain item in $D$ by a 1–1 substitution cipher and Adding fake or antonym transactions to the database.

Fig. 2. Architecture of mining-as-service paradigm.

Encryption and decryption module is responsible for transforming the input data into an encrypted database. The server conducts data mining and sends the (encrypted) patterns to the owner. There are many encryption techniques [3]. Our encryption scheme has the property that the returned supports are not true supports. The encryption and decryption(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. Fig.3. will describe the transformation of encryption and decryption. Thus, the major focus of this paper is to devise encryption schemes such that formal privacy guarantees can be proven against attacks conducted by the server using background knowledge, while keeping the resource requirements under control.

Fig.3. Architecture of the scheme

We make the following contributions. First, we formally define an attack model for the adversary and make the background knowledge the adversary may possess precise. Our notion of privacy requires that, for each ciphertext item, there are at least $k$–1 distinct cipher items that are indistinguishable from the item regarding their supports. Second, we develop an encryption scheme, called Rob Frugal with some improvement that the encryption/decryption module can employ to transform client data before it is shipped to the server.

Third, to allow the encryption and decryption(E/D) module to recover the true patterns and their correct support, we propose that it creates and keeps a compact structure, called synopsis. We also provide the encryption and decryption(E/D) module with an efficient strategy for incrementally maintaining the synopsis against updates in the form of appends. Fourth, we conduct a formal analysis based on our 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 Rob Frugal scheme.Last but not least, we conduct experimental analysis of our schema using a large real dataset from the Coop store chain in Italy. Our results show that our encryption schema is effective, scalable, and achieve the desired level of privacy.

II. RELATED WORKS

A key distinction between this problem and the aforementioned PPDM problems is that, in our setting, not only the underlying data but also the mined results are not intended for sharing and must remain
private. In particular, when the server possesses 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 works that are most related to ours are [6] and [7]. Similar to our study, they assume that the adversary possesses prior knowledge of the frequency of items or item sets, which can be used to try to reidentify the encrypted items. The work [6] utilizes a 1- n item mapping together with nondeterministic addition of cipher items to protect the identification of individual items. A recent paper [8] has formally proven that the encoding system in [6] can be broken without using context-specific information. The success of the attacks in [8] mainly relies on the existence of unique, common, and fake items, defined in [6]; our scheme does not create any such items, and the attacks in [8] are not applicable to our scheme.

A. Data Preprocessing

Data preprocessing explains any type of processing performed on raw data to prepare it for another processing procedure. Commonly used as a preliminary data mining practice, data preprocessing transforms the data into a format that will be more easily and effectively processed for the purpose of the user. Data pre-processing is an often neglected but important step in the data mining process. Data gathering methods are often loosely controlled, resulting in out-of-range values (e.g., Income: -100), impossible data combinations (e.g., Gender: Male, Pregnant: Yes), missing values, etc. Analyzing data that has not been carefully screened for such problems can produce misleading results. Thus, the representation and quality of data is first and foremost before running an analysis. If there is much irrelevant and redundant information present or noisy and unreliable data, then knowledge discovery during the training phase is more difficult. Data preparation and filtering steps can take considerable amount of processing time. Data pre-processing includes cleaning, normalization, transformation, feature extraction and selection, etc.,

B. Frequent pattern Mining

Association Rules Mining (ARM) algorithms are designed to find sets of frequently occurring items in large databases. ARM applications have found their way into a variety of fields, including medicine, biotechnology, and marketing. This class of algorithm is typically very memory intensive, leading to prohibitive runtimes on large databases. We investigate a popular tree-based ARM algorithm (FP-growth), and make use of a systolic tree structure, which mimics the internal memory layout of the original software algorithm while achieving much higher throughput. The FP-growth algorithm extracts the frequent item set in a recursive enumeration manner. Starting from the highest level of the FP-tree toward the root, the item sets ending with item D are checked first. This problem is further divided into several sub problems of finding frequent item set ending with (C,D), (B,D) and (A,D). Each sub problem ending with defined item sets is further divided into smaller problems by building its conditional FP-tree. Each PE in the systolic tree has three modes: WRITE mode, SCAN mode and COUNT mode. The last two modes will be discussed in later sections. When the tree is in building phase, PEs is in WRITE mode. An item is loaded into the control PE each cycle which in turn transfer each item into the general PEs. If the item is already contained in a PE, the corresponding count value will be increased.

C. Encryption

We introduce the encryption scheme, called improved Rob Frugal, which transforms a TDB D into its encrypted version D*. Our 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 encryption and decryption (E/D) module in the form of a compact synopsis.
D. Decryption

When the client requests the execution of a pattern mining query to the server, specifying a minimum support threshold $\sigma$, the server returns the computed frequent patterns from $D^*$. Clearly, for every item set $S$ and its corresponding cipher item set $E$, we have that support $D(S) \leq$ support $D^*(E)$. For each cipher pattern $E$ returned by the server together with support $D^*(E)$, the encryption and decryption (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 encryption and decryption (E/D) module adjusts the support of $E$ by removing the effect of the fake transactions. Support $D(S) = support D^*(E) - support D^\setminus D(E)$. This follows from the fact that support of an item set is additive over a disjoint union of transaction sets. Finally, the pattern $S$ with adjusted support is kept in the output if support $D(S) \geq \sigma$. The calculation of support $D^\setminus D(E)$ is performed by the encryption and decryption (E/D) module using the synopsis of the fake transactions in $D^\setminus D$.

E. Constructing Fake Transactions

Given a noise table specifying the noise $N(e)$ needed for each cipher item $e$, we generate the fake transactions as follows. First, we drop 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, we sort the remaining rows in downward order of noise. This concept shown in Fig.4.

III. APPROACHES TO PRIVACY PRESERVING DATA MINING

Constructing decision trees from private training data, the concepts of privacy were quite different. One was based on data obscuration, i.e., modifying the data values so real values are not disclosed. The other used Secure Multiparty Computation to encrypt data values, ensuring that no party learns anything about another’s data values. We first describe Secure Multiparty Computation, and then give additional background on data obscuration.

A. Secure multiparty computation

The idea of Secure Multiparty Computation (SMC) [2] is that the parties involved learn nothing but the results. Informally, imagine we have a trusted third party to which all parties give their input. The trusted party computes the output and returns it to the parties. SMC enables this without the trusted third party. There may be considerable communication between the parties to get the final result, but the parties don’t learn anything from this communication. The computation is secure if given just one party’s input and output from those runs, we can simulate what would be seen by the party. In this case, to simulate means that the distribution of what is actually seen and the distribution of the simulated view over many runs are computationally indistinguishable. We may not be able to exactly simulate every run, but over time we cannot tell the simulation from the real runs. Since we could simulate the runs from knowing only our input and output, it makes sense to say that we don’t learn anything from the run other than the output. This seems like a strong guarantee of privacy, and has been used in privacy preserving data mining work. We must be careful when using Secure Multiparty computation to define privacy. For example, suppose we use
a SMC technique to build a decision tree from databases at two sites [Lindell & Pinkas2000], classifying people into high and low risk for a sensitive disease. Assume that the non-sensitive data is public, but the sensitive data (needed as training data to build the classifier) cannot be revealed.

Fig.6. Decryption scheme

The SMC computation won't reveal the sensitive data, but the resulting classifier will enable all parties to estimate the value of the sensitive data. It isn't that the SMC was broken, but that the result itself violates privacy.

B. Obscuring data

Another approach to privacy is to obscure data: making private data available, but with enough noise added that exact values (or approximations sufficient to allow misuse) cannot be determined. One approach, typically used in census data, is to aggregate items. Knowing the average income for a neighborhood is not enough to determine the actual income of a resident of that neighborhood. An alternative is to add random noise to data values, then mine the distorted data. While this lowers the accuracy of data mining results, research has shown that the loss of accuracy can be small relative to the loss of ability to estimate an individual item. This would enable data collected from a web survey to be obscured at the source the correct values would never leave the respondent’s machine ensuring that exact (misusable) data doesn't exist. Techniques have also been developed for association rules, enabling valid rules to be learned from data where items have been randomly added to or removed from individual transactions.

IV. IMPROVED ROB FRUGAL SCHEME

Rob Frugal transforms a TDB $D$ into its encrypted version $D^\ast$. Our 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 $D^\ast$, and transmitted to the server. A record of the fake transactions, i.e., $DF = D^\ast \setminus D$, is stored by the encryption and decryption (E/D) module in the form of a compact outline.

1) The new transactions in $\Delta D$ are inserted into the prefix tree $T$, obtaining a cumulative representation of $D \cup \Delta D$. Also, a cumulative item support table IST is constructed by adding the support of each item in IST* and IST$\Delta$. In particular, for each item $e_i \in$ IST*, the support of $e_i$ is added to the support of $e_i \in$ IST$\Delta$. Clearly, IST$\Delta$ 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$\Delta$. 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$\Delta$, 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.

2) 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 $D^\ast \cup F^\ast$. 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.

3) The encryption and decryption(E/D) module uses both old and new synopses to reconstruct the exact support of a pattern from the server. Our 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.

V. EXPERIMENTS

In this section, we report our empirical evaluation to assess the encryption/decryption overhead and the overhead at the server side incurred by the proposed schema.

A. Large Real world Datasets

We experimented on a large real-world database, which is donated to us by Coop, a cooperative of consumers that is today the largest supermarket. We selected 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 15713 different products grouped into 366 marketing categories. Transactions are item sets, i.e., no product occurs twice in the same transaction. We consider two distinct kinds of TDBs:

1) Product-level Coop TDBs, denoted by CoProd, where items correspond to products, and
2) Category level Coop TDBs that we denote by CoCat, where items correspond to the category of the products in the original transactions.

In these datasets, \( l_{max} = 188 \) for CoProd, while \( l_{max} = 90 \) for CoCat. Also, the two kind of TDBs exhibit very different sparsity/density properties, as made evident in Fig. 7(a) and (b), in which we depict the support distribution of the items in CoProd and in CoCat with 300k transactions; we only show the support distribution on these two TDBs because the others are very similar. The heavy-tailed distribution in Fig. 7(a) (many items with very low support) indicates that CoProd is much sparser than CoCat[shown in Fig. 7(b)]. Sparsity/density of the two TDBs has a dramatic effect on pattern mining: the number of frequent patterns found in CoCat tends to explode for higher support thresholds, compared to CoProd. We experimented with our algorithms for both CoProd and CoCat.

![Fig. 7(a) Item support distribution.- CoProd 300k trans.](image1)

![Fig. 7(b) Item support distribution.- CoCat 300k trans.](image2)

B. Experimental Evaluation

We implemented the Rob Frugal encryption scheme, as well as the decryption scheme. We adopted the a priori implementation by Christian Borgelt,2 written in C and one of the most highly optimized implementations.

1) Encryption Overhead:

First, we assessed the total time needed by the ED module to encrypt the database (grouping,synopsis construction, creation of fake transactions): timings are reported for CoProd and CoCat, 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 CoProd TDB, and below 0.8 s for the biggest CoCat TDB. Indeed, it is always less than the time of a single mining query, which is at least 1 s by A priori. Therefore, when there are multiple mining queries, which is always the case for the outsourcing system, the encryption overhead of our 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 $O(n^2)$ is indeed over pessimistic. To see this point, we counted the number of queries (to check that each group is unsupported) performed by the ED module (Rob Frugal), over the two TDBs for the different values of $k$, and we discovered that such number always coincides with $n$, except for CoCat 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 Rob Frugal reaches a solution very fast, with complexity far below the $O(n^2)$ worst case: e.g., for CoCat with $k = 10$ and 400 transactions, Rob Frugal only needs to check a total of 3826 queries, while 3662 = 133, 956! Second, we assessed the size of fake transactions added to the databases after encryption. The sizes of fake transactions for different values of $k$ in CoProd* and CoCat* with 300k transactions. We observe that the size of fake transactions increases linearly with $k$. Also, we observe that sparsity/density affects the generation of fake transactions: e.g., we have that CoProd*, for $k = 30$, is only 8% larger than CoProd while, for the same $k$, CoCat* is 80% larger than CoCat. We also assessed the size of the fake transactions on synthetic databases.

Finally, we assessed the overhead of incremental encryption, which occurs when a new TDB is appended; to this end, we split CoProd with 500k transactions into two halves CoProd1 and CoProd2, and treat CoProd1 as the original TDB and CoProd2 as the appended one. We consider the nonincremental method, which is to encrypt CoProd 1 ∪ CoProd 2 from scratch, and compare its encryption time with that of the incremental approach. We ignore the time for transmitting TDBs between the client and server as we 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. The results are positive: essentially, for any value of $k$, the incremental procedure always achieves better performance than the nonincremental approach. 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 our approach more robust against the possible attack based on the comparison of multiple versions of the encrypted TDB.

2) Mining Overhead:

We studied the overhead at the server side for the pattern mining task over CoProd* with respect to CoProd with 300K transactions. Instead of measuring performance in run time, we 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. We observe that the number of frequent patterns, at a given support threshold, increases with $k$, as expected. However, mining over CoProd* 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 CoCat with 300k transactions and CoCat* is more demanding, given the far higher density, but we have similar observation, although at higher support thresholds. In either case, we 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 patterns from CoProd* 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 CoProd* requires to use a smaller support threshold to have the same patterns that one could have from the original data. For example, for $k = 10$ CoProd* 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 $2 \times 300k/306k$. The need to use a smaller support could make harder the discovery of frequent patterns. But in our experiments, given the sparsity of the real TDBs, we found that the number of fake transactions does not change the support threshold too much, making the problem still tractable.

3) Decryption by the ED Module:

We 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, we need to compare the decryption time with the time of directly executing a priori over the original database. This comparison is particularly challenging, as we have chosen one of the most optimized versions of a priori (written in C), while our decryption method is written in Java without particular optimizations, except for the use of hash tables for the synopsis. 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 CoCat.

4) Crack Probability:

We also analyze the crack probability for transactions and patterns over the Coop TDBs. We discovered that in both CoCat and CoProd TDBs encrypted by Rob Frugal, around 90% of the transactions can be broken with probability strictly less than 1/k. For example, considering the encrypted version of CoProd with 300K transactions, we 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) ≤ 1/10. Around 5% of transactions have exactly a crack probability 1/10, while 95% have a probability strictly smaller than 1/10. Around 90% have a probability strictly smaller than 1/10. No single transaction contains any pattern consisting exactly of the items in a group created by Rob Frugal.

VI. CONCLUSION

In this paper, we studied the problem of corporate Privacy preserving mining of frequent patterns on an encrypted outsourced TDB. We assumed that a conservative model where the adversary knows the domain of items and their exact frequency and can use this knowledge to identify cipher items and cipher itemsets. We proposed an encryption scheme with some upgrade version, called improved Rob Frugal, 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. We also proposed a strategy for incremental maintenance of the synopsis against updates consisting of appends and dropping of old transaction batches. We could study the privacy guarantees of our method in case of known-plaintext attacks, chosen-plaintext attacks, and chosen-cipher text attacks. Another interesting direction is to relax our 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. Our current framework assumes that the attacker does not possess such knowledge. Any relaxation may break our encryption scheme and bring privacy vulnerabilities.

REFERENCES