Privacy-Preserving Mining of Association Rules From Outsourced Transaction Databases

Fosca Giannotti, Laks V. S. Lakshmanan, Anna Monreale, Dino Pedreschi, and Hui (Wendy) Wang

Abstract—**Spurred by developments such as cloud computing, there has been considerable recent interest in the paradigm of data mining-as-a-service. A company (data owner) lacking in expertise or computational resources can outsource its mining needs to a third party service provider (server). However, both the items and the association rules of the outsourced database are considered private property of the corporation (data owner). To protect corporate privacy, the data owner transforms its data and ships it to the server, sends mining queries to the server, and recovers the true patterns from the extracted patterns received from the server. In this paper, we study the problem of outsourcing the association rule mining task within a corporate privacy-preserving framework. We propose an attack model based on background knowledge and devise a scheme for privacy preserving outsourced mining. Our scheme ensures that each transformed item is indistinguishable with respect to the attacker's background knowledge, from at least $k-1$ other transformed items. Our comprehensive experiments on a very large and real transaction database demonstrate that our techniques are effective, scalable, and protect privacy.**

Index Terms—**Association rule mining, privacy-preserving outsourcing.**

I. Introduction

With the advent of cloud computing and its model for IT services based on the internet and big data centers, the outsourcing of data and computing services is acquiring a novel relevance, which is expected to skyrocket in the near future. Business intelligence and knowledge discovery services, such as advanced analytics based on data mining technologies, are expected to be among the services amenable to be externalized on the cloud, due to their data intensive nature, as well as the complexity of data mining algorithms. Thus, the paradigm of mining and management of data as service will presumably grow as popularity of cloud computing grows [1]. This is the 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 [2], [3].

Although it is advantageous to achieve sophisticated analysis on tremendous volumes of data in a cost-effective way, there exist several serious security issues of the data-mining-as-a-service paradigm. 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 are always copurchased. 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 [4]. Unlike personal privacy, which only considers the protection of the personal information recorded about individuals, corporate privacy requires that both the individual items and the patterns of the collection of data items are regarded as corporate assets and thus must be protected.

In this paper, we study the problem of outsourcing the association rule mining task within a corporate privacy-preserving framework. A substantial body of work has been done on privacy-preserving data mining (PPDM) in a variety of contexts. A common characteristic of most of the previously studied frameworks is that the patterns mined from the data (which may be distorted, encrypted, anonymized, or otherwise transformed) are intended to be shared with parties other than the data owner. The key distinction between such bodies of work and our problem is that, in the latter, both the underlying data and the mined results are not intended for sharing and must remain private to the the data owner.

We adopt a conservative frequency-based attack model in which the server knows the exact set of items in the owner's data and additionally, it also knows the exact support of every item in the original data. Wong *et al.* [2] was one of the early works on defending against the frequency-based attack in the data mining outsourcing scenario. They introduced the idea of using fake items to defend against the frequency-based attack; however, it was lacking a formal theoretical analysis of privacy guarantees, and has been shown to be flawed very recently in [5], where a method for breaking the proposed encryption is given. Therefore, in our previous and preliminary work [6], we 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, our goal is to devise an encryption scheme which enables formal privacy guarantees to be proved, and to validate this model over large-scale real-life transaction

Manuscript received September 30, 2011; revised April 26, 2012; accepted July 26, 2012. Date of publication November 30, 2012; date of current version July 3, 2013.

F. Giannotti is with the Information Science and Technology Institute, National Research Council, Pisa 56124, Italy (e-mail: fosca.giannotti@isti.cnr.it).

L. V. S. Lakshman is with the University of British Columbia, Vancouver, BC V1V 1V7, Canada (e-mail: laks@cs.ubc.ca).

A. Monreale and D. Pedreschi are with the University of Pisa, Pisa 56126, Italy (e-mail: annam@di.unipi.it; pedre@di.unipi.it).

H. Wang is with the Stevens Institute of Technology, Hoboken, NJ 07030 USA (e-mail: hui.wang@stevens.edu).

Digital Object Identifier 10.1109/JSYST.2012.2221854

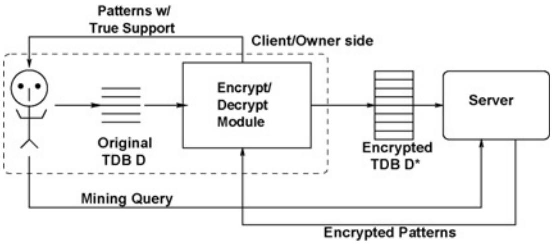


Fig. 1. Architecture of mining-as-service paradigm.

databases (TDB). The architecture behind our model is illustrated in Fig. 1. 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. While the details of this module will be explained in Section V, it is responsible for transforming the input data into an encrypted database. The server conducts data mining and sends the (encrypted) patterns to the owner. Our 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 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.

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 *RobFrugal*, that the E/D module can employ to transform client data before it is shipped to the server.

Third, to allow the 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 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 *RobFrugal* 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.

Related work is described in the next section. The background on frequent pattern mining is quickly reviewed in Section III. Our privacy-preserving outsourcing model and the associated problem statement are given in Section IV. Section V develops the encryption/decryption scheme we use. Section VI provides the key 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, we conclude this paper and discuss directions for future research in Section VIII.

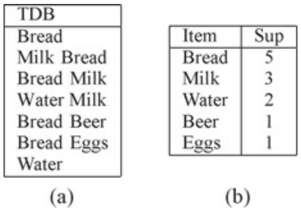
II. Related Work

The research of PPDM has caught much attention recently. The main model here is that private data is collected from a number of sources by a collector for the purpose of consolidating the data and conducting mining. The collector is not trusted with protecting the privacy, so 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 body of work was pioneered 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 disclosed.

Another related issue is secure multiparty mining over distributed datasets. Data on which mining is to be performed is partitioned, horizontally or vertically, and distributed among several parties. 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 consider third parties. This approach partially implements corporate privacy, as local databases are kept private, but it is too weak for our outsourcing problem, as the resulting patterns are disclosed to multiple parties.

The particular problem attacked in our paper is outsourcing of pattern mining within a corporate privacy-preserving framework. 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 itemset corresponding to a given cipher item or itemset with a probability above a given threshold.

The works that are most related to ours are [2] and [12]. 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 [2] utilizes a one-to-n item mapping together with nondeterministic addition of cipher items to protect the identification of individual items. A recent paper [5] has formally proven that the encoding system in [2] can be broken without using context-specific information. The success of the attacks in [5] mainly relies on the existence of unique, common, and fake items, defined in [2]; our scheme does not create any such items, and the attacks in [5] are not applicable to our scheme. Tai *et al.* [12] assumed the attacker knows exact frequency of single items, similarly to us. They use a similar privacy model as ours, which requires that each real item must have the same



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)

Fig. 2. Example of TDB and its support table. (a) TDB. (b) Item support table.

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, we have formal analysis to show that our scheme can always achieve provable privacy guarantee with respect to the background knowledge of the attacker and the notion of privacy. In general, it is prohibitively expensive to achieve perfect secrecy of outsourced frequent itemset mining [5]. We show that with less strict privacy models, we can achieve practical privacy-preserving methods that provide reasonable privacy guarantee. Our empirical 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. Furthermore, an important issue in association rule mining (or frequent item set mining) outsourcing is the ability to deal with updates. Neither of the works above addresses this concern. In contrast, we propose an incremental method for updating the compact synopsis maintained by the owner against updates to the database.

III. Pattern Mining Task

The reader is assumed to be familiar with the basics of association rule mining. We let $I = i_1, \ldots, i_n$ be the set of items and $D = t_1, \ldots, t_m$ a TDB of transactions, each of which is a set of items. We denote the support of an itemset $S \subseteq I$ as $\text{supp}_D(S)$ and the frequency by $\text{freq}_D(S)$. Recall that $\text{freq}_D(S) = \text{supp}_D(S)/|D|$. For each item i, $\text{supp}_D(i)$ and $\text{freq}_D(i)$ denote, respectively, the individual support and frequency of i. The function $\text{supp}_D(.)$, projected over items, is also called the item support table of D represented in tabular form [see, the support table in Fig. 2(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. In this paper, we confine ourselves to the study of a (corporate) privacy-preserving outsourcing framework for frequent pattern mining.

IV. Privacy Model

We let D denote the original TDB that the owner has. To protect the identification of individual items, the owner applies an encryption function to D and transforms it to D^*, the encrypted database. We refer to 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. We use $\mathcal{I}$ to denote the set of plain items and $\mathcal{E}$ to refer to the set of cipher items.

A. Adversary Knowledge

The server or an intruder who gains access to it may possess some background knowledge using which they can conduct attacks on the encrypted database D^*. We generically refer to any of these agents as an attacker. We adopt a conservative model and assume that the attacker knows exactly the set of (plain) items $\mathcal{I}$ in the original TDB D and their true supports in D, i.e., $\text{supp}_D(i)$, $\forall i \in \mathcal{I}$. The attacker may have access to similar data from a competing company, may read published reports, etc. In reality, the attacker may possess 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, we adopt the conservative assumption that he knows the exact support of every item.

Note that as the attacker has access to the encrypted database D^*, he also knows the supports $\text{supp}_{D^*}(e)$, $e \in \mathcal{E}$, where $\mathcal{E}$ is the set of cipher items in the encrypted database D^*. The encryption schema proposed in this paper are based on: 1) replacing each plain item in D by a 1–1 substitution cipher and 2) adding fake transactions to the database. In particular, no new items are added. We assume the attacker knows this and thus he knows that $|\mathcal{E}| = |\mathcal{I}|$. Essentially, compared to [2], our adversary knowledge model corresponds to a $(100\%, 0\%)$ knowledge model, confined to single items. However, we assume the attacker 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

We assume the service provider (who can be an attacker) is semihonest in the sense that although he does not know the details of our encryption algorithm, he can be curious and thus can use his background knowledge to make inferences on the encrypted transactions. We also assume that the attacker always returns (encrypted) item sets together with their exact support.

The data owner (i.e., the corporate) considers the true identity of: 1) every cipher item; 2) every cipher transaction; and 3) every cipher frequent pattern as the intellectual property which should be protected. We consider the following attack model.

1) Item-based attack: $\forall$ cipher item $e \in \mathcal{E}$, the attacker constructs a set of candidate plain items $Cand(e) \subset \mathcal{I}$. The probability that the cipher item e can be broken $\text{prob}(e) = 1/|Cand(e)|$.
2) Set-based attack: Given a cipher itemset E, the attacker constructs a set of candidate plain itemsets $Cand(E)$, where $\forall X \in Cand(E)$, $X \subset \mathcal{I}$, and $|X| = |E|$. The probability that the cipher itemset E can be broken $\text{prob}(E) = 1/|Cand(E)|$.

We refer to $\text{prob}(e)$ and $\text{prob}(E)$ 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 we 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. We say D^* satisfies the property of item k-privacy provided for every cipher item $e \in \mathcal{E}$, if there are at least $k-1$ other distinct cipher items $e_1, ..., e_{k-1} \in \mathcal{E}$ such that $\text{supp}_{D^*}(e) = \text{supp}_{D^*}(e_i)$, $1 \leq i \leq k-1$. ■

The concept of item k-anonymity is similar to the k-support anonymity [12] (based on the well-known k-anonymity [14], [15]) as we also require that for each ciphertext item e, there are at least $k-1$ distinct cipher items that are indistinguishable from e regarding their supports.

C. Problem Statement

To quantify the privacy guarantees of an encrypted database, we define the following notion.

Definition 2: Given a database D and its encrypted version D^*, we say D^* is k-private if: 1) for each cipher item $e \in D^*$, $\text{prob}(e) \leq 1/k$; and 2) for each cipher itemset E with support $\text{supp}_{D^*}(E) > 0$, $\text{prob}(E) \leq 1/k$. ■

Formally, the problem we study 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, we would 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, we introduce the encryption scheme, called *RobFrugal*, 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., $DF = D^* \setminus D$, 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, we have that $\text{supp}_D(S) \leq \text{supp}_{D^*}(E)$. For each cipher pattern E returned by the server together with $\text{supp}_{D^*}(E)$, 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. $\text{supp}_D(S) = \text{supp}_{D^*}(E) - \text{supp}_{D^* \setminus D}(E)$. 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 $\text{supp}_D(S) \geq \sigma$. The calculation of $\text{supp}_{D^* \setminus D}(E)$ is performed by the E/D module using the synopsis of the fake transactions in $D^* \setminus D$.

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 $\text{supp}_{D^* \setminus D}(E)$ for every cipher pattern. In order to realize the encryption scheme efficiently, we need to address the following technical issues.

1) How do we cluster items into groups of k?
2) How do we create the needed fake transactions?
3) How is the synopsis represented and stored?
4) How is the true support recovered efficiently?

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. We start from a simple grouping method called Frugal. We assume the item support table is sorted in descending order of support and refer to cipher items in this order as e_1, e_2, 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 e_1. ■

Assume $e_1, e_2, \ldots, e_n$ is the list of cipher items in descending order of support (with respect to D), the groups created by Frugal are $\{e_1, \ldots, e_k\}$, $\{e_{k+1}, \ldots, e_{2k}\}$, and so on. The last group, if less than k in size, is merged with its previous group. We denote the grouping obtained using the above definition as G^{frug}. For example, consider the example TDB and its associated (cipher) item support shown in Fig. 2. For $k = 2$, G^{frug} has two groups: $\{e_2, e_4\}$ and $\{e_5, e_1, e_3\}$. This corresponds to the partitioning groups shown in Table I(a). Thus, in D^*, the support of e_4 will be brought to that of e_2; and the support of e_1 and e_3 brought to that of e_5.

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 $\frac{1}{k}$? 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 $k = 2$, $\{e_2, e_4\}$ [see Table I(a)] is supported in D, because e_2, e_4 occur together in a transaction of D. Therefore, there

TABLE I

GROUPING WITH $k = 2$

(a) Frugal

Item	Support
e_2	5
e_4	3
e_5	2
e_1	1
e_3	1

(b) *RobFrugal*

Item	Support
e_2	5
e_5	2
e_4	3
e_1	1
e_3	1

TABLE II

NOISE TABLE AND ITS HASH TABLE

(a) Noise table for $k = 2$

Item	Support	Noise
e_2	5	0
e_5	2	3
e_4	3	0
e_1	1	2
e_3	1	2

(b) Hash tables of items of nonzero noise in (a)

	Table1
0	$\langle e_5, 1, 2\rangle$
1	$\langle e_3, 2, 0\rangle$

	Table2
0	$\langle e_1, 2, 0\rangle$

only exists one itemset candidate of $\{e_2, e_4\}$, i.e., the privacy guarantee is 1-privacy.

To fix the privacy vulnerabilities of Frugal, we 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 $G^{\text{frug}} = (G_1, ..., G_m)$, the grouping method *RobFrugal* consists in modifying the groups of G^{frug} by repeating the following operations, until no group of items is supported in D: 1) select the smallest $j \geq 1$ such that $\text{supp}_D(G_j) > 0$; 2) find the most frequent item $i' \notin G_j$ such that, for the least frequent item i of G_j we have: $\text{supp}_D(G_j \setminus \{i\} \cup \{i'\}) = 0$; 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 e_4 and e_5 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, we 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 G_i of the initial grouping G^{frug} 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 our 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, we have found that even for small values of $k = 10$ to 50, a *RobFrugal* grouping scheme does exist. This was the case in all our 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. We denote 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^*.

D. 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 descending order of noise. Let $e'_1, \ldots, e'_m$ be the obtained ordering of (remaining) cipher items, with associated noise $N(e'_1), \ldots, N(e'_m)$. The following fake transactions are generated:

1) $N(e'_1) - N(e'_2)$ instances of the transaction $\{e'_1\}$;
2) $N(e'_2) - N(e'_3)$ instances of the transaction $\{e'_1, e'_2\}$;
3) $\ldots$;
4) $N(e'_{m-1}) - N(e'_m)$ instances of the transaction $\{e'_1, \ldots, e'_{m-1}\}$;
5) $N(e'_m)$ instances of the transaction $\{e'_1, \ldots, e'_m\}$.

Continuing the example, we consider cipher items of nonzero noise in Table II(a). The following two fake transactions are generated: two instances of the transaction $\{e_5, e_3, e_1\}$ and one instance of the transaction $\{e_5\}$. 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, we use a hash table generated with a minimal perfect hash function [16]. 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 $[0 \dots n-1]$. Hence, h is a minimal perfect hash function over a set S if and only if $\forall i, j \in S$, $h(j) = h(i)$ implies $j = i$, and there exists an integer p such that the range of h is $p, \dots, p + |S| - 1$. A minimal perfect hash function h is order-preserving if for any keys j and i, $j < i$ implies $h(j) < h(i)$.

In our scheme, the items of the noise table e_i with $N(e_i) > 0$ are the keys of the minimal perfect hash function. Given e_i, function h computes an integer in $[0 \dots n-1]$, denoting the position of the hash table storing the triple of values $\langle e_i, \mathrm{times}_i, \mathrm{occ}_i \rangle$, where times_i represents the number of times that the fake transaction $\{e_1, e_2, \dots, e_i\}$ occurs in the set of fake transactions, and occ_i is the number of times that e_i occurs altogether in the future fake transactions after the transaction $\{e_1, e_2, \dots, e_i\}$.

Given a noise table with m items with nonnull noise, our approach generates hash tables for the group of items. In general, the ith entry of a hash table HT containing the item e_i has $\mathrm{times}_i = N(e_i) - N(e_{i+1})$, $\mathrm{occ}_i = \sum_{j=i+1}^{g} N(e_j)$, 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 $g \leq l_{\max}$ items. The hash tables for the items of nonzero noise in Table II(a) are shown in Table II(b). Finally, we use 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, we can define the function RS that allows an efficient computation of the real support of a pattern $E = \{e_1, e_2, \dots, e_n\}$ with fake support s as follows: $\mathrm{RS}(E) = s - (\mathrm{HT}[h(e_{\max})].\mathrm{times} + \mathrm{HT}[h(e_{\max})].\mathrm{occ})$, where: *i*) $e_{\max}$ is the item in E such that for $1 \leq j \leq n$, we have $h(e_j) \leq h(e_{\max})$, and *ii*) $\mathrm{HT} = H(e_i)$ is the hash table associated by H to any item e_i of E. For example, in Table I(b), for $E_1 = \{e_5\}$, $\mathrm{RS}(E_1) = s_1 - (1 + 2)$, whereas for $E_2 = \{e_5, e_3\}$, $\mathrm{RS}(E_2) = s_2 - (2 + 0)$, where s_i is the fake support of E_i. This is exactly right since e_5 is fakely added three times while e_3 is fakely added two times.

E. Incremental Maintenance

We now consider incremental maintenance of the encrypted TDB. The E/D module is responsible for this. We focus on batches of appends, which are very natural in data warehouses. Let D be an initial TDB and ΔD 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 $\mathrm{syn}(D, D^*)$ 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 ΔD arrives, together with its item support table IST_Δ. 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 ΔD 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_Δ. In particular, for each item $e_i \in \mathrm{IST}^*$ the support of e_i is added to the support of $e_i \in \mathrm{IST}_\Delta$. Clearly, IST_Δ 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_Δ. 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_Δ, 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 $D^* \cup F^*$. 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.
3) The 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. Details can be found in [17].

VI. Analysis

We now provide the main technical results on the robustness and the effectiveness of our encryption/decryption schema.

A. Complexity

Our complexity analysis shows that the encryption method requires $O(n)$ storage and $O(n^2)$ 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, we find 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 we assume the attack does not know the details of our 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, $\text{supp}_D(i) \leq \text{supp}_{D^*}(e)$, 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 $\text{supp}_{D^*}(e)$ and $\text{supp}_D(i)$, and that $\text{supp}_D(i) \leq \text{supp}_{D^*}(e)$. 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 $\text{supp}_D(i') \leq \text{supp}_{D^*}(e)$ are candidates for e. However, this can be narrowed down substantially as follows. Let e_n 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 e_n and let $S = \{i' \mid \text{supp}_D(i) \leq \text{supp}_{D^*}(e_n)\}$. By the grouping established using *RobFrugal*, we must have $|S| = |E|$ and $\forall e \in E$, the set of candidate items must be S. Now, consider any cipher item e with support $\text{supp}_{D^*}(e) > \text{supp}_{D^*}(e_n)$. It is easy to see that any item $i \in S$ corresponding to e_n 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 $ICand(e) = \{i' \mid \text{supp}_D(i') \leq \text{supp}_{D^*}(e)\}$ be the initial candidate set $\forall e \in \mathcal{E}$. The attacker can sort the cipher items in nondecreasing order of their frequency in D^*. Let $S = \{e_1, ..., e_m\}$ be the set of cipher items with the smallest support in D^*. He can infer $ICand(e_1) = \cdots = ICand(e_m)$ and $|ICand(e_i)| = m (1 \leq i \leq m)$. Clearly, every cipher item $e_i \in S$ must be mapped to a plain item in $ICand(e_i)$, and no cipher item in $\mathcal{E} - S$ can be mapped to a plain item in $ICand(e_i)$, 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 $e \in \mathcal{E} - S$. The attack can repeat the procedure on the remaining list of $\mathcal{E} - S$ until he prunes the initial candidate set of every cipher item. Denote the set of candidates for a cipher item e as $Cand(e) \forall e \in \mathcal{E}$. Define a mapping $\iota : \mathcal{E} \rightarrow \mathcal{I}$ to be consistent provided $\text{supp}_D(\iota(e)) \leq \text{supp}_{D^*}(e)$. An assignment $e \mapsto i$ of an item to a cipher item is feasible if there is a consistent mapping $\iota : \mathcal{E} \rightarrow \mathcal{I}$ such that $\iota(e) = i$. 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.

Theorem 1: For every cipher item $e \in \mathcal{E}$, let $Cand(e)$ be the corresponding candidate set computed by the above pruning procedure. Then, every candidate set $Cand(e)$ is minimal. Furthermore, $Cand(e) = \{i' \mid \text{supp}_{D^*}(e') = \text{supp}_{D^*}(e)\}$, where i' is the true plain item corresponding to e'. ■

The proof of Theorem 1 is straightforward from the pruning procedure. Following the construction details of fake items for *RobFrugal*, it is easy to see that for each ciphertext item $e \in \mathcal{E}$, $Cand(e)$ must contain at least k items. Assuming every candidate item is equally likely to be the true plain item corresponding to a given cipher item. We have:

Theorem 2: Let D be a TDB and D^* the encrypted database produced by the *RobFrugal* scheme. Then, for every cipher item e, the probability of its crack is bounded by $\text{prob}(e) \leq 1/k$, where k is a given parameter for item k-anonymity. ■

The theorem shows that the probability that an individual item is broken can always be controlled to be below a threshold chosen by the owner. By controlling the parameter k, the owner can control the crack probability of cipher items. This is exactly in the same spirit as in the classical notion of k-anonymity in the case of microdata [14].

Against the set-based attack: Consider a cipher itemset $E = \{e_1, e_2, \ldots, e_m\}$ in D^*, and suppose that this itemset has support $\text{supp}_{D^*}(E) > 0$ (patterns with zero support are uninteresting). Note that the itemset can be a transaction or a pattern. The attacker can construct the possible candidate sets for E as follows.

Definition 5: The set of possible plain item candidate sets $Cand(E)$ for E are defined as follows: $\forall$ cipher item $e_j \in E$, pick any plain item i_j from $Cand(e_j)$, making sure that for any $e_j, e_\ell \in E$, $j \neq \ell$, the chosen plain items $i_j \in Cand(e_j)$ and $i_\ell \in Cand(e_\ell)$ are distinct. A plain itemset belongs to $Cand(E)$ iff it is generated using the above step. ■

It is worth emphasizing that $Cand(e)$ for a cipher item is a candidate set of items whereas $Cand(E)$ for a cipher itemset denotes the set of candidate itemsets for E. Note that by the construction of D^*, each cipher item is indistinguishable from at least $k - 1$ other cipher items, based on support. Thus, given a cipher itemset E, the attacker can map each cipher item $e_j \in E$ independently to some distinct item plain item i_ℓ such that e_ℓ is indistinguishable from e_j. This is the intuition behind the candidate set of itemsets $Cand(E)$.

Given a cipher itemset E, the attacker finds the candidate set of itemsets $Cand(E)$. Assuming equal likelihood, he can guess the correct itemset corresponding to the given cipher itemset with probability $\text{prob}(E) = 1/|Cand(E)|$. We refer to the probability $\text{prob}(E)$ as the crack probability of E. Given an encrypted database, determining the crack probability $\text{prob}(E)$ for a cipher itemset E requires that we determine the size $|Cand(E)|$ of its candidate set of possible itemsets. We make use of the following notion.

Definition 6: Let E be any cipher itemset and $e_i, e_j \in E$ any two cipher items. Then, $e_i \equiv e_j$ iff $Cand(e_i) = Cand(e_j)$. We denote by $[e_i]$ the equivalence class containing e_i, i.e., the set $\{e \in E \mid e \equiv e_i\}$. ■

To determine the size of $Cand(E)$, consider the hypergraph H_E with nodes E whose edges are the sets $Cand'(e)$, where $e \in E$ and $Cand'(e)$ denote the set obtained by replacing every plain item in $Cand(e)$ by its substitution cipher. Clearly, E is a transversal of this hypergraph, i.e., it has a nonempty overlap with every edge of the hypergraph. The size of the set $Cand(E)$ can be determined as follows. For every edge S of the hypergraph H_E, the contribution of that edge to the number of candidates is given by $\binom{|S|}{|S \cap E|}$. The size of $Cand(E)$ is the product of the contributions from all the hyperedges of H_E.

We call an equivalence class $C \subseteq E$ of cipher items in E complete if $\exists e \in C : Cand'(e) = C$, that is, the equivalence class includes all cipher items in the set $Cand'(e)$. Clearly, the contribution of a complete equivalence class to the size of $Cand(E)$ is a factor of 1. Let C be an equivalence class in E. We denote by $Cand'(E)$ the set $Cand'(e)$ for any element $e \in C$. This is well defined since $\forall e, e' \in C$, we have $Cand'(e) = Cand'(e')$. We now have Theorem 3.

Theorem 3: Given a cipher itemset $E = \{e_1, e_2, \dots, e_m\}$, let C_1, ..., C_t be the collection of equivalence classes of E. Then, the size of the candidate set of itemsets is $|Cand(E)| = \Pi_{i=1}^{t} \binom{|Cand(C_i)|}{|C_i|}$. ■

Proof: Recall that E is a union of one or more equivalence classes. Since construction of candidate itemsets from each equivalence class is independent of each other, $|Cand(E)|$ equals to product of $\binom{|Cand(C_i)|}{|C_i|}$, the size of candidate itemsets constructed from the equivalent class C_i. Since $|Cand(C_i)| > |C_i|$ and $|Cand(C_i)| \geq k$, the result follows. ■

In *RobFrugal*, cipher itemsets that are complete cannot exist with nonzero (fake) support. In fact, we can show Theorem 4.

Theorem 4: Given the original TDB D, let D_r^* be its encrypted version obtained using any robust grouping scheme. Then $\forall$ itemset E with nonzero support in D_r^*, the crack probability prob$(E) \leq 1/k$, where k is the given threshold for k-anonymity. ■

The key to prove the correctness of Theorem 4 is to show that no cipher itemset can be complete under the *RobFrugal* scheme. Assume there is a complete cipher itemset. Then, E must be the union of one or more complete equivalence classes. In other words, every equivalence class in E has nonzero support in D_r^* . This contradicts the property ensured by the construction of *RobFrugal*. Thus, there must exist at least one equivalence class that is not complete. Theorem 3 has shown that the bound of the candidate itemset for each incomplete equivalence class is at least k. Thus the size of candidate itemset for E must be at least k. The theorem follows.

From Theory to Practice. Although the theoretical results demonstrate a remarkable guarantee of protection against the two kind of attacks, presented in Section IV, and the practicability and the effectiveness of the proposed schema, through our experiments on both real-world and synthetic transactional databases we observed that both privacy protection and run time performance are much better than the theoretical worst-cases suggested by the above results. Why?

Concerning privacy, the explanation is that the probability of crack generally decreases with the size of the itemset: $\frac{1}{k}$ is an upper bound that essentially applies only to individual items, not itemsets (under the hypothesis that the adopted grouping is robust). Concerning performance, the explanation is in the item support distribution. In real-life transaction datasets, the item support distribution (as well as the itemset support distribution) follows a power law: the item at rank x in the item support table has a support that is proportional to $\frac{\alpha}{x^\beta}$, for some parameters α and β. This is a natural assumption in real-life TDB, studied in depth in [18]; in our experiments over the Coop TDB, described in Section VII, we found $\beta \approx 0.5$ and $\alpha \approx 30.000$. The power-law distribution implies that there are a few items with large support and a heavy right-skewed tail of items with very low support [see Fig. 3(a)].

Concerning the run time performance, the power-law distribution also facilitates the search for a robust grouping: the identification of robust k-groups becomes quicker and quicker while proceeding from left to right in the item distribution, as the probability that k items in the same group do not co-occur in any transactions grows fast. All our experiments confirm that for all values of k, the actual incurred overhead of using *RobFrugal* is negligible, far below the theoretical worst-case $O(n^2)$ complexity.

VII. 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. Datasets

We experimented 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.[1] 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 15 713 different products grouped into 366 marketing categories. Transactions are itemsets, i.e., no product occurs twice in the same transaction. We consider two distinct kinds of TDBs: 1) product-level Coop TDBs, denoted by *CoopProd*, where items correspond to products, and 2) category-level Coop TDBs, that we denote by *CoopCat*, where items correspond to the category of the products in the original transactions. In these datasets, $l_{\max} = 188$ for *CoopProd*, while $l_{\max} = 90$ for *CoopCat*. Also, the two kind of TDBs exhibit very different sparsity/density properties, as made evident in Fig. 3(a) and (b), in which we depict the support distribution of the items in *CoopProd* and in *CoopCat* with 300 000 transactions; we only show the support distribution on these two TDBs because the others are very similar. The heavy-tailed distribution in Fig. 3(a) (many items with very low support) indicates that *CoopProd* is much sparser than *CoopCat* [shown in Fig. 3(b)]. 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*. We experimented with our algorithms for both *CoopProd* and *CoopCat*.

B. Experimental Evaluation

We implemented the *RobFrugal* encryption scheme, as well as the decryption scheme, as described in Section V,

[1] Available at http://www.e-coop.it, in Italian.

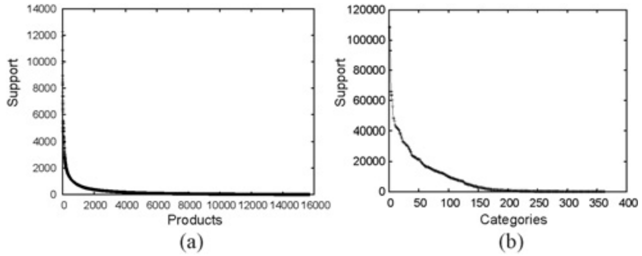


Fig. 3. Item support distribution. (a) *CoopProd* 300*k* trans. (b) *CoopCat* 300*k* trans.

in Java. All experiments were performed on an intel Core2 Duo processor with a 2.66 GHz CPU and 6 GB RAM over a Linux platform (ubuntu 8.10). 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 in Fig. 4 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. Indeed, it is always less than the time of a single mining query, which is at least 1 s by Apriori, as shown in Fig. 5(d). 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 overpessimistic. To see this point, we 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 we discovered that such number always coincides with $\frac{n}{k}$, 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 $O(n^2)$ worst case: e.g., for *CoopCat* with $k = 10$ and 400 transactions, *RobFrugal* only needs to check a total of 3826 queries, while $366^2 = 133,956$!

Second, we assessed the size of fake transactions added to the databases after encryption. Fig. 5(c) reports the sizes of fake transactions for different values of k in *CoopProd** and *CoopCat** with 300*k* 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 *CoopProd**, for $k = 30$, is only 8% larger than *CoopProd* while, for the same k, *CoopCat** is 80% larger than *CoopCat*. 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 *CoopProd* with 500*k* transactions into two halves $CoopProd_1$ and $CoopProd_2$, and treat $CoopProd_1$ as the original TDB and $CoopProd_2$ as the appended one. We consider the nonincremental method, which is to encrypt $CoopProd_1 \cup CoopProd_2$ 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, shown in Fig. 4(c), 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 *CoopProd** with respect to *CoopProd* 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 in Fig. 5(a), 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 (300*k*); 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 *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 300*k* transactions and *CoopCat** is more demanding, given the far higher density, but we have similar observation, although at higher support thresholds [see Fig. 5(b)]. 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 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 (300*k* 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 Overhead 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

[2] Available at http://www.borgelt.net.

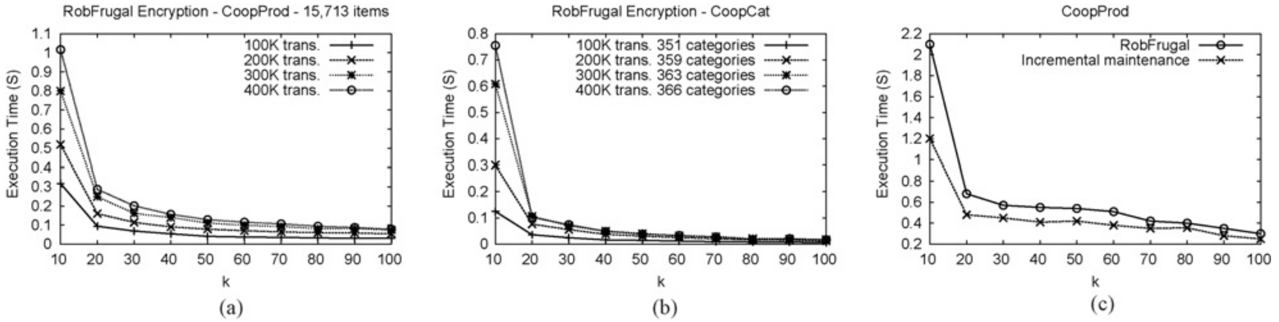


Fig. 4. Encryption overhead. (a) Encryption overhead on CoopProd. (b) Encryption overhead on CoopCat. (c) Overhead of incremental encryption.

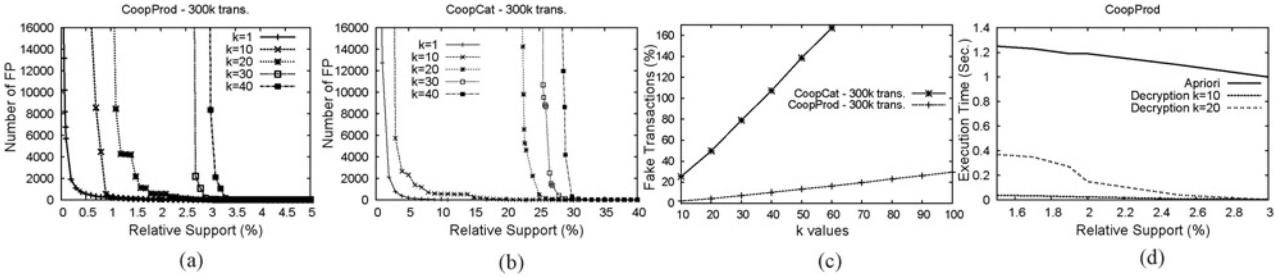


Fig. 5. Overhead at server side and decryption overhead. (a) Mining overhead CoopProd. (b) Mining overhead CoopCat. (c) Fake transactions. (d) Decryp. versus mining time.

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. Nevertheless, as shown in Fig. 5(d), 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:* We also analyze the crack probability for transactions and patterns over the Coop TDBs. We discovered that in both *CoopCat* and *CoopProd* TDBs encrypted by *RobFrugal*, around 90% of the transactions can be broken with probability strictly less than $\frac{1}{k}$. For example, considering the encrypted version of *CoopProd* 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., $\text{prob}(E) \leq \frac{1}{10}$. Around 5% of transactions have exactly a crack probability $\frac{1}{10}$, while 95% have a probability strictly smaller than $\frac{1}{10}$. Around 90% have a probability strictly smaller than $\frac{1}{100}$. 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, we studied the problem of (corporate) privacy-preserving mining of frequent patterns (from which association rules can easily be computed) 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, 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 $\geq k-1$ 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. Unlike previous works, such as [2] and [12], we formally proved that our method is robust against an adversarial attack based on the original items and their exact support. Our experiments based on both large real and synthetic datasets yield strong evidence in favor of the practical applicability of our approach.

Currently, our privacy analysis is based on the assumption of equal likelihood of candidates. It would be interesting to enhance the framework and the analysis by appealing to cryptographic notions such as perfect secrecy [19]. Moreover, our work considers the ciphertext-only attack model, in which the attacker has access only to the encrypted items. It could be interesting to consider other attack models where the attacker knows some pairs of items and their cipher values. For example, we could study the privacy guarantees of our method in case of known-plaintext attacks (where the adversary knows some item, cipher item pairs), chosen-plaintext attacks (where the attacker knows some item and cipher pairs for selected items), and chosen-ciphertext attacks (where the adversary knows some itemset and cipher pairs for selected ciphers). 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. We will investigate encryption schemes that can resist such privacy vulnerabilities. We are also interested in exploring how to improve the *RobFrugal* algorithm to minimize the number of spurious patterns.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] R. Agrawal and R. Srikant, "Privacy-preserving data mining," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 439–450.

[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] B. Gilburd, A. Schuster, and R. Wolff, "k-ttp: A new privacy model for large scale distributed environments," in *Proc. Int. Conf. Very Large Data Bases*, 2005, pp. 563–568.

[11] 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.

[12] 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.

[13] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. Int. Conf. Very Large Data Bases*, 1994, pp. 487–499.

[14] P. Samarati, "Protecting respondents' identities in microdata release," *IEEE Trans. Knowledge Data Eng.*, vol. 13, no. 6, pp. 1010–1027, Nov. 2001.

[15] V. Ciriani, S. D. C. di Vimercati, S. Foresti, and P. Samarati, "*k*-anonymity," in *Proc. Secure Data Manage. Decentralized Syst.*, 2007, pp. 323–353.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2001.

[17] F. Giannotti, L. V. Lakshmanan, A. Monreale, D. Pedreschi, and H. Wang, "Privacy-preserving outsourcing of association rule mining," ISTI-CNR, Pisa, Italy, Tech Rep. 2009-TR-013, 2009.

[18] K.-T. Chuang, J.-L. Huang, and M.-S. Chen, "Power-law relationship and self-similarity in the itemset support distribution: Analysis and applications," *Very Large Data Bases J.*, vol. 17, no. 5, pp. 1121–1141, 2008.

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

Fosca Giannotti received the Masters degree in computer science from the University of Pisa, Pisa, Italy.

She is currently a Senior Researcher with the Information Science and Technology Institute, National Research Council, Pisa, where she leads the Knowledge Discovery and Data Mining Laboratory—a joint research initiative with the University of Pisa, Pisa. She has been the Coordinator of various European-wide research projects, including Geographic Privacy-Aware Knowledge Discovery and Delivery. Her current research interests include data mining query languages, mobility data mining, privacy preserving data mining, and complex network analysis.

Ms. Giannotti has served as the Program Committee Chair and a Program Committee Member in the main conferences on databases and data mining.

Laks V. S. Lakshmanan received the B.E. degree in electronics and communications from the A. C. College of Engineering and Technology, Karaikudi, India, and the M.E. and Ph.D. degrees in computer science from the Indian Institute of Science, Bangalore, India.

He is currently a Professor of computer science with the University of British Columbia, Vancouver, BC, Cananda. He collaborates with both the industry and academia around the world. His current research interests include relational, object-oriented, XML databases, data models, data warehousing, data cleaning and mining, data integration, social or information networks, search, and recommender systems. He is a Research Fellow with the British Columbia Advanced Systems Institute, Vancouver.

Mr. Lakshmanan has served on the program committees of all top database and data mining conferences, chaired several, and edited several special issues of top journals. Currently, he is an Associate Editor of the *Very Large Data Bases Journal*.

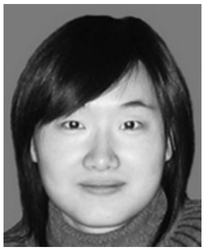
Anna Monreale received the B.S., M.S., and Ph.D. degrees in computer science from the University of Pisa, Pisa, Italy.

She is currently a Post-Doctoral Researcher with the Department of Computer Science, University of Pisa, and a member of the Knowledge Discovery and Data Mining Laboratory—a joint research group with the Information Science and Technology Institute of the National Research Council, Pisa. She has been a Visiting Student with the Department of Computer Science, Stevens Institute of Technology, Hoboken, NJ, since 2010. Her current research interests include anonymity of complex forms of data, including sequences, trajectories of moving objects and complex networks, and privacy-preserving outsourcing of analytical and mining tasks.

Dino Pedreschi received the Masters and Ph.D. degrees in computer science from the University of Pisa, Pisa, Italy.

He is currently a Full Professor of computer science with the University of Pisa, where he has served as the Coordinator of undergraduate studies in computer science and as the Vice Rector. His current research interests include data mining and privacy-preserving data mining (PPDM).

Mr. Pedreschi is a member of program committees of the main international conferences on data mining and knowledge discovery and is an Associate Editor of the *Journal of Knowledge and Information Systems*. He was a recipient of the Google Research Award in 2009 for his research on PPDM and anonymity-preserving data publishing.



Hui (Wendy) Wang received the B.S. degree in computer science from Wuhan University, Wuhan, China, in 1998, and the M.S. and Ph.D. degrees in computer science from the University of British Columbia, Vancouver, BC, Canada, in 2002 and 2007, respectively.

She has been an Assistant Professor with the Department of Computer Science, Stevens Institute of Technology, Hoboken, NJ, since 2008. Her current research interests include data management, database security, and data privacy.