Privacy-preserving Mining of Association Rules from Outsourced Transaction Databases*

Fosca Giannotti[1], Laks V.S. Lakshmanan[2], Anna Monreale[1,3], Dino Pedreschi[3], and Hui Wang[4]

[1] ISTI-CNR, Pisa, Italy, fosca.giannotti@isti.cnr.it
[2] University of British Columbia, Vancouver, Canada laks@cs.ubc.ca
[3] University of Pisa, Pisa, Italy, annam@di.unipi.it, pedre@di.unipi.it
[4] Stevens Institute of Technology, NJ, USA hwang@cs.stevens.edu

Abstract. Spurred by developments such as cloud computing, there has been considerable recent interest in the paradigm of data mining-as-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 a scheme for privacy preserving outsourced mining and show that the owner can recover the true patterns as well as their support by maintaining a compact synopsis.

1 Introduction

In recent years, there has been considerable interest in the data mining-as-service paradigm for enabling organizations with limited computational resources and/or data mining expertise to outsource their data mining needs to a third party service provider [2, 9, 6, 5, 13]. As an example, the operational transactional data from various outlets of Safeway, a grocery chain operating in the US and Canada, can be shipped to a third party which provides mining service for Safeway. The Safeway management need not employ an in-house team of data mining experts. Besides, they can cut down their local data management requirements because periodically data is shipped to the service provider who is in charge of maintaining it and conducting mining on it in response to requests from Safeway's business analysts. In this example, Safeway, the *client*, is a data *owner* and the service provider is referred to as the *server*. One of the main issues with this paradigm is that the server has access to valuable data of the owner and may learn sensitive information from it. E.g., by looking at the transactions, the server (or an intruder who gains access to the server) can learn which items are co-purchased, and in turn, the mined patterns. However, *both the transactions and the mined patterns*

* Extended Abstract

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

are the property of Safeway and should remain safe from the server. This problem of protecting important private information of organizations/companies is referred to as "*corporate privacy*" [7]. 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 (already presented in [8]). We propose

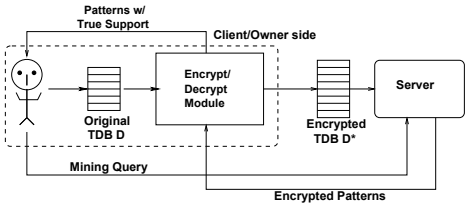


Fig. 1. Architecture of Mining-as-Service Paradigm.

an encryption scheme which enables privacy guarantees, and show some preliminary results obtained applying this model over large-scale, real-life transaction databases. The architecture behind our model is illustrated in Fig. 1. The client/owner encrypts its data using an encrypt/decrypt (ED) module, which can be treated as a "black box" from its perspective. This 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. Our encryption scheme has the property that the returned supports are not true supports. The ED module recovers the true identity of the returned patterns as well their true supports.

2 Related Work

In this section we outline the work on privacy-preserving data publishing and mining.
Privacy-preserving data publishing (PPDP): The idea is that data is published with appropriate suppression, generalization, distortion, and/or decomposition such that individual privacy is not compromised and yet the published data is useful for mining [12, 10, 14].
Privacy-preserving data mining (PPDM): 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, so data is subjected to a random perturbation as it is collected. This body of work was pioneered by [2] and has been followed up by several papers since [11, 3].
Privacy-preserving pattern publishing (PPPP): The central question is how to publish results of mining such as frequent patterns without revealing any sensitive information about the underlying data [4], but the resulting patterns are disclosed.

A key distinction between our problem and the above mentioned PPDM problems is that, in our setting, not only the underlying data but also the mined results are not

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

intended for sharing and must remain private. The work that is most related to ours is [13]. Similar to our work, first, they utilize a one-to-n item mapping together with non-deterministic addition of cipher items to protect the identification of individual items. Second, they assume that the adversary may possess some prior knowledge of frequency of the itemsets, which can be used to decipher the encrypted items. While our attack model focuses on single items with the assumption that the attacker knows the exact frequency of every single item. The major issue left open by [13] is a formal protection result: their security analysis is entirely conducted empirically on various synthetic datasets.

3 Preliminaries: pattern mining

The reader is assumed to be familiar with the basics of association rule mining. Two major steps in mining association rules are: (i) finding frequent patterns and (ii) com-

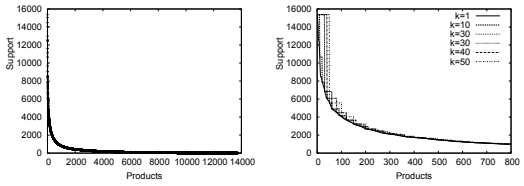


Fig. 2. Item Support Distribution: *CoopProd* (left); Encrypted *CoopProd* (right)

puting the association rules from them. Step (i) is the computationally dominant step. We briefly review frequent pattern mining below. Let $\mathscr{I} = i_1, ..., i_n$ be the set of items and $D = t_1, \ldots, t_m$ a transaction database (TDB) of transactions, each of which is a set of items. We denote the support of an itemset $S \subseteq \mathscr{I}$ as $supp_D(S)$ and the frequency by $freq_D(S)$. Recall, $freq_D(S) = supp_D(S)/|D|$. For each item i, $supp_D(i)$ and $freq_D(i)$ denote respectively the individual support and frequency of i. The whole function $supp_D(.)$, projected over items, is also called the *item support table* of D. It can be either represented in tabular form (see, e.g., Table 1 (a)), or plotted as a histogram, as in Fig. 2, where the item support distribution of the real-life Coop TDB is reported; both in support tables and in histograms items are listed in decreasing order of their support. The length of a transaction $t \in D$ is the number of items in t. We define the *size* of a TDB D as the sum of lengths of its transactions, i.e., $||D|| = \sum_{t \in D} |t|$. It is easy to see that $||D|| = \sum_{i \in \mathscr{I}} supp_D(i)$. This corresponds to the area under the support distribution graph (e.g., see Fig. 2). The frequent pattern mining problem [1] is: given a TDB D and a support threshold σ, find all patterns whose support in D is at least σ. We study a (corporate) privacy-preserving outsourcing framework for frequent pattern mining.

4 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

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6, Copyright (c) 2012 - Edizioni Libreria Progetto and the authors

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 $\mathscr{I}$ to denote the set of plain items and $\mathscr{E}$ to refer to the set of cipher items.

Adversary Knowledge. The server or an intruder (*attacker*) who gains access to the database may possess some background knowledge using which they can conduct attacks on the encrypted database D^* in order to make inferences.

We adopt a conservative model and assume that the attacker knows exactly the set of (plain) items $\mathscr{I}$ in the original transaction database D and their true supports in D, i.e., $supp_D(i), i \in \mathscr{I}$. The attacker may have access to similar data from a competing company, may read published reports, etc. Moreover we assume the attacker has access to the encrypted database D^*. Thus, he also knows the set of cipher items and their support in D^*, i.e., $supp_{D^*}(e)$, $e \in \mathscr{E}$.

In this paper we propose an encryption scheme based on: (i) replacing each plain item in D by a 1-1 substitution cipher (ii) adding fake transactions to the database. In particular, no new items are added. We assume the attacker knows this and thus he knows that $|\mathscr{E}| = |\mathscr{I}|$. We also assume the attacker knows the details of our encryption algorithm.

Attack Model. 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. If the cipher items are broken, i.e., their true identification is inferred by the attacker, then clearly cipher transactions and cipher patterns are broken, so they also must remain protected. The attack model is two-fold:

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

We refer to $prob(e)$ and $prob(E)$ as *probabilities of crack*. 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. In fact, we have a large increase in the size of D^* compared to D, i.e., a large size of the fake transactions. This in turn leads to a dramatic explosion of the frequent patterns, making pattern mining at the server side computationally prohibitive. This is the motivation for relaxing the equal-support constraint and introducing k-anonymity as a compromise.

Definition 1 (Item k-anonymity). *Let D be a transaction database and D^* its encrypted version. We say D^* satisfies the property of* item k-anonymity *provided for every*

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6, Copyright (c) 2012 - Edizioni Libreria Progetto and the authors

cipher item $e \in \mathscr{E}$*, there are at least* $k-1$ *other distinct cipher items* $e_1, ..., e_{k-1} \in \mathscr{E}$ *such that* $supp_{D^*}(e) = supp_{D^*}(e_i)$*,* $1 \leq i \leq k-1$. □

Fig.2 (right) shows the effect of grouping together cipher items into groups of k items. For a given value of k, the support distribution resembles a descending staircase. With small k, the graph tends to the original support distribution in D; while as k increases, the graph gets closer to the horizontal line discussed above. As the size of D^* is the area below the graph, we can control the size of D^* by an appropriate choice of k.

To quantify the privacy guarantee of an encrypted database, we define the following notion:

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

This definition does not constrain the crack probability of cipher itemsets which have no support in D^*. Intuitively, such cipher itemsets are not interesting. This will be exploited in Sec. 5 in designing effective k-private encryption schemes. Formally, the problem we study is the following: 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.

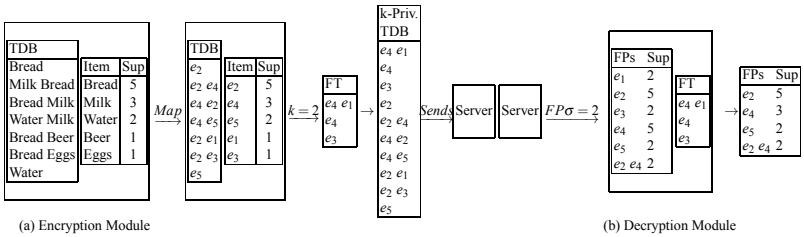


Fig. 3. E/D Module

5 Encryption/Decryption Scheme

In this section, we describe the ED module, originally presented in [8], responsible for the encryption of TDB and for the decryption of the cipher patterns coming from the server. The general idea of our Encryption/Decryption method is shown with an example in Figure 3.

5.1 Encryption

In this section, we introduce the encryption scheme, which transforms a TDB D into its encrypted version D^*. Our scheme is parametric w.r.t. $k > 0$ and consists of three main

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6, Copyright (c) 2012 - Edizioni Libreria Progetto and the authors

steps: (1) using 1-1 substitution ciphers for each plain item; (2) using a specific item k-grouping method; (3) using a method for adding new fake transactions for achieving k-privacy. The encryption scheme is a countermeasure to the item-based and set-based attacks discussed in Sec. 4: since the attacker knows the exact support of each item, we create a k-private D^*, such that the cipher items cannot be broken based on their support.

k-Grouping Method. Given the items support table, several strategies can be adopted to cluster the items into groups of k. 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. To obtain the formal protection that itemsets (or transactions) cannot be cracked with a probability higher than $\frac{1}{k}$, we need to use only grouping methods that yield groups of items that are *unsupported in D*. We call such grouping methods *robust*:

Definition 3. *Given a TDB D and a grouping G of the items occurring in D, G is called robust for D iff, for any group G_i of G, $supp_D(G_i) = 0$.* □

The above definition directly suggests a procedure for checking whether a given grouping G for a TDB D is robust: it is sufficient to check that the support in D of each group G_i in G is 0. If this is the case, the grouping can be safely used to obtain the maximum privacy protection guaranteed by our method. The following definition introduces our grouping method.

Definition 4. *Given the TDB D and its item support table in decreasing order of support, our grouping method:*

STEP1: groups together cipher items into groups of k adjacent items starting from the most frequent item e_1, obtaining the grouping $G = (G_1, ..., G_m)$.
STEP2: modifies the groups of G by repeating the following operations, until no group of items is supported in D: (1)Select the smallest $j \geq 1$ such that $supp_D(G_j) > 0$; (2) Find the most frequent item $i' \notin G_j$ such that, for the least frequent item i of G_j we have: $supp_D(G_j \setminus \{i\} \cup \{i'\}) = 0$; (3) Swap i with i' in the grouping. □

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 $N(e)$. The noise column indicates, for each cipher item e, the number of occurrences of e that are needed in D^* in order to bring e to the same support as the most frequent item of e's group. As such, the noise table represents the tool for generating the fake transactions to be added to D to obtain D^*. In particular, the total size of the needed fake transactions is exactly the summation of all the values in the Noise column of the noise table.

The noise table provides a compact *synopsis* (using $O(n)$ space, where n is the number of items) that can be stored by the ED module, to support both the creation of the fake transaction and the decryption step. For example, consider the example TDB in Figure 3, and its associated (cipher) item support table in Table 1 (a). For $k = 2$, the grouping method generates two groups: $\{e_2, e_5\}$ and $\{e_4, e_1, e_3\}$ (Table 1 (b)), that is

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6, Copyright (c) 2012 - Edizioni Libreria Progetto and the authors

(a) IST

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

(b) Grouping

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

(c) Noise Table

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

(d) Hash Tables

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

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

Table 1. Encryption with $k = 2$

robust: none of the two groups, considered as itemsets, is supported by any transaction in D.

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:

- $N(e'_1) - N(e'_2)$ instances of the transaction $\{e'_1\}$
- $N(e'_2) - N(e'_3)$ instances of the transaction $\{e'_1, e'_2\}$
- $\ldots$
- $N(e'_{m-1}) - N(e'_m)$ instances of the transaction $\{e'_1, \ldots, e'_{m-1}\}$
- $N(e'_m)$ instances of the transaction $\{e'_1, \ldots, e'_m\}$

Continuing the example, we consider cipher items of non-zero noise in Table 1 (c). The following two fake transactions are generated: 2 instances of the transaction $\{e_5, e_3, e_1\}$ and 1 instance of the transaction $\{e_5\}$. We observe that fake transactions introduced by this method may be longer than any transactions in the original TDB D. So, we consider shortening the lengths of the added fake transactions so that they are in line with the transaction lengths in D. In our running examples above, we obtain $\{e_5, e_3\}$, 2 of $\{e_1\}$ and 1 instance of $\{e_5\}$.

To implement the synopsis efficiently we use a hash table generated with a *minimal perfect hash function*. Minimal perfect hash functions are widely used for memory efficient storage and fast retrieval of items from static sets. 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 $[0 \ldots n-1]$, denoting the position of the hash table storing the triple of values $\langle e_i, times_i, occ_i\rangle$, where: $times_i$ represents the number of times the fake transaction $\{e_1, e_2, \ldots, e_i\}$ 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 $\{e_1, e_2, \ldots, e_i\}$.

Given a noise table with m items with non-null noise, our approach generates hash tables for the group of items. In general, the i-th entry of a hash table HT containing the item e_i has $times_i = N(e_i) - N(e_{i+1}), occ_i = \sum_{j=i+1}^{g} N(e_j)$, where g is the number of items in the current group. Notice 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 non-zero noise in Table 1 (c) are shown in Table 1(d). Given that in our example, $l_{max} = 2$, we need to split the 3 items of non-zero noise in Table 1 into two sets, each with associated fake transactions, coded by the two hash tables.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

Notice that any pattern consisting of items from different hash tables is *not* supported by any fake transactions. Finally, we use a (second-level) ordinary hash function H to map each item e to the hash table HT containing e.

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 ED module, by the compact synopsis described above.

5.2 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 $supp_D(S) \leq supp_{D^*}(E)$. Therefore, our encryption scheme guarantees that all itemsets frequent in D will be returned, in cipher version, by the server. But additional patterns frequent in D^*, but not in D, are returned as well. For each cipher pattern E returned by the server together with $supp_{D^*}(E)$, the ED module trivially recovers the corresponding plain pattern S as follows: $supp_D(S) = supp_{D^*}(E) - supp_{D^* \setminus D}(E)$.

This calculation is efficiently performed by the ED module using the synopsis of the fake transactions in $D^* \setminus D$ described above.

6 Preliminary Experimental Results

Data Set: We empirically assess our encryption method with respect to a real-life transaction database donated by Coop, a cooperative of consumers that is today the largest supermarket chain in Italy[5]. We selected 300,000 transactions involving 13,730 different *products*.

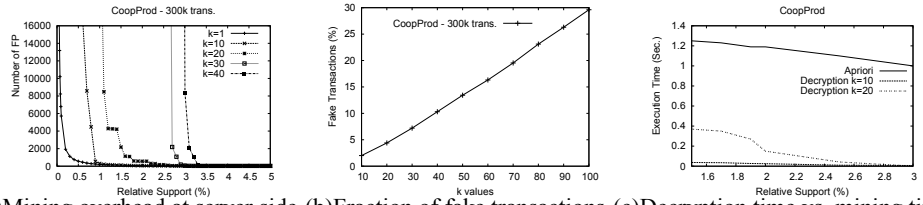
Encryption overhead: we assess the size of fake transactions added to *CoopProd** after encryption; Fig. 4 (b) reports the sizes of fake transactions for different k values. We observe that the size of fake transactions increases linearly with k.

Mining overhead: We study the overhead at server side in the pattern mining task over *CoopProd** w.r.t. *CoopProd*. We adopted the *Apriori* implementation by Christian Borgelt[6]. 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. 4(a), for different values of k; notice that $k = 1$ means that the original and encrypted TDB are the same. 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 $k = 20$. We found that, for reasonably small values of the support threshold, the incurred overhead at server side is kept under control; clearly, a trade-off exists between the level of privacy, which increases with k, and the minimum affordable support threshold for mining, which also increases with k.

[5] http://www.e-coop.it/, in Italian

[6] http://www.borgelt.net

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,



(a)Mining overhead at server side (b)Fraction of fake transactions (c)Decryption time vs. mining time

Fig. 4. Overhead at server side and Decryption overhead

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 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 apriori over the original database. As shown in Fig. 4(c), 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.

7 Conclusion and Future Work

We studied the problem of (corporate) privacy-preserving outsourcing of association rule mining. Our encryption scheme is based on 1-1 substitutions and fake transactions such that the transformed database satisfies k-anonymity w.r.t. items and itemsets. Moreover we showed some preliminary empirical results that are encouraging; naturally, there are many interesting open issues to be investigated. The next steps include: (1) The study of a formal analysis based on our attack model and the proof that the probability that an itemset 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; (2) The complexity analysis of our encryption/decryption scheme; (3) The definition of an strategy for incrementally maintaining the synopsis at the client side against updates in the form of appends. (4) scheme using large real data set with different sparsity/density properties to understand how it works in different settings; (5) The analysis of the scalability of the proposed approach and comparison of the execution time of the mining step with that of the decryption step by using a mining algorithm like FP-growth, that could be more efficient than an apriori-based algorithm.

References

1. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. VLDB 1994.
2. R. Agrawal and R. Srikant. Privacy-preserving data mining. SIGMOD 2000.
3. S. Agrawal, J. R. Haritsa: A Framework for High-Accuracy Privacy-Preserving Mining. ICDE 2005.
4. M. Atzori, F. Bonchi, F. Giannotti, D. Pedreschi: Anonymity preserving pattern discovery. VLDB J. 17(4): 703-727 (2008).

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

5. S. Bu et al. Preservation of patterns and input-output privacy. ICDE 2007
6. K. Chen et al. Toward attack-resilient geometric data perturbbation. SDM, 2007.
7. C. Clifton, M. Kantarcioglu, J. Vaidya. Defining Privacy for Data Mining. NSF Workshop on Next Generation Data Mining, 2002.
8. F. Giannotti, L.V.S. Lakshmanan, A. Monreale, D. Pedreschi and H. Wang. Privacy-Preserving Data Mining from Outsourced Databases. CPDP 2011.
9. L.V.S. Lakshmanan, R.T. Ng, G. Ramesh. To Do or Not To Do: The Dilemma of Disclosing Anonymized Data. SIGMOD 2005.
10. A. Machanavajjhala, J. Gehrke, and D. Kifer. l-diversity: Privacy beyond k-anonymity. In ICDE, 2006.
11. S. Rizvi, J. R. Haritsa: Maintaining Data Privacy in Association Rule Mining. VLDB 2002.
12. P. Samarati. Protecting respondents' identities in microdata release. TKDE, 13(6):1010-1027, 2001.
13. W. K. Wong, D. W. Cheung, E. Hung, B. Kao, N. Mamoulis. Security in Outsourcing of Association Rule Mining. VLDB 2007.
14. X. Xiao and Y. Tao. Anatomy: Simple and Effective Privacy Preservation. VLDB 2006.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,