Security in Outsourcing of Association Rule Mining

W. K. Wong
The University of Hong Kong
wkwong2@cs.hku.hk

David W. Cheung
The University of Hong Kong
dcheung@cs.hku.hk

Edward Hung
The Hong Kong Polytechnic University
csehung@comp.polyu.edu.hk

Ben Kao
The University of Hong Kong
kao@cs.hku.hk

Nikos Mamoulis
The University of Hong Kong
nikos@cs.hku.hk

ABSTRACT

Outsourcing association rule mining to an outside service provider brings several important benefits to the data owner. These include (i) relief from the high mining cost, (ii) minimization of demands in resources, and (iii) effective centralized mining for multiple distributed owners. On the other hand, security is an issue; the service provider should be prevented from accessing the actual data since (i) the data may be associated with private information, (ii) the frequency analysis is meant to be used solely by the owner. This paper proposes substitution cipher techniques in the encryption of transactional data for outsourcing association rule mining. After identifying the non-trivial threats to a straightforward one-to-one item mapping substitution cipher, we propose a more secure encryption scheme based on a one-to-n item mapping that transforms transactions non-deterministically, yet guarantees correct decryption. We develop an effective and efficient encryption algorithm based on this method. Our algorithm performs a single pass over the database and thus is suitable for applications in which data owners send streams of transactions to the service provider. A comprehensive cryptanalysis study is carried out. The results show that our technique is highly secure with a low data transformation cost.

1 Introduction

Association rule mining aims at the discovery of itemsets that co-occur frequently in transactional data. Centralized mining has been well studied in the past (e.g., see [2], [12]). The problem has a large worst-case complexity, a fact that motivates business to outsource the mining process to service providers, who have developed efficient, specialized solutions. The data owner, apart from the mining cost relief, has additional motives for outsourcing. First, it requires minimal computational resources, since the owner is only required to produce and to send the transactions to the miner. This makes the outsourcing model also attractive to applications in which data owners produce transactions as streams and they have limited resources to maintain them. Second, assume that the owner has multiple production sources of transactions, e.g., consider a chain of supermarkets which generate transactions at different locations. All transactions can be sent to a single provider for mining association rules. The provider could compute association rules that are local to the individual stores or global rules for the whole organization. Therefore, the cost of transferring transactions among the sources and performing the global mining in a distributed manner is saved.

On the other hand, the service provider becomes a single point of security attack. If the service provider is not trusted, he should be prevented from accessing the actual data because the data may be associated with private information. In addition, even if the items (e.g., store products) are public, the computed association rules are property of the owner and they are meant to be known only to him. Therefore, protecting both raw data and the resulting association rules from the service provider is a key issue in outsourcing of data mining.

There are two approaches that can protect sensitive information. The first is to apply an *encryption* function that transforms the original data to a new format [13, 7, 14]. The second is to apply *data perturbation*, which modifies the original raw data randomly [3]. The perturbation approach is less attractive since it can only provide approximate results; on the other hand, the use of encryption allows the exact rules to be recovered. In this paper we propose and evaluate appropriate encryption techniques for outsourcing of association rules mining. In order for an encryption to be appropriate for the problem, the following conditions should be satisfied. First, there should be a correct, complete, and deterministic decryption method that transforms the association rules found in the encrypted database to the true association rules in the original database. Second, the encryption and decryption processes must be reasonably fast; otherwise, owners may choose to apply association rules mining locally (if cost is the only concern). Third, the encryption method must be secure enough to prevent the service provider (or an attacker) from recovering the original transactions and the true association rules among the actual items by processing the encrypted data.

We adopt the idea of substitution ciphers [10, 18, 9] in transaction encryption. A direct application of this idea is

VLDB '07, September 23-28, 2007, Vienna, Austria.

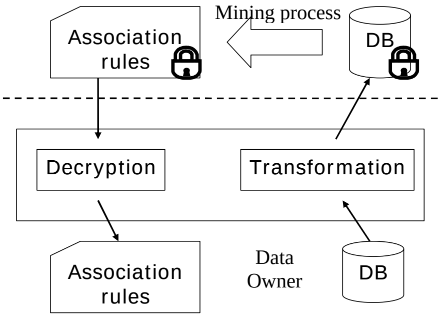

Figure 1: The architecture of the scheme

to replace each original item by a unique symbol (one-to-one item mapping); for example, the instances of a specific item (e.g., "bread") in transactions are substituted by a unique integer (e.g., 54). One possibility to enhance the security is to use a one-to-n item mapping as the substitution function; for example, "bread" is substituted by a set of integers (e.g., $\{54, 103\}$). As we will prove later, in order for decryption to be done unambiguously, the substitution of an item must contain at least one unique symbol. For example, the number 54 should not appear in any substitution of any item besides that of "bread". Our result thus shows that simple one-to-n substitutions are in fact no more secure than one-to-one substitution. Motivated by this discovery, we propose a *non-deterministic* one-to-n substitution scheme. In our scheme, random "fake" items are added to the transformed transactions. We will show that the use of fake items in non-deterministic one-to-n substitutions greatly enhances security.

Figure 1 is an overview of the outsourcing scheme. The data owner applies a one-pass transformation technique to encrypt the original transactions, which are sent to the service provider (miner). The miner computes the strong association rules for the transformed data, which are sent back to the owner. The owner then uses a decryption process to convert the rules to actual association rules involving the original items.

2 Related Work

Privacy protection is an important issue in data mining. Businesses usually do not want to share their own private (statistical) information with service providers [6]. A stream of past research has focused on protecting privacy against third-party players in distributed data mining [15, 19, 11]. These studies are not directly related to the problem we study in this paper, since our interest is to protect data and results against a service provider, who alone should perform the mining task.

Different data mining models (such as association-rule mining, decision tree classification, etc.) have different security requirements. Specialized approaches for the protection of sensitive information under different models have been designed [16, 3, 21]. In our case, we need a simple scheme that enables a third-party miner to find association rules in a transformed database while preventing it from accessing the private information in the original database. The perturbation technique proposed in [8] can be adapted for our problem. However, this solution returns only approximate results. Also, the miner knows the exact number of itemsets (and their frequencies) that will be used by the owner (i.e., the actual support of the perturbed itemsets). This can be considered a breach of privacy. On the other hand, our data encryption technique ensures the accuracy of the results and at the same time hides from the miner the original number of items and the exact frequencies of the itemsets.

Substitution cipher is a well-known method that is used in the encryption of plain text. Each letter in the text is replaced by another letter. Although the number of possible substitutions (i.e., letter orderings) is very large (26!), the encryption is in fact not difficult to break if a dictionary of words with their expected frequencies is available [10, 18, 9]. Due to the extreme size of the search space, adversaries usually resort to local search techniques to break the cipher. Genetic algorithms [20] have been widely used in cryptanalysis to attack security schemes [4, 5, 18]. In order to validate the security of our encryption scheme, we evaluate its resiliency to attacks by a genetic algorithm.

3 Problem Definition

Let I be a set of items and T be a database of transactions. A transaction t_i is a subset of I. (In our model, t_i could be $\emptyset$). A transaction t_i *contains* an itemset X if and only if $X \subseteq t_i$.

DEFINITION 1. *(Support Count of Itemset) The support count of an itemset X in the database T, denoted $supp^T(X)$, is the number of transactions in T that contain the itemset X.*

Given a support threshold $s\%$, an itemset X is *large* if and only if $supp^T(X) \geq |T| \times s\%$, where $|T|$ is the number of transactions in T. An association rule $X \Rightarrow Y$ has $s\%$ support and $c\%$ confidence if $s\%$ of transactions in T contain $X \bigcup Y$ and $c\%$ of the transactions that contain X also contain Y where X, $Y \subseteq I$. The problem of mining association rules is to discover all the association rules that have support and confidence greater than the given support and confidence thresholds, respectively.

Association rules are discovered by dividing the problem into two subproblems [1]. The first is to find all of the large itemsets in T (i.e., all itemsets whose supports are not less than the support threshold). The second is to find the association rules from the discovered large itemsets. For each large itemset Y and every subset X of Y, "$X \Rightarrow (Y - X)$" is a strong association rule if its confidence (i.e., $\frac{supp^T(Y)}{supp^T(X)}$) is not smaller than the confidence threshold.

Assume that a party p_{owner} owns a set of transactions T, where each transaction $t_i \in T$ is a subset of I. Another party (service provider) p_{miner} acts as a third-party miner to help p_{owner} to compute the set of strong association rules Γ in T. The problem is to find a transformation P_T that encrypts each transaction in T, and that it possesses the following two properties:

1. Let T' be the set of encrypted transactions computed by p_{owner} by applying P_T on T, i.e., $T' = P_T(T)$; let Γ' be the set of strong association rules in T' (computed by p_{miner}); there exists a recovery function P_R such that $P_R(\Gamma') = \Gamma$.

2. p_{miner} cannot find out T, P_T, or P_R from T'.

If such a transformation P_T is used by p_{owner} to encrypt the transactions in T, p_{miner} can only compute Γ'. Since p_{miner} does not know P_T or P_R, he cannot derive Γ from Γ'. On the other hand, p_{owner} knows P_T and P_R, and hence can recover the entire set of valid association rules Γ from Γ'. In Property 2, we assume p_{miner} can leverage background knowledge to guess P_T and P_R, if he attempts to attack the encrypted data. (We will discuss in detail the role of background knowledge in Section 6.1).

4 Item Mapping

A simple encryption method is to directly apply a substitution cipher m: $I \rightarrow J$, which is a one-to-one mapping from the original set I of items to another dictionary J, where $|I| = |J|$. A transaction t_i is transformed to $M(t_i) = \{m(x) \mid x \in t_i\}$. For example, consider the transaction $t_i = \{bread, chocolate\}$ and assume that $m(bread) = 165$ and $m(chocolate) = 54$. Then, t_i is transformed to $M(t_i) = \{54, 165\}$. This scheme allows the association rule mining algorithm of p_{miner} to be applied with 100% accuracy. Since p_{miner} knows only the transformed transactions, the original transactions cannot be directly determined.

Substitution ciphers are considered to be insecure for the encryption of languages like English text. Given a reasonable long piece of text, the mappings between letters can be easily found by frequency analysis. Exemplary algorithms for attacking substitution ciphers include relaxation[17], genetic algorithms[18] and scatter search[10]. Unlike encryption of text, item substitutions for association-rule mining are more secure because:

1. the number of possible mappings is much larger ($|I|!$);
2. p_{miner} may not have the knowledge of the relative frequencies of items; in this case, when attempting to guess the transformation, p_{miner} will have no information to verify the validity of the guessed mappings.

On the other hand, this simple scheme may not be secure enough in some applications. First, the probability to get the correct mapping of an item by a random guess is $\frac{1}{|I|}$, which could be considered too large. In addition, p_{miner} knows the number of large itemsets and their distribution which are private statistics to p_{owner}. Finally, if p_{miner} has some background knowledge, like the global frequencies of the real items from other databases (e.g., the actual frequencies of itemsets mined from another supermarket), he could easily attack the simple one-to-one substitution scheme.

4.1 Fake items and one-to-n mappings

To increase the difficulty to break a substitution cipher and to protect the statistical information of the large itemsets, we can add fake items to the transformation. Under this scheme, we add a set of *fake items* F to the dictionary J (i.e., $|J| = |I| + |F|$) and use the mapping $M(t_i) = \{m(x) \mid x \in t_i\} \bigcup F'$ to transform each transaction t_i. Here F' is a random (i.e., non-deterministic) subset of F. In this way, the probability of a correct random guess of the mapping $m(x)$ for some $x \in I$ is decreased to $\frac{1}{|I|+|F|}$. Also, even if p_{miner} has some rough estimates of the relative frequencies of itemsets, he will have difficulties in finding out the distribution of the true large itemsets in T and in recovering the mapping.

Besides adding fake items to the cipher, we can map an item to more than one item, i.e., we can use a one-to-n mapping instead of a one-to-one mapping. Intuitively, a one-to-n mapping cipher should be more difficult to break than a one-to-one mapping cipher. In the following, we study the correctness and effectiveness of one-to-n mapping ciphers.

Definition 2. *(Item Mapping) Given a set of items I and another set of items B, where $|B| \geq |I|$, we say $m : I \rightarrow 2^B$ is a one-to-n item encryption mapping if for all $x, y \in I$ where $x \neq y$, $m(x) \neq m(y)$. (For short, we call m a one-to-n item mapping).*

Example 1. $I = \{a, b, c\}$, $B = \{1, 2, 3, 4, 5\}$. A possible item encryption mapping m is defined as follows:[1]

$$m(a) = \{1, 4, 5\}$$
$$m(b) = \{2\}$$
$$m(c) = \{3, 5\}$$

Note that, the intersection of the mappings of the items may not be empty. For example, 5 occurs in the mapping of a and the mapping of c. Similar to item mapping, we can define an itemset mapping based on a given one-to-n item mapping.

Definition 3. *(Itemset Mapping) Given a one-to-n item mapping $m : I \rightarrow 2^B$, for an itemset $X = \{x_1, x_2, ..., x_n\} \subseteq I$, we define an itemset mapping $M : 2^I \rightarrow 2^B$ such that $M(X) = m(x_1) \bigcup m(x_2) \bigcup ... \bigcup m(x_n)$.*

Definition 4. *(Inverse Itemset Mapping) Given a one-to-n item mapping $m : I \rightarrow 2^B$, let $J = \{Y \subseteq B \mid \exists X \subseteq I, M(X) = Y\}$. We define the inverse itemset mapping $M^{-1} : J \rightarrow 2^I$ of M such that $M^{-1}(Y) = \{x \in I \mid m(x) \subseteq Y\}$, for $Y \in J$.*

Example 2. Here we show some examples of an itemset mapping and its inverse that is derived from the item mapping used in Example 1.

- Itemset mapping

$$M(\{a, b\}) = \{1, 2, 4, 5\}$$
$$M(\{a, c\}) = \{1, 3, 4, 5\}$$
$$M(\{b, c\}) = \{2, 3, 5\}$$

- Inverse itemset mapping

$$M^{-1}(\{1, 3, 4, 5\}) = \{a, c\}$$
$$M^{-1}(\{1, 2, 3, 4, 5\}) = \{a, b, c\}$$
$$M^{-1}(\{1, 2, 3\}) \text{ is not defined (because } \{1, 2, 3\} \notin J)$$

Note that given an item mapping m and its associated itemset mapping M, there is no guarantee that the inverse mapping (M^{-1}) correctly reverts M. In other words, if we encrypt an itemset X by M, M^{-1} may not be able to decrypt $M(X)$ correctly. In particular, if there is an item $x \in I$ such that $m(x)$ is a subset of the mappings of other items, there can be an encrypted transaction which is decrypted to a superset of the original transaction.

[1] This mapping will be used as the default mapping for the other examples used in the rest of this paper unless otherwise specified.

Example 3. Consider the following one-to-n item mapping m where $I = \{a, b, c\}$ and $B = \{1, 2, 3\}$.

$m(a) = \{1, 2\}$

$m(b) = \{2, 3\}$

$m(c) = \{1, 3\}$

Let $t = \{a, b\}$. We have $t' = M(t) = \{1, 2, 3\}$. $M^{-1}(t') = \{a, b, c\}$. So, $M^{-1}(M(t)) \neq t$, therefore M^{-1} decrypts $M(t)$ incorrectly.

In the following, we develop a result which guarantees the correctness of a one-to-n mapping cipher.

Definition 5. *(Admissible Item Mapping) A one-to-n item mapping $m : I \rightarrow 2^B$ is admissible if $\forall x \in I$, $m(x) - \bigcup_{y \in I - \{x\}} m(y) \neq \emptyset$.*

Intuitively, if m is an admissible item mapping, for any $x \in I$, there always exists an item $e \in m(x)$ such that $e \notin m(y)$ for all $y \in I - \{x\}$.

Theorem 1. *Given an itemset mapping M that is based on a one-to-n item mapping m, the cipher defined by M can be decrypted correctly by the inverse of M, i.e., $\forall X \subset I$, $M^{-1}(M(X)) = X$ if and only if m is admissible.*

Proof. Suppose m is not admissible. Then, there exist an item x and items y_1, y_2, ..., y_k such that $x \neq y_i$, $\forall i = 1, .., k$ and $m(x) \subseteq m(y_1) \bigcup m(y_2) \bigcup ... \bigcup m(y_k)$. Consider the itemset $X = \{y_1, y_2, ..., y_k\}$, and $Y = M(X)$, since $Y = m(y_1) \bigcup m(y_2) \bigcup ... \bigcup m(y_n)$, $m(x) \subseteq Y$. As a result, $x \in M^{-1}(Y)$. Since $x \notin X$, $M^{-1}(Y) \neq X$. The decryption is incorrect. Suppose m is admissible. For any itemset $X = \{x_1, x_2, ..., x_k\}$, let $Y = M(X)$ and $Z = M^{-1}(Y)$. $x_i \in Z$ because $m(x_i) \subseteq Y$ for $i = 1, 2, ..., k$. Thus $X \subseteq Z$. Since m is admissible, $\forall y \in I - X, m(y) - Y \neq \emptyset$. Hence $y \notin M^{-1}(Y)$. Thus $Z \subseteq X$ and so $X = Z$. □

Theorem 2. *Consider an admissible one-to-n item mapping m, which is applied on a transaction database $T = \{t_1, t_2, ..., t_n\}$ and which generates the encrypted database $T' = \{M(t_1), M(t_2), ..., M(t_n)\}$. For any itemset $X \subset I$, $supp^T(X) = supp^{T'}(M(X))$.*

Proof. Let $T = \{t_1, t_2, ..., t_n\}$ be the transaction database. Let $tr^T(X)$ denote the set of transactions in T that contain X. Note that $supp^T(X) = |tr^T(X)|$. Consider an arbitrary itemset $X \subseteq I$, where $X = \{x_1, x_2, ..., x_k\}$.

If $t_i \in tr^T(X)$, we have $X \subseteq t_i$ and thus $M(X) \subseteq M(t_i)$, so $M(t_i) \in tr^{T'}(M(X))$.

If $t_i \notin tr^T(X)$, then $X \not\subseteq t_i$. Hence, there exists an x_i such that $x_i \notin t_i$. Now if $m(x_i) \subseteq M(t_i)$ then by the definition of M^{-1}, we have $x_i \in M^{-1}(M(t_i))$. Since m is admissible, By Theorem 1, $M^{-1}(M(t_i)) = t_i$ and thus $x_i \in t_i$, which leads to a contradiction. Therefore, $m(x_i) \not\subseteq M(t_i)$. So, $M(X) = \bigcup_{j \in [x_1, x_k]} m(x_j) \not\subseteq M(t_i)$. This implies $M(t_i) \notin tr^{T'}(M(X))$.

As a result, $|tr^T(X)| = |tr^{T'}(M(X))|$ and thus $supp^T(X) = supp^{T'}(M(X))$. □

For an *admissible* one-to-n item mapping m from a set I to a set B, each mapping of an I-item must contain at least one unique B-item. Let $U \subset B$ be the set of these unique B-items (the minimum size of U is $|I|$). An I-item may be m-mapped to one B-item (${}_{|B|}C_1$ possible mappings) up to $|B| - |I| + 1$ B-items (${}_{|B|}C_{|B|-|I|+1}$ possible mappings). Therefore, the probability of making a correct guess for a mapping of an item is $\frac{1}{{}_{|B|}C_1 + {}_{|B|}C_2 + \cdots + {}_{|B|}C_{|B|-|I|+1}}$, which is much smaller than that in the case of a one-to-one item mapping.

4.2 Transaction transformation

We have shown that a one-to-n mapping cipher, based on an admissible item mapping m, is correct. In this section we show that an admissible one-to-n mapping cipher can be decrypted by a one-to-one mapping. Since a one-to-one mapping is relatively easy to break, a simple one-to-n mapping is not adequate for secure encryption. Motivated by this result, we introduce a powerful transaction transformation scheme that can significantly strengthen the security of admissible one-to-n mapping ciphers.

Definition 6. *(Coverage of Item Mappings and Itemset Mappings) Given two itemset mappings, $M_1 : 2^I \rightarrow 2^{D_1}$ and $M_2 : 2^I \rightarrow 2^{D_2}$, M_1 covers M_2, denoted by $M_1 \Rightarrow M_2$, if and only if $\forall X \subseteq I$, $M_2^{-1}(Y) = M_1^{-1}(Y \bigcap D_1)$ where $Y = M_2(X)$.*

Given two item mappings $m_1 : I \rightarrow D_1$ and $m_2 : I \rightarrow D_2$, we construct two itemset mappings M_1 and M_2 from the two item mappings, $M_i(t) = \bigcup_{x \in t} m_i(x), \forall t \subseteq I$ for $i = 1$, 2. We say that m_1 covers m_2, denoted by $m_1 \Rightarrow m_2$, if and only if $M_1 \Rightarrow M_2$.

Example 4. $I = \{a, b, c\}$, $D = \{1, 2, 3\}$. An item mapping $m' : I \rightarrow D$ is defined as follow:

$m'(a) = \{1\}$

$m'(b) = \{2\}$

$m'(c) = \{3\}$

Let M be the itemset mapping constructed using m in Example 1 and M' be the itemset mapping constructed using m'. We consider all possible subsets of I.

If $Y = M(\emptyset) = \emptyset$, then $M'^{-1}(Y \bigcap D) = \emptyset = M^{-1}(Y)$.

If $Y = M(\{a\}) = \{1, 4, 5\}$, then $M'^{-1}(Y \bigcap D) = \{a\} = M^{-1}(Y)$.

If $Y = M(\{b\}) = \{2\}$, then $M'^{-1}(Y \bigcap D) = \{b\} = M^{-1}(Y)$.

...

If $Y = M(\{a, b, c\}) = \{1, 2, 3, 4, 5\}$, then $M'^{-1}(Y \bigcap D) = \{a, b, c\} = M^{-1}(Y)$.

Hence, $M' \Rightarrow M$.

Intuitively, if an item mapping m_2 is covered by another mapping m_1, the encryption done by m_2 can be simulated by m_1. Hence, the cryptanalysis against m_2 can be done as if the encryption was done by m_1. In particular, an adversary can recover the original data if he is able to find m_1, even if he fails to find m_2.

Theorem 3. *Given an admissible one-to-n item mapping m and a transaction database $T = \{t_1, t_2, ..., t_n\}$, let $T' = \{M(t_1), M(t_2), ..., M(t_n)\}$ be the encrypted database. $\forall x \in I$, if $|m(x)| \geq 2$, then there exist an item $e \in m(x)$ such that $supp^{T'}(m(x)) = supp^{T'}(e)$.*

Proof. Since m is admissible, for any item $x \in I$, there exists a unique item e in $m(x) - \bigcup_{y \in I - \{x\}} m(y)$. If a transaction t contains x, we have $m(x) \subseteq M(t)$ and thus $e \in M(t)$ in the encrypted database T'. If transaction t does not contain x, $M(t) \subseteq \bigcup_{y \in I - \{x\}} m(y)$ and thus $e \notin M(t)$. Therefore, $supp^{T'}(e) = supp^{T}(x) = supp^{T'}(m(x))$. □

Theorem 4. *Given an admissible one-to-n item mapping m, there exists a one-to-one item mapping m' such that $m' \Rightarrow m$.*

Proof. Given an admissible one-to-n item mapping $m : I \rightarrow 2^B$, we will construct a one-to-one item mapping $m' : I \rightarrow D$, where $D \subset B$, that covers m. We use M, M^{-1} (resp. M', M'^{-1}) to denote the itemset mapping and inverse itemset mapping of m (resp. m'). For each item $x \in I$, if $|m(x)| = 1$, set $m'(x) = m(x)$; if $|m(x)| \geq 2$, from Theorem 3, $\exists e \in m(x)$ such that $supp^{T'}(m(x)) = supp^{T'}(e)$ and $e \in m(x) - \bigcup_{y \in I - \{x\}} m(y)$. In this case, set $m'(x) = e$. Also, let $D = \bigcup_{x \in I} m'(x)$. For every itemset $X = \{x_1, x_2, ..., x_n\} \subseteq I$, let $Y = M(X) = \bigcup_{i=1}^{n} m(x_i)$. We have $M^{-1}(Y) = X$. Note that $M'^{-1}(S)$ is defined for all $S \subseteq D$ because m' is a one-to-one item mapping. Let $Z = M'^{-1}(Y \bigcap D)$. By construction, $m'(x) \subseteq m(x)$ for all $x \in I$, and $m'(x_i) \subseteq M(X) = Y$ for $i = 1$ to n. Therefore, $X \subseteq M'^{-1}(Y \bigcap D) = Z$. On the other hand, for any item $e \in I - X$, $m'(e) - \bigcup_{x \in I - \{e\}} m(x) \neq \emptyset$. Hence $e \notin Z$ and thus $Z \subseteq X$. Therefore, $Z = X$ and $m' \Rightarrow m$. □

Theorem 4 is a counter-intuitive result. It implies that a one-to-n item mapping cipher is no better than a one-to-one item mapping cipher. When an adversary attempts to break the encryption of an admissible one-to-n mapping, it can attack the encryption as if it were done by a one-to-one mapping. In Example 1, c is mapped to {3, 5} and {3} is the unique item in $m(c)$, and $supp^{T}(c) = supp^{T'}(3,5) = supp^{T'}(3)$. Hence decryption derived from the mapping $c \rightarrow$ {3, 5} can be replaced by a decryption derived from the one-to-one mapping $c \rightarrow$ {3} in the encryption. So, the security of an admissible one-to-n mapping is only as high as that provided by a one-to-one mapping. One way to eliminate this undesirable feature of a one-to-n mapping is to add B-items and fake items randomly into the transformation of transactions. In the above example, if the item {3} is added to some transactions in T', it will make $supp^{T'}(3)$ larger and it is no longer possible to deduce $supp^{T'}(3,5)$ from $supp^{T'}(3)$.

Definition 7. *(Transaction Transformation) Given an admissible one-to-n item mapping $m : I \rightarrow 2^B$, and a set of items F (fake items), where $F \bigcap B = \emptyset$, we define a transaction transformation N as a mapping from 2^I to $2^{B \bigcup F}$ which maps a transaction $t \in 2^I$ to $M(t) \bigcup E$, where E is a random subset (including $\emptyset$) of $B \bigcup F$, i.e., $N(t) = M(t) \bigcup E$.*

Note that the random itemset E added to the transformation of a transaction t is non-deterministic. Two transactions t_1, t_2 with the same content (i.e., $t_1 = t_2$) give $M(t_1) = M(t_2)$. However, it could happen that $N(t_1) \neq N(t_2)$ due to different extra items E added to their mappings. On the other hand, we need to ensure that this randomness does not affect the correctness of the cipher based on the mapping m.

Definition 8. *(Inverse Transaction Transformation and Valid Transaction Transformation) Given N is a transaction transformation, let $t' \in 2^{B \bigcup F}$, we define the inverse transaction transformation $N^{-1}(t') = \{x \in I \mid m(x) \subseteq t'\}$. Note that $N^{-1}(t') = M^{-1}(t')$ when $t' \in J$ where $J = \{Y \subseteq B \mid \exists X \subseteq I, M(X) = Y\}$. We say that the transaction transformation N is valid if the transformed transactions can be decrypted correctly, i.e., $N^{-1}(N(t)) = t, \forall t \subseteq I$.*

A valid transaction transformation can easily be obtained by setting E to $\emptyset$ for all transactions. However, as explained in Theorem 4, such a transformation can be decrypted by a one-to-one mapping. Given a transaction t, a good transaction transformation N should consider all possible subsets E of $B \bigcup F$ when it adds extra items to $M(t)$, provided that the addition of E would still allow correct decryption. We formulate this requirement by defining the concept of "complete" transformation and extend the definition of coverage to include transaction transformations.

Definition 9. *(Coverage of transaction transformation) Given an itemset mapping, $M : 2^I \rightarrow 2^{D_1}$ and a transaction transformation $N : 2^I \rightarrow 2^{D_2}$, M covers N, denoted by $M \Rightarrow N$, if and only if $\forall X \subseteq I$, $N^{-1}(Y) = M^{-1}(Y \bigcap D_1)$ where $Y = N(X)$.*

Definition 10. *(Complete Transaction Transformation) Given a valid transaction transformation N, for all $t \subseteq I$, let $\hat{N}(t)$ be the set of all possible values returned by $N(t)$. We say that N is a complete transformation iff $\forall t \subseteq I$, $\forall E \subseteq B \bigcup F$, if $N^{-1}(M(t) \bigcup E) = t$ then $M(t) \bigcup E \in \hat{N}(t)$. In other words, any transformation $(M(t) \bigcup E)$ of a transaction t that can be correctly decrypted has the potential of being returned by $N(t)$.*

Example 5. Consider the one-to-n cipher defined in Example 1. Suppose $t = \{c\}$ and $F = \emptyset$. So, we have $M(t) = \{3, 5\}$. Here are some examples of E under a valid N:

$E = \{1\}$ and $N(t) = \{1, 3, 5\}$; or

$E = \{4\}$ and $N(t) = \{3, 4, 5\}$.

Some examples of E that lead to an invalid N:

$E = \{1, 4\}$ and $N(t) = \{1, 3, 4, 5\}$; $N^{-1}(N(t)) = \{a, c\}$.

$E = \{2\}$ and $N(t) = \{2, 3, 5\}$; $N^{-1}(N(t)) = \{b, c\}$.

If N is complete, any subset of $\{1, 2, 3, 4, 5\}$ (a total of $2^5 = 32$ sets) can be returned by N. For $t = \{c\}$, one can easily verify that $out_E(t) = \{\emptyset, \{1\}, \{4\}\}$ is the set of all possible E under a valid N. So, when N transforms t, an E in $out_E(t)$ will be picked in a non-deterministic fashion to construct $N(t) = M(t) \bigcup E$. Conversely, as long as E is picked from $out_E(t)$, N is valid with respect to t.

Theorem 5. *Given a valid transaction transformation N applied on a transaction database $T = \{t_1, t_2, ..., t_n\}$ to generate $T' = \{N(t_1), N(t_2), ..., N(t_n)\}$. For any itemset $X \subseteq I$, $supp^{T}(X) = supp^{T'}(M(X))$.*

Proof. Let $T = \{t_1, t_2, ..., t_n\}$ be the transaction database. Let $tr^T(X)$ be the set of transactions in T such that $X \subseteq t$. Note that $supp^T(X) = |tr^T(X)|$. Consider an arbitrary itemset $X \subseteq I$, where $X = \{x_1, x_2, ..., x_k\}$.

If $t_i \in tr^T(X)$, we have $X \subseteq t_i$. This implies $M(x) \subseteq M(t_i)$ and hence $M(x) \subseteq N(t_i)$. So, $N(t_i) \in tr^{T'}(M(X))$.

If $t_i \notin tr^T(X)$, then $X \not\subseteq t_i$. Now, if $N(t_i) \in tr^{T'}(M(X))$, $M(X) \subseteq N(t_i)$. So, $X \subseteq N^{-1}(N(t_i))$ and thus $X \subseteq t_i$ (since N is valid). This leads to a contradiction and thus $N(t_i) \notin tr^{T'}(M(X))$.

As a result, $|tr^T(X)| = |tr^{T'}(M(X))|$. So, $supp^T(X) = supp^{T'}(M(X))$. □

Theorem 6. *If N is a valid and complete transaction transformation obtained from an admissible one-to-n mapping m such that m is not a one-to-one mapping, there does not exist a one-to-one itemset mapping M' that covers N.*

Proof. Assume there is a one-to-one itemset mapping $M' : 2^I \rightarrow 2^D$ such that $M' \Rightarrow N$. Since m is not a one-to-one item mapping, there exists an item $x \in I$ such that $|m(x)| \geq 2$. Let $m(x) = \{y_1, y_2, ..., y_n\}$ for some $n \geq 2$. Consider the empty itemset $\emptyset$. Since N is valid, we have $N^{-1}(N(\emptyset)) = \emptyset$. Since $M' \Rightarrow N$, we can use M'^{-1} to decipher N, i.e., $M'^{-1}(N(\emptyset) \bigcap D) = N^{-1}(N(\emptyset)) = \emptyset$. Let $E_i = \{y_i\}$ for $i = 1, 2, ..., n$. We will show that $N^{-1}(E_i) = \emptyset$. If there exists an item $e \in N^{-1}(E_i)$, by definition of inverse transaction transformation, $m(e) \subseteq E_i \subset m(x)$. Hence, we have an item e whose mapping $(m(e))$ is a proper subset of that of another item x. This contradicts to the fact that m is an admissible mapping. By contradiction, $N^{-1}(E_i) = \emptyset$ and therefore, $N^{-1}(E_i) = N^{-1}(N(\emptyset))$.

Since $N^{-1}(E_i) = N^{-1}(N(\emptyset))$, when we apply the nondeterministic transformation N to $\emptyset$, E_i is a possible output for $N(\emptyset)$ given that N is complete. If $M' \Rightarrow N$, we can use M' to decrypt $N(\emptyset)$. Thus, $M'^{-1}(E_i \bigcap D) = \emptyset$. Since M' is a one-to-one mapping, we have $E_i \bigcap D = \emptyset$. Therefore, $y_i \notin D$ for $i = 1, 2, ..., n$. Now consider $M'^{-1}(N(\{x\}) \bigcap D)$. Note that by definition, $N(\{x\}) = M(\{x\}) \bigcup E$ where E is a subset of $B \bigcup F$ and it is always possible to set E to the empty set because N is complete. In that case, $N(\{x\}) = \{y_1, y_2, ..., y_n\}$. Since $y_i \notin D$, $M'^{-1}(N(\{x\}) \bigcap D) = M'^{-1}(\emptyset) = \emptyset$. Note that $N^{-1}(N(\{x\})) = \{x\}$ (because N is valid). Therefore, $M'^{-1}(N(\{x\}) \bigcap D) \neq N^{-1}(N(\{x\})$. M' cannot cover N. □

The importance of Theorem 6 is that a valid and complete transaction transformation derived from a one-to-n item mapping is more secure than a one-to-one mapping; it is not possible to simulate such a transformation by a one-to-one mapping. We are not simply performing substitution, the transaction transformation process becomes nondeterministic but yet it guarantees correctness of the decryption of the cipher. Note that the number of possible transformed transactions is $2^{|B \bigcup F|}$ which is much larger than the number of possible original transactions $(2^{|I|})$. Since an adversary can no longer simulate a one-to-n item mapping by a one-to-one item mapping, in general, we can fully utilize the search space of a one-to-n item mapping to increase the cost of attack and prevent the adversary to easily guess the correct mapping. We will report a set of experiments in Section 7.1.1 that compare the strength of a one-to-one item mapping against a one-to-n item mapping in terms of how difficult it is to break the ciphers.

Note that in our proof, we assumed that a database may contain empty transactions. In case there are no empty transactions in the physical dataset, the owner may choose to add some empty transactions randomly. This increases the number of transactions slightly. To obtain accurate support counts, the support threshold should be decreased accordingly.

5 Algorithms

In this section we first introduce an intuitive algorithm for generating a one-to-n mapping scheme m. Then, we describe an algorithm that generates a valid and complete transaction transformation T' from an original database T based on the mapping m. Finally, we describe a decryption technique for the association rules generated at p_{miner}.

5.1 Generate one-to-n item mappings

We now present an algorithm for generating admissible one-to-n item mapping $m : I \rightarrow B$. B is divided into two sets of items, U and C. U is a set of items (called *unique items*), each of which is a unique item in a mapping. C is a set of items (called *common items*) which is shared among the mappings. $U \bigcap C = \emptyset$. For simplicity, we set $|U| = |I|$. First, we generate the unique item of each mapping; each original item in I is associated with a unique random item in U. Next, we add the common items into the mappings. For each item c_i in C, we randomly pick b mappings, where the expected value of b is given by the parameter N_B. We then add c_i into each of the picked mappings. Thus, N_B allow us to control the occurrences of each item c_i among the mappings. Algorithm 1 is a pseudocode for this process.

Algorithm 1 Algorithm to generate one-to-n admissible mappings

```
Require: $I$, the original items
Require: $U$, the unique items
Require: $C$, the common items
Require: $N_B$, expected number of mappings extended
  generate a random permutation in $U$
  for $x_i$ in $I$ do
    fetch $u_i$ in $U$ at $i$-th location
    initialize $m(x_i)$ to $\{u_i\}$
  end for
  for $c_j$ in $C$ do
    generate $b$ with expected mean $N_B$
    randomly pick $b$ mappings from $\{x_i \rightarrow m(x_i), i = 1, ...., |I|\}$
    for each picked mapping $x_i \rightarrow m(x_i)$ do
      $m(x_i) = m(x_i) \bigcup \{c_j\}$
    end for
  end for
  return $m$
```

5.2 Transaction transformation

We now present an algorithm for generating a valid and complete transaction transformation from a one-to-n admissible mapping. For every transaction t, the itemset mapping $M(t)$ is extended to $N(t) = M(t) \bigcup E$. The key is to construct a random E without affecting the decryption result; we need to prevent $N(t)$ from containing $m(x)$ for any $x \notin t$; otherwise, such x will appear in $N^{-1}(t)$. To avoid this to happen, we must ensure that some items in $m(x)$ are excluded from E.

Let et be the transformed transaction of t. Initially $et = M(t)$. To create E, we process the items $x \in I - t$ iteratively. Conceptually, we process one mapping $x \rightarrow m(x)$ in each iteration. All items in $m(x)$ are candidates to be added to E. However, some items in $m(x)$ may have been added to *et* already and some may have been identified as unsuitable to be added in a previous iteration (because they result in incorrect decryption). We use *bt* to record the list of items identified as unsuitable to be included in *et*. The items of $m(x)$ that have been included into the transformed transaction in previous iterations can be found in $m(x) \bigcap (et \bigcup E)$. Those that have been identified as not suitable can be found in $m(x) \bigcap bt$. So $dt = m(x) - et - E - bt$ are the remaining candidates. Note that by considering only items in *dt*, we have ensured that the decryption of mappings that have alrady been processed will not be affected. This is because *dt* consists of items that are not contained in the mappings that have been processed in previous iterations. If $|m(x) \bigcap bt| \geq 1$, some items in $m(x)$ are already marked unsuitable. In this case, all members in *dt* can be safely added to E without making N invalid. We pick a random subset $st \subseteq dt$ and add *st* to E. If $|m(x) \bigcap bt| = 0$, then a proper subset $st \subset dt$ is picked to be added to E (otherwise $N(t)$ would be incorrectly decrypted to contain x). In both cases, the remaining candidates in $dt - st$ are added to *bt*, so that they will not be included in any E in subsequent iterations (and thus to prevent the entire $m(x)$ from being added to *et*).

The expected size of $E \bigcap B$ is an input parameter N_E. The algorithm first generates n_e in $[0, |B - M(t)|]$ such that its expected value is N_E. The algorithm continues adding items until we fill the quota or all mappings are processed. Finally, a subset of n_f fake items from F with expected size N_F is randomly picked to extend the transaction. Algorithm 2 is a pseudocode of our valid and complete transformation algorithm.

Algorithm 2 Algorithm to generate a valid and complete transformation for a transaction t

```
Require: $m : I \rightarrow 2^B$, an admissible one-to-n item mapping
Require: $t$, input transaction
Require: $F$, fake items
Require: $N_E$, expected size of $E \bigcap B$
Require: $N_F$, expected size of $E \bigcap F$
  {$et$ denotes the resulting transformation $N(t) = M(t) \bigcup E$ of $t$}
  $et = \emptyset$
  for each item $i \in t$ do
    $et = et \bigcup m(i)$
  end for
  {$E$ contains the items to be added to $et$}
  {$bm$ contains the items $x \in I - t$ such that items in $m(x)$ have been considered as candidates for insertion into $E$}
  {$bt$ contains the items in $B$ identified as not suitable to be added to $et$}
  $E = bm = bt = \emptyset$
  generate $n_e$ in $[0, |B - M(t)|]$ with expected mean $N_E$
  while $|E| < n_e$ and $|I| - |t| - |bm| > 0$ do
    pick item $x \in I - t - bm$ {$x$ is selected to be processed}
    $bm = bm \bigcup x$
    $dt = m(x) - et - E - bt$
    $st = \emptyset$
    if $|m(x) \bigcap bt| \geq 1$ then
      if $|dt| \geq 1$ then
        randomly pick $st$ such that $st \subseteq dt$
      end if
    else if $|dt| \geq 2$ then
      randomly pick $st$ such that $st \subset dt$
    end if
    $E = E \bigcup st$
    $bt = bt \bigcup (dt - st)$
  end while
  $et = et \bigcup E$
  {Adding fake items}
  generate $n_f$ with expected mean $N_F$
  pick a set $sf \subseteq F$ with size equal to $n_f$
  $et = et \bigcup sf$
  return $et$
```

Example 6. We use the mapping m in Example 1 to perform transaction transformation. We assume $F = \{6, 7\}$. Suppose $t = \{b\}$, and we are to compute $N(t)$ with Algorithm 2. Before processing any item, $et = M(t) = \{2\}$. In the first iteration, assume a and the mapping $a \rightarrow \{1, 4, 5\}$ are picked for processing. No items in $m(a) = \{1, 4, 5\}$ were processed before. So $dt = \{1, 4, 5\}$ is the set of candidates. Suppose coin tossing determines subset $st = \{1\}$. Then, $E = \{1\}$ and $bt = \{4, 5\}$.

In the second iteration, c and the mapping $c \rightarrow \{3, 5\}$ are picked. Then $dt = \{3\}$ and this reveals that only 3 in $m(c) = \{3, 5\}$ was not processed before. Note that $|m(c) \bigcap bt| = 1$, hence we proceed to toss coin to pick a subset from *dt*. Assume $st = \{3\}$ is picked, then $E = \{1, 3\}$ and *bt* remains unchanged.

Since there are no more items $x \notin t$ left, E is finalized to $\{1, 3\}$ and *et* becomes $\{1, 2, 3\}$. We proceed to pick fake items in F and assume $\{7\}$ is picked, then $et = \{1, 2, 3, 7\}$. So, the resulting transformation is $N(\{b\}) = \{1, 2, 3, 7\}$. Note that $N^{-1}(N(\{b\})) = \{b\}$.

THEOREM 7. *Given an admissible one-to-n item mapping m, Algorithm 2 generates a valid and complete transaction transformation.*

PROOF. First, we will prove the transformation is valid. Given an admissible one-to-n item mapping m, for any transaction t, Algorithm 2 returns $t' = N(t) = M(t) \bigcup E$ for some $E \subset B \bigcup F$. Since $M(t) \subseteq t'$, $t \subseteq N^{-1}(t')$. Suppose the decryption on t' is not correct, i.e., $N^{-1}(t') - t \neq \emptyset$. Let x be an item in $N^{-1}(t')$ but not t, we will show that $m(x)$ cannot be contained entirely in t' and hence a contradiction.

Since $x \notin t$, but $m(x) \subseteq t'$, $x \rightarrow m(x)$ must have been processed at an iteration i of Algorithm 2. Since m is admissible, $m(x)$ has a unique item which has not been processed at any iteration before i. Note that once an item is added to *bt*, i.e., it is marked not suitable, it will never be added into E. Hence, once an item in $m(x)$ is added to *bt*, $m(x)$ will not be contained in t'. Now we look at several cases when x is being processed in iteration i. First, assume $|m(x) \bigcap bt| \geq 1$, then some item in $m(x)$ is in *bt* and $m(x)$ cannot be a subset of t'. If $|m(x) \bigcap bt| = 0$ and $dt \geq 2$, a proper subset of *dt* is added to E and the nonempty difference $(dt - st)$ is added to *bt*. Hence $m(x)$ cannot be a subset of t'. Finally, if $|dt| = 1$, then *dt* contains the unique element in $m(x)$. In this case, $st = \emptyset$ and nothing is added to E but *dt* is added to *bt*, i.e., the unique element of $m(x)$ is in *bt*. Hence $m(x)$ cannot be a subset of t' in all cases. By contradiction, $N^{-1}(t')$ must equal to t and the transformation is valid.

Next, we show that the transformation is complete. Given an input transaction t, Algorithm 2 first adds $M(t)$ into *et*. Note that *et* contains all those items in the transformed transaction generated before the while loop. For the transformation to be complete, the algorithm should be able to generate any $t' \subseteq B \bigcup F$ such that $N^{-1}(t') = t$. Consider any t' where $N^{-1}(t') = t$, and let $X = t' - M(t)$. X represents the items the algorithm adds into E in the while loop. We first show by induction that $X \bigcap B$ can be generated in the while loop.

Let R be the required items we need to add to E where E is the variable in the while loop holding the added items, i.e., $R = X \bigcap B - E$. Let the induction statement $S(n)$ be: the algorithm can generate R where $|R| = n$ given that $R \bigcap bt = \emptyset$. Consider the base case $S(0)$, $R = \emptyset$ and there is no need to add any item into E. So, $S(0)$ is true. Assume $S(0)$, $S(1)$, ..., $S(k)$ is true. Consider $S(k+1)$, since the algorithm has not filled up the quota, the algorithm goes into the while loop. Note that if $y \in R$, $\exists x \in I - t - bm$ such that $y \in m(x)$ and x will be selected in the top of the while loop. Let $K = m(x) \bigcap R = \{e_1, e_2, ..., e_j\}$. $dt = m(x) - et - E - bt$. Since $x \notin t$ and $N^{-1}(t') = t$, so $x \notin N^{-1}(t')$. Therefore $m(x) \not\subseteq t'$, $m(x) - R = \emptyset$. In the following steps of the loop, with some coin tossing results, the algorithm picks $st = K$ and adds $dt - K$ to bt. Since $(m(x) - K) \bigcap (R - K) = \emptyset$, items in $(R - K)$ are not in bt. The remaining problem is S$((k+1-|K|))$. Since $|K| \geq 1$, $S(k+1)$ is true. By MI, the algorithm can generate X of any possible sizes with initial $E = \emptyset$.

The last part to is show the algorithm can generate the required $X \bigcap F$. This is obvious since the algorithm can pick any possible subset of F for different coin tossing results. So, the algorithm generate the required X. In conclusion, Algorithm 2 is complete and valid. □

5.3 Association rule recovery

A valid transaction transformation ensures that the decrypted result is correct; it also ensures that p_{owner} can "translate" the association rules mined by p_{miner} correctly.

THEOREM 8. *Given a valid transaction transformation N on a transaction database $T = \{t_1, t_2, ..., t_n\}$, which generates $T' = \{N(t_1), N(t_2), ..., N(t_n)\}$. Let M be the admissible itemset mapping that N is generated from, and let X, Y be two itemsets in I. $X \Rightarrow Y$ is an association rule in the original database T if and only if $M(X) \Rightarrow M(Y) - M(X)$ is an association rule in the encrypted database T'.*

PROOF. Let $Z = X \bigcup Y$. By Theorem 5, $supp^T(X) = supp^{T'}(M(X))$ and $supp^T(Z) = supp^{T'}(M(Z))$. If $X \Rightarrow Y$ is an association rule in T, X and Z are both large and $supp^T(Z) \geq c\% \times supp^T(X)$, where $c\%$ is the confidence threshold. So, $M(X)$ and $M(Z)$ are both large and $supp^{T'}(M(Z)) \geq c\% \times supp^{T'}(M(X))$. This gives us an association rule $M(X) \Rightarrow M(Z) - M(X)$ in T'. Since $M(Z) = M(X) \bigcup M(Y)$, $M(Z) - M(X) = M(Y) - M(X)$. Note that $M(Y) - M(X)$ cannot be further simplified to $M(Y)$ as $M(X)$ and $M(Y)$ may not be mutually exclusive. The if part can be proved similarly. □

We can easily design an association rule decryption algorithm using Theorem 8. Note that not all association rules in T' are representing a rule in T. Given a rule in T', $A \Rightarrow B$, if $M^{-1}(A)$ or $M^{-1}(A \bigcup B)$ is not defined, it does not result in any rule in T. If both are defined, we can output $M^{-1}(A) \Rightarrow M^{-1}(A \bigcup B) - M^{-1}(A)$ as a rule in T.

Example 7. Suppose that the encryption is based on the mapping of Example 1 and that we have found the following rules in T':

$1 \Rightarrow 4$ (rejected, $M^{-1}(\{1\})$ is not defined)

$2 \Rightarrow 1, 3, 5$ (rejected, $M^{-1}(\{1, 2, 3, 5\})$ is not defined)

$2 \Rightarrow 1, 3, 4, 5$

$M^{-1}(\{2\}) = \{b\}$
$M^{-1}(\{1, 2, 3, 4, 5\}) = \{a, b, c\}$
Decrypted rule: $b \Rightarrow ac$

$2, 3, 5 \Rightarrow 1, 4$

$M^{-1}(\{2, 3, 5\}) = \{b, c\}$
$M^{-1}(\{1, 2, 3, 4, 5\}) = \{a, b, c\}$
Decrypted rule: $bc \Rightarrow a$
Note that $M^{-1}(\{1, 4\})$ is not defined.

Note that, in this example, there are only two rules in T while there are four rules in T'. This is expected, due to the introduction of additional items in the encryption (i.e., $|B \bigcup F| > |I|$).

5.4 Cost analysis

Generating mapping and recovering association rules are relatively cheap since they are independent from database size. Algorithm 1 has a time complexity of $O(|I| + N_B \times |C|)$. It takes $O(|I|)$ time for generating a random permutation and initializing mapping of items and $O(N_B \times |C|)$ for inclusion of common items. The cost of recovering association rules is $O(c_d \times |AR'|)$ where AR' is association rules found by p_{miner} and c_d is the time to compute the inverse of an encrypted itemset. For an admissible one-to-n mapping m, each mapping of item must contain an unique item. Let y be an unique item in $m(x)$ where $x \in I$. Given an encrypted itemset X', if $x \in M^{-1}(X')$, $y \in M^{-1}(X')$. We can make use the set of unique items to construct a partial inverse which maps U to I. This gives us a list of candidates which are possible items in $M^{-1}(X')$. We verify each candidate x_c by checking if $m(x_c) \subseteq X'$ and this gives us the final itemset X. Finally, we must make sure $M(X) = X'$ or $M^{-1}(X')$ is undefined since $X' \notin J$. So, c_d is bounded by $O(|l|)$ where l is average length of association rule.

Algorithm 2 can be divided into two parts. The first part computes $M(X)$ given an input itemset X. It takes $O(|X|)$ time. The second part generates a random E for the input itemset X. We used a "quota" approach. The algorithm iteratively adds items to E and stops when enough items are added. So, the expected time complexity is linear to the expected quota $(N_E + N_F)$ and has a worst case of $O(|I|)$ (every mapping is processed). In conclusion, it takes $O(|T| \times (|t| + N_E + N_F))$ time to transform a database T where $|t|$ is the average length of transaction.

5.5 Settings of outsourcing parameters

To avoid excessive increase in the mining cost and at the same time achieve adequate security, we suggest the following methodology to set appropriate values for the transformation parameters. We inject as many additional items in the transformed transactions as the number F_1 of size-one large itemset in the original database (this number can be determined by sampling). So, we may set $|B \bigcup F| = |I| + |F_1|$. In addition, we control the length of each mapping not to be too high since a high value will cause the common items C to have very high frequency compared to the unique items U and to be easily identified in T'. A rule of the thumb is to set N_B to be close to 1, which causes common items to have a similar expected support as the unique items (and the original database items). With the above settings the expected support of an item in B, if we only use the one-to-n itemset mapping, becomes $E^1_{sup} = \frac{|U| \cdot avsup + |C| \cdot N_B \cdot avsup}{|B|}$, where $avsup$ is the average support of an I-item in the original database T. $avsup$ can be estimated by $\frac{|t_{avg}|}{|I|}$, where

$|t_{avg}|$ is the average transaction length in T. The while-loop of Algorithm 2 adds more B-items to the transformed transaction, which increases this expected support. Parameter N_E is estimated to increase E^1_{sup} for each B-item by $\frac{N_E}{|B|}$. We want this increase to be large enough in order to inject enough randomness and small enough in order to make mining inexpensive. A good strategy is to set $\frac{N_E}{|B|}$ to be a fraction of E^1_{sup}, or $N_E = \lambda \times (|U| \cdot avsup + |C| \cdot N_B \cdot avsup)$, for certain $\lambda \in (0, 1)$. Similarly, items in F should be given the same expected support as those of the B-items, which is $(1+\lambda) \times E^1_{sup}$. The support of F-items is expected to be $\frac{N_F}{|F|}$. Therefore, an appropriate value for N_F is $(1+\lambda) \times E^1_{sup} \times |F|$. For example, assuming that $|I| = 1000$, $|t_{avg}| = 10$, and $F_1 = 200$, we can set $|B| = 1100$, $|F| = 100$, $N_B = 1$, $E^1_{sup} = avsup = 0.01$, $\lambda = 0.5$, $N_E = 5.5$, $N_F = 1.5$.

6 Security Analysis

In outsourcing, we need to prevent p_{miner} from recovering the information on the large itemsets and association rules in T by using information in the transformed database T'. In cryptanalysis against substitution cipher on text, an adversary may use dictionary information to acquire word frequencies to attack the encryption. In association rule mining, we may have the advantage that dictionary information might be unavailable to attackers. However, there could be *background knowledge* that an adversary can use to guess the information on association rules and large itemsets. For example, the large itemsets and rules in an individual supermarket can be similar to some extent to those of the entire industry, which might have been published. In this case, public knowledge would become background knowledge to the attacker. In the following, we study the impact of this type of background knowledge on the proposed security scheme.

6.1 Background knowledge

We assume the adversary has acquired the knowledge of a subset of large itemsets and their corresponding supports. The accuracy of this knowledge may vary. We formally define the notion of *background knowledge* as follows.

Definition 11. *(Background Knowledge) Let L be the set of large itemsets in T. An (α, β)-knowledge of T is a subset of L, such that (i) the subset contains α% of the itemsets in L and (ii) the adversary has a count for each itemset X in this subset, which is in the range of $supp^T(X) * (1 \pm \beta\%)$.*

Based on an (α, β)-knowledge, an adversary can perform frequency analysis and recover some of the mappings and attempt to decrypt the database. Our goal is to compare the relative strength of one-to-one mappings and one-to-n mappings under the attack of an adversary with a given (α, β)-knowledge.

7 Experimental Evaluation

In this section, we present an experimental evaluation that demonstrates the security and cost effectiveness of the proposed encryption scheme. In the experiments, we used synthetically generated transactional databases using the IBM data generator. All the programs are implemented in Java. The experiment is done on a Pentium 4 2.26GHz computer with 512 MB RAM running on Windows.

7.1 Security analysis

In order to measure the security of the proposed approach, we carried out two sets of experiments. The first one studies the effectiveness of a one-to-n item mapping based transaction transformation over a one-to-one item mapping, if the attacker knows the frequencies of all itemsets in the database (and attempts to identify the identities of the items). The second one assesses the security of our scheme for different background (α, β)-knowledges.

7.1.1 One-to-one item mapping vs one-to-n item mapping

For the first experiment, we implemented and used a genetic algorithm (GA) for cryptanalysis [18]. We compare two encryption strategies: one-to-one item mapping vs one-to-n item mapping with transaction transformation. The first strategy replaces every item in a transaction by a unique item of a different dictionary. Some fake items are added to each transaction. The second strategy implements the one-to-n mapping based transaction transformation of Algorithm 2. Note that the adversary attacks the second strategy as if it was encrypted by a pure one-to-n item mapping.

We generated two synthetic databases with $100k$ transactions each. One of the databases has 20 items ($|I| = 20$) and the other one has 35 items ($|I| = 35$). Note that the number of items we used here is relatively small because the time required for each step of GA is exponential to number of items and the experiment already takes a long time to finish for the 35-items case. For the one-to-one item mapping, the size of fake items is 5. For the one-to-n item mapping, we randomly generate a set of items B, where $|B| = |I| + 5$, i.e, there are 5 common items. We set N_B to 2, and limit the size of each mapping to 2, i.e., $|m(x)| \leq 2$ for $x \in I$. No fake items are added.

GA considers only the frequencies of itemsets up to bigrams (groups of two items). Given the exact counts of the original itemsets in T, GA attempts to identify the mappings of the items in the encryption in 100 runs. During each run, 100 candidate mappings are generated and evaluated. The fitness function is defined by the average count difference: $-\sum_{X \subset I and |X| \leq 2} |supp^T(X) - supp^{T'}(M'(X))|$ where M' is a candidate mapping. We execute GA with two different mapping initialization methods. In the first method, we initialize the candidate mappings to some "good" values (i.e., we assume that GA is lucky enough to start with some true assignments). Each mapping of an item in such a good initial candidate tosses a coin. If it is head, the correct mapping of the item is included in the candidate; otherwise, the candidate mapping of the item is randomly assigned. The second candidate initialization method is done totally in random. We study the converging power of GA under two initialization schemes.

The crossover operator in each (subsequent to the initial) step of GA generates a new candidate mapping from two randomly chosen parents, i.e., candidate mappings from the previous step. The probability of a parent to be chosen is associated with its fitness. In the generated candidate (offspring), each item randomly chooses one of the two parents and inherits its mapping from the chosen parent. Finally, each item in the offspring has a probability to "mutate". A mutation operation assigns a new randomly generated mapping to the item.

Tables 1 and 2 show the results of this GA experiment for $|I| = 20$ and $|I| = 35$, respectively. Both one-to-one and one-to-n item mappings result in an encrypted database with the same number of items, i.e., both schemes create an encrypted database with $|I| + 5$ different items. However, the one-to-n item mapping shows a higher resistance against GA in the experiments. Note that even after 100 iterations, GA cannot converge to an accurate mapping. For $|I| = 35$ and if we start from a random candidate mapping, GA can identify only one item correctly. Even if GA starts with a very lucky guess (half of the assignments are given), it can only identify 2 additional items in the one-to-n encryption, while all items are eventually identified in the one-to-one encryption. This is explained by the fact that, since each original item may map to more than one items, GA needs to search a larger space of possible mappings, compared to the one-to-one case.

Apart from being less accurate, GA is also significantly slower when attacking the one-to-n mapping. In the experiment, we fix the length of each mapping to 2 and thus we need to prepare the information up to 4-itemset if we carry out frequency analysis using bigrams in the original database. This results in a large difference in the amount of time for GA to prepare the possible mappings and to count the frequencies of them, compared to the one-to-one scheme (see the second line of each table). In addition, due to the large number of itemsets, decryption of itemsets for the one-to-n case takes more time (see the third line of each table). Thus, each iteration of GA against a one-to-n item mapping takes more time compared to attacking a one-to-one scheme. This result indicates that if the adversary has limited time to attack, he will be able to execute GA for fewer iterations against a one-to-n item mapping and it is likely that GA cannot converge to an accurate mapping.

We assessed how fast GA converges to its solution and also the appropriateness of the fitness function in modeling the accuracy of a candidate assignment for both one-to-one and one-to-n schemes. Figure 2 shows the value of fitness function of the best candidate at each run for the 20 items case, while Figure 3 shows this value for the 35 items case. If GA starts with a random initial population, although it shows a similar progress in both one-to-one and one-to-n item mappings in terms of fitness value, the actual accuracy of the best candidate mapping for one-to-n case is much lower (compare with the number of correct guesses in Tables 1 and 2). This happens because the space of possible one-to-n mappings is huge and it is possible to find many candidate mappings having similar (i.e., slightly lower) fitness value. This makes it very difficult for GA to identify the correct mapping for an item. In the case of a good initial population, GA starts with a fairly good solution, since the correct mappings for half of items have already been found. In this case, GA converges very fast to the actual mappings in the one-to-one cases. (These are identified in 13 and 5 runs for the 20 and 35 items cases, respectively.) Also, GA's progress is much slower in the one-to-n case because it is more difficult to find the correct mapping for the incorrectly assigned items in the initial population due to the large search space. The initial boost in the fitness value is due to the effectiveness of the crossover operator in the early stages; each candidate has a different set of individual items that are correctly mapped and crossover helps to bring them together in a single candidate.

Summing up, in this section we have shown that a one-to-n item mapping has a much resistance to attacks by a genetic algorithm than a one-to-one mapping and at the same time brings a much higher cost burden to each iteration of GA.

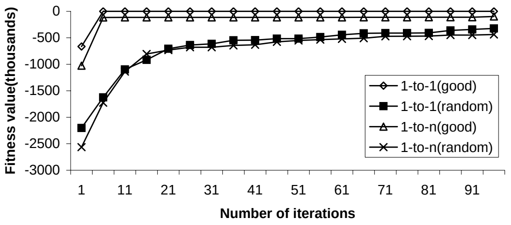

Figure 2: Fitness of the best candidate per iteration for $|I| = 20$

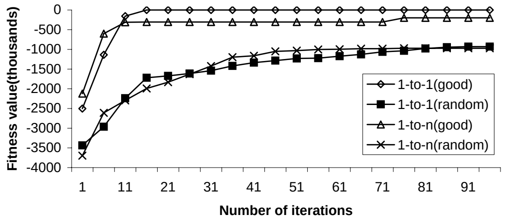

Figure 3: Fitness of the best candidate per iteration for $|I| = 35$

7.1.2 Security with (α, β)-knowledge

In the second experiment, we used a modified genetic algorithm as an adversary who has acquired an (α, β)-knowledge. The genetic algorithm we used in this experiment is more intelligent than the one used in the first experiment because we assumed the adversary knows the value of β. So, the adversary can reduce the search space for each mapping of item. Since the adversary only has information about the large itemsets, he can only find the mappings for items that appear in the background knowledge. We define a set of possible mappings for each item using the value of β. When a mapping of an item mutates, it randomly picks a number of samples in the set of possible mappings and performs a local search to find the best one. So, the adversary (who knows β) can progress easily. Let K be the set of itemsets for which the frequencies are known from the background knowledge, $supp_K(t)$ be the support count of itemset t known in the background knowledge, and M' be a candidate mapping. The fitness function is defined using the average count difference given by $-\frac{\sum_{X \in K} |supp_K(X) - supp^{T'}(M'(X))|}{|K|}$.

We set $|B| = 1200$, $|F| = 0$, $N_B = 2.5$, and $N_E = 5$ to generate a test encrypted database which contains 1000 items, $100k$ transactions and 106 size one large items. We run the adversary program with α varying from 100% to 50% and β varying from 0% to 50%. For each set of (α, β)-knowledge, we check how many item mappings in each candidate are correct. We measure the accuracy of the adversary with respect to the number of such correct mappings. Let K_1 be the set of size-one itemsets in K. We define *mapping accuracy* as $\frac{\text{no. of correct item mappings}}{|K_1|}$. The mapping accuracy of the best candidate, if the genetic algorithm is applied with different values of α and β is shown in Figure 4. If we use the candidate mapping with the higher mapping accuracy

	One-to-one item mapping		One-to-n item mapping	
Candidate Initialization	good	random	good	random
Time to compute itemset counts in T'	25s		809s	
Time spent by genetic algorithm	14s	13s	513s	513s
Number of iterations per second	7.14	7.69	0.19	0.19
Number of correct guesses	20	12	12	3

Table 1: One-to-one vs. one-to-n item mapping on a 100k-transaction database ($|I| = 20$)

	One-to-one item mapping		One-to-n item mapping	
Candidate Initialization	good	random	good	random
Time to compute itemset counts in T'	117s		20856s	
Time spent by genetic algorithm	78s	81s	5842s	4840s
Number of iterations per second	1.28	1.23	0.017	0.021
Number of correct guesses	35	4	22	1

Table 2: One-to-one vs. one-to-n item mapping on a 100k-transaction database ($|I| = 35$)

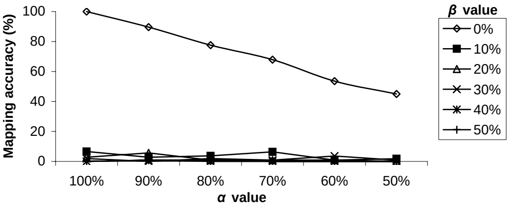

Figure 4: The accuracy of guessed item mappings in automated frequency analysis

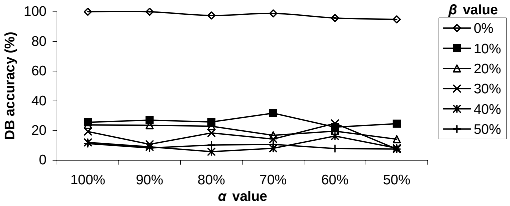

Figure 5: The accuracy on decrypting the database in automated frequency analysis

to decrypt the database, we can measure how much information the adversary can correctly recover. Since the adversary does not have the mappings for all items, we consider only items in K_1 to assess accuracy. For each transaction t, let t' be the corresponding encrypted transaction. M'^{-1} is the inverse of itemset mapping constructed by the candidate mapping with the highest mapping accuracy. The accuracy of a decrypted transaction is measured by $\frac{M^{-1}(t') \bigcap t_k}{|t_k|}$ where $t_k = t \bigcap K_1$. The accuracy of the decrypted database is then assessed by the average decryption accuracy for the transactions for which $t_k \neq \emptyset$ (i.e., those containing at least one item in K_1). The result is shown in Figure 5.

Figure 4 shows that when the adversary does not have an accurate values of support counts (when β is more than 0%), the performance of the attack degrades rapidly. This is because for a higher value of β, the number of possible mappings for each item greatly increases, as there is a large number of itemset frequencies in T' that are within the range allowed by β. α limits the number of correct mappings for the case where $\beta = 0\%$, but its impact in other cases is minor. Note that frequency analysis cannot generate additional information for items that are not included in the background knowledge (i.e., if $\beta = 0\%$, only α items are identified). Figure 5 shows a similar result; the accuracy of the database decryption drops rapidly when β increases from 0%. On the other hand, the database decryption accuracy is generally higher than the mapping accuracy. This happens because even the mapping of an item is incorrect, the item may still appear in the decrypted transaction.

7.2 Cost of encryption and mining

To verify the low cost of the proposed one-to-n encryption scheme in practice, we applied the transaction transformation algorithm on a set of five databases with varying sizes ($100k$ to $500k$ transactions). The total number of items is $|I| = 1000$, the average transaction length is 10, and the databases contain about 400 large itemsets. We run the transformation algorithm with two sets of input parameters. In both sets, $|B| = 1150$ and $|F| = 50$. The first set of encryptions has a higher expected number of mappings extended by a common item (N_B), a higher expected size of $E \bigcap B$ (N_E), and a higher number of expected fake items in F added to a transaction (N_F) than the second. In order to assess the cost benefit of outsourcing we also mined the association rules locally at p_{owner} and compared its cost with that of transforming the data and decrypt the rules produced by p_{miner}. As we assume that p_{owner} is not an expert in data mining, we used simple implementation of Apriori for this purpose. The mining task is to discover all association rules with support threshold 5% and confidence threshold 75%. Table 3 shows the results of the experiments.

When the problem is outsourced to p_{miner}, p_{owner} spends much less time in preparing the encryption function, applying the transformation, and decrypting the rules, than he would spend if he performed the mining locally. In addition, the cost at p_{owner} side is not sensitive to the mining parameters: only the recovery part is related to the number of rules, but its cost is very small compared to the transformation cost. Thus, p_{owner} can save a lot more when there are more large itemsets in the original databases. Additional benefits of outsourcing are (i) p_{owner} does not require high memory resources (required by Apriori) and (ii) transactions

Number of transactions		100k	200k	300k	400k	500k
$N_B = 4$, $N_E = 8$, $N_F = 2$						
p_{owner}	Time to generate mapping	0.1s	0.1s	0.1s	0.1s	0.1s
	Time to transform database	6.0s	11.9s	18.3s	23.9s	30.0s
	Time to recover association rules	0.2s	0.1s	0.1s	0.1s	0.1s
p_{miner}	Time spent in mining	453.8s	1083.0s	1592.9s	2099.9s	2629.6s
$N_B = 2.5$, $N_E = 4$, $N_F = 1$						
p_{owner}	Time to generate mapping	0.1s	0.1s	0.1s	0.1s	0.1s
	Time to generate mapping and transform database	2.5s	5.3s	9.3s	10.9s	12.3s
	Time to recover association rules	0.2s	0.1s	0.1s	0.2s	0.1s
p_{miner}	Time spent in mining	194.6s	488.3s	738.2s	944.8s	1121.9s
Without outsourcing						
p_{owner}	Time spent in mining	79.7s	203.9s	292.9s	383.1s	464.8s

Table 3: Cost of one-to-n item mapping with varying database sizes and parameter settings

can be transformed and sent to p_{miner} *progressively*, as they are being created; i.e., p_{owner} is not required to have the complete database before he can start the encryption and transmission.

On the other hand, the mining overhead at p_{miner}, due to the transformation, is high compared to the mining cost on the original data. In this experiment, we ran Apriori also on the p_{miner} side in order to analyze the cost overhead. For the case $N_B = 4$, $N_E = 8$, and $N_F = 2$, the size of the transformed database is about 2.4 that of times of the original database and the mining time is about 5.6 times of that of mining the original database. For the case $N_B = 2.5$, $N_E = 4$, $N_F = 1$, the size and mining time factors are 1.6 and 2.4 respectively. Note that the mining overhead due to the transformation is not too bad and could be easily handled by p_{miner}, especially if the service provider is equipped with more sophisticated mining algorithms.

8 Conclusion

In this paper we present a set of encryption methods for transactional databases that are suitable for outsourcing association rule mining. Starting from a simple one-to-one substitution cipher, which is susceptible to attacks, we developed a sophisticated transformation algorithm that injects non-deterministic information to the deterministic results of a one-to-n item mapping scheme. We proved that our encryption technique cannot be broken by one-to-one decryption attacks. To test the vulnerability of the proposed scheme against adversaries, we ran a generic algorithm that has background knowledge about the frequencies of the itemsets. The results show that our encryption technique is very robust to attacks as opposed to simple one-to-one ciphers, which can be easily broken with the help of background knowledge. Finally, we showed through experimentation that the encryption cost is affordable and much lower than the mining cost if the task were to be performed at the owner side.

9 References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 1994.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD*, 2000.

[4] A. Albasall and A. Wahdan. Genetic algorithm cryptanalysis of a feistel type block cipher. In *ICEEC*, 2004.

[5] A. Albassall and A. Wahdan. Genetic algorithm cryptanalysis of the basic substitution permutation network. In *MWSCAS*, 2003.

[6] C. Clifton and D. Marks. Security and privacy implications of data mining. In *SIGMOD Workshop on Data Mining and Knowledge Discovery*, 1996.

[7] G. I. Davida, D. L. Wells, and J. B. Kam. A database encryption system with subkeys. *ACM TODS*, 6(2):312–328, 1981.

[8] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *SIGKDD*, 2002.

[9] W. Forsyth and R. Safavi-Naini. Automated cryptanalysis of substitution ciphers. *Cryptologia*, 17(4):407–418, 1993.

[10] M. A. Garici and H. Drias. Cryptanalysis of substitution ciphers using scatter search. In *IWINAC*, 2005.

[11] B. Gilburd, A. Schuster, and R. Wolff. A new privacy model and association-rule mining algorithm for large-scale distributed environments. In *VLDB*, 2005.

[12] G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *FIMI*, 2003.

[13] J. He and M. Wang. Cryptography and relational database management systems. In *IDEAS*, 2001.

[14] B. Iyer, S. Mehrotra, E. Mykletun, G. Tsudik, and Y. Wu. A framework for efficient storage security in rdbms. In *EDBT*, 2004.

[15] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *TKDE*, 16(9):1026–1037, 2004.

[16] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. *ACM TOS*, 2(2):107–138, 2006.

[17] S. Peleg and A. Rosenfeld. Breaking substitution ciphers using a relaxation algorithm. *Communications of the ACM*, 22(11):598–605, 1979.

[18] R. Spillman, M. Janssen, B. Nelson, and M. Kepner. Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia*, XVII(1):31–44, 1993.

[19] J. Vaidya and C. Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Security*, 13(4):593–622, 2005.

[20] D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994.

[21] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *VLDB*, 2006.