

# Privacy-preserving Distributed Mining of Association Rules on Horizontally Partitioned Data

Murat Kantarcioglu  
kanmurat@cs.purdue.edu

Purdue University

Chris Clifton  
clifton@cs.purdue.edu

## Abstract

Data mining can extract important knowledge from large data collections – but sometimes these collections are split among various parties. Privacy concerns may prevent the parties from directly sharing the data, and some types of information about the data. This paper addresses secure mining of association rules over horizontally partitioned data. The methods incorporate cryptographic techniques to minimize the information shared, while adding little overhead to the mining task.

## 1 Introduction

Data mining technology has emerged as a means of identifying patterns and trends from large quantities of data. Data mining and data warehousing go hand-in-hand: most tools operate by gathering all data into a central site, then running an algorithm against that data. However, privacy concerns can prevent building a centralized warehouse – data may be distributed among several custodians, none of which are allowed to transfer their data to another site.

This paper addresses the problem of computing association rules within such a scenario. We assume homogeneous databases: All sites have the same schema, but each site has information on different entities. The goal is to produce association rules that hold globally, while limiting the information shared about each site.

Computing association rules without disclosing individual data items is straightforward. We can compute the global support and confidence of an association rule  $AB \Rightarrow C$  knowing only the local supports of  $AB$  and  $ABC$ , and the size of each database:

$$\begin{aligned} \text{support}_{AB \Rightarrow C} &= \frac{\sum_{i=1}^{\text{sites}} \text{support\_count}_{ABC}(i)}{\sum_{i=1}^{\text{sites}} \text{database\_size}(i)} \\ \text{support}_{AB} &= \frac{\sum_{i=1}^{\text{sites}} \text{support\_count}_{AB}(i)}{\sum_{i=1}^{\text{sites}} \text{database\_size}(i)} \end{aligned}$$

$$\text{confidence}_{AB \Rightarrow C} = \frac{\text{support}_{AB \Rightarrow C}}{\text{support}_{AB}}$$

Note that this doesn't require sharing any individual items. We can easily extend an algorithm such as a-priori [2] to the distributed case using the following lemma: If a rule has  $\text{support} > k\%$  globally, it must have  $\text{support} > k\%$  on at least one of the individual sites. A distributed algorithm for this would work as follows: Request that each site send all rules with support at least  $k$ . For each rule returned, request that all sites send the count of items they have that support the rule, and the total count of all items at the site. From this, we can compute the global support of each rule, and (from the lemma) be certain that all rules with support at least  $k$  have been found. More thorough studies of distributed association rule mining can be found in [5, 6].

The above approach protects individual data privacy, but it does require that each site disclose what rules it supports, and how much it supports each potential global rule. What if this information is sensitive? For example, suppose the Centers for Disease Control (CDC), a public agency, would like to mine health records to try to find ways to reduce the proliferation of antibiotic resistant bacteria. Insurance companies have data on patient diseases and prescriptions. Mining this data would allow the discovery of rules such as *Augmentin&Summer*  $\Rightarrow$  *Infection&Fall*, i.e. people taking Augmentin in the summer seem to have recurring infections.

The problem is that insurance companies will be concerned about sharing this data. Not only must the privacy of patient records be maintained, but insurers will be unwilling to release rules pertaining only to them. Imagine a rule indicating a high rate of complications with a particular medical procedure. If this rule doesn't hold globally, the insurer would like to know this – they can then try to pinpoint the problem with their policies and improve patient care. But if the fact that the insurer's data supports this rule is revealed (say, under a Freedom of Information Act request to the CDC), the insurer could be exposed

to significant public relations or liability problems. This potential risk could exceed their own perception of the benefit of participating in the CDC study.

This paper presents a solution that preserves such secrets – the parties learn (almost) nothing beyond the global results. The solution is efficient: The additional cost relative to previous non-secure techniques is  $O(\text{candidate\_itemsets} * \text{sites})$  encryptions, and a constant increase in the number of messages.

We assume three or more parties. In the two party case, knowing a rule is supported globally and not supported at one’s own site reveals that the other site supports the rule. Thus much of the knowledge we try to protect is revealed even with a completely secure method for computing the global results. We discuss the two party case further in Section 4. By the same argument, we assume no collusion, as colluding parties can reduce this to the two party case.

## 1.1 Related Work

Previous work in privacy-preserving data mining has addressed two issues. In one, the aim is preserving customer privacy by distorting the data values [3]. The idea is that the distorted data does not reveal private information, and thus is “safe” to use for mining. The key result is that the distorted data, and information on the distribution of the random data used to distort the data, can be used to generate an approximation to the original data *distribution*, without revealing the original data *values*. The distribution is used to improve mining results over mining the distorted data directly, primarily through selection of split points to “bin” continuous data. Later refinement of this approach tightened the bounds on what private information is disclosed, by showing that the ability to reconstruct the distribution can be used to tighten estimates of original values based on the distorted data [1].

More recently, the data distortion approach has been applied to boolean association rules [10]. Again, the idea is to modify data values such that reconstruction of the values for any individual transaction is difficult, but the rules learned on the distorted data are still valid. One interesting feature of this work is a flexible definition of privacy; e.g., the ability to correctly guess a value of ‘1’ from the distorted data can be considered a greater threat to privacy than correctly learning a ‘0’.

The data distortion approach addresses a different problem from our work. The assumption with distortion is that the values must be kept private from whoever is doing the mining. We instead assume that *some* parties are allowed to see *some* of the data, just

that nobody is allowed to see *all* the data. In return, we are able to get exact, rather than approximate, results.

The other approach uses cryptographic tools to build decision trees.[8] In this work, the goal is to securely build an ID3 decision tree where the training set is distributed between two parties. The basic idea is, finding the attribute that maximizes information gain is equivalent to finding the attribute that minimizes the conditional entropy. Since the conditional entropy for an attribute for two-party can be written as a sum of the expression of the form  $(v_1 + v_2) \times \log(v_1 + v_2)$ . Authors suggest a way to securely calculate the expression  $(v_1 + v_2) \times \log(v_1 + v_2)$  and show how to use this function for building the ID3 securely. Clearly this approach treats privacy-preserving data mining as a special case of secure multi-party computation[7] and not only aims for preserving individual privacy but also tries to preserve leakage of any information other than the final result. We follow this approach, but address a different problem (association rules), and emphasize the efficiency of the resulting algorithms. A particular difference is that we recognize that some types of information can be exchanged without violating security policies - secure multi-party computation treats leakage of any information other than the final result as a violation. The ability to share non-sensitive data enables highly efficient solutions.

The problem of privately computing association rules in *vertically* partitioned distributed data has also been addressed[11]. Here the problem is how to count support when *transactions* are split across sites, without revealing the contents of individual transactions. The change in the way the data is distributed makes this a much different problem from the one we address here.

## 1.2 Private Association Rule Mining Overview

Our method follows the basic approach outlined on Page 1 except that values are passed between the local data mining sites rather than to a centralized combiner. The two phases are discovering candidate itemsets (those that are frequent on one or more sites), and determining which of the candidate itemsets meet the global support/confidence thresholds.

The first phase (Figure 1) uses commutative encryption. Each party encrypts its own itemsets, then the (already encrypted) itemsets of every other party. These are passed around, with each site decrypting, to obtain the complete set.

In the second phase (Figure 2), an initiating party

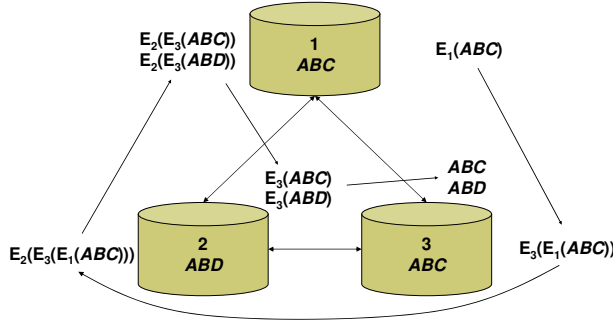


Figure 1: Determining global candidate itemsets

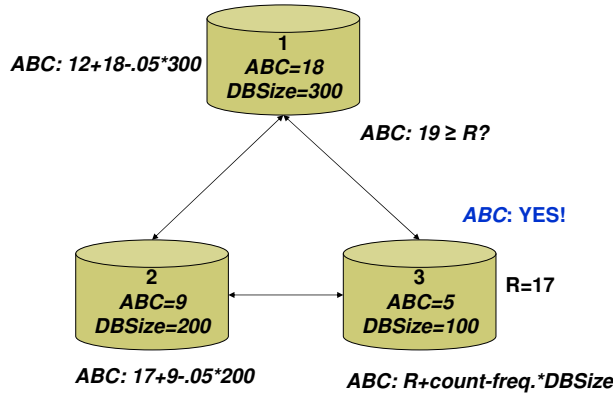


Figure 2: Determining if itemset support exceeds threshold

passes its support count, plus a random value, to its neighbor. The neighbor adds its support count and passes it on. The final party then engages in a secure comparison with the initiating party to determine if the final result is greater than the threshold plus the random value.

This gives a brief, oversimplified idea of how the method works. Section 3 gives full details. Before going into the details, we give background and definitions on techniques from association rule mining, cryptography, and secure multi-party computation used in our method.

## 2 Background

There are several bodies of work that serve as a basis for our work. Here we briefly summarize the related data mining and secure-multi party computation concepts.

### 2.1 Distributed Mining of Association Rules

We address association rule mining as defined in [2]: Let  $I = \{i_1, i_2, \dots, i_n\}$  be a set of items and  $DB$  be a set of transactions, where each transaction  $T \in DB$  is an itemset such that  $T \subseteq I$ . Given an itemset  $X \subseteq I$ , a transaction  $T$  contains  $X$  if and only if  $X \subseteq T$ . An association rule is an implication of the form  $X \Rightarrow Y$  where  $X \subseteq I, Y \subseteq I$  and  $X \cap Y = \emptyset$ . The rule  $X \Rightarrow Y$  has support  $s$  in the transaction database  $DB$  if  $s\%$  of transactions in  $DB$  contain  $X \cup Y$ . The rule has confidence  $c$  if  $c\%$  of the transactions in  $DB$  that contain  $X$  also contains  $Y$ . An itemset  $X$  with  $k$  items is called a  $k$ -itemset. The problem of mining association rules is to find all rules whose support and confidence are higher than a specified minimum support and confidence.

In a horizontally partitioned database, The transactions are distributed among  $n$  sites  $S_1, S_2, \dots, S_n$  such that each site  $S_i$  contains a disjoint subset  $DB_i$  of the transactions.  $DB = DB_1 \cup DB_2 \cup \dots \cup DB_n$ . The itemset  $X$  has a local support count of  $X.sup_i$  at site  $S_i$  if  $X.sup_i$  of the transactions at  $S_i$  contains  $X$ . The global support count of  $X$  is given as  $X.sup = \sum_{i=1}^n X.sup_i$ . An itemset  $X$  is globally supported if  $X.sup \geq s * |DB| = s * \sum_{i=1}^n DB_i$ . The global confidence of a rule  $X \Rightarrow Y$  can be given as  $\{X \cup Y\}.sup / X.sup$ . A  $k$ -itemset is called globally large  $k$ -itemset if it is globally supported and referred as  $L_{(k)}$ . The  $k$ -itemset is called locally large if it is supported locally at some site  $S_i$  and referred as  $LL_{i(k)}$ . If a  $k$ -itemset is locally large at some site  $S_i$  and is globally large it is referred to as  $GL_{i(k)}$ . The aim of distributed association rule mining is to find all rules whose global support and global confidence are higher than the user specified minimum support and confidence.

The FDM algorithm [5] is a fast method for distributed mining of association rules. We summarize this below:

- Candidate Set Generation:** Intersect the globally large itemsets of size  $k-1$ ,  $G_{(k-1)}$ , with locally large  $k-1$  itemsets to get  $GL_{i(k-1)}$ . From these, use the classic apriori candidate generation algorithm to get the candidate  $k$  itemsets  $CG_{i(k)}$ .
- Local Pruning:** For each  $X \in CG_{i(k)}$ , scan the database  $DB_i$  to compute  $X.sup_i$ . If  $X.sup_i / DB_i \geq s\%$ ,  $X$  is locally large and is included in  $LL_{i(k)}$  set.
- Itemset Exchange:** Broadcast  $LL_{i(k)}$  to all sites – the union is  $LL_{(k)}$ , a superset of the possi-

ble global frequent itemsets. Each site computes (using apriori) the support of items in  $LL_{(k)}$ .

4. **Support Count Exchange:** Broadcast the computed supports. From these, each site computes  $G_{(k)}$ .

The above algorithm avoids disclosing individual transactions, but does expose significant information about the rules supported at each site. Our goal is to approximate the efficiency of the above algorithm, without requiring that any site disclose  $LL_{i(k)}$ ,  $X.sup_i$ , or  $|DB_i|$ .

## 2.2 Cryptographic Tools and Secure Multi-party Computation

Secure multi-party computation involves computing a function where multiple parties hold the inputs, and at the end of the computation each party should know the result of the function, their own input, *and nothing else*[12]. It has been shown that given certain cryptographic assumptions, any function that can be represented by a polynomial size circuit can be computed securely. The basic idea behind the proof is to show how to securely compute a logic gate – then represent the desired function as a circuit and evaluate it. A full discussion can be found in [7]. While effective, this is not an efficient method for problems with large inputs.

A key component of the general secure multi-party computation solution is commutative encryption. As we also use commutative encryption, we give a brief description here. We start with a definition of one way accumulators. The basic idea is to come up with quasi-commutative hash functions. A function  $h$  is said to be quasi-commutative if for given  $x$  and  $y_1, y_2, \dots, y_m$ , the value

$$z = h(h(\dots h(h(x, y_1), y_2) \dots, y_{m-1}), y_m) \quad (1)$$

is the same for every permutation of  $y_i$ . A family of one-way hash functions that are quasi-commutative are said to be one-way accumulators. Since a one-way accumulator is a hash function, if  $x \neq \hat{x}$  they will have different hash values  $z$  for fixed  $y_i$ . In addition to the one to one mapping and quasi-commutative properties of one-way accumulators, we need to be able to retrieve the  $x$  value given  $(z, y_1, \dots, y_m)$ , giving us commutative encryption.

The construction of one-way accumulators of Benaloh and de Mare [4] can be easily modified to let us to retrieve the  $x$  value. Their construction uses the RSA [9] assumption with a slight modification. Instead of using arbitrary primes  $p, q$  for generation

of  $n = pq$ , they require that  $p$  and  $q$  be *safe* primes to prevent collisions. In this context, a prime  $p$  is said to be safe if  $p = 2\hat{p} + 1$  where  $\hat{p}$  is an odd prime. They define the one-way accumulator function as below.

$$h(x, y_i) = x^{y_i} \text{ mod } n \quad (2)$$

Assuming the difficulty of the discrete logarithm, this function may be seen as one-way. We use the above construction with one more addition. Instead of generating random  $y_i$  values, we generate  $y_i, t_i$  pairs such that  $y_i * t_i = 1 \text{ mod } (\varphi(n))$ . This construction is equal to the RSA algorithm except that  $p$  and  $q$  have the above safe property.

## 3 Secure Association Rule Mining

We will now use the tools described above to construct a distributed association rule mining algorithm that preserves the privacy of individual site results. The algorithm given is for three or more parties – the two party case is discussed further in Section 4.

### 3.1 Problem Definition

Let  $i \geq 3$  be the number of sites. Each site has a private database  $DB_i$  with  $d_i$  transactions. We are given support threshold  $s$  and confidence  $c$  as percentages. The goal is the discovery of all association rules satisfying the thresholds, as defined in Section 2.1. We further desire that disclosure be limited: No site should be able to learn contents of a transaction at any other site, what rules are supported by any other site, or the specific value of support/confidence for any rule at any other site, unless that information is revealed by knowledge of one's own data and the final result. (e.g., if a rule has 100% support globally, we know it is supported 100% by all sites.) Here we assume no collusion (this is discussed further in Section 4.)

### 3.2 Method

Our method follows the general approach of the FDM algorithm [5], with special protocols replacing the broadcasts of  $LL_{i(k)}$  and the support count of items in  $LL_{(k)}$ . First we give a method for finding the union of locally supported itemsets without revealing the originator of the particular itemset. We then give a method for securely testing if the support count exceeds the threshold.

### 3.2.1 Secure union of locally large itemsets

In the FDM algorithm (Section 2.1), step 3 reveals the large itemsets supported by each site. We instead exchange locally large itemsets in a way that obscures the source of each itemset. We assume a commutative encryption algorithm with negligible collision probability (Section 2.2). The algorithm is given in Protocol 1.

---

#### Protocol 1 Finding secure union of large itemsets

---

**Require:**  $N$  is number of sites and  $\text{Rule\_set} = \emptyset$  initially {Encryption of all the rules by all sites}

**for** each site  $i$  **do**  
    generate  $LL_{i(k)}$  ;  
    **for** each  $X \in LL_{i(k)}$  **do**  
         $M = \text{newarray}[N]$  ;  
         $Xp = \text{encrypt}(X, e_i)$  ;  
         $M[i] = 1$  ;  
         $\text{Rule\_set} \cup (Xp, M)$  ;  
    **end for**  
**end for** {Site  $i$  encrypts its locally large  $k$ -itemsets and adds them to the global set. Each site then encrypts the itemsets it has not encrypted before}

**for** each site  $i$  **do**  
    **for** each tuple  $(r, M) \in \text{Rule\_set}$  **do**  
        **if**  $M[i] == 0$  **then**  
             $rp = \text{encrypt}(r, e_i)$  ;  
             $M[i] = 1$  ;  
             $Mp = M$  ;  
             $\text{Rule\_set} = (\text{Rule\_set} - \{(r, M)\}) \cup \{(rp, Mp)\}$  ;  
        **end if**  
    **end for**  
**end for**

**for**  $(r, M) \in \text{Rule\_set}$  and  $(rp, Mp) \in \text{Rule\_set}$  **do** {check for duplicates}  
    **if**  $r == rp$  **then**  
         $\text{Rule\_set} = \text{Rule\_set} - \{(r, M)\}$  {Eliminate duplicate itemsets before decrypting};  
    **end if**  
**end for**

**for** each site  $i$  **do** { Each site decrypts every item to get the rule set}  
    **for all**  $(r, M) \in \text{Rule\_set}$  **do**  
         $rd = \text{decrypt}(r, d_i)$  ;  
         $\text{Rule\_set} = (\text{Rule\_set} - \{(r, M)\}) \cup \{(rd)\}$  ;  
    **end for**  
    permute elements in the  $\text{Rule\_set}$   
**end for**  
return  $\text{Rule\_set}$

---

The main idea is that each site encrypts the locally supported itemsets. Each site then encrypts the itemsets from other sites. Since equation 1 holds, dupli-

cates in the locally supported itemsets will be duplicates in the encrypted itemsets, and can be deleted. In addition, the decryption can occur in any order, so by permuting the encrypted itemsets we prevent sites from tracking the source of each itemset.

An example of Protocol 1 in action is given below.

**Example 1** Assume we have three parties for data mining. After local support pruning, site  $S_1$  supports  $\{3, 4, 5\}$ ,  $S_2$  supports  $\{2, 5, 7\}$ , and  $S_3$  supports  $\{5, 7, 9\}$ . Let  $n = 7 \times 11$  where  $7 = 2 * 3 + 1, 11 = 2 * 5 + 1$  are safe primes. Each site has key pairs, both secret:  $S_1$  has  $(13, 37)$ ,  $S_2$  has  $(17, 53)$  and  $S_3$  has  $(19, 19)$ . First, each party encrypts its candidate itemsets with its own key. It then sends these to the other parties. Each party then encrypts the items it had not encrypted before, and sends them on for the third encryption. So after the encryption step, we have the encrypted items  $(26, 58, 31, 63, 31, 63, 60)$ . The duplicate items are removed, giving  $(26, 31, 58, 60, 63)$ . Let us assume decryption starts with  $S_2$ .  $S_2$  permutes and decrypts:  $(D_2(31), D_2(58), D_2(26), D_2(63), D_2(60)) \rightarrow (47, 60, 31, 28, 37)$ . Clearly by looking at the decrypted set,  $S_2$  cannot predict the items.  $S_2$  sends the result to  $S_1$ . Under the assumption that RSA is a trapdoor function,  $S_1$  cannot find any correlation between the initial set before decryption phase and the set it received.  $S_1$  decrypts and permutes the set and send it to  $S_3$ . The set sent by  $S_1$  will look like  $(59, 25, 75, 63, 16)$ . Finally  $S_3$  will output the result,  $(3, 4, 5, 7, 9)$ .

Clearly Protocol 1 finds the union without revealing which itemset belongs to which site. It is not, however, secure under the definitions of secure multi-party computation. It reveals the number of itemsets having common support, e.g. two itemsets are supported by two sites, and the rest by only one site. It does not reveal *which* itemsets these are, but a truly secure computation (as good as each site giving its input to a “trusted party”) could not reveal even this count. Allowing innocuous information leakage (the number of itemsets having common support) allows an algorithm that is sufficiently secure with much lower cost than a fully secure approach.

### 3.2.2 Testing support threshold without revealing support count

Protocol 1 gives the full set of locally large itemsets  $LL_{(k)}$ . We still need to determine which of these itemsets are supported globally. Step 4 of the FDM algorithm forces each site to reveal its own support count for every itemset in  $LL_{(k)}$ . All we really need

to know is for each itemset  $X \in LL(k)$ , is  $X.sup \geq s\% \times |DB|$ ? The following allows us to reduce this to a comparison against a sum of local values (the *excess support* at each site):

$$\begin{aligned} X.sup &\geq s * |DB| = s * \left( \sum_{i=1}^n d_i \right) \\ \sum_{i=1}^n X.sup_i &\geq s * \left( \sum_{i=1}^n d_i \right) \\ \sum_{i=1}^n (X.sup_i - s * d_i) &\geq 0 \end{aligned}$$

Therefore checking for support is equivalent to checking if  $\sum_{i=1}^n (X.sup_i - s * d_i) \geq 0$ . The challenge is to do this without revealing  $X.sup_i$  or  $d_i$ . An algorithm for this is given in Protocol 2.

---

**Protocol 2** Finding the global support counts securely

---

**Require:** rule\_set is  $\emptyset$  initially

**if** site  $i$  is the starting site **then**

**for** each  $r \in$  candidate\_set **do**

generate random number  $x_r$ ;

$t = r.sup_i - s * d_i + x_r$ ;

rule\_set = rule\_set  $\cup$   $\{ (r,t) \}$ ;

**end for**

send the rule\_set to the next site ;

**end if**

**if** site  $i$  is neither starting nor ending site **then**

**for** each  $(r,t) \in$  rule\_set **do**

$\bar{t} = r.sup_i - s * d_i + t$ ;

rule\_set = rule\_set -  $\{ (r,t) \} \cup (r, \bar{t})$  ;

**end for**

send the rule\_set to the next site ;

**end if**

**if** site  $i$  is the last site **then**

**for** each  $(r,t) \in$  rule\_set **do**

$\bar{t} = r.sup_i - s * d_i + t$ ;

securely compute whether  $\bar{t} - x_r \geq 0$  with the first site  $\{$  First site knows the  $x_r \}$

**if**  $\bar{t} - x_r \geq 0$  **then**

multi-cast  $r$  as a globally large itemset.

**end if**

**end for**

**end if**

---

The idea is that the first site generates a random number  $x_r$  for each itemset  $X$ , adds that number to its  $(X.sup_i - s * d_i)$ , and sends it to the next site. The random number masks the actual excess support, so the second site learns nothing about the first site's actual database size or support. The second site adds

its excess support and sends the value on. The random value now hides both support counts. The last site in the change now has  $\sum_{i=1}^n (X.sup_i - s * d_i) + x_r$ , and needs to test if this is  $\geq x_r$ . This can be done securely using Yao's generic method[12]. Clearly this algorithm is secure as long as there is no collusion, as no site can distinguish what it receives from a random number. Alternatively the first site can simply send  $x_r$  to the last site. Now the last site learns the actual excess support, but still does not learn the support values for any single site. In addition, if we consider the excess support to be a valid part of the global result, this method is still secure.

We now sketch a proof of the security of Protocol 2. This proof is based on the definitions of secure given in [7]; due to space considerations we do not repeat those definitions and thus are able to give only a sketch of the proof. In particular, the *semi-honest* model assumes each site follows the protocol, but remembers everything it sees during the protocol – informally, a protocol is secure if an indistinguishable view to this history can be generated solely from the output and one's own data.

**Theorem 1** Protocol 2 privately computes globally supported itemsets in the semi-honest model.

**PROOF SKETCH:** To show that Protocol 2 is secure under the semi-honest model, we have to show that a polynomial time simulator can simulate the view of the parties during the execution of the protocol, based on their local inputs and the global result. We also use the general composition theorem for semi-honest computation. The theorem says that if  $g$  privately reduces to the  $f$  and if there is way to compute  $f$  securely than there is a way to compute  $g$  securely. In our context,  $f$  is the secure comparison of two integers, and  $g$  is Protocol 2. First we show that the view of any site during the addition phase can efficiently simulated given the input of that site and the global output. Site  $i$  uniformly chooses a random number  $s_r$ . Now we show that view and the output of the simulator are computationally indistinguishable by showing that the probability of seeing a given  $x$  in both of them is equal. In the following equations,  $x_r$  is the random number added at the beginning of Protocol 2, selected uniformly among the  $m$  bit two's complement numbers. This  $m$  is chosen such that  $2^{m-1} > |DB|$ . The arithmetic is assumed to be two's compliment, ignoring overflows. The random number in the simulator,  $s_r$ , is chosen uniformly from the same domain. Also note that  $X.sup_i$  is fixed for each site.

$$\begin{aligned}
Pr [VIEW_i^{Protocol2} = x] &= Pr \left[ x_r = x - \sum_{k=1}^{k=i-1} X.sup_i \right] \\
&= \frac{1}{2^m} \\
&= Pr [s_r = x] \\
&= Pr [Simulator_i = x]
\end{aligned}$$

We showed that what each site sees during the addition phase is indistinguishable from that simulated with a random number generator. Since during the comparison phase we can use the generic secure method, from the composition theorem we can conclude that Protocol 2 is secure in the semi-honest model.

### 3.3 Securely Finding Confidence of a Rule

To find if the confidence of a rule  $X \Rightarrow Y$  is higher than the given confidence threshold  $c$ , we have to check if  $\frac{\{X \cup Y\}.sup}{Y.sup} \geq c$ . We will denote the support of  $\{X \cup Y\}.sup_i$  as  $XY.sup_i$  in the following equations.

$$\begin{aligned}
\frac{\{X \cup Y\}.sup}{Y.sup} \geq c &\Rightarrow \frac{\sum_{i=1}^{i=n} XY.sup_i}{\sum_{i=1}^{i=n} X.sup_i} \geq c \\
&\Rightarrow \sum_{i=1}^{i=n} XY.sup_i \geq c * \left( \sum_{i=1}^{i=n} X.sup_i \right) \\
&\Rightarrow \sum_{i=1}^{i=n} (XY.sup_i - c * X.sup_i) \geq 0
\end{aligned}$$

Since each site knows  $XY.sup_i$  and  $X.sup_i$ , we can easily use Protocol 2 to securely calculate the confidence of a rule.

### 3.4 Communication and Computation costs of Mining with Protocols 1 and 2

Let the total number of locally large candidate itemsets be  $|CG_{(k)}|$ , and the number of candidates that can be directly generated by the globally large  $(k-1)$  itemsets be  $|GLC_k|$  ( $apriori\_gen(L_{(k-1)}) = GLC_k$ ). The number of sites is  $n$  and we assume  $m$  bits can represent the  $X.sup_i - d_i$  for every large itemset  $X$ .

Here we give the cost associated with Protocols 1 and 2, ignoring the cost of computing locally frequent itemsets. We compare this with the cost of the (non-secure) FDM algorithm.

The communication cost of protocol 1 totals  $O(|CG_k| * n)$ , divided into  $n$  rounds. The FDM algorithm is also  $O(|CG_k| * n)$ , but in a single

round. The computation cost is  $O(|CG_k| * n)$  encryptions/decryptions plus the underlying computation required in FDM.

Protocol 2 requires communication cost  $O(|GLC_k| * n)$  in  $n$  rounds for the basic addition, as opposed to the same  $O(|GLC_k| * n)$  in a single broadcast round for FDM. If secure comparison is desired at the end of Protocol 2, this requires  $O(|GLC_k| * m)$  1 out of 2 oblivious transfers (the main cost in the comparison phase). Each oblivious transfer takes  $O(t^3)$  where  $t$  is length of the RSA key.

## 4 Conclusions and Further Work

Cryptographic tools can be used to do data mining that would otherwise be prevented due to security concerns. We have given procedures to mine distributed association rules on horizontally partitioned data. We showed that distributed association rule mining can be done efficiently under reasonable security assumptions.

The two party case poses difficulties. If we have only two parties, knowing that a rule is supported globally and not supported at one's own site reveals that the other site supports the rule. This is true no matter how secure computation, it is an artifact of the result. Thus extending to secure computation in the two party case is unlikely to be of use. In addition, Yao's Millionaire's problem can be reduced to securely testing if an itemset is supported globally with two parties. No better solution to this problem than the generic method is known.

Another problem is with collusion. In Protocol 1,  $n - 1$  sites can collude to learn the value of a single site, as the problem reduces to two parties. This is simply a problem with releasing the result, as the colluding parties can simply generate no input values, and the result is exactly the compromised parties values, *regardless of the protocol used*. However, if we have an honest majority we can avoid compromise. Since each site encrypts it's values before exchanging them, the only way of using the protocol to reveal values is to maintain a correspondence between the encrypted values and the final results. When each site decrypts the results, it can scramble the order. Thus the only option for the dishonest collaborators is to have seen the partially encrypted values before, so they can be separated from the target's values. The target's values were never seen without encryption by the target before, so the only separation is to be able to recognize all encrypted values other than

those of the target. Properly ordering encryption and decryption among the honest majority can prevent this, however the details are beyond the scope of this paper.

The collusion problem with Protocol 2 is that the parties  $i$  and  $i + 2$  in the chain can collude to find the exact excess support of party  $i + 1$ . If we instead run Protocol 2 multiple times, with randomly chosen *shares* of the excess support from each site in each run, we can avoid this. By varying the path followed with each share, the number of colluding sites required to compromise a particular site grows. The end result is achieved by summation of the partial results. Again this can be done with an honest majority, however at increased communication cost. Details are beyond the scope of this paper.

We also plan to implement and test this protocol to validate the efficiency in practical terms.

We believe that need for data mining in presence of privacy concerns will increase. Examples include knowledge discovery among intelligence services of different countries and collaboration among corporations without revealing trade secrets. Even within a single multi-national company, privacy laws in different jurisdictions may prevent sharing individual data. Many more examples can be imagined. We would like to see secure algorithms for classification, clustering, etc. Another possibility is secure *approximate* data mining algorithms. Allowing error in the results may enable more efficient algorithms that maintain the desired level of security.

The secure multi-party computation definitions from the cryptography domain may be too restrictive for our purposes. More suitable security definitions that allows parties to choose their desired level of security are needed, allowing *efficient* solutions that maintain the desired security. One line of research is to predict the value of information for a particular organization, allowing tradeoff between disclosure cost, computation cost, and benefit from the result. We believe some ideas from game theory and economics may be relevant.

## References

[1] D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Santa Barbara, California, USA, May 21-23 2001. ACM.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago, Chile, Sept. 12-15 1994. VLDB.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD Conference on Management of Data*, Dallas, TX, May 14-19 2000. ACM.

[4] J. C. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures (extended abstract). In T. Heleseth, editor, *Advances in Cryptology (Proceedings of EuroCrypt '93)*, pages 274–285, Lofthus, Norway, May 1993.

[5] D. W.-L. Cheung, J. Han, V. Ng, A. W.-C. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *Proceedings of the 1996 International Conference on Parallel and Distributed Information Systems (PDIS'96)*, Miami Beach, Florida, USA, Dec. 1996.

[6] D. W.-L. Cheung, V. Ng, A. W.-C. Fu, and Y. Fu. Efficient mining of association rules in distributed databases. *Transactions on Knowledge and Data Engineering*, 8(6):911–922, Dec. 1996.

[7] O. Goldreich. Secure multi-party computation, 1998.

[8] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO 2000*, pages 36–54. Springer-Verlag, Aug. 20-24 2000.

[9] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2), 1978.

[10] S. J. Rizvi and J. R. Haritsa. Privacy-preserving association rule mining. In *Proceedings of 28th International Conference on Very Large Data Bases*. VLDB, Aug. 20-23 2002.

[11] J. S. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, July 23-26 2002.

[12] A. C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE, 1986.