

Privacy-Preserving Algorithms for Distributed Mining of Frequent Itemsets

Sheng Zhong

August 15, 2003

Abstract

Standard algorithms for association rule mining are based on identification of *frequent itemsets*. In this paper, we study how to maintain privacy in distributed mining of frequent itemsets. That is, we study how two (or more) parties find frequent itemsets in a distributed database without revealing each party's portion of the data to the other. The existing solution for vertically partitioned data leaks a significant amount of information, while the existing solution for horizontally partitioned data only works for three parties or more. In this paper, we design algorithms for vertically and horizontally partitioned data, respectively. We give two algorithms for vertically partitioned data: one of them reveals only the support count and the other reveals nothing. Both of them have computational overheads linear in the number of transactions. Our algorithm for horizontally partitioned data works for two parties and above and is more efficient than the existing solution.

Keywords: Data Mining; Association Rule; Frequent Itemset; Distributed Database; Privacy

1 Introduction

Data mining has been studied extensively and applied widely. Through the use of data mining techniques, businesses can discover hidden patterns and rules from a database and then employ them to predict about the future. An important scenario of data mining is *distributed data mining*, in which a database is distributed between two (or more) parties and each party owns a portion of the data. These parties need to collaborate with each other, so that they can jointly mine the data and produce results that are interesting to both of them. Privacy concerns are of great importance in this scenario, because each party does not want to reveal her own portion of the data, although she would like to participate in the mining.

This paper is concerned with a major category of data mining, namely mining of *association rules*. Look at the transaction database of a supermarket. We may find that most of those who buy bread also buy milk. Therefore, “bread \Rightarrow milk,” which means “buying bread implies buying milk,” is a candidate of association rule. Two metrics are defined to measure such a candidate rule: *confidence* and *support*. Here confidence means the number of transactions where both bread and milk are bought divided by the number of transactions where bread is bought. Support means the number of transactions where bread and milk are bought divided

by the overall number of transactions. A candidate is considered a valid association rule if both its confidence and its support are sufficiently high.

Standard algorithms for association rule mining are based on identification of *frequent itemsets* [3]. We say that bread and milk constitute a frequent itemset if, in a sufficiently large percentage of transactions, both of them are bought. If all frequent itemsets can be computed, then all association rules can be computed easily from the frequent itemsets.

In this paper, we study how to maintain privacy in distributed mining of frequent itemsets. That is, we study how two (or more) parties find frequent itemsets in a distributed database without revealing each party's portion of the data to the other. We will formally specify what we mean by "privacy." We will also give solutions for two major types of data partition, namely *vertical partition* and *horizontal partition* (to be defined rigorously in Section 2), respectively, and show that our algorithms preserve privacy.

Related Work To the best of our knowledge, Clifton and his students were the first to study privacy-preserving distributed mining of association rules/frequent itemsets. In [19], Vaidya and Clifton gave a nice algebraic solution for vertically partitioned data. However, this solution can *leak many linear combinations* of each party's private data to the other. Furthermore, to process one candidate of frequent itemset, its computational overhead is quadratic in the number of transactions. In [13, 14], Kantarcioglu and Clifton gave a solution for horizontally partitioned data, which uses Yao's *generic* secure computation protocol as a subprotocol. However, as Goldreich pointed out in [11], generic secure computation protocols are highly expensive for practical purposes. (In data mining problems, because the input size is huge, they can be even more expensive than in other applications.) Furthermore, the solution in [13, 14] only works for three parties or more, not for two parties.

Privacy-preserving data mining has been a topic of active study (see, *e.g.*, papers by Agrawal and his collaborators [2, 1]). In particular, many papers have addressed the privacy issues in mining of association rules/frequent itemsets. Some examples are [7, 9, 17, 16, 18]. However, these papers are concerned with privacy of individual transactions and/or hiding of sensitive rules, rather than privacy in distributed mining.

Privacy-preserving distributed mining was first addressed by Lindell and Pinkas [15]. But their paper only discusses the *classification problem* ("classifying transactions into a discrete set of categories"), not the association rule problem.

As pointed out in [8], the problems of privacy-preserving data mining can be viewed as an application of generic secure computation. Existing protocols for generic secure computation [20, 4, 10, 6] can solve such problems in theory. However, these generic protocols are highly expensive and therefore it is our goal to design special-purpose solutions that are much more efficient for our problems.

Our Contributions In this paper, we rigorously specify the problems and security requirements of privacy-preserving mining of frequent itemsets. We give algorithms for vertically and horizontally partitioned data, respectively.

For vertically partitioned data, we design algorithms with two levels of privacy, respectively. The privacy guarantee of *both* levels is superior to the existing solution. Our algorithms are very efficient in that their computational overheads are linear in the number of transactions.

For horizontally partitioned data, our algorithm is more efficient than the existing solution.

In addition, our algorithm works not only for three parties and above, but also for two parties.

Paper Organization The rest of this paper is organized as follows. In Section 2, we present the problem formulation and the definitions of privacy. In Sections 3 and 4, we describe two-party algorithms for vertically partitioned data, with weak privacy and strong privacy, respectively. In Section 5, we give a two-party algorithm for horizontally partitioned data. In section 6, we show how to extend the algorithms to distributed mining of more than two parties. We conclude in Section 7.

2 Technical Preliminaries

2.1 Problem Formulation

Association Rule and Frequent Itemset We adopt the following standard formulation of association-rule mining: Assume that $\mathcal{I} = \{I_1, \dots, I_m\}$ is a set of literals, which are called *items*. We call any subset of \mathcal{I} an *itemset*. Assume that $\mathcal{T} = \{T_1, \dots, T_n\}$ is a set of transactions, where each transaction T_i is a set of items (*i.e.*, $T_i \subseteq \mathcal{I}$). We say that a transaction T_i *contains* an itemset X if and only if $X \subseteq T_i$. An *association rule* is of the form $X \Rightarrow Y$, where X and Y are non-empty itemsets such that $X \cap Y = \Phi$.

Such an association rule *holds* in the transaction set \mathcal{T} with confidence $\alpha\%$ if $\alpha\%$ of the transactions containing X also contain Y . Such an association rule has support $\beta\%$ if $\beta\%$ of the transactions contain both X and Y .

The major technical problem in association rule mining is *frequent itemset* identification. An itemset is frequent if and only if it is contained by $\beta\%$ of the transactions.

Matrix Representation Mathematically, the transaction set \mathcal{T} can be represented by a boolean matrix \mathcal{D} . Each row of the matrix corresponds to a transaction, while each column corresponds to an item. A matrix element $\mathcal{D}(i, j)$ is 1 if the i th transaction T_i contains the j th item I_j ; it is 0 otherwise. The following example illustrates how to convert the transaction set \mathcal{T} to the boolean matrix \mathcal{D} .

	Bread	Milk	Eggs	
Transaction 1	✓		✓	1 0 1
Transaction 2	✓			1 0 0
Transaction 3	✓	✓	✓	1 1 1
Transaction 4		✓	✓	0 1 1

We define the *support count* of an itemset as the number of transactions that contain this itemset. Formally, let C be the set of columns corresponding to an itemset. The support count of the itemset $\{I_j | j \in C\}$ is $S = |\{i | \forall j \in C, \mathcal{D}(i, j) = 1\}|$. Therefore, to decide whether the itemset $\{I_j | j \in C\}$ is frequent, we actually need to decide whether $S > \beta\% \cdot n$ (recall that n is the number of transactions).

As pointed out in [19], the support count S is essentially the inner product of all columns

in set C . Because D is a *boolean* matrix,

$$\begin{aligned}
S &= |\{i | \forall j \in C, \mathcal{D}(i, j) = 1\}| \\
&= |\{i | \prod_{j \in C} \mathcal{D}(i, j) = 1\}| \\
&= \sum_{i=1}^n \prod_{j \in C} \mathcal{D}(i, j) \\
&(\ = \text{InnerProduct}_{j \in C} \vec{\mathcal{D}}_j),
\end{aligned}$$

where $\vec{\mathcal{D}}_j = (D_{1,j}, \dots, D_{n,j})$ stands for the j th column of \mathcal{D} . Therefore, the problem of frequent itemset mining amounts to comparing this inner product with the threshold $\beta\% \cdot n$.

Vertical Partition and Horizontal Partition We consider our problem with respect to two types of data partition, namely vertical partition and horizontal partition. Intuitively, vertical partition means that each party owns some columns of the matrix D , while horizontal partition means that each party owns some rows. For simplicity, at this point we only discuss two-party distributed mining, and leave the extension to more parties to Section 6. Suppose that the two parties are A and B .

Formally, if the data is vertically partitioned, then A (resp., B) owns a set C_A (resp., C_B) of *columns* of the boolean matrix D , where $C_A \cup C_B = [1, m]$ and $C_A \cap C_B = \Phi$. Recall that we are studying an itemset $\{I_j | j \in C\}$. The column set C of this itemset is partitioned into two subsets — $C \cap C_A$ which is owned by A , and $C \cap C_B$ which is owned by B . It is easy to see

$$S = \sum_{i=1}^n \prod_{j \in C} \mathcal{D}(i, j) = \sum_{i=1}^n \left(\prod_{j \in C \cap C_A} \mathcal{D}(i, j) \cdot \prod_{j \in C \cap C_B} \mathcal{D}(i, j) \right).$$

Let $x_i = \prod_{j \in C \cap C_A} \mathcal{D}(i, j)$ and $y_i = \prod_{j \in C \cap C_B} \mathcal{D}(i, j)$. Note that A can privately compute all x_i s and B can privately all y_i s. Let $t = \beta\% \cdot n$. Therefore, our problem can be formulated as follows.

Problem 1 (*Problem for Vertically Partitioned Data*) A has a private input $\vec{x} = (x_1, \dots, x_n)$ ($x_i \in \{0, 1\}$), and B has a private input $\vec{y} = (y_1, \dots, y_n)$ ($y_i \in \{0, 1\}$). A and B have a public input $t \in [0, n]$. Design a two-party algorithm to decide whether $\sum_{i=1}^n x_i y_i > t$.

If the data is horizontally partitioned, then A (resp., B) owns a set R_A (resp., R_B) of *rows*, where $R_A \cup R_B = [1, n]$ and $R_A \cap R_B = \Phi$. It is easy to see

$$S = \sum_{i=1}^n \prod_{j \in C} \mathcal{D}(i, j) = \sum_{i \in R_A} \prod_{j \in C} \mathcal{D}(i, j) + \sum_{i \in R_B} \prod_{j \in C} \mathcal{D}(i, j).$$

Let $x = \sum_{i \in R_A} \prod_{j \in C} \mathcal{D}(i, j)$ and $y = \sum_{i \in R_B} \prod_{j \in C} \mathcal{D}(i, j)$. Note that A can privately compute x and B can privately compute y . Therefore, our problem can be formulated as follows.

Problem 2 (*Problem for Horizontally Partitioned Data*) A has a private input $x \in [0, n]$, and B has a private input $y \in [0, n]$. A and B have a public input $t \in [0, n]$. Design a two-party algorithm to decide whether $x + y > t$.

2.2 Definitions of Privacy

As in existing works on privacy-preserving distributed mining [15, 19, 13, 14], we assume that the participants are *semi-honest* (i.e., *honest-but-curious*).¹ We specify our privacy requirements by adapting the standard definition of privacy for *deterministic* interactive protocols [11] to our distributed frequent-itemset mining problems. Our definitions will apply to both the problem for vertically partitioned data and the problem for horizontally partitioned data.

Let s be a security parameter. Recall that A has private input x , B has private input y , and there is a public input t . Denote by $VIEW_A(x, y, t)$ (resp., $VIEW_B(x, y, t)$) the view of A (resp., B) — note that $VIEW_A$ (resp., $VIEW_B$) is defined to include x (resp., y), t , A 's coin flips (resp., B 's coin flips), and all messages A (resp., B) receives. Use $\stackrel{C}{\equiv}$ to denote computational indistinguishability.

Definition 3 *A two-party distributed algorithm for frequent itemset mining leaks only α to each party if there exist probabilistic polynomial time simulators S_A and S_B for the views of A and B , respectively, such that*

$$\begin{aligned} \{S_A(x, t, \alpha)\}_{x,t} &\stackrel{C}{\equiv} \{VIEW_A(x, y, t)\}_{x,t}, \\ \{S_B(y, t, \alpha)\}_{y,t} &\stackrel{C}{\equiv} \{VIEW_B(x, y, t)\}_{y,t}. \end{aligned}$$

In this definition, we use α to represent the information leaked by the algorithm. The definition states that, for each party, there exists a simulator that can simulate her view, given her own private input, the public input, and α . Therefore, everything learned by this party is implied by α (and her own private input as well as the public input).

It is clear that, in the best possible case, we could have an algorithm that leaks nothing but its output. For frequent itemset mining, it is often also acceptable for an algorithm to leak the support count of a candidate. So we distinguish two levels of privacy, namely strong privacy and weak privacy, respectively.

Definition 4 *A two-party distributed algorithm for frequent itemset mining is strongly privacy-preserving if it leaks only o , where $o = 1$ if the candidate itemset is frequent; $o = 0$ otherwise.*

A two-party distributed algorithm for frequent itemset mining is weakly privacy-preserving if it leaks only S , the support count of the candidate itemset.

3 Weakly Privacy-Preserving Algorithm for Vertically Partitioned Data

3.1 Overview

Recall that, in the problem of vertically partitioned data, A has a private input (x_1, \dots, x_n) and B has a private input (y_1, \dots, y_n) . We need to design an algorithm to decide whether

¹Zero-knowledge proofs can be used to force a *malicious* party to follow the protocol. However, they are highly expensive in general. Since our goal is to develop efficient solutions to mining problems, we do not consider a malicious adversary in this paper.

$\sum_{i=1}^n x_i y_i > t$, where t is a public input.

We build our weakly privacy-preserving algorithm based on probabilistic public-key encryption. Consider a probabilistic public-key encryption scheme whose cleartext space is $\{0, 1\}$. Let $E_{k_p}(x_i, r_i)$ stand for an encryption of cleartext x_i using public key k_p and random string r_i . Let $D(Z_i, k_s)$ stand for the decryption of ciphertext Z_i using private key k_s . Assume that we have a rerandomization algorithm that can compute another encryption of the same cleartext from any ciphertext. One such encryption scheme is the well-known Goldwasser-Micali encryption scheme [12], which is semantically secure under the standard Quadratic Residue Assumption (QRA).

For weak privacy, we only need to compute $S = \sum_{i=1}^n x_i y_i$ and compare it to the threshold t . The main idea of our algorithm is that A counts the number of 1s in a random permutation of $(x_1 y_1, \dots, x_n y_n)$ — this number equals to $\sum_{i=1}^n x_i y_i$. As to privacy, A cannot learn more information because she only sees a random permutation.

More specifically, the algorithm has 3 steps. In Step 1, A encrypts (x_1, \dots, x_n) using her *own* public key (so that B cannot decrypt them) and sends the encryptions to B . In Step 2, B computes encryptions of $(x_1 y_1, \dots, x_n y_n)$ from these encryptions. Then B rerandomizes the newly computed encryptions, repermutes them, and sends them to A . In Step 3, A decrypts the encryptions she received and counts the number of 1s and compare it to the threshold.

The only thing left is how B computes encryptions of $(x_1 y_1, \dots, x_n y_n)$ from encryptions of (x_1, \dots, x_n) . Observe that $x_i y_i = x_i$ if $y_i = 1$, and $x_i y_i = 0$ otherwise. Therefore, if $y_i = 1$, B simply takes the encryption of x_i as an encryption of $x_i y_i$. Otherwise, B computes an encryption of 0.

3.2 Algorithm

Mine1(A, B, \vec{x}, \vec{y})

A 's Input: $\vec{x} = (x_1, \dots, x_n)$ ($x_i \in \{0, 1\}$); (k_s, k_p) ;

B 's Input: $\vec{y} = (y_1, \dots, y_n)$ ($y_i \in \{0, 1\}$); k_p ;

Public Input: $t \in [0, n]$.

Step 1

(1.1) For $i = 1, \dots, n$, A encrypts x_i using public key k_p : $X_i = E_{k_p}(x_i, r_i)$, where r_i is picked uniformly at random.

(1.2) A sends $\vec{X} = (X_1, \dots, X_n)$ to B .

Step 2

(2.1) For $i = 1, \dots, n$, B computes Z_i , an encryption of $z_i = x_i y_i$ as follows:

- If $y_i = 1$, $Z_i = X_i$; otherwise, $Z_i = E_{k_p}(0, 0)$.

(2.2) For $i = 1, \dots, n$, B rerandomizes Z_i .

(2.3) B repermutes \vec{Z} as follows:

- For $i = 1, \dots, n$, $Z_i = Z_{\pi(i)}$, where π is a random permutation on $[1, n]$.

(2.4) B sends $\vec{Z} = (Z_1, \dots, Z_n)$ to A .

Step 3

(3.1) For $i = 1, \dots, n$, A decrypts Z_i to get cleartext z_i : $z_i = D(Z_i, k_s)$.

(3.2) A counts the number of 1's in $\{z_1, \dots, z_n\}$. If the number is greater than t , then A outputs “This is a frequent itemset;” otherwise, A outputs “This is not a frequent itemset.”

3.3 Security Analysis

Theorem 5 *Mine1 is weakly privacy-preserving, if the encryption E is semantically secure.*

Proof: We construct S_A as follows. First, S_A simulates the coin flips of A in the algorithm. To simulate message \vec{Z} , S_A computes S encryptions of 1 and $n - S$ encryptions of 0, rerandomizes them at random and repermutes them at random.

We construct S_B as follows. First, S_B simulates the coin flips of B in the algorithm. S_B simulate message \vec{X} by generating n random ciphertexts. If $S > t$, S_B simulates the public output using “This is a frequent itemset;” if $S \leq t$, it simulates the public output using “This is not a frequent itemset.” \square

3.4 Efficiency Analysis

Computational Overhead The algorithm Mine1 needs to compute at most $2n$ encryptions, n rerandomizations, and n decryptions. Therefore, its computational overhead is linear in n , while the existing solution [13, 14] has a computational overhead quadratic in n . This is a significant improvement because n is very large in data mining problems — it is not unusual for a transaction database to have millions of transactions.

Assume that we use the Goldwasser-Micali encryption scheme. Then each encryption amounts to one modular multiplication, where the modulus is s -bit. Each rerandomization is worth two modular multiplications. Decryption is more expensive — two modular exponentiations, which are equivalent to no more than $2s$ modular multiplications. To summarize, the overall computational overhead is no more than $(2s + 4)n$ modular multiplications.

Communication Overhead The algorithm Mine1 needs to transfer $2n$ items, each of s bits. Therefore, the overall communication overhead is $2sn$ bits.

4 Strongly Privacy-Preserving Algorithm for Vertically Partitioned Data

4.1 Overview

Homomorphic Encryption We build our strongly privacy-preserving algorithm based on a class of homomorphic encryption. We need a probabilistic public-key encryption algorithm F that satisfies the following conditions:

- The cleartext space \mathcal{M} is a large field of size $\Theta(2^s)$. In particular, the size is greater than $2n + 1$.

- It is *not* necessary to have an efficient decryption algorithm; however, there exists an efficient algorithm that uses the private key to decide whether a ciphertext decrypts to 0.
- There is an efficient rerandomization algorithm.
- F is additively homomorphic. That is, for $m_1, m_2 \in \mathcal{M}$,

$$F(m_1, r_1) \uplus F(m_2, r_2)$$

is an encryption of $m_1 + m_2$, where \uplus is an “addition” operation that can be performed without decrypting $F(m_1, r_1)$ or $F(m_2, r_2)$.

- F allows homomorphic computing of constant multiplication. That is, for $m_1 \in \mathcal{M}$ and constant c_1 ,

$$c_1 \circ F(m_1, r_1)$$

is an encryption of $c_1 m_1$, where \circ is a “constant multiplication” operation that can be performed without decrypting $F(m_1, r_1)$.

One example of F is a variant of ElGamal encryption: $F_{k_p}(m_i, r_i) = (g^{m_i}(k_p)^{r_i}, g^{r_i})$, where g is a generator of a group in which discrete logarithm is hard. Note that F is semantically secure under the standard Decisional Diffie-Hellman (DDH) Assumption [5]. The second condition above is satisfied because we can use the private key to compute g^{m_i} from a ciphertext and compare it with g^0 , *i.e.*, compare with 1. To satisfy the fourth condition, we define, for any ciphertexts (M_1, G_1) and (M_2, G_2) ,

$$(M_1, G_1) \uplus (M_2, G_2) = (M_1 M_2, G_1 G_2).$$

To satisfy the fifth condition, we define, for any constant c_1 ,

$$c_1 \circ (M_1, G_1) = (M_1^{c_1}, G_1^{c_1}).$$

The reader can easily verify that \uplus implements addition and \circ implements constant multiplication.

Algorithm Design Recall that the support count of the candidate itemset is S , *i.e.*, $S = \sum_{i=1}^n x_i y_i$. For strong privacy, our algorithm needs to decide whether $S > t$ without revealing S to either A or B . The main idea of our algorithm is that $S > t$ if and only if there exists a 0 in $(S - t - 1, \dots, S - t - n)$.² We would be able to solve this problem immediately if the vector $(S - t - 1, \dots, S - t - n)$ could be revealed to A or B . However, for strong privacy, this vector cannot be revealed. Therefore, we reveal a masked vector $(r_1(S - t - 1), \dots, r_n(S - t - n))$ instead, where r_1, \dots, r_n are random non-zero elements of \mathcal{M} . Note that this masked vector has a 0 if and only if the original vector has a 0. On the other hand, all non-zero elements in the original vector has been replaced by randomized elements in the masked vector, so that no extra information is leaked. In this way, the algorithm can decide whether $S > t$ without revealing any extra information.

²We can prove this fact as follows. $S > t \Leftrightarrow 0 < S - t (\leq n) \Leftrightarrow S - t \in [1, n] \Leftrightarrow$ there exists $i \in [1, n]$ such that $S - t = i \Leftrightarrow$ there exists $i \in [1, n]$ such that $S - t - i = 0 \Leftrightarrow$ there exists a 0 in $(S - t - 1, \dots, S - t - n)$.

More specifically, the algorithm has 3 steps. In Step 1, A encrypts (x_1, \dots, x_n) using her *own* public key (so that B cannot decrypt them) and sends the encryptions to B . In Step 2, B computes encryptions of $(r_1(S - t - 1), \dots, r_n(S - t - n))$ from these encryptions. Then B rerandomizes the newly computed encryptions, repermutes them, and sends them to A . In Step 3, A checks these encryptions to see whether there is one that decrypts to 0.

The only thing left is how B computes encryptions of $(r_1(S - t - 1), \dots, r_n(S - t - n))$ from the encryptions of (x_1, \dots, x_n) . Observe that, because $x_i, y_i \in \{0, 1\}$, we have $S = \sum_{i=1}^n x_i y_i = \sum_{y_i=1} x_i$. Therefore, B can sum up all encryptions of x_i where $y_i = 1$, to get an encryption of S . Then, using the homomorphic property of F , B can compute encryptions of $(r_1(S - t - 1), \dots, r_n(S - t - n))$, because t is public and r_i s are picked by herself.

4.2 Algorithm

Mine2(A, B, \vec{x}, \vec{y})

A 's **Input**: $\vec{x} = (x_1, \dots, x_n)$ ($x_i \in \{0, 1\}$); (k_s, k_p) ;

B 's **Input**: $\vec{y} = (y_1, \dots, y_n)$ ($y_i \in \{0, 1\}$); k_p ;

Public Input: $t \in [0, n]$.

Step 1

(1.1) For $i = 1, \dots, n$, A encrypts x_i using public key k_p : $X_i = F_{k_p}(x_i, r_i)$, where r_i is picked uniformly at random.

(1.2) A sends $\vec{X} = (X_1, \dots, X_n)$ to B .

Step 2

(2.1) B computes an encryption \bar{S} of $S = \sum_{i=1}^n x_i y_i$ as follows:

- $\bar{S} = F_{k_p}(0, 0)$;
- For $i = 1, \dots, n$, if $y_i = 1$, $\bar{S} = \bar{S} \uplus X_i$.

(2.2) For $i = 1, \dots, n$, B picks $r_i \in \mathcal{M} - \{0\}$ uniformly at random and computes an encryption of $r_i(S - t - i)$:

$$U_i = r_i \circ (\bar{S} \uplus F(-t - i, 0)).$$

(2.3) For $i = 1, \dots, n$, B rerandomizes U_i .

(2.4) B repermutes $\vec{U} = (U_1, \dots, U_n)$ as follows:

- For $i = 1, \dots, n$, $U_i = U_{\pi(i)}$, where π is a random permutation on $[1, n]$.

(2.5) B sends $\vec{U} = (U_1, \dots, U_n)$ to A .

Step 3

If one of U_i s decrypts to 0, A outputs “This is a frequent itemset;” otherwise, A outputs “This is not a frequent itemset.”

4.3 Security Analysis

Theorem 6 *Mine2 is strongly privacy-preserving, if the encryption F is semantically secure.*

Proof: We construct S_A as follows. S_A simulates the coin flips of A in the algorithm. If $o = 1$, S_A simulates message \vec{U} using one random encryption of 0 and $n - 1$ random encryptions of random elements of $\mathcal{M} - \{0\}$, in a random order; otherwise, it simulates message \vec{U} using n random encryptions of random elements of $\mathcal{M} - \{0\}$, in a random order.

We construct S_B as follows. S_B simulates the coin flips of B in the algorithm. S_B simulates message \vec{X} using n random ciphertexts. If $o = 1$, S_B simulates the public output using “This is a frequent itemset;” if $o = 0$, it simulates the public output using “This is not a frequent itemset.” \square

4.4 Efficiency Analysis

Computational Overhead The algorithm Mine2 needs to compute $2n + 1$ encryptions, and n rerandomizations. It also needs to check n ciphertexts to see whether they decrypt to 0. In addition, it needs to compute \uplus for $2n$ times and \circ for n times. Therefore, its computational overhead is still linear in n and much lower than the existing solution.

Assume that we use the variant of ElGamal encryption: $F_{k_p}(m_i, r_i) = (g^{m_i}(k_p)^{r_i}, g^{r_i})$. Then each encryption amounts to three modular exponentiations plus one modular multiplications, where the modulus is s -bit. Each rerandomization is worth two modular exponentiations plus two modular multiplications. It takes one modular exponentiation plus one modular multiplication to check whether a ciphertext decrypts to 0. Computing \uplus is worth two modular multiplications, while computing \circ is worth two modular exponentiations. To summarize, the overall computational overhead is no more than $(11s + 9)n + (3s + 1)$ modular multiplications.

Communication Overhead The communication overhead of Mine2 is also $2sn$ bits.

5 Algorithm for Horizontally Partitioned Data

5.1 Overview

Recall that, in the problem for horizontally partitioned data, A has a private input x and B has a private input y . We need to design an algorithm to decide whether $x + y > t$ where t is public.

We still build a strongly privacy-preserving algorithm based on homomorphic encryption. We use the homomorphic encryption scheme F specified in Section 4.

For strong privacy, our algorithm needs to decide whether $x + y > t$ without revealing x to B or revealing y to A . The main idea of our algorithm is that $x + y > t$ if and only if there exists a 0 in $(x + y - t - 1, \dots, x + y - t - n)$ (see the footnote in Section 4.1 for why this is true). However, for strong privacy, the vector $(x + y - t - 1, \dots, x + y - t - n)$ cannot be revealed to A or B . Therefore, we reveal a masked vector $(r_1(x + y - t - 1), \dots, r_n(x + y - t - n))$ instead, where r_1, \dots, r_n are random non-zero elements of \mathcal{M} . Note that this masked vector has a 0 if and only if the original vector has a 0. On the other hand, all non-zero elements in

the original vector has been replaced by randomized elements in the masked vector. In this way, the algorithm can decide whether $x + y > t$ without revealing any extra information.

More specifically, the algorithm has 3 steps. In Step 1, A encrypts x using her *own* public key (so that B cannot decrypt them) and sends the encryptions to B . In Step 2, B computes encryptions of $(r_1(x + y - t - 1), \dots, r_n(x + y - t - n))$ from the encryption of x , using the homomorphic property of F . Then B rerandomizes the newly computed encryptions, repermutes them, and sends them to A . In Step 3, A checks these encryptions to see whether there is one that decrypts to 0.

5.2 Algorithm

Mine3(A, B, x, y)

A 's Input: $x \in [0, n]$; (k_s, k_p) ;

B 's Input: $y \in [0, n]$; k_p ;

Public Input: $t \in [0, n]$.

Step 1

(1.1) A encrypts x : $X = F_{k_p}(x, r)$, where r is picked uniformly at random.

(1.2) A sends X to B .

Step 2

(2.1) For $i = 1, \dots, n$, B picks $r_i \in \mathcal{M} - \{0\}$ and computes U_i , an encryption of $u_i = r_i(x + y - t - i)$:

$$U_i = r_i \circ (X \uplus F_{k_p}(y - t - i, 0)).$$

(2.2) For $i = 1, \dots, n$, B rerandomizes U_i .

(2.3) B repermutes $\vec{U} = (U_1, \dots, U_n)$ at random.

(2.4) B sends $\vec{U} = (U_1, \dots, U_n)$ to A .

Step 3

If one of U_i s decrypts to 0, A outputs “This is a frequent itemset;” otherwise, A outputs “This is not a frequent itemset.”

5.3 Security Analysis

Theorem 7 *Mine3 is strongly privacy-preserving, if the encryption F is semantically secure.*

Proof: We construct S_A as follows. S_A simulates the coin flips of A in the algorithm. If $o = 1$, S_A simulates message \vec{U} using one random encryption of 0 and $n - 1$ random encryptions of random elements of $\mathcal{M} - \{0\}$, in a random order; otherwise, S_A simulates \vec{U} using n random encryptions of random elements of $\mathcal{M} - \{0\}$, in a random order.

We construct S_B as follows. S_B simulates the coin flips of B in the algorithm. S_B simulates message X using a random encryption. If $o = 1$, S_B simulates the public output using “This is a frequent itemset;” otherwise, it simulates the public output using “This is not a frequent itemset.” \square

5.4 Efficiency Analysis

Computational Overhead The algorithm Mine3 needs to compute $n + 1$ encryptions, and n rerandomizations. It also needs to check n ciphertexts to see whether they decrypt to 0. In addition, it needs to compute \uplus and \circ for n times, respectively.

Assume that we use the variant of the ElGamal encryption scheme: $F_{k_p}(m_i, r_i) = (g^{m_i}(k_p)^{r_i}, g^{r_i})$. Then the overall computational overhead is $(8s + 6)n + (3s + 1)$ modular multiplications.

The existing solution for horizontally partitioned data only works for three parties or more. In Section 6, we will show how to extend our algorithm Mine3 to more parties. Our extended algorithm is more efficient than the existing solution, because the latter uses Yao's generic secure computation protocol.

Communication Overhead The communication overhead of Mine3 is $sn + s$ bits.

6 Extension to Multiparty Distributed Mining

In this section, we demonstrate how to extend our algorithms to multiparty distributed mining. To avoid overly complicated notations, instead of presenting general algorithms for k parties, we give three-party algorithms for vertically and horizontally partitioned data, respectively. It is straightforward to further extend our algorithms to more parties in a similar way.

6.1 Algorithm for Vertically Partitioned Data

Now we extend our strongly privacy-preserving algorithm Mine2 to three-party distributed mining.

Suppose that we have the third party C with private input (z_1, \dots, z_n) . The extended algorithm has 4 steps. In Step 1, A encrypts (x_1, \dots, x_n) using her own public key and sends the encryptions to B . In Step 2, B computes encryptions of (x_1y_1, \dots, x_ny_n) and sends them to C . In Step 3, C computes $(r_1(\sum_{i=1}^n x_iy_iz_i - t - 1), \dots, r_n(\sum_{i=1}^n x_iy_iz_i - t - n))$. Then C rerandomizes these newly computed encryptions, repermutes them, and sends them to A . In Step 4, A checks the encryptions she received to see whether there is one that decrypts to 0. Note that Steps 1, 3, and 4 of the extended algorithm correspond to Steps 1, 2, and 3 of algorithm Mine2, respectively. The only new step is Step 2, which is based on the fact that $x_iy_i = x_i$ if $y_i = 1$, and $x_iy_i = 0$ otherwise.

Mine4($A, B, C, \vec{x}, \vec{y}, \vec{z}$)

A 's Input: $\vec{x} = (x_1, \dots, x_n)$ ($x_i \in \{0, 1\}$); (k_s, k_p) ;

B 's Input: $\vec{y} = (y_1, \dots, y_n)$ ($y_i \in \{0, 1\}$); k_p ;

C 's Input: $\vec{z} = (z_1, \dots, z_n)$ ($z_i \in \{0, 1\}$); k_p ;

Public Input: $t \in [0, n]$.

Step 1

(1.1) For $i = 1, \dots, n$, A encrypts x_i using public key k_p : $X_i = F_{k_p}(x_i, r_i)$, where r_i is picked uniformly at random.

(1.2) A sends $\vec{X} = (X_1, \dots, X_n)$ to B .

Step 2

(2.1) For $i = 1, \dots, n$, B computes U_i , an encryption of $u_i = x_i y_i$ as follows:

- If $y_i = 1$, B sets $U_i = X_i$; otherwise, B sets $U_i = F_{k_p}(0, 0)$;
- B rerandomizes U_i .

(2.2) B sends $\vec{U} = (U_1, \dots, U_n)$ to C .

Step 3

(3.1) C computes an encryption of $v = \sum_{i=1}^n x_i y_i z_i$ as follows:

- $V = F_{k_p}(0, 0)$;
- For $i = 1, \dots, n$, if $z_i = 1$, $V = V \uplus U_i$.

(3.2) For $i = 1, \dots, n$, C picks $r_i \in \mathcal{M} - \{0\}$ uniformly at random and computes an encryption of $r_i(v - t - i)$:

$$W_i = r_i \circ (V \uplus F_{k_p}(-t - i, 0)).$$

(3.3) For $i = 1, \dots, n$, C rerandomizes W_i .

(3.4) C repermutes $\vec{W} = (W_1, \dots, W_n)$ at random.

(3.5) C sends $\vec{W} = (W_1, \dots, W_n)$ to A .

Step 4

If one of W_i 's decrypts to 0, A outputs “This is a frequent itemset;” otherwise, A outputs “This is not a frequent itemset.”

6.2 Algorithm for Horizontally Partitioned Data

Now we extend our algorithm Mine3 to three-party distributed mining.

Suppose that we have the third party C with private input z . The extended algorithm has 4 steps. In Step 1, A encrypts x using her own public key and sends it to B . In Step 2, B computes an encryption of $x + y$ and sends it to C . In Step 3, C computes encryptions of $(r_1(x + y + z - t - 1), \dots, r_n(x + y + z - t - n))$ using the homomorphic property of F . Then C rerandomizes these newly computed encryptions, repermutes them, and sends them to A . In Step 4, A checks the encryptions she received to see whether there is one that decrypts to 0. Note that Steps 1, 3, and 4 of the extended algorithm correspond to Steps 1, 2, and 3 of algorithm Mine3, respectively. The only new step is Step 2, which is also based on the homomorphic property of F .

Mine5(A, B, C, x, y, z)

A 's Input: $x \in [0, n]$; (k_s, k_p) ;

B 's Input: $y \in [0, n]$; k_p ;

C 's Input: $z \in [0, n]$; k_p ;

Public Input: $t \in [0, n]$.

Step 1

(1.1) A encrypts x : $X = F_{k_p}(x, r)$, where r is picked uniformly at random.

(1.2) A sends X to B .

Step 2

(2.1) B computes U , an encryption of $u = x + y$: $U = X \uplus F_{k_p}(y, 0)$.

(2.2) B rerandomizes U .

(2.3) B sends U to C .

Step 3

(3.1) For $i = 1, \dots, n$, C picks $r_i \in \mathcal{M} - \{0\}$ and computes V_i , an encryption of $v_i = r_i(x + y + z - t - i) = r_i(u + z - t - i)$:

$$V_i = r_i \circ (U \uplus F_{k_p}(z - t - i, 0)).$$

(3.2) For $i = 1, \dots, n$, C rerandomizes V_i .

(3.3) C repermutes $\vec{V} = (V_1, \dots, V_n)$ at random.

(3.4) C sends $\vec{V} = (V_1, \dots, V_n)$ to A .

Step 4

If one of V_i s decrypts to 0, A outputs “This is a frequent itemset;” otherwise, A outputs “This is not a frequent itemset.”

7 Conclusion

In this paper, we study privacy preserving algorithms for distributed mining of frequent itemsets. Our algorithms provide very strong privacy guarantee as defined in cryptography. They have computational overheads linear in the number of transactions and therefore are very efficient.

References

- [1] D. Agrawal and C. C. Agrawal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 247–255, 2001.
- [2] D. Agrawal and R. Srikant. Privacy preserving mining. In *Proceedings of the 2000 ACM SIGMOD Conference on Management of Data*, pages 439–450, 2000.
- [3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithm for mining association rules. In *Proceedings of 20th International Conference on Very Large Data Bases*, pages 487–499, 1994.
- [4] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 1–10, 1988.

- [5] Dan Boneh. The decision Diffie-Hellman problem. In *ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63, 1998.
- [6] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 11–19, 1988.
- [7] Elena Dasseni, Vassilios S. Verykios, Ahmed K. Elmagarmid, and Elisa Bertino. Hiding association rules by using confidence and support. In *Information Hiding Workshop*, pages 369–383, 2001.
- [8] W. Du and M. J. Atallah. Secure multi-party computation problems and their applications: a review and open problems. In *New Security Paradigms Workshop*, pages 11–20, 2001.
- [9] A. Evfimievski, R. Srikant, D. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–228, 2002.
- [10] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all alnguages in NP have zero-knowledge proof systems. *JACM*, 38:691–729, 1991.
- [11] Oded Goldreich. Secure multi-party computation. Working Draft Version 1.1, 1998.
- [12] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
- [13] Murat Kantarcioglu and Chris Clifton. Privacy preserving distributed mining of association rules on horizontally partitioned data. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2002.
- [14] Murat Kantarcioglu and Chris Clifton. Privacy preserving distributed mining of association rules on horizontally partitioned data. Manuscript, Purdue University Department of Computer Science, 2003.
- [15] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology - Proceedings of CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 36–54, 2000.
- [16] Stanley R. M. Oliveira and Osmar R. Zaiane. Privacy preserving frequent itemset mining. In *IEEE International Conference on Data Mining Workshop on Privacy, Security and Data Mining*, volume 14, 2002.
- [17] Shariq J. Rizvi and Jayant R. Haritsa. Maintaining data privacy in association rule mining. In *Proceedings of 28th International Conference on Very Large Data Bases*, 2002.
- [18] Y. Saygin, V. S. Verykios, and A. K. Elmagarmid. Privacy preserving association rule mining. In *Research Issues in Data Engineering (RIDE)*, 2002.
- [19] Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

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