2012

Privacy Preserving Distributed Data Mining

Zhenmin Lin
University of Kentucky, zlin2@uky.edu

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained and attached hereto needed written permission statements(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine).

I hereby grant to The University of Kentucky and its agents the non-exclusive license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless a preapproved embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's dissertation including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Zhenmin Lin, Student

Dr. Jerzy W. Jaromczyk, Major Professor

Dr. Raphael Finkel, Director of Graduate Studies

PRIVACY PRESERVING DISTRIBUTED DATA MINING

DISSERTATION

A dissertation submitted in partial
fulfillment of the requirements for the
degree of Doctor of Philosophy in the
College of Engineering at the
University of Kentucky

By
Zhenmin Lin
Lexington, Kentucky

Director: Dr. Jerzy W. Jaromczyk
Lexington, Kentucky 2012

ABSTRACT OF DISSERTATION

PRIVACY PRESERVING DISTRIBUTED DATA MINING

Privacy preserving distributed data mining aims to design secure protocols which allow multiple parties to conduct collaborative data mining while protecting the data privacy. My research focuses on the design and implementation of privacy preserving two-party protocols based on homomorphic encryption. I present new results in this area, including new secure protocols for basic operations and two fundamental privacy preserving data mining protocols.

I propose a number of secure protocols for basic operations in the additive secret-sharing scheme based on homomorphic encryption. I derive a basic relationship between a secret number and its shares, with which we develop efficient secure comparison and secure division with public divisor protocols. I also design a secure inverse square root protocol based on Newton's iterative method and hence propose a solution for the secure square root problem. In addition, we propose a secure exponential protocol based on Taylor series expansions. All these protocols are implemented using secure multiplication and can be used to develop privacy preserving distributed data mining protocols.

In particular, I develop efficient privacy preserving protocols for two fundamental data mining tasks: multiple linear regression and *EM* clustering. Both protocols work for arbitrarily partitioned datasets. The two-party privacy preserving linear regression protocol is provably secure in the semi-honest model, and the *EM* clustering protocol discloses only the number of iterations. I provide a proof-of-concept implementation of these protocols in C++, based on the Paillier cryptosystem.

KEYWORDS: Privacy Preserving, Data Mining, Secure Computation, Multiple Linear Regression, *EM* Clustering

Zhenmin Lin

December, 13, 2012

PRIVACY PRESERVING DISTRIBUTED DATA MINING

By
Zhenmin Lin

Jerzy Jaromczyk
Director of Dissertation

Raphael Finkel
Director of Graduate Studies

December 13, 2012
Date

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Dr. Jerzy Jaromczyk, who guided me through the whole process of dissertation writing. This work would not have been possible without his support, understanding and technical advice.

I would furthermore like to thank the rest of my committee members: Dr. Tingjian Ge, Dr. Mirek Truszczynski and Dr. Qiang Ye, for their insightful comments and useful suggestions on my work. I would also like to thank the outside examiner Dr. Uwe Nagel for his helpful comments on my dissertation.

Last but not least, I would like to express my great gratitude to my parents, my uncles and my sisters, who have all unconditionally supported and encouraged me throughout my whole life.

Contents

List of Tables

List of Figures

List of Algorithms and Protocols

Chapter 1

Introduction

1.1 Motivation

Privacy preserving distributed data mining aims to design secure protocols which allow multiple parties to conduct collaborative data mining while protecting the privacy of their data. My research focuses on the design and implementation of privacy preserving protocols based on homomorphic encryption. I have designed new secure protocols for a number of basic operations, including comparison, inverse square root, square root and the exponential function. Using these protocols I develop two fundamental privacy preserving data mining protocols: multiple linear regression and EM clustering.

Data mining attempts to discover potentially useful and interesting patterns from large quantities of data. Many useful techniques have been developed in this area, include clustering, classification, regression, association rule mining and outlier detection. They have been been successfully applied in domains ranging from business to science and engineering. For example, in market basket analysis, association rule mining is used to find the purchase patterns of customers. The supermarket can use such knowledge to plan product placement and promotional pricing.

In many applications data may be collected and owned by multiple parties. For example, for a group of persons, the employer companies have their employment information, banks have the financial information, and hospitals have the health data. Suppose that the whole dataset can be represented as a matrix whose rows correspond to subjects and whose columns correspond to attributes. It can be distributed among the parties in two typical ways:

(a) Vertical partition. All the parties have the same set of subjects, but each party has a different set of attributes. The set of data held by employer companies, banks

and hospitals is an example of vertical partition.

(b) Horizontal partition. Multiple parties have disjoint sets of subjects, and each party has the data of all the attributes for each subject he/she holds. For example, different colleges have information on disjoint sets of students.

In some cases the dataset may be arbitrarily partitioned among multiple parties. That is, each party holds part of the dataset in an arbitrary way. This includes vertical partition and horizontal partition as special cases.

When data are distributed among multiple parties, collaborative data mining has the potential to produce more accurate knowledge than the use of data owned by a single party. However, privacy concerns, due to conflict of interests or legal regulation, may prevent such applications. Consider the following examples:

(a) Two clinics, each of which has the patient data for one disease, conjecture that these two diseases may be correlated. They wish to conduct a joint analysis to verify the conjecture. However, due to law regulation, they are prohibited from disclosing their individual data to each other (Zhang et al., 2006).

(b) The garment market is highly competitive. To raise their competitiveness in this market, two companies would like to cooperate to analyze their joint sales data to obtain knowledge such as how the sale of a particular category of garment is distributed in different regions. However, these two companies compete against each other, so they may not be willing to disclose their data.

Privacy preserving distributed data mining addresses this issue and aims to design secure protocols which allow multiple parties to conduct collaborative data mining while protecting the privacy of their data. It has attracted much attention and has become an active research area in recent years.

Secure computation aims to design secure protocols so that a set of parties can perform joint computation privately. The concepts and techniques in secure computation can be applied to the area of privacy preserving distributed data mining. My research focuses on the design of privacy preserving data mining protocols based on homomorphic encryption.

1.2 Short Survey on Data Privacy

In this section, I briefly introduce some related work in the area of privacy preserving data mining, including privacy preserving distributed data mining, data perturbation, anonymization and differential privacy.

Privacy Preserving Distributed Data Mining. Lindell and Pinkas (2000) first applied the concept of secure computation in the field of data mining and developed a provably secure two-party decision tree over horizontally partitioned data. Since then, privacy preserving distributed data mining has attracted much attention and many secure protocols have been proposed for specific data mining algorithms, including support vector machines (Laur et al., 2006; Vaidya and Clifton, 2008a), Bayesian network (Yang and Wright, 2006), k-nearest neighbor (Qi et al., 2008), k-means (Vaidya et al., 2003; Bunn et al., 2007), EM-clustering (Lin et al., 2005), regression (Du et al., 2004; Sanil et al., 2004; Hall et al. 2011), association rules mining (Viadya and Clifton, 2002; Kantarcioulu and Clifton, 2004; Viadya and Clifton, 2005) and outlier detection (Vaidya and Clifton, 2004b).

Some of these protocols are provably secure in the semi-honest model, such as the secure protocols for decision tree (Lindell and Pinkas, 2000), k-means (Bunn et al., 2007) and support vector machine (Laur et al., 2006). Some protocols choose to disclose additional information to achieve better efficiency. For example, the secure support vector machine proposed by Vaidya et al. (2008) discloses the kernel matrix. Other protocols use algebraic techniques to protect the private data. For example, a vector or matrix is protected by multiplying with random matrices (Viadya and Clifton, 2002; Du et al., 2004). However, one should be cautious of using such methods because they may leak significant information (Goethals et al., 2004).

Data Perturbation. Suppose that a government agency would like to publish a set of electronic health records which may facilitate research. One strategy for protecting the privacy of the individual records is to perturb the original data. Agrawal et al. (2000) proposed an additive perturbation method which adds Gaussian noise to the data and they constructed decision tree on the perturbed data to demonstrate its utility. Chen et al. (2005) proposed a random rotation method which multiplies the original data matrix with a random orthogonal matrix. This method can preserve the distances of the original data points. Liu. et al. (2006) proposed a random projection-based multiplicative perturbation method in which the set of data points from high-dimensional space are projected to a randomly chosen low-dimensional subspace.

Anonymization. One obvious way to protect the privacy of tabular data is to remove the identity attributes such as social security number and name. However, this is not sufficient because the combination of some attributes such as age, sex and address can be linked with external data and discloses the identity of the record. These attributes are called quasi-identifiers. A table is said to be k-anonymous if

every record is indistinguishable from at least $k-1$ other records over the quasi-identifier attributes (Sweeney, 2002a). To achieve k-anonymity, we can replace quasi-identifier attributes values with values that are less specific but semantically consistent (Sweeeny, 2002b).

Differential Privacy. In the scenario of a statistical database, a trusted curator collects a set of sensitive information (e.g. medical records). He/she answers the queries issued by the users and provides the statistical information about the data. But he/she doesn't want to compromise the privacy of individual records in the database.

We can model the actions of the trusted curator as a randomized algorithm A. A randomized algorithm A guarantees ϵ-differential privacy (Dwork et al., 2006) if, when D_1 and D_2 are a pair of datasets that differ on a single element, then for all $S \subseteq Range(A)$,

$$Pr[A(D_1) \in S] \leq \exp(\epsilon) * Pr[A(D_2) \in S]$$

where the probability is taken over the coin tosses of A. This means that for any two datasets which are close to one another, a differentially private mechanism will behave approximately the same on both datasets. This definition gives a strong guarantee that the presence or absence of an individual (single element) will not affect the final output of the query significantly.

When the query is a real-valued function, one method to achieve ϵ-differential privacy is to add Laplace noise according to the sensitivity of the query function (Dwork et al., 2006). When the query maps the database to some discrete structures such as strings or trees, McSherry and Twalwar (2007) proposed an exponential mechanism to provide ϵ-differential privacy.

1.3 Contributions

The focus of my research is to design privacy-preserving distributed data mining protocols with secure computation techniques. I implemented privacy-preserving data mining protocols based on homomorphic encryption. I have designed new secure protocols for a number of basic operations and used these protocols to develop privacy preserving distributed data mining protocols. I summarize our contributions below.

1. New secure protocols for basic operations. I derived a basic relationship between a secret and its two shares when we used the additive secret-sharing scheme based on homomorphic encryption (section 3.2.1). With this relationship we

developed two efficient secure protocols: secure comparison (section 3.2) and secure division with public divisor (section 4.5). The new secure comparison protocol needs only $2L + O(1)$ secure multiplications when the comparands belong to a known interval $[0, 2^L)$. Existing protocols require at least $12L + O(1)$ secure multiplications (Bunn et al., 2007; Qi et al., 2008).

In addition, we designed a secure inverse square root protocol based on the Newton iterative method and hence we proposed a solution for secure square root (section 4.6.1). I also developed an efficient secure exponential protocol based on Taylor's series explanation (section 5.4.2). All these protocols are implemented using secure multiplication and can be used to develop privacy preserving data mining protocols.

2. Design and Implementation of fundamental privacy preserving data mining protocols. I have developed privacy preserving protocols for two fundamental data mining tasks: multiple linear regression and *EM* clustering. Privacy preserving linear regression and *EM* clustering have been studied in the literature. Sanil et al. (2004) proposed a privacy preserving linear regression protocol based on the Powell iterative method. It addresses only the case of vertical partition and discloses aggregate information during each iteration. Du et .. (2004) proposed secure matrix multiplication and secure matrix inverse protocols and hence designed privacy preserving linear regression protocols for vertically partitioned datasets. Their method protects the data matrix by multiplying with random matrices, which cannot provide theoretical guarantee about privacy. Hall et al. (2011) presented secure linear regression protocols for arbitrarily partitioned datasets based on homomorphic encryption. They designed a secure protocol to invert a matrix, which they used to invert normal matrices and solve normal equations. Generally, it is not desirable to solve normal equations by inverting the normal matrices because inverting a matrix is more expensive and the normal matrix may be ill-conditioned. I designed privacy preserving multiple linear regression protocols based on the stable QR-decomposition method (chapter 4). They work for arbitrarily partitioned datasets The two-party protocol is provably secure in the semi-honest model.

Lin et al. (2004) presented a privacy preserving *EM* clustering over horizontally partitioned datasets. Their method chooses to disclose the means and the covariance matrix. I proposed a privacy preserving *EM* clustering protocol over arbitrarily partitioned datasets, which includes vertical partition and horizontal partition as special cases. This is the first solution for vertically partitioned datasets. Our two-party protocol discloses only the number of iterations.

Due to the computational cost of encryption/decryption operations and the com-

plexity of data mining tasks, an important concern is whether privacy preserving data mining protocols are practical or they are of only theoretical interest. I have implemented our privacy preserving multiple linear regression and *EM* clustering algorithms in C++ based on the Paillier cryptosystem and evaluated their performances over benchmark datasets. Our experiments show that although the executions of secure protocols are generally slow, they are feasible for small datasets. Further improvements and new techniques are needed to make them more practical for larger datasets.

3. Application of the Schur Complement. When data are distributed among multiple parties, the dataset can be represented as a block matrix with each block held by a party. I explored the possibility of using the structure of the block matrix to design efficient privacy preserving data mining protocols. In particular, I studied the potential application of the Schur Complement in the design of efficient kernel ridge regression protocol (chapter 6).

Chapter 2

Secure Computation

2.1 Introduction

The focus of my research is to design privacy-preserving distributed data mining protocols with secure computation techniques. I design privacy-preserving data mining protocols based on homomorphic encryption. In particular, I implement secure protocols using the Paillier cryptosystem. In this chapter, I introduce the background knowledge in secure computation and the additive secret-sharing scheme based on homomorphic encryption.

This chapter is organized as follows. I first introduce the concepts in secure computation in section 2.2. In section 2.3.1, I describe the Paillier cryptosystem, which I use to implement privacy preserving data mining protocols. It is homomorphic, and semantically secure based on some computational assumption. I then describe the additive secret-sharing scheme based on homomorphic encryption in section 2.3.2. The presentation of secure computation in section 2.2 is based on the material from (Goldreich, 2004).

2.2 Concepts in Secure Computation

Let $f : (\{0,1\}^*)^m \rightarrow (\{0,1\}^*)^m$ be an m-ary functionality [1] which maps m inputs $(x_1, \ldots, x_m)$ to m outputs, $(y_1, \ldots, y_m) = f(x_1, \ldots, x_m)$. Here f can be deterministic or randomized. We write $y_i = f_i(x_1, \ldots, x_m)$. For the moment, we assume that all x_i are of the same length. I will discuss later the issues related to relax this

[1] In the area of secure computation, the notion of an m-ary functionality refers to a random process which maps m inputs to m outputs, where functions mapping m inputs to m outputs are a special case. Functionalities are randomized extension of ordinary functions.

assumption. Suppose that there are m parties, each holding a local input x_i. A multi-party protocol for computing the functionality f is a protocol such that if all the parties compute and communicate as specified by the protocol, each party i will obtain his/her desired output $y_i = f_i(x_1, ..., x_m)$ when the protocol finishes. We assume that the functionality f is polynomial-time computable and the protocol runs in polynomial time of the input length.

During the execution of the protocol, the participating parties need to communicate with each other with messages. These message may contain sensitive information. Secure computation is concerned about the privacy of the involved parties when they interact with each other. A protocol is considered to be secure if the participating parties don't disclose any information more than necessary. We require that what a party can learn from the joint computation can be inferred from his/her own input and output. More generally, we require that even if a subset of parties collude, what they can learn from the joint computation can be inferred from their own inputs and outputs.

To give a formal definition of privacy, we need the concept of **computational indistinguishability**.

Definition 2.1. *(Goldreich, 2004) Let $X = \{X_k\}_{k=1,2,...}$ and $Y = \{Y_k\}_{k=1,2,...}$ be two probabilistic ensembles. X and Y are said to be computationally indistinguishable, denoted by*

$$\{X_k\}_{k=1,2,...} \simeq \{Y_k\}_{k=1,2,...},$$

if for every family of polynomial-size circuits $\{C_k\}$, every positive polynomial p and all sufficiently large k, it holds that

$$|Prob(C_k(X_k) = 1) - Prob(C_k(Y_k) = 1)| < \frac{1}{p(k)}. \tag{2.1}$$

The formal definition of privacy can be given based on the simulation paradigm. Suppose that these m parties use a protocol Π to compute the functionality $f(x_1, \ldots, x_m)$. Initially, each party i holds a local input x_i and is supplied with some random coins r_i which are used as the random source for the execution of the protocol. In addition, we also supply each party with a security parameter 1^k. Then these parties perform the computation and communication as specified by the protocol Π and output the results when the protocol finishes.

Let $x = (x_1, \ldots, x_m)$. We denote the output of party i by $\Pi_i(1^k, x)$ and the outputs of the protocol by $\Pi(1^k, x) = (\Pi_i(1^k, x), \ldots, \Pi_m(1^k, x))$. The view of party i during

the execution of the protocol Π is defined as the collection of the security parameter 1^k, his/her input x_i, his/her internal coin toss r_i and all the messages he/she has received during the joint computation $(m_1, ..., m_{t_i})$,

$$VIEW_i^{\Pi}(1^k, x) = (1^k, x_i, r_i, m_1, \ldots, m_{t_i}).$$

Given a subset of parties $I = \{i_1, \ldots, i_s\} \subseteq \{1, 2, \ldots, m\}$, we let $f_I(x) = (f_{i_1}, \ldots, f_{i_s})$ and define

$$VIEW_I^{\Pi}(1^k, x) = (VIEW_{i_1}^{\Pi}(1^k, x), \ldots, VIEW_{i_s}^{\Pi}(1^k, x)).$$

Suppose that a subset of the parties I collude and they try to infer extra information from the computation. We assume the **semi-honest** model. That is, we assume that all the parties follow the protocol and the colluding parties only try to infer useful information from the messages they have received during the joint computation.

Note that in the semi-honest model, all the information these colluding parties obtain is contained in the view of these parties. We consider a protocol to be secure if the view of these colluding parties I can be simulated based on the their own inputs and outputs. That is, there exists a probabilistic polynomial time algorithm, given the inputs and the outputs of the colluding parties I, can simulate the view of these parties. We call this algorithm the simulator. The simulation here means that the output of the simulator is computationally indistinguishable from the view of the colluding parties I.

Definition 2.2. *Let Π be a protocol for m parties to compute the m-ary functionality f. We say that Π privately [2] computes f if there exists a probabilistic polynomial time algorithm, denoted by S, such that for every $I \subseteq \{1, \ldots, m\}$, it holds that*

$$\{(S(I, 1^k, x_I, f_I(x)), f(x))\}_{k=1,2,\ldots} \simeq \{(VIEW_I(1^k, x), \Pi(1^k, x))\}_{k=1,2,\ldots}.$$

Definition 2.2 says that even if a subset of parties I collude, the information they can obtain can be efficiently simulated based only on their own inputs and outputs. Note that when we say Π is a protocol to compute f, we mean the output of the protocol $\Pi(x)$ is identically distributed with $f(x)$. The above definition guarantees both the correctness and the security of the protocol.

An important special case is secure two-party computation.

[2] In this paper, I use the terms secure and private, securely and privately interchangeably.

Definition 2.3. *Suppose that f is a two-ary functionality $(y_1, y_2) = f(x_1, x_2)$, $y_1 = f_1(x_1, x_2)$ and $y_2 = f_2(x_1, x_2)$. Let Π be a protocol for 2 parties to compute f. We say that Π privately computes f if there exists two probabilistic polynomial time algorithms, denoted by S_1 and S_2, such that*

$$\{(S_1(1^k, x_1, f_1(x)), f(x))\}_{k=1,2,\ldots} \simeq \{(VIEW_1(1^k, x), \Pi(1^k, x))\}_{k=1,2,\ldots} \quad (2.2)$$

$$\{(S_2(1^k, x_2, f_2(x)), f(x))\}_{k=1,2,\ldots} \simeq \{(VIEW_2(1^k, x), \Pi(1^k, x))\}_{k=1,2,\ldots}. \quad (2.3)$$

In the definition of secure computation we assume that all the inputs have the same lengths. The reason is that in a protocol to compute any functionality, the program of a party usually depends on the lengths of other parties' inputs. One way to ensure this is to pad the inputs with zeros. Another method is simply to add the lengths of all the inputs as a part of the input of each party. This is more practical in the field of privacy preserving distributed data mining. For example, in the vertically partitioned datasets, we assume that each party knows the number of objects and the numbers of the attributes held by other parties.

Modular Composition of Secure Protocols. A protocol can be augmented with access to an oracle for some functionality g. The access to an oracle means that if each party i provides x_i and they invoke the oracle, they will get the results of $g(x_1, \ldots, x_m)$ instantly without any computational cost. We can define the security of an oracle-aided protocol in the same way as for the ordinary protocol except that the results of the invocations of the oracles are treated as messages and are included in the views of the parties.

Suppose that a protocol Π with an access to the oracle for the functionality g privately computes the functionality f and a secure protocol Ψ privately computes g. If we replace the access to the oracle for g in the protocol Π with the secure protocol Ψ, the composition theorem (Goldreich, 2004, page 673) says that the resulting protocol privately computes the functionality f.

To use the composition theorem, we need to show that Ψ securely computes g and that the protocol Π with an access to the oracle for g privately computes f. The composition theorem allows us to design secure protocols in a modular way. If the functionality to be computed can be implemented using a number of procedures, we can design new secure protocol or use existing protocol for each procedure and then assemble them together to design a secure protocol for the whole functionality.

Feasibility Results. Yao (1988) proposed a general construction of secure protocol for any function in two-party cases. We know that every function can be rep-

resented as a circuit. The first party constructs a "scrambled" circuit which consists of pairs of encrypted secrets that correspond to the wires of the original circuit and gadgets that correspond to the gates of the original circuit. The gadget is constructed in such a way that the knowledge of secrets corresponding to the wire entering the gates yields a secret corresponding to the wire that exists the gate. The first party sends the "scrambled" circuit to the second party. The second party "evaluates" the "scrambled" circuit from top (input wires) to bottom (output wires), obtaining the result, and sends it to the first party.

Goldreich et al. (1987) extended the results by Yao and proposed a secure protocol for any functionality in multi-party cases. The functionality is expressed as a circuit which consists of only AND and NOT gates. Each bit corresponding to a wire is shared by all the parties which each hold a random bit that sums to the secret mod 2. The computation propagates from top (input wires) to bottom (output wires) along the circuit. When a NOT gate is encountered, the first party flips its bit and all other parties maintain their bit values. The secure computation of an AND gate in the multi-party case can be reduced to the secure computation of an AND operation in two-party case, which can be implemented using oblivious transfer.

The existence of general construction of secure protocols for any functionality may be sufficient for traditional cryptographic applications, such as secure exchange and key management. However, data mining deals with large datasets and such general constructions may be not efficient. Design of efficient secure protocols for specific data mining tasks has been an active research topic in the past ten years.

2.3 Additive Secret-Sharing Scheme Based on Homomorphic Encryption

I first introduce the Paillier cryptosystem in subsection 2.3.1. The Paillier cryptosystem is homomorphic and semantically secure. I then describe the additive secret-sharing scheme based on homomorphic encryption in subsection 2.3.2. I use this scheme to design privacy preserving data mining protocols in the following chapters.

2.3.1 Paillier Cryptosystem and Homomorphic Encryption

An encryption scheme is a triple (G, E, D) of probabilistic polynomial-time algorithms, where G is the key-generator algorithm, E is the encryption algorithm and D is the decryption algorithm. Given input 1^k, the algorithm G outputs a pair of

bit strings $(e, d) = G(1^k)$, where e is the encryption key and d is the corresponding decryption key. Here k is the security parameter. We often write $G = (G_1, G_2)$ and $e = G_1(1^k)$, $d = G_2(1^k)$.

Given a pair of encryption/decryption keys (e, d), we can encrypt a message $\alpha \in \{0,1\}^k$ using $E(e, \alpha)$ and decrypt a ciphertext β using $D(d, \beta)$. We may write $E(e, \alpha)$ as $E_e(\alpha)$ and $D(d, \beta)$ as $D_d(\beta)$. When no confusion will be caused, we often omit the encryption and decryption keys and write them as $E(\alpha)$ and $D(\beta)$, respectively. The encryption algorithm E and decryption algorithm D satisfy

$$Pr[D(d, E(e, \alpha))] = \alpha] = 1,$$

where the probability is taken over the internal coin tosses of the algorithms E and D.

In the public-key encryption scheme, a party, say P, uses the algorithm G to generate a pair of keys (e, d). He/she keeps the decryption key d private and publicizes the encryption key e. Any other party can send party P private messages by encrypting them using the public key e. Only party P can decrypt these messages using his/her private key d, but nobody else can do that.

The particular cryptosystem we use in the design and implementation of secure protocols is the Paillier cryptosystem. It was invented by Pascal Paillier in 1999. The Paillier cryptosystem generates the pair of encryption/decryption keys as follows. Given the security parameter 1^k, the key generator G chooses randomly two large prime numbers p and q such that $\gcd(pq, (p-1)(q-1)) = 1$. Let the modulus $n = pq$ and $\lambda = \text{lcm}(p-1, q-1)$. The security parameter k is the bit length of the modulus n. The algorithm G then selects a random integer g from $\in Z^*_{n^2}$ and computes $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$, where the function L is defined as $L(x) = \lfloor (x-1)/n \rfloor$. The public (encryption) key is (n, g) and the corresponding private (decryption) key is (λ, μ). Note that the modulus n is contained in the public key. If p and q are of equal length, a simpler variant of the above key generation procedure is to set $g = n + 1$, $\lambda = (p-1)(q-1)$ and $\mu = \lambda^{-1} \bmod n$.

The Paillier cryptosystem is a probabilistic encryption scheme. To encrypt a message $m \in Z_n$, we select a random integer $r \in Z^*_n$ and compute the ciphertext as $c = E(m, r) = g^m r^n \bmod n^2$. We often omit the random number r and write $E(m, r)$ as $E(m)$. Given a ciphertext $c \in Z^*_{n^2}$, we can decrypt it using the formula $m = D(c) = L(c^\lambda \bmod n^2)\mu \bmod n$.

Proposition 2.1. *The Paillier cryptosystem is a* **homomorphic cryptosystem**.

That is, given two messages $m_1, m_2 \in Z_n$, we have

$$D(E(m_1) \cdot E(m_2) \bmod n^2) = (m_1 + m_2) \bmod n \tag{2.4}$$

$$D(E(m_1)^{m_2} \bmod n^2) = (m_1 m_2) \bmod n. \tag{2.5}$$

For convenience, we often write equations (2.4) and 2.5 as

$$E(m_1 + m_2) = E(m_1)E(m_2) \tag{2.6}$$

$$E(m_1 m_2) = E(m_1)^{m_2}. \tag{2.7}$$

The homomorphic property allows us to perform operations on the encrypted messages without decrypting them. This is an important property we use to design secure protocols, as I will show in next section.

The security of the Paillier cryptosystem is based on the decisional composite residuosity assumption (DCRA), which states that given a composite n and an integer z, it is computationally intractable to decide whether z is an n-residual modulo n^2 or not, i.e., whether there exists y such that $z = y^n \bmod n^2$.

Proposition 2.2. *If the decisional composite residuosity assumption holds, the Paillier cryptosystem is semantically secure. That is, for every family of polynomial-size circuits C_k and for every polynomial p, all sufficiently large k, any $x, y \in \{0,1\}^k$,*

$$|Pr[C_k(G_1(1^k), E_{G_1(1^k)}(x)) = 1] - Pr[C_k(G_1(1^k), E_{G_1(1^k)}(y)) = 1]| < \frac{1}{p(k)}. \tag{2.8}$$

Equation 2.8 states that the cryptosystem is secure against one message attack. It is known that it implies that the cryptosystem is secure against multiple messages attack (Goldreich, 2004). That is, for every family of polynomial-size circuits C_k, for every polynomial p, all sufficiently large k, any two sequences of messages $x = (x_1, \ldots, x_{t(k)})$ and $y = (y_1, \ldots, y_{t(k)})$ such that $x_i, y_i \in \{0,1\}^k$ and $t(k)$ is a polynomial of k, it holds that

$$|Pr[C_k(G_1(1^k), E_{G_1(1^k)}(x)) = 1] - Pr[C_k(G_1(1^k), E_{G_1(1^k)}(y)) = 1]| < \frac{1}{p(k)}, \tag{2.9}$$

where

$$E_{G_1(1^k)}(x) = (E_{G_1(1^k)}(x_1), \ldots, E_{G_1(1^k)}(x_{t(k)}))$$

$$E_{G_1(1^k)}(y) = (E_{G_1(1^k)}(y_1), \ldots, E_{G_1(1^k)}(y_{t(k)})).$$

Note that equation 2.9 states exactly that $(G_1(1^k), E_{G_1(1^k)}(x))$ and $(G_1(1^k), E_{G_1(1^k)}(y))$ are computationally indistinguishable.

2.3.2 Additive Secret-Sharing Scheme

One important technique for the design and implementation of two-party secure computation and privacy preserving data mining protocols is based on the **additive secret-sharing scheme** using homomorphic encryption (Bansal et al., 2011; Hall et al. 2011). To be concrete, I use the Paillier cryptosystem as the homomorphic encryption scheme. Suppose that there are two parties, Alice and Bob. Alice has the decryption (private) key d and both Alice and Bob know the encryption (public) key e. The plaintext domain is $Z_n = \{0, 1, \ldots, n-1\}$ and the corresponding ciphertext domain is $Z_{n^2} = \{0, 1, \ldots, n^2 - 1\}$. Note that in the Paillier system $e = (n, g)$ as defined in last section.

In the additive secret-sharing scheme based on homomorphic encryption, we try to maintain secret numbers between Alice and Bob in such a way that neither person knows the number but they still perform basic operations on these numbers. For each number x in the plaintext domain, Alice holds a number $x_A \in Z_n$ and Bob holds a number x_B such that $x = (x_A + x_B) \bmod n$. We call x_A Alice's share and x_B Bob's share. Since Alice only knows her share and Bob only knows his share, neither of them knows the number x. In this way we can hide x as a **secret** between Alice and Bob. As a convention, I use the subscript or superscript A to denote the shares Alice holds and the subscript or superscript B to denote the shares Bob holds. When no confusion will be caused, we omit the modulus in the expression and write $x = x_A + x_B$.

I now discuss how to perform secure addition and secure multiplication in the additive secret-sharing scheme. Given two secrets $x = (x_A + x_B) \bmod n$ and $y = (y_A + y_B) \bmod n$, it is straightforward to compute their sum. Alice adds her shares $z_A = (x_A + y_A) \bmod n$, Bob adds his shares $z_B = (x_B + y_B) \bmod n$, then z_A, z_B are shares of the secret $z = x + y$. Note that no encryption/decryption operation is needed and no communication is invoked in this procedure at all.

It is more involved to privately multiple two secrets. Lindell and Pinkas (2000) mentioned that private polynomial evaluation can be performed based on homomorphic encryption. Yang et al. (2006) and Goethals et al. (2005) presented secure scalar product based on homomorphic encryption, respectively. Secure multiplication

is a special case of secure scalar product. I present secure multiplication below.

Protocol 2.1 Secure multiplication

Input: two secrets x and y are split between Alice and Bob, $x = (x_A + x_B) \bmod n$, $y = (y_A + y_B) \bmod n$.

Output: Alice and Bob obtain their respective shares of $z = xy$.

1: Alice encrypts x_A and y_A, $m_1 = E(x_A, s_1)$, $m_2 = E(y_A, s_2)$, where s_1 and s_2 are uniformly random numbers in Z_n^*, and sends them to Bob.

2: Bob computes $p_1 = m_1^{y_B} \bmod n^2$ and $p_2 = m_2^{x_B} \bmod n^2$.
Bob encrypts a uniformly random number $r \in Z_n$, $p_3 = E(r, s_3)$, where s_3 is uniformly random in Z_n^*.
Bob computes $m_3 = (p_1 p_2 p_3) \bmod n^2$ and sends it back to Alice.
Bob sets $z_B = (x_B y_B - r) \bmod n$.

3: Alice decrypts $q = D(m_3)$.
Alice sets $z_A = (q + x_A y_A) \bmod n$.

Note that in Protocol 2.1,

$$\begin{aligned}
m_3 &= p_1 p_2 p_3 \pmod{n^2} \\
&= m_1^{y_B} m_2^{x_B} E(r, s_3) \pmod{n^2} \\
&= E(x_A, s_1)^{y_B} E(y_A, s_2)^{x_B} E(r, s_3) \pmod{n^2} \\
&= (g^{x_A} s_1^n)^{y_B} (g^{y_A} s_2^n)^{x_B} g^r s_3^n \pmod{n^2} \\
&= g^{x_A y_B + y_A x_B + r} (s_1^{y_B} s_2^{x_B} s_3)^n \pmod{n^2} \\
&= E(x_A y_B + y_A x_B + r, s_1^{y_B} s_2^{x_B} s_3) \\
q &= D(m_3) \\
&= x_A y_B + y_A x_B + r \pmod{n} \\
z_A &= q + x_A y_A \pmod{n} \\
&= x_A y_B + x_B y_A + x_A y_A + r \pmod{n} \\
z_B &= x_A y_B - r \pmod{n} \\
z_A + z_B &= xy \pmod{n}.
\end{aligned}$$

So z_A and z_B are shares of $z = xy$. We use Protocol 2.1 as a basic building block to implement other secure protocols. At the very beginning of those secure protocols, Alice generates a pair of keys $(e, d) = G(1^k)$ and sends the public key e to Bob. Then Alice and Bob use the key pair (e, d) to encrypt and decrypt messages There are two important facts about the secure multiplication which are used to prove the security of protocols.

(1) Alice receives only one message $m_3 = E(x_A y_B + y_A x_B + r, s_1^{y_B} s_2^{x_B} s_3)$ from Bob, which is an encryption of a random number. If we choose any random number r' in Z_n and another random number s' in Z_n^*, then $E(r', s')$ is identically distributed with m_3 no matter what x_A, x_B, y_A, y_B are and no matter what s_1 and s_2 Alice has chosen to encrypt messages.

(2) Bob receives two messages $m_1 = E(x_A)$ and $m_2 = E(y_A)$ from Alice. If we encrypt two messages $m_1' = E(0)$ and $m_2' = E(0)$, then according to Proposition 2.2, (e, m_1, m_2) and (e, m_1', m_2') are computationally indistinguishable ($e = G_1(1^k)$ is treated as a random variable).

Now we know how to perform secure addition and secure multiplication in the additive secret-sharing scheme based on homomorphic encryption. If any function can be computed using addition and multiplication, then we are able to implement a secure protocol for the function using secure addition and secure multiplication. Inspired by this idea, I develop a number of new secure protocols for basic operations and use these protocols to design privacy preserving distributed data mining protocols. Because secure addition is trivial in the additive secret-sharing scheme, I often say that we implement a secure protocol using secure multiplication, although secure addition is also needed.

Chapter 3

An Efficient Secure Comparison Protocol

3.1 Introduction

Number comparison is a common operation in the implementation of data mining algorithms. For example, the k-means clustering algorithm assigns a point to the closest cluster, and the k-nearest neighbor method assigns a point to the class most common in its k nearest neighbors, both of which need to compare distances. In particular, many data mining and machine learning algorithms, such as support vector machine and neural network, are implemented with iterative procedures, in which the stopping criterion is usually to compare some quantity with a threshold.

Secure comparison is a fundamental problem in the area of secure computation and privacy preserving data mining. Design of efficient secure comparison protocol is of practical importance in the implementation of privacy preserving data mining protocols. In this chapter, I propose an efficient secure comparison protocol based on homomorphic encryption (Lin and Jaromczyk, 2012). This protocol requires $2L + O(1)$ secure multiplications when the comparands belong to a known interval $[0, 2^L)$. Previous protocols require at least $12L+O(1)$ secure multiplications. To demonstrate the efficiency of the new secure comparison protocol, I implement a privacy preserving two-party k-means clustering protocol (Bunn et al., 2007). Experimental results show that the new secure comparison protocol can improve the performance of the privacy preserving k-means protocol substantially.

This chapter is organized as follows. In section 3.2, I derive a basic relationship between a secret number and its shares and hence propose an efficient secure comparison protocol. I discuss previous work on secure comparison in section 3.3 and

then present the privacy preserving k-means clustering algorithm in section 3.4. Experimental results on secure comparison and privacy preserving k-means clustering protocols are presented in section 3.5.

3.2 Secure Comparison

In this section, I first derive a basic relationship between a secret number and its shares. With this relationship I propose an efficient comparison protocol in section 3.2.2.

3.2.1 A Basic Relationship Between a Secret and Its Shares

Suppose that we are using the additive secret-sharing scheme based on homomorphic encryption to design secure protocols (section 2.3.2). We denote the plaintext domain of the underlying cryptosystem by $Z_N = \{0, 1, \ldots, N-1\}$[1]. To represent both positive and negative integers in the domain Z_N, a possible way is to make the assumption that all the considered integers x have absolute values $|x| < N/2$. If $x \geq 0$, it is represented as x in Z_N; if $x < 0$, then it is represented as $x + N$ in Z_N. For example, if $x = -1$, it is represented as $N - 1$. Now we make a little stronger assumption that all the considered integers x have absolute values $|x| < N/3$. This assumption is simple but very useful. As we shall see shortly, it enables us to derive a relationship between a secret number and its shares, with which we are able to design an efficient secure comparison protocol.

Suppose that a secret x is split between Alice and Bob, $x = x_A + x_B \pmod N$. Note that $x_A \in [0, N)$, $x_B \in [0, N)$, and, by assumption, $x \in (-N/3, N/3)$. We consider the relationship between x and its two shares x_A and x_B without referring to modular operations. There are three possibilities.

$$x = x_A + x_B;$$
$$x = x_A + x_B - 2N;$$
$$x = x_A + x_B - N.$$

We divide the interval $[0, N)$ into three sub-intervals $[0, N/3), [N/3, 2N/3)$ and $[2N/3, N)$. According to the ranges of x_A and x_B, we are able to determine which

[1]Hereafter, I use N to denote the modulus of the cryptosystem and K to denote the security parameter. I reserve n for the number of observations in datasets and k for the number of clusters in clustering tasks.

one of the above equations holds.

Theorem 3.1. *Suppose that* $x \in (-N/3, N/3)$, $x_A, x_B \in [0, N)$, $x = x_A + x_B \pmod N$. *The following holds:*

(i) If $x_A, x_B \in [0, N/3)$, *then* $x = x_A + x_B$;

(ii) If $x_A, x_B \in [2N/3, N)$, *then* $x = x_A + x_B - 2N$;

(iii) In all other cases, $x = x_A + x_B - N$.

Proof. (i) If $x_A, x_B \in [0, N/3)$, then $x_A + x_B - N \in [-N, -N/3)$, $x_A + x_B - 2N \in [-2N, -4N/3)$. Neither $[-N, -N/3)$ nor $[-2N, -4N/3)$ intersects with $(-3/N, N/3)$. So we must have $x = x_A + x_B$.

(ii) If $x_A, x_B \in [2N/3, N)$, then $x_A + x_B \in [4N/3, 2N)$, $x_A + x_B - N \in [N/3, N)$. Neither $[4N/3, 2N)$ nor $[N/3, N)$ intersects with $(-N/3, N/3)$. So we have $x = x_A + x_B - 2N$.

(iii) As an example, we consider the case when $x_A, x_B \in [N/3, 2N/3)$. In this case, $x_A + x_B \in [2N/3, 4N/3)$, $x_A + x_B - 2N \in [-4N/3, -2N/3)$. Note that $x \in (-N/3, N/3)$, so we must have $x = x_A + x_B - N$. Similar arguments hold for other cases. □

Corollary 3.2. *Define* $\alpha = 1$ *if* $x_A, x_B \in [0, N/3)$ *and* 0 *otherwise,* $\beta = 1$ *if* $x_A, x_B \in [2N/3, N)$ *and* 0 *otherwise. Let* $\gamma = 1 - \alpha + \beta$, *then*

$$x = x_A + x_B - \gamma N. \tag{3.1}$$

Proof. Note that in the first case in Theorem 3.1, $\alpha = 1$, $\beta = 0$ and $\gamma = 0$. In the second case, $\alpha = 0$, $\beta = 1$ and $\gamma = 2$. In all other cases, $\alpha = 0$, $\beta = 0$ and $\gamma = 1$. □

Equation (3.1) is a basic relationship between a secret and its two shares. It is used to develop an efficient secure comparison protocol in next subsection. Later in section 4.5, I use this relationship to design an efficient secure division with public divisor.

3.2.2 Secure Comparison

Suppose that we use the additive secret-sharing scheme based on homomorphic encryption to design secure two-party protocols. We assume that Alice has the private key and both Alice and Bob know the public key. Now suppose that two secrets $0 \le x, y < 2^L$ are split between Alice and Bob, $x = x_A + x_B \pmod N$, $y = y_A + y_B$

(mod N). Alice and Bob wish to privately compare these two secrets. We assume that the comparing result is that $r = 1$ if $x \leq y$ and $r = 0$ otherwise. Alice obtains a share r_A and Bob obtains a share r_B such that $r = r_A + r_B \pmod{N}$. Here we assume that both x and y belong to a known interval $[0, 2^L)$ and have a maximum bit length of L. The specification of this interval and the maximum bit length L depends on the application and is agreed on by both parties. The bit length L is typically much smaller than the security parameter K. We also assume that $x, y < N/3$, as I discuss in last subsection. Because N is typically a large number, this assumption is easily satisfied in real applications.

Now let $z = y - x$, $z_A = (y_A - x_A) \mod N$ and $z_B = (y_B - x_B) \mod N$. We have $z = z_A + z_B \pmod{N}$. Note that $z \in (-N/3, N/3)$. By Theorem 3.1, if we consider the ranges of z_A, z_B, we have:

(i) If $z_A, z_B \in [0, N/3)$, then $z = z_A + z_B$, so $z \geq 0$ and $x \leq y$.

(ii) If $z_A, z_B \in [2N/3, N)$, then $z = z_A + z_B - 2N$, so $-2N/3 \leq z < 0$ and $x > y$.

We can determine the order of x and y directly in these two cases. In all other cases, we know that $z = z_A + z_B - N$. Note that $-2^L < z < 2^L$ as we assume that $0 \leq x, y < 2^L$. If $z \geq 0$, then the $(L+1)$-th bit of $2^K + z$ is 0; and it is 1 if $z < 0$. This is true as long as $L < K$ holds. For example, let $K = 10, L = 4$, then $2^{10} + 3 = 10000000011$ and $2^{10} - 3 = 0111111111101$. So to determine whether $0 \leq z$ or not, we only need to compute the $(L+1)$-th bit of $2^K + z$. Note that $2^K + z = z_A + z_B + 2^K - N$.

Putting all things together, I present an efficient secure comparison protocol in Protocol 3.1. In the protocol, the comparison result r is a secret split between Alice and Bob. Alice and Bob obtain their respective shares of r but neither of them knows the secret r. This protocol will be used as a building block to design other secure protocols.

In Protocol 3.1, $\alpha = 1$ if and only if $z_A, z_B \in [0, N/3)$. In this case, we have $x \leq y$. $\beta = 1$ if and only if $z_A, z_B \in [2N/3, N)$. In this case, we have $x > y$. Lines 9-16 consider the case when $z = z_A + z_B - N$ and compute the $(L+1)$-th bit of $(2^K + z)$.

Lines 11-16 perform the usual binary addition of p and q:

$$\begin{array}{r} p_{L+1}p_L \ldots p_1 \\ + \; q_{L+1}q_L \ldots q_1 \\ \hline d = d_{L+1}d_L \ldots d_1 \end{array}$$

c_i is the i-th carry-over bit and d_i is the i-th bit of $d = p + q$. Line 14 says that if

Protocol 3.1 Secure comparison

Input: two secret integers x and y such that $0 \leq x, y < 2^L$ are split between Alice and Bob, $x = x_A + x_B \pmod N$, $y = y_A + y_B \pmod N$.

Output: Alice and Bob obtain their respective shares of the comparison result r such that $r = 1$ if $x \leq y$ and 0 otherwise.

1: Alice: $z_A = (y_A - x_A) \bmod N$.
2: Bob: $z_B = (y_B - x_B) \bmod N$.

3: Alice: $\alpha_1 = 1$ if $z_A < N/3$ and $\alpha_1 = 0$ otherwise.
4: Bob: $\alpha_2 = 1$ if $z_B < N/3$ and $\alpha_2 = 0$ otherwise.
5: Alice and Bob use secure multiplication (Protocol 2.1) to privately compute $\alpha = \alpha_1\alpha_2$.

6: Alice: $\beta_1 = 1$ if $2N/3 \leq z_A$ and $\beta_1 = 0$ otherwise.
7: Bob: $\beta_2 = 1$ if $2N/3 \leq z_B$ and $\beta_2 = 0$ otherwise.
8: Alice and Bob use secure multiplication to privately compute $\beta = \beta_1\beta_2$.

9: Alice: let $p = p_{L+1}p_L \ldots p_1$ be the lowest $L+1$ bits of z_A.
10: Bob: let $q = q_{L+1}q_L \ldots q_1$ be the lowest $L+1$ bits of $z_B + 2^K - N$.
11: $c_0 = 0$
12: Alice and Bob use secure multiplication to privately compute the following loop
13: **for** $i = 1$ to $L+1$ **do**
14: $\quad c_i = p_i c_{i-1} + q_i(p_i + c_{i-1} - p_i c_{i-1} - p_i c_{i-1})$
15: $\quad d_i = p_i + q_i + c_{i-1} - c_i - c_i$
16: **end for**

17: Alice and Bob use secure multiplication to privately compute $r = 1 - \beta - (1-\alpha)(1-\beta)d_{L+1}$.

at least two of p_i, q_i and c_{i-1} are 1, c_i is 1; otherwise, c_i is 0. Note that we need 2 multiplications to compute each c_i. d_{L+1} is the $(L+1)$-th bit of $(2^K + z)$. If $d_{L+1} = 0$, then $x \leq y$; otherwise $x > y$.

Line 17 combines all three cases and it can be interpreted as follows. If $\beta = 1$, which means $x > y$, so $r = 0$. Otherwise, $\beta = 0$ and $r = 1 - (1 - \alpha)d_{L+1}$. If $\alpha = 1$, we know that $x \leq y$, so $r = 1$; otherwise $\alpha = 0$, we have $r = 1 - d_{L+1}$.

Consider the following trivial example. Let $N = 143$, $K = 8$, $L = 4$, $x = 5, x_A = 70, x_B = 78$, $y = 2, y_A = 33, y_B = 112$. Then $z_A = (33 - 70) \bmod 143 = 106$, $z_B = (112 - 78) \bmod 143 = 34$. In this case, $\alpha = 0$ and $\beta = 0$. Now $p = 01010$, $q = 10011$ because $106 = (1101010)_2$, $2^K - N + z_B = 147 = (10010011)_2$. So $d = p + q = 11101$, and we have $r = 1 - d_5 = 0$.

Th implementation of Protocol 3.1 is based on the additive secret-sharing scheme. In the implementation, all the variables α, β, c_i, d_i and r are secrets split between Alice and Bob. Note that Protocol 3.1 involves only addition/subtraction and multiplication. Secure addition in the additive secret-sharing scheme is trivial. In the protocol of secure multiplication, both the input and the output are secrets split between Alice and Bob. We can use secure multiplication to implement Protocol 3.1. Protocol 3.1 needs $2L + O(1)$ invocations of secure multiplication.

As an example, I show how to implement line 17. We have inputs $\alpha = \alpha_A + \alpha_B \pmod N$, $\beta = \beta_A + \beta_B \pmod N$ and $d_{L+1} = d^A_{L+1} + d^B_{L+1} \pmod N$ before the execution of line 16. Alice first computes $u_A = (1 - \beta_A) \bmod N$ and $v_A = (1 - \alpha_A) \bmod N$; and Bob computes $u_B = (-\beta_B) \bmod N$ and $v_B = (-\alpha_B) \bmod N$. Then u_A and u_B, v_A and v_B, w_A are shares of $u = 1 - \beta$ and $v = 1 - \alpha$, respectively. Alice and Bob invoke secure multiplication on (u_A, v_A) and (u_B, v_B) and obtain their respective shares of $s = uv$. That is, Alice obtains a share s_A and Bob obtains a share s_B such that $s = s_A + s_B \pmod N$. Now Alice and Bob invoke secure multiplication on $(s_A, d^A_{L+1}$ and (s_B, d^B_{L+1}) and obtain their respective shares of $t = sd_{L+1}$. Alice computes $r_A = (u_A - t_A) \bmod N$ and Bob computes $r_B = (u_B - t_B) \bmod N$. Then r_A and r_B ares shares of the desired comparison result r.

In the implementation of the secure comparison protocol, we can use the following three precomputation techniques. They can reduce the running time drastically.

(1) In the secure multiplications of $\alpha = \alpha_1\alpha_2$, $\beta = \beta_1\beta_2$ and p_ic_{i-1}, Alice needs to encrypt α_1, β_1 and p_i and sends them to Bob. Note that α_1, β_1 and p_i are either 0 or 1 and they are held by Alice. Alice can compute a series of encryptions of 0 and 1 beforehand. During the execution of the protocol, Alice picks up the encrypted numbers accordingly (Yang et al., 2006).

(2) In secure multiplication, Bob needs to select and encrypt a random number. These random numbers and their encryptions are independent of the execution of secure protocols. So Bob can do these beforehand.

(3) In the Paillier cryptosystem, we encrypt a plaintext x as $g^x r^N \bmod N$, where (N, g) is the encryption key and r is a random number in Z_N^*. Note that r^N is independent of the execution of secure protocols. We can compute a large number of numbers r^N beforehand and use them when needed in the execution of secure protocols (Paillier, 1999).

3.3 Previous Work

Secure comparison is a fundamental problem in secure computation and privacy preserving data mining. In his seminal paper on secure computation (Yao, 1982), Yao proposed the Millionaires' problem, in which two millionaires wish to know who is richer but without revealing their asset values. The original solution is exponential in time and space. Several efficient protocols have been proposed and they focus on the case when each party knows one number (Lin et al., 2005). The new secure comparison protocol presented in the previous section assumes that we use the additive secret-sharing scheme based on homomorphic encryption and that the numbers are secrets split between two parties. The adoption of homomorphic encryption enables efficient implementation of secure multiplication, which is a basic and almost indispensable operation in the design of privacy preserving data mining protocols. The new protocol assumes that both the input numbers and the output result are secrets split between two parties, so that it can be used as a subprotocol in other privacy preserving data mining protocols.

Bunn et al. (2007) proposed a secure comparison protocol based on homomorphic encryption in the development of their secure k-means protocol. Their comparison protocol assumes that two secrets $x, y \in [0, 2^L)$ are split between two parties and the comparison result is that $r = 0$ if $x < y$, $r = 1$ if $x > y$, and r takes 0 or 1 randomly when $x = y$. The result r is also a secret split between Alice and Bob.

Bunn et al. first proposed a secure protocol to privately transform a secret x into its binary representation. It is assumed that the secret $x \in [0, N/2)$ and $x \in [0, 2^L)$. If we let $\alpha = 0$ if $x_A, x_B \in [0, N/2)$ and $\alpha = 1$ otherwise, then $x = x_A + x_B - \alpha N$. Let $x_L^A x_{L-1}^A \ldots x_1^A$ and $x_L^B x_{L-1}^B \ldots x_1^B$ be the lowest L bits of x_A and x_B, respectively, and $p = p_L p_{L-1} \ldots p_1$ be the lowest L bits of $2^K - N$. Then the binary representation of x is computed as the following:

$$\begin{array}{r} x_L^A x_{L-1}^A \dots x_1^A \\ x_L^B x_{L-1}^B \dots x_1^B \\ +\alpha * (p_L p_{L-1} \dots p_1) \\ \hline x = x_L x_{L-1} \dots x_1 \end{array}$$

The authors didn't give the exact details of how to compute the above formula. One possible way is to lines 11-16 in Protocol 3.1 to perform binary addition twice and it takes $4L + O(1)$ secure multiplications. I present the secure transformation protocol in Protocol 3.2. In the protocol, $\beta = 1 - \alpha$.

Protocol 3.2 Secure transformation protocol

Input: a secret integer x such that $0 \leq x < 2^L$ and $0 \leq x < N/2$ is split between Alice and Bob, $x = x_A + x_B \pmod N$.

Output: the binary representation of x, $x = x_L \dots x_1$, whose bits x_i are secrets split between Alice and Bob.

1: Alice: $\beta_1 = 1$ if $z_A < N/2$ and $\beta_1 = 0$ otherwise.
2: Bob: $\beta_2 = 1$ if $z_B < N/2$ and $\beta_2 = 0$ otherwise.
3: Alice and Bob use secure multiplication (Protocol 2.1) to securely compute $\beta = \beta_1\beta_2$. Alice obtains α_A and Bob obtains β_B such that $\beta = \beta_A + \beta_B \pmod N$.
4: Alice: let $p = p_L \dots p_1$ be the lowest L bits of x_A.
5: Bob: let $q = q_L \dots q_1$ be the lowest L bits of x_B.
6: $c_0 = 0$
7: **for** $i = 1$ to L **do**
8: $\quad c_i = p_i c_{i-1} + q_i(p_i + c_{i-1} - p_i c_{i-1} - p_i c_{i-1})$
9: $\quad d_i = p_i + q_i + c_{i-1} - c_i - c_i$
10: **end for**
11: **for** $i = 1$ to L **do**
12: $\quad$ **if** the i-th bit of $(2^K - N)$ is 1 **then**
13: $\quad\quad s_i = 1 - \beta$
14: $\quad$ **else**
15: $\quad\quad s_i = 0$
16: $\quad$ **end if**
17: **end for**
18: $c_0 = 0$
19: **for** $i = 1$ to L **do**
20: $\quad c_i = d_i c_{i-1} + s_i(d_i + c_{i-1} - d_i c_{i-1} - d_i c_{i-1})$
21: $\quad x_i = d_i + s_i + c_{i-1} - c_i - c_i$
22: **end for**

Lines 4-10 computes $d_L \dots d_1$, the binary representation of $d = x_A + x_B$. Lines 11-17 computes the binary representation of $\alpha(2^K - N)$. Note that both parties know

$2^K - N$. In line 12, if the i-th bit of $(2^K - N)$ is 1, Alice sets $s_{i,A} = (1 - \beta_A) \bmod N$ and Bob sets $s_{i,B} = (-\beta_B) \bmod N$; Otherwise, Alice and Bob set their shares of s_i as 0. $s = s_L \dots r_1$ is the binary representation of $\alpha(2^K - N)$. Lines 18-22 compute the binary representation of $x = d + s$.

Alternatively, we can compute the binary representation of $x_A + x_B$, denoted by $u_L u_{L-1} \dots u_1$, using lines 4-10; we then compute the binary representation of $x_A + x_B + 2^K - N$, denoted by $v_L v_{L-1} \dots v_1$, using lines 4-10 with x_B replaced by $x_B + 2^K - N$. Then the i-th bit of x is then $(1-\alpha)u_i + \alpha v_i$. When we use precomputation techniques as we discuss in the previous subsection, this implementation may be more efficient because in line 8, p_i and q_i are numbers known by one party and the secure multiplication can be simplified, while in line 20, both d_i and c_i are secrets.

After transforming x and y into their binary representations, $x = x_L \dots x_1$ and $y = y_L \dots y_1$, Bunn et.al used the following formula to compare x and y:

$$\begin{aligned} r =& (x_L \oplus y_L)x_L + (x_L \oplus y_L \oplus 1)(x_{L-1} \oplus y_{L-1})x_{L-1} \\ & + (x_L \oplus y_L \oplus 1)(x_{L-1} \oplus y_{L-1} \oplus 1)(x_{L-2} \oplus y_{L-2}) \\ & * x_{L-2} + \dots + (x_L \oplus y_L)...(x_2 \oplus y_2 \oplus 1)(x_1 \oplus y_1) \\ & + (x_L \oplus y_L \oplus 1) \dots (x_1 \oplus y_1 \oplus 1)t, \end{aligned}$$

where t takes 0 or 1 randomly and $\oplus$ is the XOR operation. Note that $a \oplus b = a + b - 2ab$. This formula needs $4L + O(1)$ invocations of secure multiplication. So the secure comparison protocol takes $12L + O(1)$ secure multiplications totally.

Qi et al. (2008) proposed another secure protocol for comparison when Alice holds x and Bob holds y. Let c_j (d_j) be 1 if the integer with binary representation $x_j..x_1$ is greater (smaller) than the integer with binary representation $y_j...y_1$. They used the following formula to compute c_L and d_L, If $j = 1$ then

$$c_j = x_j(1 - y_j), d_j = y_j(1 - x_j),$$

If $j > 1$ then

$$\begin{aligned} c_j &= (1 - d_{j-1})(c_{j-1} + (1 - c_{j-1}x_j(1 - y_j)))d_j \\ &= (1 - c_{j-1})(d_{j-1} + (1 - d_{j-1}y_j(1 - x_j))). \end{aligned}$$

The above formula needs $6L + O(1)$ multiplications. When the secrets x and y are split between Alice and Bob, we need to first transform them into their binary

representations. So the secure comparison protocol takes $14L + O(1)$ secure multiplications.

The three secure comparison protocols discussed so far are all based on homomorphic encryption. S. From (2006) proposed a secure multi-party comparison protocol based on the Shamir's polynomial secret-sharing scheme. Suppose that two integers $x, y \in [0, 2^L)$ have binary representations $x = x_L x_{L-1} \dots x_1$ and $y = y_L y_{L_1} \dots x_1$, respectively. To compare x and y, we only need to compute $(L+1)$-th bit of $2^L + x - y$. It is 1 if $y \leq x$ and 0 otherwise. This can be computed by usual binary additions. When the polynomials are defined on the Galois field $GF(2^8)$, which has character 2, S. From (2006) used the following formula to compute the i-th carry-over bit

$$c_i = x_i y_i + c_{i-1}(x_i + y_i),$$

which requires $2L$ multiplications. When the polynomials are defined over any prime field Z_p, S. From used the following formula:

$$c_i = x_i y_i + x_i c_{i-1} + y_i c_{i-1} - 2x_i y_i c_{i-1},$$

which requires $4L$ multiplications. As the formula in line 14 in Protocol 3.1 shows, this can be achieved with only $2L$ multiplications.

Nishide et al. (2007) proposed secure comparison protocols based on the Shamir's secret-sharing scheme. Their method is derived from interval testing. In their paper the polynomials are defined over some large field Z_p (p is a prime number) so that all the considered integers are in Z_p. In contrast, S. From used polynomials over the small field $GF(2^8)$ to represent secrets of single bits.

Note that both in the additive secret-sharing scheme based on homomorphic encryption and in the Shamir's polynomial secret-sharing scheme, the secure comparison protocols presented here are all implemented using secure multiplication. However, the implementations of secure multiplication in these two schemes are different. In the additive secret-sharing scheme, the implementation of secure multiplication is based on the homomorphic property of cryptosystems; in the Shamir's secret-sharing scheme, secure multiplication is implemented via multiplication of polynomials. Note that in the Shamir's sharing scheme, secure multiplication and hence secure comparison don't work in two-party cases. In contrast, the additive secret-sharing scheme can be used in two-party cases.

3.4 Privacy Preserving K-means Clustering

Bunn et al. (2007) proposed a privacy-preserving two-party k-means clustering protocol based on homomorphic encryption, The k-means protocol uses secure comparison as a subprotocol. To demonstrate the efficiency of the new secure comparison protocol, I implement a privacy preserving k-means clustering protocol based on the work by Bunn et al. I present the protocol in this section.

3.4.1 K-means

K-means clustering is one of the most widely used clustering techniques. Suppose that we have a dataset $X = (x_1, \ldots, x_n)^{\mathrm{T}}$ of n observation with p attributes. We wish to cluster these observations into k groups, $C = \{C_1, \ldots, C_k\}$, such that the observations in the same group are similar to each other. Let $\mu_j \in R^p$ be the center (means) of the cluster C_j. The k-means method aims to find a clustering of the dataset X which minimizes the error function $E = \sum_{j=1}^{k} \sum_{x \in C_j} \|x - \mu_j\|_2^2$.

The k-means algorithm is an iterative method for data clustering. Initially, it selects k cluster centers in some manner. Then it alternates with two steps: assigns each observation to the cluster with the closest center and then computes the new cluster centers accordingly. I present the k-means clustering algorithm in Algorithm 3.3, in which μ_j denotes the current center of the j-th cluster and ν_j denotes the new center.

Algorithm 3.3 K-means clustering

Input: a dataset $X = (x_1, \ldots, x_n)$ of n observations with p attributes and the number of clusters k.

Output: a clustering of n observations into k groups C_j $(j = 1, ..., k)$.

1: initialize the k means $\nu_1, ..., \nu_k$.
2: **repeat**
3: $\mu_j = \nu_j$ for $j = 1, 2, ..., k$
4: **for** each observation x_i **do**
5: assign x_i to cluster j such that $\|x_i - \mu_j\|_2^2$ is the minimum over all $j = 1, \ldots, k$.
6: **end for**
7: compute the new cluster centers ν_j for $j = 1, 2, ..., k$.
8: **until** convergence

The initialization of the cluster centers is an important issue. It determines the final solution and affects the speed of convergence. We can select the initial centers randomly, or we can apply the k-means algorithm on a small sample of the dataset

and use the results as the initial centers to cluster the whole dataset (Bradley et al., 1996). The stopping criterion is that the difference between ν_j and μ_j is sufficiently small, that is, $\sum_{j=1}^{k} \|\nu_j - \mu_j\|_2^2 < \epsilon$ for some predefined threshold ϵ. Here we use the squared distance $\|\nu_j - \mu_j\|_2^2$ instead of the distance $\|\nu_j - \mu_j\|_2$ to avoid square root operation.

3.4.2 Privacy Preserving Two-Party K-means

Consider a dataset $X = (x_1, ..., x_n)^{\mathrm{T}}$ consisting of n observations with p attributes. Here we represent each observation as a column vector. We assume that all the attribute values are integers. For a dataset with real numbers, we represent each real number r with $\lfloor r2^P \rfloor$, where P is the number of bits to represent the fractional parts of real numbers.

Now suppose that the dataset X is vertically partitioned between Alice and Bob. Alice has the first p_1 attributes and Bob has the last p_2 attributes ($p_1 + p_2 = p$). We denote the set of integers by Z. For each observation x_i, $x_i^{\mathrm{T}} = (x_{i,1}^{\mathrm{T}}, x_{i,2}^{\mathrm{T}})$, where $x_{i,1} \in Z^{p_1}$ is the values of the first p_1 attributes and held by Alice, $x_{i,2} \in Z^{p_2}$ is the values of the last p_2 attributes and held by Bob. Alice and Bob wish to apply the k-means algorithm to cluster their joint dataset X into k groups but without disclosing their confidential data.

I now present a secure two-party k-means clustering protocol using the additive secret-sharing scheme based on homomorphic encryption. We assume that the plaintext domain of the underlying cryptosystem is $Z_N = \{0, \ldots, N-1\}$ and Alice has the private key. As I discuss in section 3.2, a nonnegative integer m is represented as m in Z_N and a negative integer m is represented as $N + m$ in Z_N.

I first present a secure protocol to compute the squared distance between two points. Suppose that two vectors $x = (x_1, \ldots, x_p)^{\mathrm{T}}$ and $y = (y_1, \ldots, y_p)^{\mathrm{T}}$ are split between Alice and Bob, $x_i = x_{i,A} + x_{i,B} \pmod N$ and $y_i = y_{i,A} + y_{i,B} \pmod N$ ($i = 1, \ldots, p$). Alice and Bob wish to privately compute $s = \|x - y\|_2^2$ and obtain their respective shares of s. Let $z = x - y$, then the two shares of z_i are $z_{i,A} =$

$(x_{i,A} - y_{i,A}) \bmod N$ and $z_{i,B} = (x_{i,B} - y_{i,B}) \bmod N$. Note that

$$\begin{aligned}
\|x - y\|_2^2 &= \|z\|_2^2 \\
&= \langle z, z \rangle \\
&= \langle z_A + z_B, z_A + z_B \rangle \\
&= \langle z_A, z_A \rangle + \langle z_B, z_B \rangle + 2\langle z_A, z_B \rangle \\
&= \prod_{i=1}^{p} z_{i,A}^2 + \prod_{i=1}^{p} z_{i,B}^2 + 2\prod_{i=1}^{p} z_{i,A} z_{i,B}.
\end{aligned}$$

According to this formula, we can use secure multiplication to implement a secure protocol to compute squared distances. I present this protocol in Protocol 3.4.

Protocol 3.4 Secure squared distance

Input: two vectors $x = (x_1, \ldots, x_p)^{\mathrm{T}}$ and $y = (y_1, \ldots, y_p)^{\mathrm{T}}$ are split between Alice and Bob, $x_i = x_{i,A} + x_{i,B} \pmod N$ and $y_i = y_{i,A} + y_{i,B} \pmod N$.

Output: Alice and Bob obtain their respective shares of $s = \|x - y\|_2^2$.

1: Alice: $z_{i,A} = (x_{i,A} - y_{i,A}) \bmod N$ $(i = 1, \ldots, p)$.
2: Bob: $z_{i,B} = (x_{i,B} - y_{i,B}) \bmod N$ $(i = 1, \ldots, p)$.
3: Alice: $s_A = (\prod_{i=1}^{p} z_{i,A}^2) \bmod N$.
4: Bob: $s_B = (\prod_{i=1}^{p} z_{i,B}^2) \bmod N$.
5: **for** $i = 1$ to p **do**
6: Alice and Bob use Protocol 2.1 to compute $r = z_{i,A} z_{i,B}$.
Alice obtain her share r_A and Bob obtain his share r_B.
7: Alice: $s_A = (s_A + r_A + r_A) \bmod N$.
8: Bob: $s_B = (s_B + r_B + r_B) \bmod N$.
9: **end for**

To implement the privacy preserving k-means protocol, we need a secure division protocol which privately computes the quotient when both the dividend and the divisor are secrets. Bunn et al. proposed a solution based on homomorphic encryption. Suppose that both the dividend b and the divisor d belong to a known interval $[0, 2^L)$. Their solution simulates the ordinary binary division to compute $\lfloor b/d \rfloor$. Let $b_0 = b$. We find the largest $a_1 \in [0, L)$ such that $2^{a_1} d \leq b$, which is represented as a characteristic vector $\delta_1 \in Z^L$ such that the a_1-th element in δ_1 is 1 and all other elements are 0. Let $b_1 = b_0 - 2^{a_1} d$. This procedure iterates for $i = 1, 2, \ldots, L$. The quotient has the binary representation $\delta = \sum_{i=1}^{L} \delta_i$. This protocol needs $O(L^2)$ secure multiplications.

I now present the privacy preserving two-party k-means clustering protocol in Protocol 3.5. In the protocol, $\mu_j, \nu_j \in Z^p$ denote the current and the new centers of cluster j, respectively, and s_j denotes the size of cluster j. The vector $\phi \in Z^k$

indicates the assignment of an observation to the closest center. If x_i is assigned to cluster t, then $\phi_t = 1$, and $\phi_j = 0$ for $j \neq t$. The vector $c \in Z^n$ represents the clustering of the n observations. If $c_i = t$, it means that we assign the observation x_i to cluster t. In the implementation of the protocol, all the variables μ, ν, s, ϕ, c are secrets split between Alice and Bob during each iteration. Only at the end of the protocol, Alice and Bob exchange their shares of c to get the final clustering results.

The initialization of the cluster centers is an important issue in k-means clustering. The original secure k-means algorithm by Bunn et al. (2007) privately selects the initial centers according to some probability distribution (Ostrovksy et al., 2006). In Protocol 3.5, for simplicity, Alice and bob simply initialize the cluster centers randomly (lines 1 and 2).

Protocol 3.5 invokes Protocol 3.6 in line 10. Protocol 3.6 privately assigns each point x_i to the closest cluster. In Protocol 3.6, the variable ϕ_j in line 4 is the comparison result of d_j and m. In line 5, the variable m is assigned the minimum of m and d_j. Lines 7-11 set $\phi_t = 1$ if t is the closest cluster and $\phi_j = 0$ for all $j \neq t$. For example, suppose that we have a vector $\phi = (0, 1, 1, 0, 1, 0)$ after the execution of line 6, then we scan this vector from right to left. We don't change the first 1 we encounter and set all the remaining entries to be 0. The vector ϕ becomes $(0, 1, 0, 0, 0, 0)$.

The new cluster centers are computed privately in lines 18-20, Protocol 3.5. Alice and Bob then use the secure squared distance protocol to compute privately $\sum_{j=1}^{k} \|\nu_j - \mu_j\|_2^2$ and invoke secure comparison to check privately whether $\sum_{j=1}^{k} \|\nu_j - \mu_j\|_2^2 < \epsilon$. The comparison result is disclosed to both parties so that they can decide whether to stop the loop or not. When they exist the loop, Alice and Bob exchange their shares of c and both parties obtain the clustering results.

We can implement Protocol 3.5 and 3.6 using secure multiplication, secure squared distances, secure comparison and secure division. It discloses only the number of iterations. If we fix the number of iterations, the privacy preserving k-means protocol is provably secure in the semi-honest model.

The privacy preserving two-party k-means presented here is based on the work by Bunn et al. (2007). To test the performance of the privacy preserving k-means protocol, I implement the protocol in C++ based on the Paillier system. My implementation is slightly different from the original protocol by Bunn et al. I summarize the differences below.

1. The secure protocol by Bunn et al. privately selects the initial cluster centers according to some probability distribution (Ostrovksy et al., 2006). In Protocol 3.5, for simplicity, we select k observations randomly as the initial centers. For large

Protocol 3.5 Privacy preserving two-party k-means clustering

Input: a dataset consisting of n observations with p attributes, $X = (x_1, \ldots, x_n)^{\mathrm{T}}$, is vertically partitioned between Alice and Bob. Alice has the first p_1 attributes and Bob has the last p_2 attributes. Both Alice and Bob know the number of clusters k.

Output: Alice and Bob obtain a clustering of these n observations into k groups C_j $(j = 1, ..., k)$.

1: Alice randomly selects k indices from $\{0, \ldots, n\}$, denoted by $l_1, \ldots, l_k$, and sends these indices to Bob.
2: Alice and Bob set $\nu_j = x_{l_j}$ $(j = 1, ..., k)$.
3: **repeat**
4: **for** $j = 1$ to k **do**
5: $\mu_j = \nu_j$
6: $\nu_j = 0$
7: $s_j = 0$
8: **end for**
9: **for** each observation x_i **do**
10: Alice and Bob invoke Protocol 3.6 (see below) to privately assign x_i to the closest cluster t. The result is a vector $\phi \in R^k$ such that $\phi_t = 1$ and $\phi_j = 0$ for $j \neq t$.
11: $c_i = 0$
12: **for** $j = 1$ to k **do**
13: $\nu_j = \nu_j + \phi_j x_i$
14: $s_j = s_j + \phi_j$
15: $c_i = c_i + j\phi_j$
16: **end for**
17: **end for**
18: **for** $j = 1$ to k **do**
19: Alice and Bob invoke secure division to compute $\nu_j = \nu_j / s_j$
20: **end for**
21: **until** $\sum_{j=1}^{k} \|\nu_j - \mu_j\|_2^2 < \epsilon$
22: $C_j = \{i | c_i = j\}$ $(j = 1, ..., k)$

Protocol 3.6 Secure protocol to privately assign a point to the closest cluster

Input: an observation $x \in Z^p$ that vertically partitioned between Alice and Bob; k clusters centers $\mu_j \in Z^p$ $(j = 1, \ldots, k)$ that are split between Alice and Bob.

Output: $\phi \in Z^k$ such that if x is closest to cluster t, then $\phi_t = 1$ and $\phi_j = 0$ for $j \neq t$. ϕ are secrets split between Alice and Bob.

1: $m = \infty$
2: **for** $j = 1$ to k **do**
3: Alice and Bob use Protocol 3.4 to securely compute $d_j = \|x - \mu_j\|_2^2$.
4: Alice and Bob use secure comparison (Protocol 3.1) to securely compare d_j with m. The result is that $\phi_j = 1$ if $d_j \leq min$ and 0 otherwise.
5: $m = (1 - \phi_j)m + \phi_j d_j$
6: **end for**
7: $\delta = 1$
8: **for** $j = k$ downto 1 **do**
9: $\phi_j = \delta\phi_j$
10: $\delta = \delta(1 - \phi_j)$
11: **end for**

datasets, we first run the privacy preserving k-means protocol on a sample of the dataset and use the results as the initial centers to cluster the whole dataset.

2. The original protocol by Bunn et al. uses a secure comparison protocol which requires $12L + O(1)$ secure multiplications. I use the new secure comparison protocol that needs $2L + O(1)$ secure multiplications. Secure comparison is a frequent and costly operation in the privacy preserving k-means algorithm and it becomes the bottleneck in the execution of the secure protocol. Experimental results presented in next section show that the new secure comparison protocol improves the performance of the privacy preserving k-means protocol substantially.

3. The k-means protocol by Bunn et al. uses a secure division protocol which requires $O(L^2)$ secure multiplications. I implement a secure division protocol based on homomorphic encryption using Newton's iterative method, which requires $O(L)$ secure multiplications. I present the secure division protocol in section 4.6 after I discuss how to perform secure operations of real numbers. I use this secure division protocol in the implementation of the privacy preserving k-means protocol.

3.5 Experimental Results

I have implemented secure comparison protocols and the privacy preserving two-party k-means clustering protocol in C++ based on the Paillier cryptosystem (Paillier,

1999). I used the GMP library (Torbjorn Granlund et al.) for big integers. I ran the protocols on two computers both with Intel Pentium 4 CPU (3.2GHz) and the Linux operating system. These computers are in a network connected by 100Mbps Ethernet with average message latency less than 1ms.

I first compared the performances of the new secure comparison protocol with those of Bunn et al. (2007) and Qi et al. (2008) without using precomputation techniques. Table 3.1 reports the execution time when we use security parameter $K = 512$. Table 3.2 reports the results with security parameter $K = 1024$. In the tables, each column corresponds to the maximum bit length of the comparands. Then I compared the performances of these secure comparison protocols using the precomputation techniques that I present in section 3.2. I report the results in Table 3.3 and 3.4. The results show that the secure comparison protocol is several times faster than the existing protocols.

Table 3.1: Execution time of secure comparison protocols ($K = 512$)

	Maximum bit length L				
Protocol	10	20	30	40	50
new	0.13s	0.34s	0.49s	0.63s	0.71s
Bunn et al.	3.63s	7.04s	10.45s	13.88s	20.88s
Qi et al.	4.62s	9.23s	13.86s	18.47s	23.26s

Table 3.2: Execution time of secure comparison protocols ($K = 1024$)

	Maximum bit length L				
Protocol	10	20	30	40	50
new	0.63s	1.17s	1.66s	2.20s	2.71s
Bunn et al.	5.17s	10.20s	15.20s	20.42s	25.47s
Qi et al.	6.17s	12.38s	18.64s	25.04s	31.27s

Table 3.3: Execution time of secure comparison protocols with precomputation ($K = 512$)

	Maximum bit length L				
Protocol	10	20	30	40	50
new	0.034s	0.054s	0.075s	0.098s	0.12s
Bunn et al.	0.44s	0.84s	1.26s	1.67s	2.07s
Qi et al.	0.55s	1.09s	1.65s	2.18s	2.73s

Table 3.4: Execution time of secure comparison protocols with precomputation ($K = 1024$)

	Maximum bit length L				
Protocol	10	20	30	40	50
new	0.22s	0.34s	0.47s	0.60s	0.74s
Bunn et al.	2.76s	5.32s	8.01s	10.57s	13.10s
Qi et al.	3.48s	6.92s	10.41s	13.88s	17.27s

I tested the privacy preserving two-party k-means clustering protocol on 4 datasets available in the UCI machine learning repository: Wisconsin Breast Cancer (WBC), Glass Identification, Stalog Australian Credit Data and Wine (Frank and Asuncion). These datasets are vertically partitioned between Alice and Bob. I describe the datasets and their partitions in Table 3.5 in which n denotes the number of instances, p is the number of attributes, and k is the number of clusters. Alice holds the first p_1 attributes and Bob holds the last p_2 attributes. The observations in the Glass dataset are classified into 6 groups. Here we consider two broad classes: window and non-window.

In the following experiments I used security parameter $K = 512$ and set the threshold in the stopping criterion as $\epsilon = 2^{-10}$. When I apply the secure comparison protocol, I assume that all the numbers belong to $[0, 2^{50})$ ($L = 50$).

Table 3.5: Benchmark datasets for k-means

Dataset	n	p	p_1	p_2	k
WBC	699	9	5	4	2
Glass	214	9	5	4	2
Credit	690	14	7	7	2
Wine	178	13	7	6	3

I first tested the privacy preserving k-means protocol with different secure comparison protocols on small subsets of the WBC dataset. The subset of size n consists of the first n observations from the WBC dataset. On these small subsets of observations, I chose the first 2 observations as the initial cluster centers. The results are reported in Table 3.6. We can see that the privacy preserving k-means protocol using the new secure comparison protocol is several times faster than using existing comparison protocols. This verifies that secure comparison is the bottleneck in the implementation of the privacy preserving k-means clustering protocol.

I then tested the privacy preserving k-means protocol on the WBC, Glass, Aus-

Table 3.6: Execution time of privacy preserving k-means with different secure comparison protocols

	Size			
Protocol	60	90	120	150
new	72s	129s	133s	163s
Bunn et al.	540s	1010s	1072s	1339s
Qi et al.	699s	1301s	1385s	1703s

tralian Credit and Wine datasets. I normalized the Australian Credit and the Wine dataset by dividing each attribute with its standard deviation. As we assume that the datasets are vertically partitioned, each party holds all the data for his/her attributes and can normalize them locally. I used the first k observations in the datasets as the initial centers when I tested the Glass and Wine datasets. For the WBC and German Credit datasets, I first ran the protocol on the first 120 observations in the datasets and used the results as the initial cluster centers to cluster the whole datasets. I report the running time and clustering accuracies of the privacy preserving k-means protocol in Table 3.7. All these datasets contain class labels for their observations. The clustering accuracy is computed as the percentage of correctly clustered observations. I also report the clustering accuracies using the *kmeans* function in the Matlab. We can see that the clustering accuracies of the privacy preserving k-means protocol are similar to the Matlab *kmeans* function.

Table 3.7: Experimental results of privacy preserving k-means on benchmark datasets

	Dataset			
Measure	WBC	Glass	Credit	Wine
Running Time	862s	400s	1715s	831s
Secure Protocol Accuracy	95.99%	88.32%	84.20%	95.51%
kmeans (Matlab) Accuracy	95.99%	88.32%	84.06%	96.63%

The running time of secure comparison is determined by the maximum bit length of the comparands L. To see how the assumption of the maximum bit length affects the overall running time of the privacy preserving k-means clustering protocol, I tested the privacy preserving k-means protocol on the WBC and Glass dataset with different assumptions of the maximum bit length. The results are reported in Figure 3.1 and Figure 3.2, which show that the overall running time of the privacy preserving k-means protocol grows linearly with the maximum bit length.

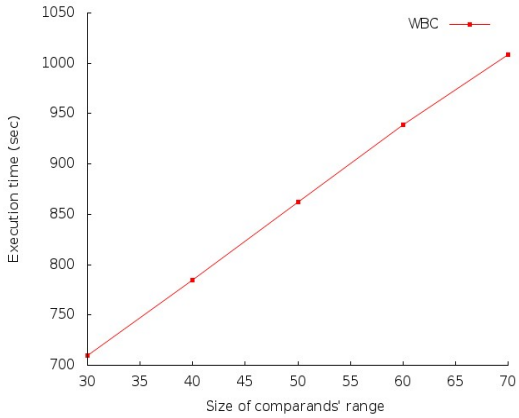

Figure 3.1: Scalability of privacy preserving k-means with respect to the maximum bit length of the comparands (WBC)

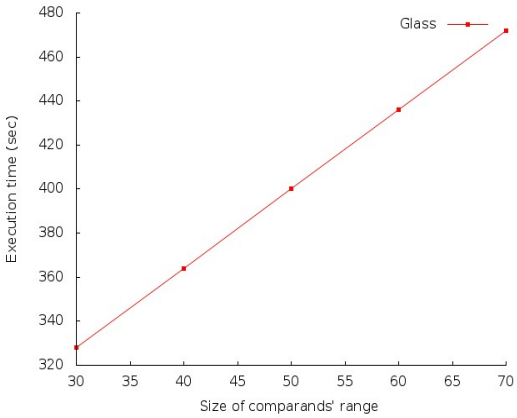

Figure 3.2: Scalability of privacy preserving k-means with respect to the maximum bit length of the comparands (Glass)

Chapter 4

Privacy Preserving Multiple Linear Regression

4.1 Introduction

Multiple linear regression is an approach to model the relationship between a response variable and a set of explanatory variables. It assumes that the response variable depends linearly on the explanatory variables and aims to find the linear predictor function. Multiple linear regression is one of the most successful tools in statistical analysis and has wide applications in many areas. For example, in finance, the well-known capital asset pricing model uses linear regression to analyze and quantify the systematic risk of investments; in economics, the predictions of consumption spending and the demand for liquid assets are also based on linear regression models.

There are three typical computational methods to solve the problem of multiple linear regression: solving normal equations, QR-decomposition and singular value decomposition (SVD) (Seber, 2003; Demmel, 1997). Normal equations can be solved using the Cholesky decomposition method. Solving normal equations is fastest and least accurate. It is adequate when the condition number of the normal matrix is small. QR-decomposition is the standard method and is employed in the software packages such as Matlab, R and S-PLUS. The SVD method is the most accurate in practice but is more expensive.

In many situations, the data to be analyzed are distributed among several parties. For example, one party has a subset of explanatory variables and the other party has the rest of explanatory variables and the response variable. If these parties cooperate with each other and analyze the data jointly, they are able to achieve more accurate statistical models. However, due to privacy concerns, the data holders may not be

willing to disclose their confidential data. In such cases, it is necessary to develop privacy preserving linear regression protocols which allow these parties to perform linear regression jointly while protecting their data privacy.

Privacy preserving multiple linear regression has been studied in the literature. Du et al. (2004) proposed secure matrix multiplication and secure matrix inverse protocols, and hence designed privacy preserving linear regression protocols for vertically partitioned datasets. Their method protects the data matrix by multiplying with random matrices, which cannot provide theoretical guarantee about privacy. Hall et al. (2011) presented secure linear regression protocols for arbitrarily partitioned datasets based on homomorphic encryption. They designed a secure protocol to invert a matrix, which they used to invert normal matrices and solve normal equations. Sanil et al. (2004) proposed another privacy preserving linear regression protocol for vertically partitioned datasets based on the Powell's iterative method. Their protocol discloses aggregate information during each iteration.

I have developed privacy preserving multiple linear regression protocols based on the QR-decomposition method. The protocols use the additive secret-sharing scheme based on homomorphic encryption. In this chapter, I first present a two-party privacy preserving linear regression protocol and prove that it is secure in the semi-honest model. I then extend the two-party protocol to the multi-party cases.

The organization of this chapter is as follows. Section 4.2 and section 4.3 introduce multiple linear regression and the QR-decomposition method, respectively. Section 4.4 discusses previous work on privacy preserving multiple linear regression. Section 4.5 describes how to perform secure operations of real numbers and section 4.6 presents secure inverse square root, secure square root and secure division protocols. Section 4.7 presents privacy preserving multiple linear regression protocols and section 4.8 presents experimental results on benchmark datasets.

4.2 Multiple Linear Regression

In regression analysis, we are interested in studying how a variable, called the response variable, depends on a set of variables called the explanatory variables. A linear regression model assumes that the relationship between the response variable and the explanatory variables is linear. The goal of linear regression analysis is to learn this linear function from a training set, with which we can predict the value of the response variable given the values of the explanatory variables.

Suppose that we have a training dataset consisting of n observations with p ex-

planatory attributes and an additional response attribute,

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix} \in R^{n \times p}, \qquad Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \in R^n,$$

where each row corresponds to an observation, each column in X corresponds to an explanatory attribute and the vector Y corresponds to the response attribute. The i-th observation consists of the values of the p explanatory attributes, $x_i = (x_{i1}, \ldots, x_{ip})^{\mathrm{T}} \in R^p$, and the corresponding response attribute value y_i. The linear regression model aims to find the coefficients

$$\beta = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} \in R^p$$

which best fit the linear relationship

$$Y = X\beta + \epsilon, \tag{4.1}$$

where

$$\epsilon = \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix} \in R^n$$

are error terms.

The regression analysis very often incorporates a constant factor in the model. That is, it tries to find $\alpha \in R$ and $\beta \in R^p$ that best fit the relationship

$$Y = \alpha \mathbf{1} + X\beta + \epsilon, \tag{4.2}$$

where $\mathbf{1}$ is the vector of all $1s$. This is equivalent that we add an additional explanatory attribute in the model 4.1 with fixed values $1s$. Without loss of generality, we focus on the model 4.1.

The least squares method estimates the regression coefficients β by minimizing the residual sum of squares (RSS)

$$f(\beta) = (X\beta - Y)^{\mathrm{T}}(X\beta - Y).$$

The estimated coefficients β are the solution of the normal equation

$$(X^{\mathrm{T}}X)\beta = X^{\mathrm{T}}Y, \tag{4.3}$$

namely,

$$\beta = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}Y. \tag{4.4}$$

Here we assume that X has full rank and $X^{\mathrm{T}}X$ is symmetric and positive. Note that in numerical computation, we typically don't use equation 4.4 to compute β because inverting the normal matrix $X^{\mathrm{T}}X$ is more expensive than solving the normal equation itself. The normal equation can be solved using the Cholesky-decomposition method. It is adequate when the condition number of the normal matrix $X^{rmT}X$ is small. However, the normal matrix is often ill-conditioned and strongly influenced by roundoff errors. Solving normal equations directly is not desirable in this case.

There are two stable methods to solve the multiple linear regression problem: QR-decomposition and singular value decomposition (SVD). QR-decomposition is the standard method and is employed in the software packages such as Matlab, R and S-PLUS. The SVD method is the most accurate in practice but is more expensive. I develop privacy preserving linear regression protocols based on the QR-decomposition method and I will describe this method in detail in next section. See the classic textbook (Seber, 2003) for details about linear regression and its computational methods.

4.3 QR-decomposition

Given an $n \times p$ matrix X, its QR-decomposition is a decomposition of the form

$$X = Q \begin{pmatrix} R \\ 0 \end{pmatrix}, \tag{4.5}$$

where Q is an $n \times n$ orthonormal matrix and R [1] is a $p \times p$ upper triangular matrix. If we write $Q = (Q_p, Q_{n-p})$, where Q_p is an $n \times p$ matrix and Q_{n-p} is an $n \times (n-p)$ matrix, then

$$X = Q_p R. \tag{4.6}$$

Equation 4.6 is called the thin form of QR-decomposition and equation 4.5 is called the fat form.

[1] In this chapter, R denotes both the upper triangular matrix in QR-decomposition and the set of real numbers. Its meaning is clear from the context.

Given the regression dataset $X \in R^{n \times p}$ and $Y \in R^n$, the regression coefficients β are estimated according to equation 4.4. If we have the QR-decomposition of X, then

$$\begin{aligned} \beta &= (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}Y \\ &= ((Q_pR)^{\mathrm{T}}(Q_pR))^{-1}(Q_pR)^{\mathrm{T}}Y \\ &= (R^{\mathrm{T}}R)^{-1}R^{\mathrm{T}}Q_p^{\mathrm{T}}Y \\ &= R^{-1}r_p, \end{aligned} \tag{4.7}$$

where we let $r_p = Q_p^{\mathrm{T}}Y$. So the coefficients are the solution of the linear system $R\beta = r_p$.

QR-decomposition can be performed via the Gram-Schmidt orthogonalization process, the Householder transformation and the Givens transformation. They provide similar computational accuracies. I develop privacy preserving regression protocols based on the Householder transformation. I describe the Householder transformation below.

To factor X into its fat form 4.5, the Householder transformation actually produces

$$Q^{\mathrm{T}}X = \begin{pmatrix} R \\ 0 \end{pmatrix}.$$

We write $X = (c_1, \ldots, c_p)$, where $c_i = (x_{1i}, \ldots, x_{ni})^{\mathrm{T}}$. To transform X into the correct form, we first convert the first column c_1 into the correct form, that is, all the entries in c_1 except the first one are zeros. Let

$$\theta = (-c_{11}/|c_{11}|)\|c_1\|_2,$$

where c_{11} is the first entry in c_1, and

$$w = c_1 - \theta e$$

where $e = (1, 0, \ldots, 0)^{\mathrm{T}} \in R^p$. Also let

$$\begin{aligned} \eta &= \sqrt{2}/\|w\|_2, \\ v &= \eta w, \\ H_1 &= I_n - vv^{\mathrm{T}}, \end{aligned}$$

where I_n is the $n \times n$ identity matrix. The Householder matrix H_1 is orthonormal and symmetric. Then

$$H_1 X = (H_1 c_1, \ldots, H_1 c_p).$$

It can be shown that the first column $H_1 c_1$ is $(\theta, 0, \ldots, 0)$. The i-th column $H_1 c_i$ is computed as

$$\begin{aligned} H_1 c_i &= (I - vv^{\mathrm{T}}) c_i \\ &= c_i - v^{\mathrm{T}} c_i v. \end{aligned} \tag{4.8}$$

We write

$$H_1 X = \begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1p} \\ 0 & & & \\ \vdots & & X_1 & \\ 0 & & & \\ , & & & \end{pmatrix}$$

where X_1 is an $(n-1) \times (p-1)$ matrix.

Similarly, let S_2 be the Householder matrix chosen to convert the first column of X_1 into the correct form and let

$$H_2 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & S_2 & \\ 0 & & & \end{pmatrix}$$

Then

$$H_2 H_1 X = \begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1p} \\ 0 & t_{22} & \cdots & t_{2p} \\ 0 & 0 & & \\ \vdots & \vdots & X_2 & \\ 0 & 0 & & \end{pmatrix}.$$

Continuing on in this way, we are able to transform X into its correct form:

$$H_p \ldots H_1 X = \begin{pmatrix} R \\ 0 \end{pmatrix}.$$

Let $Q = H_1 \dots H_p$, then

$$X = Q \begin{pmatrix} R \\ 0 \end{pmatrix}.$$

To compute $r_p = Q_p^{\mathrm{T}} Y$, we define $r = Q^{\mathrm{T}} Y \in R^n$ and $r_{n-p} = Q_{n-p}^{\mathrm{T}} Y$, then $r^{\mathrm{T}} = (r_p^{\mathrm{T}}, r_{n-p}^{\mathrm{T}})$. Note that $r = Q^{\mathrm{T}} Y = H_p \dots H_1 Y$ and, similar to equation (4.8),

$$\begin{aligned} H_1 Y &= (I_n - vv^{\mathrm{T}})Y \\ &= Y - v^{\mathrm{T}} Y v. \end{aligned}$$

The residual sum of squares is computed as

$$\begin{aligned} RSS &= (X\beta - Y)^{\mathrm{T}}(X\beta - Y) \\ &= (Q_p R\beta - Y)^{\mathrm{T}}(Q_p R\beta - Y) \\ &= (Q_p r_p - Y)^{\mathrm{T}}(Q_p r_p - Y) \\ &= r_p^{\mathrm{T}} Q_p^{\mathrm{T}} Q_p r_p - r_p^{\mathrm{T}} Q_p^{\mathrm{T}} Y - Y^{\mathrm{T}} Q_p r_p + Y^{\mathrm{T}} Y \\ &= Y^{\mathrm{T}} Y - r_p^{\mathrm{T}} r_p \end{aligned} \tag{4.9}$$

I summarize the procedure in Algorithm 4.1. The algorithm outputs R and r_p, which we use to compute the regression coefficients β. In the algorithm, I use Matlab-like notations to denote matrices and vectors. For example, $X[i,i]$ is the (i,i)-th entry in X, $X[i:n,i]$ is the sub-vector of the i-th column in X, and $X[i:n,i:p]$ denotes the sub-matrix of X. e_i denotes the vector $(1,0,\dots,0)^{\mathrm{T}}$ of length $n-i+1$.

Algorithm 4.1 Householder transformation

Input: an $n \times p$ matrix X and an n-vector Y.

Output: the $p \times p$ upper triangular matrix R in the QR-decomposition of X, $X = Q_p R$, and the p-vector $r_p = Q_p^{\mathrm{T}} Y$.

1: $r = Y$
2: **for** $i = 1$ to p **do**
3: $\quad \theta = -X[i,i]/|X[i,i]| * \|X[i:n,i]\|_2$
4: $\quad w[i:n] = X[i:n,i] - \theta * e_i$
5: $\quad \eta = \sqrt{2}/\|w[i:n]\|_2$
6: $\quad v[i:n] = \eta * w[i:n]$
7: $\quad X[i:n,i:p] = X[i:n,i:p] - (v[i:n]^{\mathrm{T}} * X[i:n,i:p]) * v[i:n]$
8: $\quad r[i:n] = r[i:n] - (v[i:n]^{\mathrm{T}} * r[i:n]) * v[i:n]$
9: **end for**
10: $\{R = X[1:p,1:p],\ r_p = r[1:p]\}$

The linear system $R\beta = r_p$ can be solved using the back-substitution method, as

presented in Algorithm 4.2.

Algorithm 4.2 Back-substitution algorithm for solving linear system of equations

Input: a $p \times p$ upper triangular matrix and a p-vector b.
Output: the solution of the linear system $R\beta = b$

1: **for** $i = p$ downto 1 **do**
2: $\quad \beta[i] = b[i]/R[i,i]$
3: $\quad b[1:i-1] = b[1:i-1] - \beta[i] * R[1:i-1,i]$
4: **end for**

4.4 Previous Work

Du et al. (2004) considered the problem of privacy preserving multivariate statistical analysis, including linear regression and classification, over vertically partitioned datasets. They proposed secure matrix multiplication and secure matrix inverse protocols which can be then used to design privacy preserving linear regression protocols.

Du et al. used algebraic techniques to design secure matrix multiplication and secure matrix inverse protocols. Suppose that Alice has a $p \times n$ matrix S and Bob has another $n \times q$ matrix T. Alice and Bob jointly generate an invertible $n \times n$ matrix M. Let $M = (M_{left}, M_{right})$, where M_{left}, M_{right} are $n \times n/2$ matrices, and $M^{-1} = (M^{\mathrm{T}}_{inv-top}, M^{\mathrm{T}}_{inv-bottom})^{\mathrm{T}}$, where $M_{inv-top}, M_{inv-bottom}$ are $n/2 \times n$ matrices. Alice computes $S_1 = SM_{left}$, $S_2 = SM_{right}$ and sends S_1 to Bob. Bob computes $T_1 = M_{inv-top}T$, $T_2 = M_{inv-bottom}T$ and sends T_2 to Alice. Now Alice computes $V_A = S_2T_2$ and Bob computes $V_B = S_1T_1$. It can be shown that $V_A + V_B = ST$.

When S and T are both square matrices, Alice and Bob can use the secure matrix multiplication to privately compute $(S+T)^{-1}$. Bob first chooses two random matrices F and G. Alice and Bob use secure matrix multiplication to multiply $S+T$ with F and G and let Alice obtain the matrix $F(S+T)G$. The purpose of random matrices F and G is to prevent Alice from learning the matrix $S+T$. Now Alice computes $W = G^{-1}(S+T)^{-1}F^{-1}$. Then Alice and Bob use secure matrix multiplication to multiply W with F and G. The result is that Alice obtain V_A and Bob obtain V_B such that $V_A + V_B = (S+T)^{-1}$.

The solution of the multiple linear regression is $\beta = (X^{\mathrm{T}}X)^{-1}XY$. It is clear that we can privately compute β using secure multiplication matrix and secure matrix inverse protocols. Note that the matrix multiplication and matrix inverse protocols try to protect matrices by multiplying them with random matrices. However, it is

generally not a good idea to hide a number or matrix by multiplication because such methods may leak information (Kiltz et al., 2005).

Hall et al. (2011) proposed secure multiple linear regression protocols for arbitrarily partitioned datasets based on homomorphic encryption. They noticed that the inverse of a matrix S can be computed using the following coupled iteration:

$$U_{t+1} = 2U_t - U_t V_t, U_0 = c^{-1} I$$

$$V_{t+1} = 2V_t - V_t^2, V_0 = c^{-1} S$$

where $V_t = U_t S$ and c is chosen by the user, for example, as the trace of S.

Based on these iterative formulas, Hall et al. developed secure protocols for linear regression based on homomorphic encryption. Their protocols disclose the number of iterations, which is related to the condition number of the covariance matrix $X^{\mathrm{T}} X$.

Sanil et al. (2004) considered the scenario where the explanatory attributes are partitioned among m ($m > 2$) parties and the response attribute is known by all parties. That is, all the parties know Y and the dataset X is vertically partitioned among these parties, $X = (X_1, \ldots, X_m)$, where the j-th party holds X_j. As we know, the least squares method estimates the regression coefficients β by minimizing the function $f(\beta) = (X\beta - Y)^{\mathrm{T}}(X\beta - Y)$. Sanil et al. noticed that the Powell's iterative method can be used to minimize the quadratic function $f(\beta)$. A basic step in the Powell minimization process is to compute $X\beta$. If each party j knows the regression coefficients corresponding to the attributes he/she holds, denoted by β_j, then each party j can compute $X_j\beta_j$ locally and all the participating parties can use secure sum to jointly compute $X\beta = \sum_{j=1}^{m} X_j\beta_j$. Based on this idea, Sanil et al. proposed a secure linear regression protocol. Their protocol doesn't employ cryptographic techniques and can be implemented efficiently. However, each party obtains aggregate information such as $X\beta$ during each iteration. If we only want to compute the regression coefficients β, the protocol will disclose more information than necessary.

4.5 Representing Real Numbers in the Plaintext Domain

My research focuses on the design and implementation of privacy preserving distributed data mining protocols based on homomorphic encryption. The plaintext

domain of the cryptosystem, denoted by $Z_N = \{0, 1, \ldots, N-1\}$, is a set of natural numbers. However, in data mining tasks, we deal mostly with real numbers, and we need a way to represent them in the plaintext domain.

In section 3.2, I discuss how to represent a signed integer v in the plaintext domain Z_N. The integer v is represented in Z_N as v if $v \geq 0$ and as $N + v$ if $v < 0$. To represent a real number x in the plaintext domain Z_N, we use the representation of the signed integer $\lfloor x2^P \rfloor$ in Z_N, where P is some positive integer. That is, if $x \geq 0$, it is represented in Z_N as $\lfloor x2^P \rfloor$; if $x < 0$, it is represented as $N + \lfloor x2^P \rfloor$. Such representation is similar to the way we represent real numbers in computers and P is the number of bits we use to represent the fractional parts of real numbers and determine the accuracy of the representations. We use the notation $[x]$ to denote the representation of a real number x in the plaintext domain. Hereafter, when we say that an integer x is split between Alice and Bob, we mean that $x = x_A + x_B \pmod{N}$. When we say that a real number x is split between Alice and Bob, we mean that its representation $[x]$ is split between Alice and Bob, $[x] = x_A + x_B \pmod{N}$, where Alice holds the share x_A and Bob holds the share x_B.

We now consider secure addition and secure multiplication of real numbers. Suppose that two real numbers, x and y, are split between Alice and Bob, $[x] = x_A + x_B \pmod{N}$, $[y] = y_A + y_B \pmod{N}$. Alice and Bob wish to obtain their respective shares of $s = x + y$ and the shares of $r = xy$. The secure addition of x and y is just the secure addition of $[x]$ and $[y]$. Alice computes $s_A = (x_A + y_A) \bmod N$ and Bob computes $s_B = (x_B + y_B) \bmod N$, then s_A and s_B are shares of $s = x + y$.

To privately compute $r = xy$, we first use secure multiplication of integers (Protocol 2.1) to multiply $[x]$ and $[y]$. Note that $[x]$ and $[y]$ are the representations of the signed integers $\lfloor x2^P \rfloor$ and $\lfloor y2^P \rfloor$, respectively. The secure multiplication of $[x]$ and $[y]$ results in two shares of $v = \lfloor x2^P \rfloor \lfloor y2^P \rfloor$. Note that $[xy] = \lfloor xy2^P \rfloor \approx \lfloor x2^P \rfloor \lfloor y2^P \rfloor / 2^P$. To obtain (the shares of) $[xy]$, we need to divide v by 2^P privately. Before I present a secure division with public divisor protocol, I first prove the following theorem.

Theorem 4.1. *Suppose that a secret signed integer v ($|v| < N/3$) is split between Alice and Bob, $v = v_A + v_B \pmod{N}$, and M is a public positive integer. Let $\alpha = 1$ if $v_A, v_B \in [0, N/3)$ and 0 otherwise, $\beta = 1$ if $v_A, v_B \in [2N/3, N)$ and 0 otherwise, $\gamma = 1 - \alpha + \beta$. Let*

$$r = \lfloor v_A/M \rfloor + \lfloor v_B/M \rfloor - \gamma \lfloor N/M \rfloor.$$

Then

$$|r - v/M| < 2.$$

Proof. According to Corollary 3.2,

$$v = v_A + v_B - \gamma N.$$

So

$$v/M = v_A/M + v_B/M - \gamma N/M.$$

Let

$$v_A/M = \lfloor v_A/M \rfloor + \epsilon_1$$

$$v_B/M = \lfloor v_B/M \rfloor + \epsilon_2$$

$$N/M = \lfloor N/M \rfloor + \epsilon_3$$

such that $\epsilon_1, \epsilon_2, \epsilon_3 \in [0, 1)$. Then

$$\begin{aligned} v/M &= v_A/M + v_B/M - \gamma N/M \\ &= \lfloor v_A/M \rfloor + \epsilon_1 + \lfloor v_B/M \rfloor + \epsilon_2 - \gamma(\lfloor N/M \rfloor + \epsilon_3). \\ &= r + \epsilon_1 + \epsilon_2 - \gamma\epsilon_3 \end{aligned}$$

So

$$|r - v/M| = |\epsilon_1 + \epsilon_2 - \gamma\epsilon_3|.$$

Note that $0 \leq \gamma \leq 2$. We have

$$|\epsilon_1 + \epsilon_2 - \gamma\epsilon_3| < 2,$$

namely,

$$|r - v/M| < 2.$$

□

Suppose that γ_A, γ_B are the shares of γ, $\gamma = \gamma_A + \gamma_B \pmod N$. Then

$$\begin{aligned} r &= \lfloor v_A/M \rfloor + \lfloor v_B/M \rfloor - (\gamma_A + \gamma_B)\lfloor N/M \rfloor \pmod N \\ &= \lfloor v_A/M \rfloor - \gamma_A\lfloor N/M \rfloor + \lfloor v_B/M \rfloor - \gamma_B\lfloor N/M \rfloor \pmod N \end{aligned}$$

If we let $r_A = (\lfloor v_A/M \rfloor - \gamma_A\lfloor N/M \rfloor) \bmod N$ and $r_B = (\lfloor v_B/M \rfloor - \gamma_B\lfloor N/M \rfloor) \bmod N$, then r_A and r_B are the two shares of r. If v/M is sufficiently large, r is a good approx-

imation of v/M. Based on this idea, I propose a secure division with public divisor protocol (Protocol 4.3). We can implement Protocol 4.3 using secure multiplication (Protocol 2.1). Note that this protocol only computes an approximation of v/M and it is only useful when we know that v/M is sufficiently large.

Protocol 4.3 Secure division with public divisor

Input: a signed integer v is split between Alice and Bob, $v =_A +v_B \pmod N$, and a public positive integer M.

Output: Alice and Bob obtain their respective shares of $r = v/M$.

1: Alice: $\alpha_1 = 1$ if $x_A < N/3$ and $\alpha_1 = 0$ otherwise;
2: Bob: $\alpha_2 = 1$ if $x_B < N/3$ and $\alpha_2 = 0$ otherwise;
3: Alice and Bob use Protocol 2.1 to securely compute $\alpha = \alpha_1\alpha_2$ and obtain the shares α_A and α_B, respectively.
4: Alice: $\beta_1 = 1$ if $2N/3 \leq x_A$ and $\beta_1 = 0$ otherwise;
5: Bob: $\beta_2 = 1$ if $2N/3 \leq x_B$ and $\beta_2 = 0$ otherwise;
6: Alice and Bob use Protocol 2.1 to securely compute $\beta = \beta_1\beta_2$ and obtain the shares β_A and β_B, respectively;
7: Alice: $\gamma_A = (1 - \alpha_A + \beta_A) \bmod N$;
8: Bob: $\gamma_B = (-\alpha_B + \beta_B) \bmod N$;
9: Alice: $r_A = (\lfloor x_A/M \rfloor - \gamma_A \lfloor N/M \rfloor) \bmod N$;
10: Bob: $r_B = (\lfloor x_B/M \rfloor - \gamma_B \lfloor N/M \rfloor) \bmod N$.

I continue to discuss the secure multiplication of two real numbers. After we use secure multiplication of integers (Protocol 2.1) to multiple $[x]$ and $[y]$, we can now use Protocol 4.3 to privately divide the product $v = \lfloor x2^P \rfloor \lfloor y2^P \rfloor$ by 2^P. Note that typically we tens of bits to represent the fractional parts of real numbers. The difference between $v/2^P$ and r is negligible and r is a good approximation of $[xy]$.

Some clarification is in order. The representation of a signed integer x in the plaintext domain Z_N is x if $x \geq 0$ and $N + x$ if $x < 0$, while the representation of a real number x in Z_N is $\lfloor x2^P \rfloor$ if $x \geq 0$ and $N + \lfloor x2^P \rfloor$ if $x < 0$. Secure multiplication of two integers x and y means that we use Protocol 2.1 to multiple x and y. When we say secure multiplication of two real numbers x and y, we mean that we first use Protocol 2.1 to multiply $[x]$ and $[y]$ and then use Protocol 4.3 to privately divide the product by 2^P. In some cases, we use Protocol 2.1 to multiple an integer x and the representation of a real number y. The result is the (shares of) representation of the real number xy.

4.6 Subprotocols

In this section, I first propose new secure protocols for inverse square root and square root operations. I then present the general secure division protocol.

4.6.1 Secure Inverse Square Root and Secure Square Root

Inverse square root $1/\sqrt{x}$ is a basic operation in data mining algorithms and statistical analysis. For example, the correlation coefficient of two random variables u and v is defined as $\rho_{uv} = \sigma_{uv}/\sqrt{\sigma_u \sigma_v}$, where σ_u, σ_v and σ_{uv} are the variances and covariance of u and v, respectively. We want to design a secure protocol to privately compute $1/\sqrt{x}$ when x is a secret real number split between two parties. For ease of presentation, I first consider the case when x is an integer. Suppose that the integer x is split between Alice and Bob, $x = x_A + x_B \pmod N$. Alice and Bob wish to privately compute $z = 2^M/\sqrt{x}$, for some public integer M and obtain their respective shares of z. I propose a secure inverse square root based on Newton's iterative method (Kincaid, 2002). Let

$$f(z) = 2^{2M}/z^2 - x.$$

Then $2^M/\sqrt{x}$ is the zero of the function $f(z)$. We can use Newton's iterative method to find this zero:

$$\begin{aligned} z_{t+1} &= z_t - f(z_t)/f'(z_t) \\ &= z_t(3 \cdot 2^{2M} - x z_t^2)/2^{2M+1}. \end{aligned}$$

Note that the operations involved in the iterative formula are multiplication, subtraction and division with constant divisor. We can compute this formula privately using secure multiplication and secure division with public divisor.

One issue with the iterative method is that we need to choose a good initial point. We assume that x belongs to a known interval $(0, 2^L)$. We find an integer B such that $2^{2B} \leq x < 2^{2(B+1)}$ and the initial point can be chosen as $3/2 \cdot 2^{M-B-1} = 3 \cdot 2^{M-B-2}$. Note that Newton's iterative method converges quadratically. If we execute $T = O(\log M)$ iterations , it will compute a good approximation of $2^M/\sqrt{x}$.

I present the secure protocol of inverse square root in Protocol 4.4. Lines 1-5 compute the initial point z_1. We first use Protocol 3.2 to securely transform x into its binary representation $x = x_L x_{L-1} \ldots x_1$ and then scan this binary string from the lowest bits to the highest bits. If $x_i = 0$, we don't change z_1; otherwise, z_1 is set to

be $3/2 \cdot 2^{M-(i+1)/2} = 3 \cdot 2^{M-(i+1)/2-1}$. We then execute line 7 with a fixed number of iterations. Clearly, we can implement Protocol 4.4 using secure multiplication and secure division with public divisor.

In Protocol 4.4 we assume that x is an integer. When we consider a real number x and apply Protocol 4.4 on its representation $[x] = \lfloor x2^P \rfloor$, then the result is $2^M/\sqrt{\lfloor x2^P \rfloor}$. Note that $2^M/\sqrt{\lfloor x2^P \rfloor} \approx 2^{M-P/2}/\sqrt{x}$, so the result is the representation of the real number $2^{M-3P/2}/\sqrt{x}$. When we need to compute the representation of the real number $2^M/\sqrt{x}$, we can replace M with $M + 3P/2$ in the protocol.

Protocol 4.4 Secure inverse square root

Input: a secret integer $x \in (0, 2^L)$ is split between Alice and Bob.
Output: Alice and Bob obtain their respective shares of $z = 2^M/\sqrt{x}$.

1: Alice and Bob use Protocol 3.2 to securely transform x into it binary representation $x = x_L x_{L-1} \dots x_1$, whose bits x_i are split between Alice and Bob.
2: $z_1 = 3 \cdot 2^{M-1}$
3: **for** $i = 1$ to L **do**
4: $\quad z_1 = (1 - x_i) \cdot z_1 + x_i \cdot 3 \cdot 2^{M-(i+1)/2-1}$
5: **end for**
6: **for** $t = 1$ to T **do**
7: $\quad z_{t+1} = z_t(3 \cdot 2^{2M} - x z_t^2)/2^{2M+1}$
8: **end for**

Noticing that $\sqrt{x} = 1/\sqrt{x} \cdot x$, we are able to implement a secure protocol for the square root operation, which I present below.

Protocol 4.5 Secure square root

Input: a secret integer x is split between Alice and Bob, $x = x_A + x_B \pmod N$.
Output: Alice and Bob obtain their respective shares of $\sqrt{x}$.

1: Alice and Bob use Protocol 4.4 to privately compute $y = 2^M/\sqrt{x}$ for some sufficiently large M and obtain their respective shares of y.
2: Alice and Bob use Protocol 2.1 to privately compute $z = xy$ and obtain their respective shares of z.
3: Alice and Bob use Protocol 4.3 to privately divide z by 2^M. Alice and Bob return the shares they obtain at this step.

In Protocol 4.5, M is chosen sufficiently large to provide sufficient accuracy. For example, if we assume that $x < 2^L$ and we use P bits to represent the fractional parts of real numbers, we can choose $M = P + L$. Protocol 4.5 privately computes the square root of an integer. If we wish to compute the square root of a real number

x, we can apply Protocol 4.5 on $[x]$, the result is $\sqrt{[x]} = \sqrt{\lfloor x2^P \rfloor}$. We can securely multiply it with $2^{P/2}$ to obtain the representation of the real number $\sqrt{x}$.

Both inverse square root and square root are basic operations in many data mining and machine learning algorithms. Han et al. (2009) proposed secure protocols for both operations in their design of a secure SVD protocol. In their solution, Alice first selects a random r. Alice and Bob use secure multiplication to privately compute $t = rx_B$ and obtain their respective shares of t. Alice sends $rx_A + t_A$ to Bob. Bob computes $rx = rx_A + t_A + t_B$. Then Alice and Bob use secure multiplication to compute $\sqrt{x} = 1/\sqrt{r} \cdot \sqrt{rx}$. Kiltz et al. (2004) pointed out that it is not a good idea to hide a secret by multiplication. We notice that if x is a small number as in most applications, then Bob will be able to factor rx and guess the secret x.

4.6.2 Secure Division

Suppose that two secrets x and y are split between Alice and Bob; they wish to privately compute $z = y/x$ and obtain their respective shares of z. Note that $y/x = ((2^M/x) \cdot y)/2^M$. So we only need a secure protocol to compute $2^M/x$ for some public integer M.

Bunn et al. (2007) proposed a secure division protocol to compute $\lfloor y/x \rfloor$ based on the general division procedure, which I present in section 3.4. Their protocol takes $O(L^2)$ secure multiplications when x, y belong to a known interval $[0, 2^L)$. When L is large, this protocol is not efficient.

S. From (2006) proposed a secure division protocol based on Newton's iterative method. Let $f(z) = 1/z - x/2^M$. Then $2^M/x$ is the zero of $f(z)$. This zero can be computed using Newton's iterative formula:

$$z_{t+1} = z_t(2^{M+1} - z_t x)/2^M.$$

The initial point is chosen as $z_1 = 3/2 \cdot 2^{M-B-1} = 3 \cdot 2^{M-B-2}$ if $2^B \leq x < 2^{B+1}$. The protocol is implemented using the Shamir's polynomial secret-sharing scheme, which can guarantee privacy as long as the majority of the parties are honest. Note that the protocols in the Shamir's secret-sharing scheme don't work in the two-party case.

It is easy to adapt the idea to implement a secure division protocol based on homomorphic encryption, which I present in Protocol 4.6. We can implement Protocol 4.6 using secure multiplication, secure division with public divisor. This protocol computes an approximation of $2^M/x$. As Newton's iterative method converges quadratically, we execute line 7 with a fixed number of iterations $T = O(\log M)$.

Protocol 4.6 Secure division

Input: a secret integer $x \in (0, 2^L)$ is split between Alice and Bob, $x = x_A + x_B \pmod N$.

Output: Alice and Bob obtain their respective shares of $z = 2^M/x$.

1: Alice and Bob use Protocol 3.2 to securely transform x into it binary representation $x = x_L x_{K-1} \dots x_1$, whose bits x_i are split between Alice and Bob.
2: $z_1 = 3 \cdot 2^{M-1}$
3: **for** $i = 1$ to L **do**
4: $\quad z_1 = (1 - x_i) \cdot z_1 + x_i \cdot 3 \cdot 2^{M-i-1}$
5: **end for**
6: **for** $t = 1$ to T **do**
7: $\quad z_{t+1} = z_t(2^{M+1} - xz_t)/2^M$
8: **end for**

Protocol 4.6 securely computes $2^M/x$ when x is an integer. If we apply Protocol 4.6 on the representation of a real number x, the result is $2^M/[x]$. Note that $2^M/[x] = 2^M/\lfloor x2^P \rfloor \approx 2^{M-P}/x$. So the result is the representation of $2^{M-2P}/x$. If we want to privately compute the real number $2^M/x$, we simply replace M with $M + 2P$ in the protocol.

4.7 Privacy Preserving Multiple Linear Regression

In this section, I first present a privacy preserving two-party multiple linear regression protocol for arbitrarily partitioned datasets. I prove that it is secure in the semi-honest model. I then show how to extend it to the multi-party cases.

4.7.1 Two-Party Cases

The linear regression model aims to estimate the coefficients β given the training set (X, Y), where $X \in R^{n \times p}$ consists of n observations with p explanatory attributes and $Y \in R^n$ consists of n corresponding response attribute values. Now we suppose that the regression dataset (X, Y) is arbitrarily distributed between Alice and Bob, $X = X_1 + X_2$ and $Y = Y_1 + Y_2$, where Alice holds X_1 and Y_1 and Bob holds X_2 and Y_2. We assume that the attribute values held by Alice and Bob don't overlap. If one party doesn't know a attribute value, he/she simply sets the corresponding entry in (X_i, Y_i) as 0. Although both Alice and Bob would like to know the regression coefficients β, they are not willing to disclose their confidential data to each other. I use the additive secret-sharing scheme based on homomorphic encryption to design a two-

party privacy preserving multiple linear regression protocol, which allows Alice and Bob to conduct multiple linear regression on the joint dataset (X, Y) while protecting the privacy of their individual data.

We consider a more general scenario where X and Y are secrets split between Alice and Bob, $X = X_A + X_B \pmod N$, $Y = Y_A + Y_B \pmod N$. This setting is useful when some private preprocessing procedures such as variable selection are first applied on the original dataset and then we perform privacy preserving linear regression on the resulting secret dataset. It also provides a way to reduce multi-party protocols to two-party protocols, which I present in next subsection.

The computation of the regression coefficients consists of two steps: the Householder transformation (Algorithm 4.1) and the back-substitution procedure (Algorithm 4.2). We can design privacy preserving protocols for these two procedures separately and then combine them to implement a privacy preserving multiple linear regression protocol.

First consider the Householder transformation (Algorithm 4.1). In line 3, we need to compute $X[i,i]/|X[i,i]|$, the sign of $X[i,i]$. If we define $r = 1$ iff $X[i,i] \leq 0$ and 0 otherwise, the sign of $X[i,i]$ equals $1 - 2r$. So we use can secure comparison to compute $X[i,i]/|X[i,i]|$. Other operations involved in Algorithm 4.1 include addition/subtraction, multiplication (line 3), scalar-vector multiplication (line 4, 8), vector-matrix multiplication (line 7), scalar product (line 8), square root (line 3) and inverse square root (line 5). Note that scalar-vector multiplication, vector-matrix multiplication and scalar product can be computed using a series of multiplications. So we can use secure multiplication, secure inverse square root, square root and secure comparison to implement a privacy preserving two-party protocol for the Householder transformation.

In the implementation of the secure protocol, all the variables are secrets split between Alice and Bob, $r = r_A + r_B \pmod N$, $\theta = \theta_A + \theta_B \pmod N$, $w = w_A + w_B \pmod N$, $\eta = \eta_A + \eta_B \pmod N$, $v = v_A + v_B \pmod N$, $R = R_A + R_B \pmod N$. Note that for all the subprotocols used here, both the inputs and the outputs are assumed to be split between Alice and Bob and we are able to compose them sequentially to implement a secure protocol for the Householder transformation. Remember that when we need to securely multiply two real numbers, we first use Protocol 2.1 to multiply their representations in the plaintext domain and then use Protocol 4.3 to privately divide the product by 2^P.

Similarly, we use secure multiplication and secure division to implement a privacy preserving protocol for the back-substitution algorithm (Algorithm 4.2), in which

both the inputs R and b and the output β are secrets split between Alice and Bob.

We combine the secure protocols for the Householder transformation and the back-substitution algorithm to implement a privacy preserving two-party multiple linear regression protocol (Protocol 4.7).

Protocol 4.7 Privacy preserving two-party multiple linear regression protocol

Input: a regression dataset (X, Y), where $X \in R^{n \times p}$ and $Y \in R^n$, is split between Alice and Bob, $X = X_1 + X_2 \pmod N$, $Y = Y_1 + Y_2 \pmod N$. Alice has input $(1^K, X_1, Y_1)$ and Bob has input $(1^K, X_2, Y_2)$, where K is the security parameter.

Output: Both Alice and Bob obtain the regression coefficients β.

1: Alice generates a pair of keys $(e, d) = G(1^K)$ and sends the public key e to Bob.
2: Alice and Bob privately compute $R \in R^{p \times p}$ and $r_p \in R^p$ based on the Householder transformation (Algorithm 4.1) and obtain their respective shares of R and r_p.
3: Alice and Bob privately solve the linear system $R\beta = r_p$ based on the back-substitution method (Algorithm 4.2) and obtain their respective shares of β.
4: Alice and Bob exchange their shares of β so both parties know the regression coefficients β.

Theorem 4.2. *Protocol 4.7 is secure in the semi-honest model.*

Proof. To prove that the two-party privacy preserving multiple linear regression protocol is secure in the semi-honest model, we need to show that:

(1) Given Alice's input and output, we are able to simulate her view during the execution of the protocol;

(2) Given Bob's input and output, we are able to simulate his view during the execution of the protocol.

Note that we use secure multiplication, secure division, secure inverse square root and secure square root to implement Protocol 4.7. All these subprotocols are implemented using secure multiplication, so we actually implement Protocol 4.7 using secure multiplication only.

We first examine the message exchanges during the execution of Protocol 4.7.

(1) At the beginning (step 1), Alice generates a pair of keys $(e, d) = G(1^K)$ and sends the public key e to Bob. As usual, the modulus associated with this key pair is denoted by N.

(2) During each invocation of secure multiplication, Alice sends two encrypted messages to Bob and Bob sends the encryption of a random number to Alice. We denote the total number of invocations of secure multiplication by v.

(3) At the last step (step 4), Alice and Bob exchange their shares of β_i $(i = 1, \ldots, p)$. Alice sends $\beta_{i,A}$ to Bob and Bob sends $\beta_{i,B}$ to Alice.

To simulate Alice's view during the execution of Protocol 4.7, we first check the elements of Alice's view.

(1) Alice has input $(1^K, X_1, Y_1)$.

(2) Alice uses a sequence of random coins $r_{1,A}$ to generate the key pair $(e, d) = G(1^k, r_{1,A})$. During the invocations of secure multiplication, Alice uses random numbers in Z_N^* to encrypt messages. Denote by $r_{2,A} = (r_{2,1,A}, \ldots, r_{2,2v,A})$ the sequences of random coins used to generate those random numbers. Denote all these sequences of random coins by $r_A = (r_{1,A}, r_{2,A})$.

(3) During each invocation of secure multiplication, Alice receives the encryption of a random number from Bob. Denote all these messages by $M_{1,A} = (m_1, \ldots, m_v)$.

(4) Alice receives $\beta_{i,B}$ $(i = 1, \ldots, p)$ from Bob at step 4. Denote these messages by $M_{2,A} = (\beta_{1,B}, \ldots, \beta_{p,B})$.

So Alice's view is $VIEW_A = (1^K, X_1, Y_1, r_A, M_{1,A}, M_{2,A})$. I now show how to simulate Alice's view based on her input $(1^K, X_1, Y_1)$ and her output β. The simulator generates a number of sequences of independent random coins $r'_{1,A}$ and $r'_{2,A} = (r'_{2,1,A}, \ldots, r'_{2,2v,A})$, which correspond to the sequences $r_{1,A}, r_{2,A}$ that Alice uses in the real execution. Let $r'_A = (r'_{1,A}, r'_{2,A})$. It is identically distributed with $r_A = (r_{1,A}, r_{2,A})$. The simulator generates $(e', d') = G(1^K, r'_{1,A})$, which is identically distributed with (e, d). Denote the modulus associated with the key pair (e', d') by N'.

I now show how to simulate the messages $M_1 = (m_1, \ldots, m_v)$. As I discuss in section 2.3, each message m_i Alice receives during the invocation of secure multiplication is the encryption of a random number and is identically distributed with $E_e(b_1, b_2)$, where b_1 is uniformly random in Z_N and b_2 is uniformly random in Z_N^*. The simulator generates a uniformly random number b'_1 in $Z_{N'}$ and another uniformly random number b'_2 in $Z_{N'}^*$ and computes $m'_i = E_{e'}(b'_1, b'_2)$. m'_i is identically distributed with m_i. Let $M'_{1,A} = (m'_1, \ldots, m'_v)$. $M'_{1,A}$ is identically distributed with $M_{1,A}$.

I next show how to simulate the messages in $M_2 = (\beta_{1,B}, \ldots, \beta_{p,B})$. Before Alice and Bob exchange their shares of β_i, Alice has the share $\beta_{i,A}$ such that $(\beta_{i,A} + \beta_{i,B}) \bmod N = \beta_i$. $\beta_{i,A}$ is computed as $\beta_{i,A} = h_{i,A}(1^K, X_1, Y_1, r_A, M_{1,A})$ for some function $h_{i,A}$. The simulator computes $\beta'_{i,A} = h_{i,A}(1^K, X_1, Y_1, r'_A, M'_{1,A})$, which is identically distributed with $\beta_{i,A}$. Since the simulator is given the final result β_i, it can compute $\beta'_{i,B} = (\beta_i - \beta'_{i,A}) \bmod N'$, which is identically distributed with $\beta_{i,B} = (\beta_i - \beta_{i,A}) \bmod N$. Let $M'_{2,A} = (\beta'_{1,B}, \ldots, \beta'_{p,B})$. It is identically distributed with $M_{2,A}$.

Let $M'_A = (1^K, X_1, Y_1, r'_A, M'_{1,A}, M'_{2,A})$. From the above discussion we know that M'_A is identically distributed with Alice's view $VIEW_A$ and thus it is computationally indistinguishable from $VEIW_A$. So we can simulate Alice's view from her input

$(1^K, X_1, Y_1)$ and her output β.

I now show how to simulate Bob's view during the execution of Protocol 4.7 from Bob's input $(1^K, X_2, Y_2)$ and his output β. Bob's view is divided into five parts.

(1) Bob has input $(1^K, X_2, Y_2)$.

(2) During each invocation of secure multiplication, Bob uses a sequence of random coins r_i to generate a uniformly random number b_1 in Z_N and another uniformly random number b_2 in Z_N^* and then computes $E_e(b_1, b_2)$. Here b_1 and b_2 correspond to r and s_3 in Protocol 2.1, respectively. Denote these sequences of random coins by $r_B = (r_{2,1,B}, \ldots, r_{2,v,B})$. Here v is the total number of invocations of secure multiplication. We use the same subscript 2 here as we use in the simulation of Alice's view because they are both used to simulate the invocations of secure multiplication.

(3) At the beginning (step 1), Bob receives the public key e from Alice.

(4) During each invocation of secure multiplication, Bob receives the encryptions of two numbers from Alice. Bob receives altogether $2v$ messages during the v invocations of secure multiplication. Denote the i-th message by $m_i = E_e(a_i, r_{2,i,A})$, where a_i is some number only known by Alice and $r_{2,i,A}$ is the random coins Alice uses to encrypt the message. Denote all these messages by $M_{1,B} = (m_1, \ldots, m_{2v}) = (E_e(a_1, r_{2,1,A}), \ldots, E_e(a_{2v}, r_{2,2v,A}))$.

(5) At step 4, Bob receives $\beta_{i,A}$ $(i = 1, \ldots, p)$ from Alice. Denote these messages by $M_{2,B} = (\beta_{1,A}, \ldots, \beta_{p,A})$.

So Bob' view is $VIEW_B = (1^K, X_2, Y_2, r_B, e, M_{1,B}, M_{2,B})$. I now show how to simulate Bob's view based on his input $(1^k, X_2, Y_2)$ and his output β. The simulator generates a number of sequences of independent random coins $r'_B = (r'_{2,1,B}, \ldots, r'_{2,v,B})$, which correspond to the sequences of random coins $r_B = (r_{2,1,B}, \ldots, r_{2,v,B})$ that Bob uses in the real execution. r'_B and r_B are identically distributed.

The simulator generates a pair of keys $(e', d') = G(1^K, r'_{1,A})$, where $r'_{1,A}$ is a sequence of independent random coins which the simulator generates to simulate $r_{1,A}$ that Alice uses in the real execution. Because $r'_{1,A}$ is identically distributed with $r_{1,A}$, (e', d') is also identically distributed with the key pair (e, d) that Alice generates in the real execution, and e' is also identically distributed with e. Denote the modulus associated with the key pair (e', d') by N'.

I now show how to simulate the messages $M_{1,B} = (E_e(a_1, r_{2,1,A}), \ldots, E_e(a_{2v}, r_{2,2v,A}))$. The simulator computes $M'_{1,B} = (E_{e'}(0, r'_{2,1,A}), \ldots, E_{e'}(0, r'_{2,2v,A}))$, where $r'_{2,i,A}$ is a sequence of independent random coins which the simulate generates to simulate $r_{2,i,A}$. To show that $M'_{1,B}$ is computationally indistinguishable from $M_{1,B}$, we define $M''_{1,B} = (E_e(0, r_{2,1,A}), \ldots, E_e(0, r_{2,2v,A}))$. Since e' and e are identically distributed

and $r'_{2,i,A}$ is uniformly distributed in $Z^*_{N'}$ and $r_{2,i,A}$ is uniformly distributed in Z^*_N, we know that $(e, M''_{1,B})$ is identically distributed with $(e', M'_{1,B})$ and hence $(e, M''_{1,B})$ is computationally indistinguishable from $(e', M'_{1,B})$. Besides, because the Paillier cryptosystem is semantically secure, $(e, M''_{1,B})$ and $(e, M_{1,B})$ are computationally indistinguishable. So $(e', M'_{1,B})$ is also computationally indistinguishable from $(e, M_{1,B})$.

To simulate the messages $M_{2,B} = (\beta_{1,A}, \ldots, \beta_{p,A})$, note that before Alice and Bob exchange their shares of β_i, Bob has the share $\beta_{i,B}$ such that $(\beta_{i,A} + \beta_{i,B}) \bmod N = \beta_i$. $\beta_{i,B}$ is computed as $\beta_{i,B} = h_{i,B}(1^K, X_2, Y_2, r_B, e, M_{1,B})$ for some function $h_{i,B}$. The simulator computes $\beta'_{i,B} = h_{i,B}(1^K, X_2, Y_2, r'_B, e', M'_{1,B})$, which is indistinguishable from $\beta_{i,B}$. Since the simulator is given the final result β_i, it can compute $\beta'_{i,A} = (\beta_i - \beta'_{i,B}) \bmod N'$, which is indistinguishable from $\beta_{i,A}$. Let $M'_{2,B} = (\beta'_{1,A}, \ldots, \beta'_{p,A})$. It is indistinguishable from $M_{2,B}$.

Let $M'_B = (1^K, X_2, Y_2, r'_B, e', M'_{1,B}, M'_{2,B})$. From the above discussion we know that M'_B is computationally indistinguishable from Bob's view $VEIW_B$. So we can simulate Bob's view using his inputs $(1^K, X_2, Y_2)$ and his output β.

□

4.7.2 Multi-Party Cases

We now consider the multi-party cases. Suppose that the regression dataset (X, Y) is distributed among m parties, $X = X_1 + \cdots + X_m$ and $Y = Y_1 + \cdots + Y_m$, where party l holds (X_l, Y_l). We assume that the attribute values held by different parties don't overlap. If party l doesn't know a attribute value, he/she simply sets the corresponding entry in (X_l, Y_l) as 0. These m parties wish to conduct multiple linear regression on the joint dataset (X, Y) so that all of them obtain the regression coefficients β, but they don't want to disclose their confidential data to each other. A secure multi-party protocol is necessary to achieve this goal.

I present a multi-party privacy preserving multiple linear regression protocol in Protocol 4.8. We assume that two parties, for example, party 1 and party 2, will never collude and we reduce the multi-party protocol to the two-party case.

Note that after the execution of line 13, $X = X_1 + X_2 \pmod N$ and $Y = Y_1 + Y_2 \pmod N$. Protocol 4.8 invokes Protocol 4.7 and computes β correctly. Here we assume that Party 1 and Party 2 never collude. The original definition of secure multi-party computation (Definition 2.2) requires the collusion of any subset of parties should not disclose any information except what can be inferred from their own inputs and outputs. Protocol 4.8 is not secure according to definition 2.2 because if Party

Protocol 4.8 Privacy preserving multi-party multiple linear regression protocol

Input: the regression dataset (X, Y), where $X \in R^{n \times p}$ and $Y \in R^n$, is distributed among m parties, $X = X_1 + \cdots + X_m$ and $Y = Y_1 + \cdots + Y_m$, Parties l holds $(1^K, X_l, Y_l)$, where K is the security parameter.

Output: All the parties obtain the regression coefficients β.

1: Party 1 generates a pair of keys $(e, d) = G(1^K)$ and sends the public key e to all other parties.
2: **for** each party $l \neq 1, 2$ **do**
3: **for** each entry $X_l[i, j]$ in X_l **do**
4: Party l selects a random number r in Z_N.
5: Party l sends r to Party 1 and Party 1 sets $X_1[i, j] = (X_1[i, j] + r) \bmod N$.
6: Party l sends $(X_l[i, j] - r) \bmod N$ to Party 2 and Party 2 sets $X_2[i, j] = (X_2[i, j] + X_l[i, j] - r) \bmod N$.
7: **end for**
8: **for** each entry $Y_l[i]$ in Y_l **do**
9: Party l selects a random number r in Z_N.
10: Party l sends r to Party 1 and Party 1 sets $Y_1[i] = (Y_1[i] + r) \bmod N$.
11: Party l sends $(Y_l[i] - r) \bmod N$ to Party 2 and Party 2 sets $Y_2[i] = (Y_2[i] + Y_l[i] - r) \bmod N$.
12: **end for**
13: **end for**
14: Party 1 and Party 2 run Protocol 4.7 on $(1^K, X_1, Y_1)$ and $(1^K, X_2, Y_2)$ and both parties obtain the regression coefficients β.
15: Party 1 sends the regression coefficients β to all other parties.

1 and Party 2 collude, they will know all the data. The definition of secure multi-party computation can be modified to make the explicit assumption that two specific parties never collude.

Definition 4.1. *(Modified from (Goldreich, 2004)) Let* $f : (\{0,1\}^*)^m \to (\{0,1\}^*)^m)$ *be an* m*-ary functionality where* $f_i(x_1, \ldots, x_m)$ *denotes the* i*-th element of* $f(x_1, \ldots, x_m)$*. For* $I = \{i_1, \ldots, i_t\} \subseteq \{1, \ldots, m\}$*, we let* $f_I(x_1, \ldots, x_m)$ *denote the subsequence of* $f_{i_i}(x_1, \ldots, x_m), \ldots, f_{i_t}(x_1, \ldots, x_m)$*. Let* Π *be an* m*-party protocol for computing* f*. We assume that each party is supplied with a security parameter* 1^K *besides his/her input* x_i*. Define the view of the* i*-th party during an execution of* Π *on* $x = (x_1, \ldots, x_m)$ *as* $VIEW_i^{\Pi}(1^K, x) = (1^K, x_i, r_i, m_1, \ldots, m_{s_i})$*, where* r_i *represents the outcome of the* i*-th party internal coin toss and* m_j *represents the* j*-th message he/she has received. For* $I = \{i_1, \ldots, i_t\}$*, let* $VIEW_I(1^K, x) = (I, VIEW_{i_1}(1^K, x), \ldots, VIEW_{i_t}(1^K, x))$*.*

We say that Π *privately computes* f *if there exists a probabilistic polynomial time algorithm, denote by* S*, such that for every* $I \subseteq \{1, \ldots, m\}$ *that doesn't include both* 1 *and* 2*, it holds that*

$$(S(I, 1^K, (x_1, \ldots, x_i), f_I(x)), f(x)) \simeq (VIEW_I(1^K, x), \Pi(x))$$

where $\Pi(x)$ *denotes the output sequence of all parties during the execution and* $\simeq$ *means computational indistinguishability.*

Theorem 4.3. *Multi-party protocol 4.8 privately computes* β *according to definition 4.1.*

Proof. Suppose that S is any subset of parties that doesn't include both party 1 and 2. For Party $l \in S$, if $l \neq 1, 2$, what he/she receives is the public key e and the regression coefficients. We simply compute $e' = G_1(1^K)$ to simulate the message e. If Party 1 is in the subset S, he/she receives a message for each entry in the matrix X and the vector Y from each of the other parties at the beginning. These message are just random numbers, so we can choose random numbers to simulate them. After the execution of line 13, Party 1 has (X_1, Y_1) and Party 2 has (X_2, Y_2) such that $X = X_1 + X_2 \pmod N$ and $Y = Y_1 + Y_2 \pmod N$. Then Party 1 and Party 2 invoke Protocol 4.7 on (X_1, Y_1) and (X_2, Y_2). Because Protocol 4.7 is secure in the semi-honest model, we can simulate those messages Party 1 receives during the invocation of Protocol 4.7, based on his/her inputs X_1 and Y_1 and his/her output β. So we can simulate all the messages Party 1 receives during the execution of Protocol 4.8. Similarly, if Party 2 is in S, we can simulate all the messages Party 2 receives

during the execution of Protocol 4.8. So Protocol 4.8 privately computes β according to Definition 4.1.

□

4.8 Experimental Results

The goal of this experimental analysis is to access the practicality and the accuracy of the privacy preserving two-party multiple linear regression protocol. I have implemented the protocol in C++ based on the Paillier cryptosystem and tested its performance on benchmark datasets. I used the GMP libary (Torbjorn Granlund et al.) for big integers. I ran the experimental study on two separating computers both with Intel Pentium 4 CPU (3.2GHz) and the Linux operating system. These computers are in a network connected by 100Mps Ethernet with average message latency less 1ms. I

In Protocol 4.7, Alice generates a pair of keys for each running of the protocol. In the experiments, the pair of keys were fixed so that I could use the precomputation techniques presented in section 3.2. The key security parameter I used was $K = 512$.

I used benchmark datasets from the UCI machine learning repository (Frank and Asuncion). Specifically, I used 5 datasets to test the privacy preserving linear regression protocol: Housing, Auto-mpg, Servo, Computer hardware and Slump datasets. I assume that the datasets are vertically partitioned between Alice and Bob, each running a computer. The execution time for other partitions are similar to vertical partitions. I describe these datasets and their partitions in Table 4.1, in which n is the number of observations and p is the number of explanatory attributes. Alice holds the first p_1 attributes and Bob holds the last p_2 attributes. In addition, I used an additional explanatory attribute with fixed values $1s$ to accommodate the constant factor in the model 4.2 and assume that Alice holds this attribute. I assume that the response attribute is held by Bob. The Slump dataset consists of 7 explanatory attributes and 3 response attributes. I only report the regression results on the first response attribute. The results on the other two response attributes are similar.

I tested the privacy preserving linear regression protocols on these 5 datasets. I report the execution time in Table 4.2. I also implemented a Matlab program to compute the regression coefficients based on the Householder transformation. The running times of the Matlab program on all these 5 datasets are less than 0.1 second. The regression coefficients computed by the secure protocol are the same as those computed by the Matlab program.

Table 4.1: Benchmark datasets for linear regression

Dataset	n	p	p_1	p_2
Housing	506	13	6	7
Auto-mpg	392	7	3	4
Servo	167	4	2	2
Slump	103	7	3	4
Computer hardware	209	6	3	3

Table 4.2: Experimental results of privacy preserving linear regression on benchmark datasets

Dataset	Housing	Auto-mpg	Servo	Slump	Computer hardware
Running Time	625s	229s	75s	106s	126s

I then tested how the execution time scales with the number of observations in the datasets. I report the results on the housing and auto-mpg datasets in Figure 4.1 and Figure 4.2. The dataset of size n consists of the first n observations in the whole datasets. We observe that the running time scales linearly with the size of the dataset.

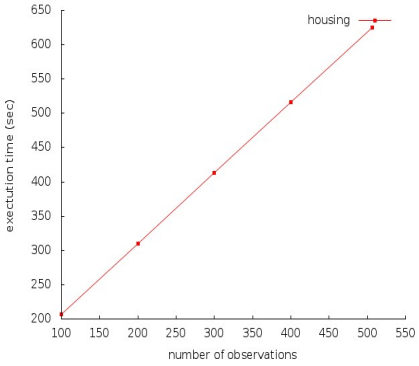

Figure 4.1: Scalability of privacy preserving multiple linear regression with respect to the number of observations (Housing)

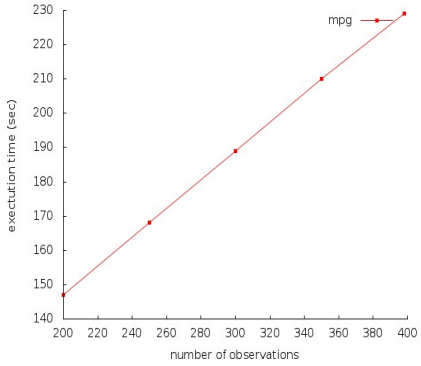

Figure 4.2: Scalability of privacy preserving multiple linear regression with respect to the number of observations (Auto-mpg)

I then tested how the execution time scales with the number of attributes. I used the Housing dataset. To observe the asymptotic behavior, I assembled two copies of the Housing datasets into one which has 506 observations and 28 explanatory

attributes. Note that the explanatory attributes in the new dataset are dependent. The regression coefficients in this model may not be useful and the computation may not be accurate. Now I used a subset of the generated dataset to test the asymptotic behavior. The subset with p attributes consist of the first p explanatory attributes and the additional response attribute. I assume that the explanatory attributes are evenly distributed between Alice and Bob. I report the results in Figure 4.3.The line in Figure 4.3 shows that the execution time is super-linear in the number of explanatory attributes. The execution time is expected to exhibit quadratic asymptotic behavior with larger numbers of attributes.

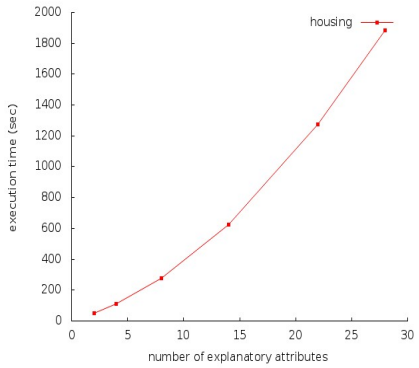

Figure 4.3: Scalability of privacy preserving multiple linear regression with respect to the number of attributes (Housing)

As one of the conclusions, our experiments demonstrated that the proposed privacy preserving multiple linear regression protocol have practical potential. The current proof-of-concept implementation allowed us to handle instances with hundreds of observations and tens of attributes. However, new techniques are needed to design practical protocols for larger datasets.

Chapter 5

Privacy Preserving *EM* Clustering

5.1 Introduction

Expectation maximization (*EM*) is an iterative method for finding unknown parameters in statistical models which depend on unobserved latent variables. Since Dempster, Laird, and Rubin published their classic paper on the *EM* algorithm in 1977, it has become a popular method in the AI and statistics communities. Particularly, the *EM* algorithm is frequently used for data clustering in data mining and machine learning. *EM* clustering can often achieve better clustering performances than other clustering methods such as k-means.

In many situations, the data to be clustered are distributed among several parties. To achieve more accurate clustering results, these parties wish to use the *EM* algorithm to cluster their joint dataset. However, due to privacy concerns, they may not be willing to disclose their confidential data. In this chapter, we propose a privacy preserving two-party *EM* clustering protocol which allows two parties to jointly perform *EM* clustering without disclosing their individual data. The protocol is based on the additive secret-sharing scheme using homomorphic encryption. This protocol works for arbitrarily partitioned datasets. and discloses only the number of iterations besides the clustering results. The existing secure protocol on *EM* clustering considers only the scenario where the datasets are horizontally partitioned among multiple parties (Lin et al., 2005) and it discloses the means and the covariance matrix, which may contain sensitive information.

The organization of this chapter is as follows. Section 5.2 introduces the *EM* clustering method. Section 5.3 discusses previous work on privacy preserving *EM* clustering. Section 5.4 presents two basic cryptographic building blocks, secure logarithm and secure exponential function. Section 5.5 presents the privacy preserving

two-party EM clustering protocol and gives a formal proof about its security. Section 5.6 presents experimental results on benchmark datasets.

5.2 EM Clustering

Suppose that a statistical model consists of a set $X = (x_1, \ldots, x_n)^T$ of observed data, a set of unobserved latent data $Z = (z_1, \ldots, z_n)'$, and a vector of unknown parameters θ. We assume that z_i are discrete random variables taking values from $\{1, \ldots, k\}$ and the parameters θ are continuous. The observed data x_i can be discrete or continuous.

Assume that the joint likelihood function of X and Z is

$$L(\theta; X, Z) = p(X, Z|\theta),$$

where p is the probability density function. Then the marginal likelihood function of X is

$$L(\theta; X) = \sum_Z p(X, Z|\theta) \tag{5.1}$$

and the marginal log-likelihood function of X is

$$\log L(\theta; X) = \log \sum_Z p(X, Z|\theta). \tag{5.2}$$

We denote the conditional distribution of z_i given x_i and θ by

$$s_{ij} = p(z_i = j|x_i, \theta).$$

The maximum likelihood estimate (MLE) method aims to find the parameters θ which maximize the marginal likelihood function 5.1 or equivalently the marginal log-likelihood function 5.2. This can be achieved by taking the derivative of the marginal log-likelihood function. However, it is usually difficult and very often impossible to obtain an analytical solution.

The EM algorithm is an iterative method to find the maximum likelihood estimates of the unknown parameters θ. It alternates with two steps. In the E step, we compute the conditional distribution of Z given X and the current estimates of parameters $\theta^{(t)}$,

$$s_{ij}^{(t)} = p(z_i = j|x_i; \theta^{(t)}),$$

and construct a function

$$Q(\theta, \theta^{(t)}) = E_{Z|X,\theta^{(t)}}(\log L(\theta; X, Z)). \tag{5.3}$$

This function is a lower bound of the marginal log-likelihood function $\log L(\theta; X)$. In the M step, we find the parameters θ which maximize $Q(\theta, \theta^{(t)})$. It is often likely to obtain an analytic solution to maximize equation 5.3. It can be proved that the EM algorithm converges and finds a maxima of the marginal log-likelihood function. The readers are referred to the textbook by Mclachlan and Krishnan (2008) for a detailed account for the EM algorithm.

One of the important applications of the EM algorithm is data clustering. Suppose that we have a sample of n independent observations from a mixture of k multivariate normal distributions of dimension q, $X = (x_1, \ldots, x_n)^{\mathrm{T}}$. Let $Z = (z_1, z_2, \ldots, z_n)^{\mathrm{T}}$ be the latent variables which determine the populations from which these observations are generated. We assume that $p(z_i = j) = \tau_j$ for $j = 1, \ldots, k$. That is, the percentage of these observations coming from the j-th population is τ_j. We also assume that

$$x_i|(z_i = j) \sim \mathcal{N}_q(\mu_j, \Sigma_j).$$

So the conditional distribution of x_i given $z_i = j$ has density function

$$f(x_i; \mu_j, \Sigma_j) = \frac{1}{\sqrt{(2\pi)^q |\Sigma_j|}} \exp(-\frac{1}{2}(x_i - \mu_j)^{\mathrm{T}} \Sigma_j^{-1} (x_i - \mu_j)) \tag{5.4}$$

where $\mu_j \in R^q$ is the mean vector of the j-th normal distribution and $\Sigma_j \in R^{q \times q}$ is the covariance matrix.

In this model, the unknown parameters are

$$\theta = (\tau_1, \ldots, \tau_k, \mu_1, \ldots, \mu_k, \Sigma_1, \ldots, \Sigma_k),$$

which represent the means and covariances of the normal distributions and the "mixing" values between them. Given these parameters θ, the conditional distribution of z_i given x_i, $s_{ij} = p(z_i = j|x_i, \theta)$, is the probability that the point x_i is generated from the j-th population. To cluster these n observations into k groups, we can use the EM algorithm to estimate the parameters θ and compute s_{ij} from the observed data X. The values of s_{ij} provide a soft clustering of these observations. To obtain a hard clustering, we assign each observation x_i to cluster j such that s_{ij} is the largest among all s_{il} for $l = 1, \ldots, k$.

Note that in this model the likelihood function of X and Z is:

$$\begin{aligned} L(\theta; X, Z) &= p(X, Z|\theta) \\ &= \prod_{i=1}^{n} p(x_i, z_i|\theta) \\ &= \prod_{i=1}^{n} p(z_i|\theta) p(x_i|z_i; \theta) \\ &= \prod_{i=1}^{n} \sum_{j=1}^{k} I(z_i = j) p(z_i = j|\theta) p(x_i|z_i = j; \theta) \\ &= \prod_{i=1}^{n} \sum_{j=1}^{k} I(z_i = j) \tau_j f(x_i; \mu_j, \Sigma_j) \end{aligned} \tag{5.5}$$

where I is the indicator function and f is the density function of normal distribution as defined in equation 5.4. The log-likelihood function is

$$\log L(\theta; X, Z) = \sum_{i=1}^{n} \sum_{j=1}^{k} I(z_i = j)[\log \tau_j - \frac{q}{2} \log(2\pi) - \frac{1}{2} \log |\Sigma_j| - \frac{1}{2}(x_i - \mu_j)^{\mathrm{T}} \Sigma_j^{-1} (x_i - \mu_j)]. \tag{5.6}$$

The marginal likelihood function of X is

$$\begin{aligned} L(\theta; X) &= \sum_{Z} L(\theta; X, Z) \\ &= \sum_{Z} p(X, Z|\theta) \\ &= \sum_{Z} \prod_{i=1}^{n} p(x_i, z_i|\theta) \\ &= \prod_{i=1}^{n} \sum_{j=1}^{k} p(x_i, z_i = j|\theta) \\ &= \prod_{i=1}^{n} \sum_{j=1}^{k} p(z_i = j|\theta) p(x_i|z_i = j; \theta) \\ &= \prod_{i=1}^{n} \sum_{j=1}^{k} \tau_j f(x_i; \mu_j, \Sigma_j). \end{aligned} \tag{5.7}$$

The marginal log-likelihood function of X is

$$\begin{aligned}\log L(\theta; X) &= \sum_{i=1}^{n} \log \sum_{j=1}^{k} \tau_j f(x_i; \mu_j, \Sigma_j) \\ &= \sum_{i=1}^{n} \log \sum_{j=1}^{k} \tau_j \frac{1}{\sqrt{(2\pi)^q |\Sigma_j|}} \exp(-\frac{1}{2}(x_i - \mu_j)^{\mathrm{T}} \Sigma_j^{-1} (x_i - \mu_j))\end{aligned} \tag{5.8}$$

We now use the EM algorithm to estimate the parameters θ, which we can use to cluster the n observations $X = (x_1, \ldots, x_n)^{\mathrm{T}}$ into k groups. In the E step, given the current estimate of parameters

$$\theta^{(t)} = (\tau_1^{(t)}, \ldots, \tau_k^{(t)}, \mu_1^{(t)}, \ldots, \mu_k^{(t)}, \Sigma_1^{(t)}, \ldots, \Sigma_k^{(t)}),$$

the conditional distribution z_i given x_i is computed using the Bayes' formula

$$s_{ij}^{(t)} = p(z_i = j | x_i; \theta^{(t)}) = \frac{\tau_j^{(t)} \ f(x_i; \mu_j^{(t)}, \Sigma_j^{(t)})}{\sum_{l=1}^{k} \tau_l^{(t)} f(x_i; \mu_l^{(t)}, \Sigma_l^{(t)})}. \tag{5.9}$$

The corresponding Q function (defined in equation 5.3) is:

$$Q(\theta, \theta^{(t)}) = \sum_{i=1}^{n} \sum_{j=1}^{k} s_{ij}^{(t)} [\log \tau_j - \frac{q}{2} \log(2\pi) - \frac{1}{2} \log |\Sigma_j| - \frac{1}{2}(x_i - \mu_j)^{\mathrm{T}} \Sigma_j^{-1} (x_i - \mu_j)] \tag{5.10}$$

In the M step, we find the parameters θ which maximize $Q(\theta, \theta^{(t)})$. The solutions are:

$$\tau_j^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} s_{ij}^{(t)} \tag{5.11}$$

$$\mu_j^{(t+1)} = \frac{\sum_{i=1}^{n} s_{ij}^{(t)} x_i}{\sum_{i=1}^{n} s_{ij}^{(t)}} \tag{5.12}$$

$$\Sigma_j^{(t+1)} = \frac{\sum_{i=1}^{n} s_{ij}^{(t)} (x_i - \mu_j^{(t+1)})(x_i - \mu_j^{(t+1)})^{\mathrm{T}}}{\sum_{i=1}^{n} s_{ij}^{(t)}} \tag{5.13}$$

The EM algorithm alternates with the E and M steps. In the implementation of this algorithm, we first initialize s_{ij}, the conditional distribution of z_i given x_i. This can be achieved by an initial clustering of these n observations into k groups. For each observation i, we can set $s_{ij}^{(1)} = 1$ if we assign observation i to cluster j and

set $s_{il}^{(1)} = 0$ for all $l \neq j$. We can perform the initial clustering in several ways. We may assign each data point to a cluster randomly, or we may randomly choose k data points as the initial cluster centers and assign each point to the closest center. We may even run other clustering algorithms such as k-means on the dataset and use the clustering results as the initial clustering of the EM algorithm.

The EM algorithm is used to maximize the marginal log-likelihood function. The natural stopping criterion is to check whether the difference of the marginal log-likelihoods between two successive iterations is sufficiently small, that is, to check whether the following condition holds,

$$|\log L(\theta^{(t+1)}; X) - \log L(\theta^{(t)}; X)| < \epsilon \tag{5.14}$$

where ϵ is some predetermined threshold and $\log L(\theta; X)$ is as defined in equation 5.8. Alternatively, we can check the condition on the relative difference

$$|\frac{\log L(\theta^{(t+1)}; X) - \log L(\theta^{(t)}; X)}{L(\theta^{(t+1)}; X)}| < \epsilon. \tag{5.15}$$

After the loop with the E and M steps terminates, we have the estimates of the parameters θ. W can compute $s_{ij} = p(z_i|x_i, \theta)$, the conditional distribution of z_i given x_i, according to equation 5.9. They provide a soft clustering of these n observations. As a final step, we compute hard clustering results and assign each observation x_i to cluster j such that s_{ij} is the largest among all s_{il} $(l = 1, \ldots, k)$.

The EM clustering algorithm is summarized in Algorithm 5.1.

Algorithm 5.1 EM clustering

Input: a dataset of n observations $X = (x_1, \ldots, x_n)^{\mathrm{T}} \in R^{n \times q}$.
Output: cluster these n observations into k groups.

initialize the conditional distribution $s_{ij}^{(1)}$ of z_i given x_i.
for $t = 1$ to MAX **do**
  M step: compute $\tau_j^{(t+1)}, \mu_j^{(t+1)}, \Sigma_j^{(t+1)}$ as in equations (5.11),(5.12) and (5.13).
  E step: compute $s_{ij}^{(t+1)}$ as in equation (5.9).
  compute the marginal log-likelihood $\log L(\theta^{(t+1)}; X)$ as in equation (5.8).
  if $|\log L(\theta^{(t+1)}; X)/n - \log L(\theta^{(t)}; X)/n| < \epsilon$ **then**
    break
  end if
end for
compute the final clustering results.

5.3 Previous Work

Lin et al. (2004) considered the scenario where the datasets are horizontally partitioned among multiple parties. Suppose that there are m ($m > 2$) parties which jointly hold a dataset of n observations $X = (x_1, \ldots, x_n)^{\mathrm{T}}$. Each party h ($1 \leq h \leq m$) owns n_h observations ($\sum_{h=1}^{m} n_h = n$). Lin et al. proposed a privacy preserving EM clustering protocol which allows these m parties to securely perform EM clustering. I briefly discuss their protocol below.

Note that in the M-step, the parameters are updated as in equations (5.11), (5.12) and (5.13). To obtain global estimates $\tau_j^{(t+1)}$, $\mu_j^{(t+1)}$ and $\Sigma_j^{(t+1)}$, we only need the global values $\sum_{i=1}^{n} s_{ij}^{(t)}$, $\sum_{i=1}^{n} s_{ij}^{(t)} x_i$ and $\sum_{i=1}^{n} s_{ij}^{(t)} (x_i - \mu_j^{(t+1)})(x_i - \mu_j^{(t+1)})^{\mathrm{T}}$. Let x_{ih} ($i = 1, \ldots, n_h$) be the i-th observation held by party h and define

$$
\begin{aligned}
A_{jh} &= \sum_{i=1}^{n_h} s_{ihj}^{(t)} \\
B_{jh} &= \sum_{i=1}^{n_h} s_{ihj}^{(t)} x_{ih} \\
C_{jh} &= \sum_{i=1}^{n_h} s_{ihj}^{(t)} (x_{ih} - \mu_j^{(t+1)})(x_{ih} - \mu_j^{(t+1)})^{\mathrm{T}}.
\end{aligned}
$$

Then we have

$$
\begin{aligned}
\sum_{h=1}^{m} A_{jh} &= \sum_{i=1}^{n} s_{ij}^{(t)} \\
\sum_{h=1}^{m} B_{jh} &= \sum_{i=1}^{n} s_{ij}^{(t)} x_i \\
\sum_{h=1}^{m} C_{jh} &= \sum_{i=1}^{n} s_{ij}^{(t)} (x_i - \mu_j^{(t+1)})(x_i - \mu_j^{(t+1)})^{\mathrm{T}}.
\end{aligned}
$$

A_{jh}, B_{jh}, C_{jh} can be computed locally by each party. To compute the global estimates, we can invoke the following secure sum protocol. Suppose that each party h owns a number v_h and they wish to obtain the sum of these numbers without disclosing their individual numbers to each other. We assume that their sum belong to $[0, N)$. The secure summation proceeds as follows. Party 1 selects a random number $r \in [0, N)$ and then sends $w_1 = r + v_1$ to party 2. Each party h adds its number v_h to the message it received and then sends it to the next party. Party m sends $w_m = w_{m-1} + v_m$ to party 1. Now party 1 computes $w_m - r$, which equals

$\sum_{h=1}^{m} v_h$, and sends this result to all other parties. Using this secure sum procedure, each party h doesn't need to disclose A_{jh}, B_{jh}, C_{jh} and all the parties obtain the global estimates.

After each party h obtains the global parameters $(\tau_j^{(t+1)}, \mu_j^{(t+1)}, \Sigma_j^{(t+1)})$, he/she computes the conditional distribution of z_{ih} given x_{ih}

$$s_{ihj}^{(t+1)} = \frac{\tau_j^{(t+1)} f(x_{ih}; \mu_j^{(t+1)}, \Sigma_j^{(t+1)})}{\sum_{l=1}^{k} \tau_l^{(t+1)} f(x_{ih}; \mu_l^{(t+1)}, \Sigma_l^{(t+1)})}.$$

Lin et al. used the following formula to check the convergence criterion:

$$|\log H(\theta^{(t+1)}; X) - \log H(\theta^{(t)}; X)| < \epsilon$$

where

$$\log H(\theta^{(t)}; X) = \sum_{i=1}^{n} \sum_{j=1}^{k} \log \tau_j^{(t)} f(x_i; \mu_j^{(t)}, \Sigma_j^{(t)})$$

and ϵ is some predetermined threshold. This criterion is not exactly the same as equation 5.14. Note that

$$\log H(\theta^{(t)}; X) = \sum_{h=l}^{m} D_h$$

where

$$D_h = \sum_{i=1}^{n_h} \sum_{j=1}^{k} [\log \tau_j f(x_{ih}; \mu_j^{(t)}, \Sigma_j^{(t)})].$$

So each party h can compute D_h locally. All these parties use secure sum to compute $\log H(\theta; X)$. Each party can check the stopping criterion locally.

In this protocol, the values of individual data items are not disclosed and that no information can be traced to a specific party. However, this protocol discloses the means and the covariance matrix of each population, which may contain sensitive information. If what we want to compute is just the clustering results, then this protocol discloses more information than necessary. Besides, if there are only two parties, then each party will learn the means and the covariance matrix of the other party. Also note that this protocol doesn't work over vertically partitioned datasets. I will develop a privacy preserving EM clustering protocol which works for arbitrarily partitioned datasets.

5.4 Subprotocols

In this section, I present two secure protocols, secure logarithm and secure exponential function, which I use in the design of privacy preserving EM clustering protocol.

5.4.1 Secure Logarithm

Suppose that a positive real numbers x, represented as $[x] = \lfloor x2^P \rfloor$ in the plaintext domain, is split between Alice and Bob, $[x] = x_A + x_B \pmod N$. Alice holds the share x_A and Bob holds the share x_B. They wish to privately compute $\log x$ with the result also split between Alice and Bob.

Lindell and Pinkas (2000) proposed a protocol to securely compute $\log x$. Let 2^m be the power of 2 which is closest to x and we write $x = 2^m(1+\epsilon)$ where $-1/2 \leq \epsilon \leq 1/2$. Then

$$\log x = \log(2^m(1+\epsilon)) = m \log 2 + \log(1+\epsilon).$$

The Taylor series for $\log(1+\epsilon)$ is:

$$\log(1+\epsilon) = \sum_{i=1}^{\infty} \frac{(-1)^{i-1}\epsilon^i}{i} = \epsilon - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3} - \frac{\epsilon^4}{4} + \cdots .$$

The truncated error of this series is bounded by

$$|\log(1+\epsilon) - \sum_{i=1}^{l} \frac{(-1)^{i-1}\epsilon^i}{i}| < \frac{|\epsilon|^{l+1}}{l+1} \cdot \frac{1}{1-|\epsilon|}.$$

Lindell and Pinkas used Yao's private circuit evaluation to compute m and ϵ and then used private polynomial evaluation to compute $\log(1+\epsilon)$. Here I give an implementation all based on homomorphic encryption.

I first show how to privately compute m and ϵ. We assume that $[x]$ $(= \lfloor x2^P \rfloor)$ $< 2^L$, where L is known to both parties. Let $x_L \ldots x_1$ be the binary representation of $[x]$. We define a series of real numbers y_i $(1 \leq i \leq L)$ such that $[y_i]$ has binary representation $x_i \ldots x_1$. Note that $[y_i] < 2^i$. For each real number y_i, we define m_i and ϵ_i such that 2^{m_i} is the power of 2 closest to y_i and $y_i = 2^{m_i}(1+\epsilon_i)$. Note that $x = y_L$, $m = m_L$ and $\epsilon = \epsilon_L$.

We can compute m_i and ϵ_i iteratively from $i = 1$ to L. Note that when $x_i = 0$, $y_i = y_{i-1}$, so $m_i = m_{i-1}$ and $\epsilon_i = \epsilon_{i-1}$. We now consider the case when $x_i = 1$. In this case we know that $[y_i] = 2^{i-1} + [y_{i-1}]$. If $x_{i-1} = 0$, $[y_{i-1}] < 2^{i-2}$ and $0 \leq [y_{i-1}]/2^{i-1} <$

1/2. Since

$$\begin{aligned} y_i &= [y_i]/2^P \\ &= (2^{i--1} + [y_{i-1}])/2^P \\ &= 2^{i-P-1}(1 + [y_{i-1}]/2^{i-1}), \end{aligned}$$

we have

$$\begin{aligned} m_i &= i - P - 1 \\ \epsilon_i &= [y_{i-1}]/2^{i-1} = ([y_i] - 2^{i-1})/2^{i-1} \\ [\epsilon_i] &= 2^P([y_i] - 2^{i-1})/2^{i-1}. \end{aligned}$$

If $x_{i-1} = 1$, then $2^{i-2} \leq [y_{i-1}] < 2^{i-1}$ and $-1/2 < ([y_{i-1}] - 2^{i-1})/2^i < 0$. Note that

$$\begin{aligned} y_i &= [y_i]/2^P \\ &= (2^{i-1} + [y_{i-1}])/2^P \\ &= (2^i + [y_{i-1}] - 2^{i-1})/2^P \\ &= 2^{i-P}(1 + ([y_{i-1}] - 2^{i-1})/2^i). \end{aligned}$$

We have

$$\begin{aligned} m_i &= i - P \\ \epsilon_i &= ([y_{i-1}] - 2^{i-1})/2^i \\ &= ([y_i] - 2^{i-1} - 2^{i-1})/2^i \\ &= ([y_i] - 2^i)/2^i \\ [\epsilon_i] &= 2^P([y_i] - 2^i)/2^i. \end{aligned}$$

I present secure protocol to compute m and ϵ in Protocol 5.2. For convenience, we set $m = -(P + 1)$ and $\epsilon = 0$ when $x = 0$.

In Protocol 5.2, m is an integer and ϵ is a real number. Lines 5 and 6 consider the case when $x_i = 1$ and $x_{i-1} = 0$. Lines 7 and 8 consider the case when $x_i = 1$ and $x_{i-1} = 1$. Lines 9 and 10 combine these two cases to compute m_i and ϵ_i. Lines 11-12 combine the case when $x_i = 1$ with the case when $x_i = 0$. In lines 6 and 8 we use $[x]$ instead of $[y_i]$. Note that $[x]$ is equal to $[y_i]$ when i is the largest index such that $x_i = 1$. For this index i, line 10 computes ϵ_i correctly and line 12 multiplies ϵ_{i-1} with 0. So only the computation in the i-th iteration matters and the protocol computes

Protocol 5.2 Secure computation of m and ϵ such that $x = 2^m(1+\epsilon)$ and $-1/2 < \epsilon < 1/2$

Input: a positive real number x such that $[x] < 2^L$ is split between Alice and Bob, $[x] = x_A + x_B \pmod N$.

Output: an integer m and a real number ϵ such that 2^m is the power of 2 closest to x and $x = 2^m(1+\epsilon)$. m and ϵ are also secrets split between Alice and Bob.

1: Alice and Bob use Protocol 3.2 to securely transform $[x]$ into its binary representation $x_L \dots x_1$ whose bits x_i are split between Alice and Bob.
2: $m_0 = -(P+1)$
3: $\epsilon_0 = 0$
4: **for** $i = 1$ to L **do**
5: $\quad u_1 = i - P - 1$
6: $\quad \eta_1 = 2^P([x] - 2^{i-1})/2^{i-1}$
7: $\quad u_2 = i - P$
8: $\quad \eta_2 = 2^P([x] - 2^i)/2^i$
9: $\quad m_i = (1 - x_{i-1}) \cdot u_1 + x_{i-1} \cdot u_2$
10: $\quad [\epsilon_i] = (1 - x_{i-1}) \cdot \eta_1 + x_{i-1} \cdot \eta_2$
11: $\quad m_i = (1 - x_i) \cdot m_{i-1} + x_i \cdot m_i$
12: $\quad [\epsilon_i] = (1 - x_i) \cdot [\epsilon_{i-1}] + x_i \cdot [\epsilon_i]$
13: **end for**$\{m = m_L,\ \epsilon = \epsilon_L\}$

m and ϵ correctly.

The operations in Protocol 5.2 include addition, multiplication and division with public divisor. So we can use secure multiplication (Protocol 2.1) and secure division with public divisor (Protocol 4.2) to implement Protocol 5.2.

Once we know ϵ, we can use Protocol 5.3 to privately compute $\log(1+\epsilon)$. In Protocol 5.3, l is the fixed number of iterations. Note that line 4 is the multiplication of two real numbers. We need to use secure multiplication of real numbers. We can implement line 5 using secure division with public divisor (Protocol 4.3).

Protocol 5.4 combines Protocol 5.2 and 5.3 to privately compute $\log x$.

5.4.2 Secure Exponential Function

In the development of the privacy preserving EM clustering protocol, we need a protocol to securely compute $y = \exp x$ when the real number $x \le 0$ is a secret split between Alice and Bob. Note that the Taylor series for the exponential function is

$$\exp x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots .$$

Protocol 5.3 Secure protocol for computing $\log(1+\epsilon)$ $(-1 < \epsilon < 1)$

Input: a secret real number ϵ such that $-1 < \epsilon < 1$ is split between Alice and Bob, $[\epsilon] = \epsilon_A + \epsilon_B \pmod{N}$.

Output: Alice and Bob obtain their respective shares of $s = \log(1+\epsilon)$.

1: $s = 0$
2: $t = 1$
3: **for** $i = 1$ to l **do**
4: $\quad t = t\epsilon$
5: $\quad r = t/i$
6: $\quad$ **if** $i \mod 2 == 1$ **then**
7: $\quad\quad s = s + r$
8: $\quad$ **else**
9: $\quad\quad s = s - r$
10: $\quad$ **end if**
11: **end for**

Protocol 5.4 Secure logarithm

Input: a secret real number $x \geq 0$ is split between Alice and Bob.

Output: Alice and Bob obtain their respective shares of $y = \log x$.

1: Alice and Bob use Protocol 5.2 to privately compute m and ϵ such that $x = 2^m(1+\epsilon)$ and $-1/2 < \epsilon < 1/2$.
2: Alice and Bob use Protocol 5.3 to privately compute $s = \log(1+\epsilon)$.
3: $t = m \log 2$
4: $y = s + t$

We can truncate the series up to the l-th place to approximate the exponential function and then use secure multiplication to privately compute the polynomial. However, when the secret number x has large absolute value, the series converges slowly. As we will see shortly, we will use the secure exponential protocol to compute $r = \sum_{i=q} \exp x_i$, where $x_i \leq 0$ $(i = 1, \ldots, q)$ and one of x_i is 0. Since $r \geq 1$, we can ignore sufficiently small terms. So if x is negative and sufficiently small, for example $x \leq -32$, we simply approximate it with 0.

We then compute $z = x/64$. Now that $|z| < 1/2$, the Taylor series on $\exp z$ will converge faster. Once we have $s = \exp z$,

$$y = \exp x = \exp(64z) = (\exp z)^{64} = s^{64}.$$

I present the secure exponential protocol in Protocol 5.5. The multiplications in lines 6 and 12 refer to real numbers and we need to use secure multiplication of real numbers. We can implement line 7 using secure division with public divisor.

Protocol 5.5 Secure exponential function for non-positive real numbers

---

Input: a secret real number $x \leq 0$ is split between Alice and Bob, $[x] = x_A + x_B \pmod{N}$.

Output: Alice and Bob obtain their respective shares of $y = \exp x$.

1: Alice and Bob use Protocol 3.1 to privately compare x and -32. Let $\alpha = 1$ if $x \leq -32$ and 0 otherwise.
2: Alice and Bob use Protocol 4.2 to securely compute $z = x/64$.
3: $s = 1$
4: $t = 1$
5: **for** $i = 1$ to l **do**
6: $\quad t = tz$
7: $\quad t = t/i$
8: $\quad s = s + t$
9: **end for**
10: $y = s$
11: **for** $i = 1$ to 6 **do**
12: $\quad y = y^2$
13: **end for**
14: $y = (1 - \alpha)y$

I present another secure exponential protocol which computes $z = x^y$ when $x > 0$ and $y \geq 0$ are both secret integers split between Alice and Bob. I will not use this protocol in the development of privacy preserving EM clustering protocol.

Let $y_L y_{L-1} \ldots\ldots y_1$ be the binary representation of y. Then

$$y = y_L 2^{L-1} + \cdots + y_i 2^{i-1} + \cdots + y_1,$$

and we have the equality:

$$\begin{aligned} z &= x^y \\ &= x^{(y_L 2^{L-1} + \cdots + y_i 2^{i-1} + \cdots + y_1)} \\ &= x^{(y_L 2^{L-1})} \ldots x^{(y_i 2^{i-1})} \ldots x^{y_1}. \end{aligned}$$

Note that when $y_i = 1$, $x^{(y_i * 2^{i-1})} = x^{2^{i-1}}$; otherwise $x^{(y_i 2^{i-1})} = 1$. So

$$x^{(y_i 2^{i-1})} = y_i x^{2^{i-1}} + (1 - y_i).$$

Let $t_i = x^{2^{i-1}}$. Then $t_{i+1} = t_i^2$. I present the secure exponential protocol in Protocol 5.6.

Protocol 5.6 Secure exponential function for nonnegative integers

Input: two secret integers $x > 0$ and $y \geq 0$ are split between Alice and Bob.
Output: Alice and Bob obtain their respective shares of $z = x^y$.

1: Alice and Bob use Protocol 3.2 to securely transform y into its binary representations $y_L \ldots y_1$.
2: $z = 1$
3: $t = x$
4: **for** $i = 1$ to L **do**
5: $\quad r = y_i t + (1 - y_i)$
6: $\quad z = zr$
7: $\quad t = t^2$
8: **end for**

5.5 Privacy Preserving EM clustering

Suppose that a dataset of n observations with q attributes, $X = (x_1, \ldots, x_n)^{\mathrm{T}} \in R^{n \times q}$, is arbitrarily distributed between Alice and Bob, $X = X_1 + X_2$. Alice owns $X_1 \in R^{n \times q}$ and Bob owns $X_2 \in R^{n \times q}$. They wish to perform the EM algorithm to cluster the joint dataset X into k groups but without disclosing their confidential dataset to each other. The purpose of this section is to present a privacy preserving two-party EM clustering protocol to enable such applications.

The design of secure protocols is based on the specific homomorphic encryption, the Paillier cryptosystem. We use the additive secret-sharing scheme as we present in section 2.3. We assume that both parties are supplied with a key parameter K. Alice has input $(1^K, k, X_1)$ and Bob has input $(1^K, kX_2)$. We assume that Alice has the private (decryption) key and both parties know the public (encryption) key.

As I discuss in section 5.2, the EM clustering algorithm consists of five steps: initialization, M-step, E-step, checking convergence criterion and final clustering. I develop secure protocols for these individual procedures and assemble them together to implement a privacy preserving EM clustering algorithm. Note that the proposed EM protocol works for arbitrarily partitioned datasets, which include vertical partition and horizontal partition as special cases. At the end of this section, I prove that this protocol discloses only the number of iterations.

1. Initialization.

The first step in EM-clustering is to find the initial clustering and initialize s_{ij}, the conditional distribution of z_i given x_i. As I discuss in section 5.2, there are several ways for initialization. Based on different methods, we can implement different privacy preserving initialization procedures accordingly. As an example, we assume that we select k points randomly as the initial cluster centers and then assign each point to the closest cluster. If the point x_i is closest to the j-th cluster center, we set $s_{ij} = 1$ and $s_{il} = 0$ for $l \neq j$. Based on this method, I implement a privacy preserving initialization procedure, which I present in Protocol 5.7.

Protocol 5.7 is implemented as follows. In line 1, Alice sends k indices $(d_1, \ldots, d_k)$ to Bob. Since these indices are randomly selected, they don't disclose any information about Alice. The k points $(x_{d_1}, \ldots, x_{d_k})$ are used as the initial cluster centers. In lines 4 and 6, we use Protocol 3.4 to privately compute the squared distance between the point x_i and the cluster centers x_{d_l}. In line 7 secure comparison (Protocol 3.1) is invoked to privately compare m and d. In line 8, m is assigned the minimum of m and d. We can implement line 8 using secure multiplication.

The vector $(s_{i1}, \ldots, s_{ik})$ records the comparison results of m and $\|x_i, x_{d_l}\|_2^2$. Lines 10-14 scan the vector from the k-th entry to the first entry. Once it encounter a 1, it will set the remaining entries to be 0. So if x_i is closest to cluster j, we have $s_{ij} = 1$ and $s_{il} = 0$ for $l \neq j$. For example, suppose that the squared distances from x_i to the k cluster centers x_{d_l} are $(3, 4, 2, 5)$. After the execution of line 9 the vector $(s_{il}, \ldots, s_{ik})$ is $(1, 0, 1, 0)$. Lines 10-14 sets the vector to be $(0, 0, 1, 0)$.

Protocol 5.7 Privacy preserving initialization in EM clustering

Input: a dataset $X = (x_1, \ldots, x_n)^{\mathrm{T}} \in R^{n \times q}$ is split between Alice and Bob.

Output: initial clustering $s_{ij} \in R$ for $i = 1, \ldots, n$, $j = 1, \ldots, k$, which are secrets split between Alice and Bob.

Alice randomly selects k indices d_l $(l = 1, \ldots, k)$ from $\{1, \ldots, n\}$ and sends these indices to Bob.
for $i = 1$ to n **do**
  $s_{i1} = 1$
  $m = \|x_i, x_{d_1}\|_2^2$
  for $l = 2$ to k **do**
    $d = \|x_i, x_{d_l}\|_2^2$
    Alice and Bob privately compare m and d using Protocol 3.1.
    Let $s_{il} = 1$ if $m \leq d$ and 0 otherwise.
    $m = s_{il} m + (1 - s_{il}) d$
  end for
  $r = s_{ik}$
  for $l = k - 1$ to 1 **do**
    $s_{il} = (1 - r) s_{il}$
    $r = r + s_{il}$
  end for
end for

2. M step.

In the M step we need to compute

$$\tau_j = \frac{1}{n}\sum_{i=1}^{n} s_{ij}$$
$$\mu_j = \frac{\sum_{i=1}^{n} s_{ij}x_i}{\sum_{i=1}^{n} s_{ij}}$$
$$\Sigma_j = \frac{\sum_{i=1}^{n} s_{ij}(x_i-\mu_j)(x_i-\mu_j)^{\mathrm{T}}}{\sum_{i=1}^{n} s_{ij}}$$

The above formulas involve addition, multiplication and division. We can compute them privately using secure multiplication and secure division subprotocols. We can improve the performance significantly using the following optimization techniques:

1. To compute μ_j and Σ_j, we need to divide them by $s = \sum_{i=1}^{n} s_{ij}$. Note that $\mu_j \in R^q$ and $\Sigma_j \in R^{q\times q}$, we need to invoke secure division (q^2+q) times. As secure division is a costly operation we try to reduce its invocation as much as possible. We can invoke secure division to compute $t = 2^M/s$ for some sufficiently large M. Then we multiply t with the entries in μ_j and Σ_j and then divide them by 2^M using secure division with public divisor.

2. In order to compute Σ_j, we need to compute $s_{ij}(x_i-\mu_j)(x_i-\mu_j)^{\mathrm{T}}$. We first compute $w = s_{ij}(x_i-\mu_j)$ and then compute $w(x_i-\mu_j)^{\mathrm{T}}$. This method needs $(q+q^2)$ secure multiplications. If instead we first compute $\Lambda = (x_i-\mu_j)(x_i-\mu_j)^{\mathrm{T}}$ and then compute $s_{ij}\Lambda$, it requires $2q^2$ secure multiplications.

3. The above formulas refer to the operations of real numbers. For each secure multiplication of real numbers, we need to invoke secure multiplication on the representations of real numbers (Protocol 2.1) and then invoke secure division with public divisor (Protocol 4.3) to scale them down accordingly. A more efficient implementation is to first invoke secure multiplication only. We scale the representations of real numbers properly only at the end of the M steps. For example, in the computation of Σ_j, we simply use secure multiplications (Protocol 2.1) to compute $\Psi = \sum_{i=1}^{n}[s_{ij}]([x_i]-[\mu_j])([x_i]-[\mu_j])^{\mathrm{T}}$ and we don't invoke secure division with public divisor at this moment. Then we multiply it with $2^M/\sum_{i=1}^{n}[s_{ij}]$. Only this time we divide them by 2^{M+P} using secure division with public divisor (P is the number of bits to represent the fractional part of real numbers). Similarly, to compute μ_j, we can first compute $\nu_j = \sum_{i=1}^{n}[s_{ij}][x_i]$, multiply it with $2^M/\sum_{i=1}^{n}[s_{ij}]$, and then divide it by 2^M.

4. The computation of μ_j and Σ_j involves scalar-vector multiplication and vector-

vector multiplication. They can be implemented using a series of secure multiplications. However, we don't invoke Protocol 2.1 individually for each multiplication. We encrypt the vectors and the scalar only once and reuse them in the computation of several entries.

Combining all these strategies together, I present a privacy preserving protocol for the M step in Protocol 5.8. We can implement the secure protocol using secure multiplication (Protocol 2.1), secure division with public divisor (Protocol 4.3), secure division (Protocol 4.6). Note that in this protocol, we explicitly deal with the scaling of the representations of real numbers. For multiplication in the protocol, we simply use secure multiplication of integers (Protocol 2.1) and we don't need to invoke secure division with public divisor right after it. The scaling is done explicitly in line 6 and 15.

Protocol 5.8 Privacy preserving M step in EM clustering

Input: a dataset of n observations $X = (x_1, \ldots, x_n)^{\mathrm{T}} \in R^{n \times q}$, a soft clustering $s_{ij} \in R$, all of which are secrets split between Alice and Bob.

Output: $\tau_j \in R$, $\mu_j \in R^q$, $\Sigma_j \in R^{q \times q}$ for $j = 1, \ldots, k$; These results are secrets split between Alice and Bob.

1: **for** $j = 1$ to k **do**
2: $s = \sum_{i=1}^{n} [s_{ij}]$
3: $[\tau_j] = s/n$
4: $r = 2^M/s$
5: $\nu = \sum_{i=1}^{n} [s_{ij}][x_i]$
6: $[\mu_j] = (\nu r)/2^M$
7: $\Psi = 0$
8: **for** $i = 1$ to n **do**
9: $\nu = [x_i] - [\mu_j]$
10: $\xi = [s_{ij}]\nu$
11: $\Phi = \xi \nu^{\mathrm{T}}$
12: $\Psi = \Psi + \Phi$
13: **end for**
14: $\Psi = \Psi r$
15: $[\Sigma_j] = \Psi_j / 2^{M+P}$
16: **end for**

3. E step.

In the E step, we need to compute

$$s_{ij} = \frac{\tau_j f(x_i; \mu_j, \Sigma_j)}{\sum_{l=1}^{k} \tau_j f(x_i; \mu_l, \Sigma_l)} \tag{5.16}$$

where

$$f(x_i, \mu_j, \Sigma_j) = \frac{1}{\sqrt{(2\pi)^q |\Sigma_j|}} \exp\left(-(x_i - \mu_j)^{\mathrm{T}} \Sigma_j^{-1} (x_i - \mu_j)\right). \tag{5.17}$$

The E step can be further divided into four procedures: the Cholesky decomposition of Σ_j, the computation of $-(x_i - \mu_j)\Sigma_j^{-1}(x_i - \mu_j)^{\mathrm{T}}$, the computation of $\log|\Sigma_j|$ and the final computation of s_{ij}. I describe these procedures below and also show how to implement a secure protocol for each procedure.

(a) Cholesky decomposition of Σ_j. We assume that Σ_j is a symmetric definite matrix. In the implementation, we add a regularization term λI for a small $\lambda > 0$ to Σ_j to ensure this condition. The standard Cholekey decomposition factors $\Sigma_j = LL^{\mathrm{T}}$ where L is a lower triangular matrix and it involves square root operation. We use an alternative procedure to perform Cholesky decomposition without using square root operation. This procedure finds a lower triangular matrix $L \in R^{q \times q}$ whose diagonal entries are all $1s$ and a diagonal matrix D such that $\Sigma_j = LDL^{\mathrm{T}}$. Based on this procedure we are able to implement a secure protocol for the Cholesky decomposition using secure multiplication and secure division only. I present this protocol in Protocol 5.9. Note that we use real numbers in the description of the protocol, so we need to use secure multiplication of real numbers.

Protocol 5.9 Cholesky decomposition without square root

Input: a symmetric definite matrix $\Sigma \in R^{q \times q}$ is split between Alice and Bob.

Output: a lower triangular matrix $L \in R^{q \times q}$ with all diagonal entries $1s$ and a diagonal matrix $D \in R^{q \times q}$ such that $LDL^{\mathrm{T}} = \Sigma$.

1: **for** $j = 1$ to q **do**
2: $\quad D_j = \Sigma_{jj} - \sum_{l=1}^{j-1} L_{jk}^2$
3: $\quad$ **for** $i = j + 1$ to q **do**
4: $\quad\quad L_{ij} = (\Sigma_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk})/D_j$
5: $\quad$ **end for**
6: **end for**

(b) Computation of $-(x_i - \mu_j)^{\mathrm{T}}\Sigma_j^{-1}(x_i - \mu_j)$. Suppose that the Cholesky decomposition of Σ_j is $\Sigma_j = LDL^{\mathrm{T}}$. Denote the diagonal entries of D by $D_1, \ldots, D_q$. we know that $\Sigma_j^{-1} = (L^{-1})^{\mathrm{T}} D^{-1} L^{-1}$. We can invert L and D, compute Σ_j^{-1}, and then compute $-(x_i - \mu_j)^{\mathrm{T}}\Sigma_j^{-1}(x_i - \mu_j)$. However, we can use a more efficient procedure. Note that $(x_i - \mu_j)^{\mathrm{T}}\Sigma_j^{-1}(x_i - \mu_j) = (x_i - \mu_j)^{\mathrm{T}}(L^{-1})^{\mathrm{T}} D^{-1} L^{-1}(x_i - \mu_j)$. We can first compute $\nu = L^{-1}(x_i - \mu_j)$ and then compute $\nu^{\mathrm{T}} D^{-1} \nu$. Notice that L^{-1} is a triangular matrix with diagonal entries $1s$ and the matrix-vector multiplication $L^{-1}(x_i - \mu_j)$ is more efficient than the multiplication with a full matrix.

I present the secure protocol to compute $-(x_i-\mu_j)^{\mathrm{T}}\Sigma_j^{-1}(x_i-\mu_j)$ in Protocol 5.10. Note that we use real numbers in the description of Protocol 5.10. So we need to use secure multiplication of real numbers. We can implement line 14 using secure division (Protocol 4.6) and implement the scalar product in line 16 using a series of secure multiplications.

Protocol 5.10 Secure computation of $-(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)$

Input: $x \in R^q, \mu \in R^q$ and the Cholesky decomposition of $\Sigma = LDL^{\mathrm{T}}$, $L \in R^{q\times q}$ is a lower triangular matrix and $D \in R^{q\times q}$ is a diagonal matrix. All the inputs are secrets split between Alice and Bob.

Output: Alice and Bob obtain their respective shares of $r = -(x-\mu)\Sigma^{-1}(x-\mu)$.

{Line 1-10 compute $U = L^{-1}$}
1: $U = I$ { $I \in R^{q\times q}$ is the identity matrix}
2: **for** $i = 1$ to q **do**
3: **for** $j = i+1$ to q **do**
4: **for** $k = 1$ to i **do**
5: $t = U_{ik}L_{ji}$
6: $U_{jk} = U_{jk} - t$
7: **end for**
8: $U_{jk} = U_{jk} - L_{ji}$
9: **end for**
10: **end for**
11: $\nu = x - \mu$
12: $\nu = U\nu$
13: **for** $i = 1$ to q **do**
14: $w_i = \nu_i/D_i$.
15: **end for**
16: $r = -\langle w, \nu\rangle$

(c) Computation of $\log|\Sigma_j|$. As will be clear in next procedure, we need to compute $\log|\Sigma_j|$. Suppose that we have the Cholesky decomposition of Σ_j, $|\Sigma_j| = LDL^{\mathrm{T}}$ and the diagonal entries of D are $(D_1, \ldots, D_q)$. We know that $|\Sigma_j| = \prod_{l=1}^{q} D_l$. In some cases, the diagonal entries D_l are small numbers. If we compute $\prod_{l=1}^{q} D_l$ directly, it may cause underflow problems. So instead we compute

$$\begin{aligned}\log|\Sigma_j| &= \log\prod_{l=1}^{q} D_l \\ &= \sum_{l=1}^{q}\log D_l\end{aligned}$$

Here we need to privately compute the logarithm q times. Since secure computation of the logarithm function is costly, we try to avoid its invocation as much as possible. Note that for each D_l, we can find an integer m_l such that $2^{m_l} \leq D_l < 2^{m_l+1}$. We can write $D_l = 2^{m_l}\delta_l$ where $1 \leq \delta_l < 2$. Now we have

$$\begin{aligned}
\log|\Sigma_j| &= \log\prod_{l=1}^{q} D_l \\
&= \log\prod_{l=1}^{q} 2^{m_l}\delta_l \\
&= \log(\prod_{l=1}^{q} 2^{m_l}\prod_{l=1}^{q}\delta_l) \\
&= \log\prod_{l=1}^{q} 2^{m_l} + \log\prod_{l=1}^{q}\delta_l \\
&= \sum_{l=1}^{q} m_l \log 2 + \log\prod_{l=1}^{q}\delta_l
\end{aligned}$$

Now that $1 \leq \prod_{l=1}^{q}\delta_l < 2^q$ and we only need to invoke a secure logarithm on it.

I first give a protocol to privately compute m_l and δ_l (Protocol 5.11). In Protocol 5.11, P is the number of bits to represent the fractional part of real numbers. We scan the binary representation of $[x]$ from the lowest bits to the highest bits. If the bit we encounter x_i is 1, the current values of m and η are computed in line 4 and in lines 6-10, respectively, and we update them in lines 11 and 12. Otherwise we don't change m and δ. Clearly we can implement Protocol 5.11 using secure multiplication and secure division with public divisor.

I next give a protocol to privately compute $\log|\Sigma_j|$ (Protocol 5.12). We use secure multiplication of real numbers to implement line 6. The values m_l and u are integers. In line 8, Alice computes $\mu_A = \nu_A[\log 2]$ and Bob computes $\mu_B = \nu_B[\log 2]$, where $[\log 2] = \lfloor(\log 2)2^P\rfloor$, ν_A and ν_B are the shares of ν and μ_A and μ_B are the shares of μ.

(d) Computation of s_{ij}. s_{ij} can be computed using equations (5.16), (5.17) directly. However,

$$r_{ij} = -(x_i - \mu_j)\Sigma_j^{-1}(x_i - \mu_j) \tag{5.18}$$

may be a small negative number and the value $\exp r_{ij}$ is close to 0. We may not be able to compute s_{ij} accurately if all of $\exp r_{ij}$ are close to 0. We can employ a

Protocol 5.11 Secure protocol for computing m and δ such that $2^m \leq x < 2^{m+1}$ and $x = 2^m\delta$.

Input: a positive real number x such that $[x] < 2^L$ is split between Alice and Bob, $[x] = x_A + x_B \pmod N$.

Output: an integer m and a real number δ such that $2^m \leq x < 2^m\delta$ and $1 \leq \delta < 2$. m and δ are secrets split between Alice and Bob.

1: Alice and Bob use secure protocol 3.4 to privately transform $[x]$ into its binary representation $x_L \dots x_1$, whose bits x_i are secrets split between Alice and Bob.
2: $m = -P - 1$
3: $\delta = 0$
4: **for** $i = 1$ to L **do**
5: $\quad u = i - 1 - P$
6: $\quad$ **if** $i \leq P$ **then**
7: $\quad\quad \eta = ([x] - 2^{i-1})2^{P-i+1}$
8: $\quad$ **else**
9: $\quad\quad \eta = ([x] - 2^{i-1})/2^{i-1-P}$
10: $\quad$ **end if**
11: $\quad m = x_i u + (1 - x_i)m$
12: $\quad \delta = x_i\eta + (1 - x_i)\delta$
13: **end for**

Protocol 5.12 Secure protocol for computing $\log|\Sigma|$ when Σ is a positive definite matrix

Input: Cholesky decomposition of Σ, $\Sigma = LDL^{\mathrm{T}}$. L and D are secrets split between Alice and Bob.

Output: Alice and Bob obtain their respective shares of $r = \log|\Sigma|$.

1: $u = 0$
2: $v = 1$
3: **for** $l = 1$ to q **do**
4: $\quad$ Alice and Bob use Protocol 5.11 to privately compute m_l and δ_l such that $2^{m_l} \leq D_l < 2^{m_l+1}$ and $D_l = 2^{m_l}\delta_l$.
5: $\quad u = u + m_l$
6: $\quad v = v\delta_l$
7: **end for**
8: $\mu = u \log 2$
9: Alice and Bob invoke secure logarithm to privately compute $\nu = \log v$.
10: $r = \mu + \nu$

standard method to prevent such underflow problems. Let

$$\begin{aligned} t_{ij} &= \log \tau_j f(x_i; \mu_j, \Sigma_j) \\ &= \log \tau_j + \log f(x_i; \mu_j, \Sigma_j) \\ &= \log \tau_j + \log \frac{1}{\sqrt{(2\pi)^q |\Sigma_j|}} \exp(-(x_i - \mu_j)\Sigma_j^{-1}(x_i - \mu_j)) \\ &= \log \tau_j - \frac{q}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma_j| + r_{ij}. \end{aligned} \tag{5.19}$$

Let

$$t_i = \max_j t_{ij} \tag{5.20}$$

and

$$t'_{ij} = t_{ij} - t_i. \tag{5.21}$$

Then

$$\begin{aligned} s_{ij} &= \frac{\tau_j f(x_i; \mu_j, \Sigma_j)}{\sum_{l=1}^k \tau_l f(x_i; \mu_l, \Sigma_l)} \\ &= \frac{\exp t_{ij}}{\sum_{l=1}^k \exp t_{il}} \\ &= \frac{\exp t_{ij} / \exp t_i}{(\sum_{l=1}^k \exp t_{il}) / \exp t_i} \\ &= \frac{\exp t'_{ij}}{\sum_{l=1}^k \exp t'_{il}}. \end{aligned}$$

Let

$$v_{ij} = \exp t'_{ij} \tag{5.22}$$

and

$$s_i = \sum_{l=1}^k v_{il}. \tag{5.23}$$

Then

$$s_{ij} = \frac{v_{ij}}{s_i}.$$

Note that all of t'_{ij} $(j = 1, \ldots, k)$ are non-positive numbers and at least one of them is 0. So we have $1 \leq s_i < k$.

I summarize the procedure in Protocol 5.13. Lines 1-4 compute $\mu_j = \log \tau_j$ and $w_j = \log|\Sigma_j|$ for each cluster j. These quantities can be used for the computation of each point x_i. Lines 5-26 compute s_{ij} for each point x_i. Lines 10-14 compute

$t_i = \max_j t_{ij}$. In line 13, if $t_i \leq t_{ij}$, then $\alpha = 1$ and we have $t_i = t_{ij}$. Otherwise we don't change t_i. Note that $t'_{ij} \leq 0$. So we can invoke Protocol 5.5 to privately compute $v_{ij} = \exp t'_{ij}$ and then compute $s_i = \sum_{l=1}^{k} v_{il}$. Also note that in the design of Protocol 5.5, when the exponent is smaller than -32, we simply approximate the value of the exponential function with 0. Because at least one of t'_{ij} is 0 and $s_i \geq 1$, we ignore the sufficiently small terms.

Note that we use real numbers in the description in the protocol, so we need to use secure multiplication of real numbers. The exception is in line 12 and 13 where α is either 0 or 1 and it is represented as it is in the plaintext domain and we only need to use secure multiplication of integers in line 13. We can use secure division to implement line 24. So we can implement the privacy preserving E step using secure multiplication, secure division, secure logarithm and secure exponential function.

4. Checking convergence criterion.

The stopping criterion is that the marginal log-likelihood in successive iterations is sufficiently close, as we presented in equation (5.15). For convenience, we repeat the equation here: we can check the condition on the relative difference

$$|\frac{\log L(\theta^{(t+1)}; X) - \log L(\theta^{(t)}; X)}{L(\theta^{(t+1)}; X)}| < \epsilon, \tag{5.24}$$

where ϵ is some predefined threshold and

$$\log L(\theta; X) = \sum_{i}^{n} \log \sum_{j=1}^{k} \tau_j \exp\left(-(x_i - \mu_j)^{\mathrm{T}} \Sigma_j^{-1} (x_i - \mu_j)\right). \tag{5.25}$$

Protocol 5.13 Privacy preserving E step in EM clustering

Input: $X = (x_1, \ldots, x_n)^{\mathrm{T}} \in R^{n \times q}$, $\tau_j \in R$, $\mu_j \in R^q$ and $\Sigma_j \in R^{q \times q}$ for $j = 1, \ldots, k$, all of which are secrets split between Alice and Bob.

Output: $s_{ij} \in R$ $(i = 1, \ldots, n, j = 1, \ldots, k)$; the results are secrets split between Alice and Bob.

1: **for** $j = 1$ to k **do**
2: Alice and Bob use Protocol 5.4 to securely compute $u_j = \log \tau_j$.
3: Alice and Bob use Protocol 5.12 to securely compute $w_j = \log |\Sigma_j|$.
4: **end for**
5: **for** $i = 1$ to n **do**
6: **for** $j = 1$ to k **do**
7: Alice and Bob use Protocol 5.10 to securely compute $r_{ij} = -(x_i - \mu_j)^{\mathrm{T}} \Sigma_j^{-1} (x_i - \mu_j)$.
8: $t_{ij} = r_{ij} + u_j - \frac{1}{2} w_j - \frac{q}{2} \log(2\pi)$
9: **end for**
10: $t_i = t_{i1}$
11: **for** $j = 2$ to k **do**
12: Alice and Bob use Protocol 3.1 to securely compare t_i and t_{ij}. Let $\alpha = 1$ if $t_i \leq t_{ij}$ and 0 otherwise.
13: $t_i = (1 - \alpha) t_i + \alpha t_{ij}$
14: **end for**
15: **for** $j = 1$ to k **do**
16: $t'_{ij} = t_{ij} - t_i$.
17: **end for**
18: $s_i = 0$.
19: **for** $j = 1$ to k **do**
20: Alice and Bob use Protocol 5.5 to securely compute $v_{ij} = \exp t'_{ij}$.
21: $s_i = s_i + v_{ij}$
22: **end for**
23: **for** $j = 1$ to k **do**
24: $s_{ij} = v_{ij} / s_i$
25: **end for**
26: **end for**

Note that

$$\begin{aligned}
\log L(\theta; X) &= \sum_{i=1}^{n} \log \sum_{j=1}^{k} \tau_j f(x_i; \mu_j, \Sigma_j) \\
&= \sum_{i=1}^{n} \log \sum_{j=1}^{k} \exp t_{ij} \\
&= \sum_{i=1}^{n} \log \sum_{j=1}^{k} \exp(t'_{ij} + t_i) \\
&= \sum_{i=1}^{n} \log \sum_{j=1}^{k} (\exp t_i \exp t'_{ij}) \\
&= \sum_{i=1}^{n} \log(\exp t_i (\sum_{j=1}^{k} \exp t'_{ij})) \\
&= \sum_{i=1}^{n} (t_i + \log \sum_{j=1}^{k} \exp t'_{ij}) \\
&= \sum_{i=1}^{n} t_i + \sum_{i=1}^{n} \log \sum_{j=1}^{k} \exp t'_{ij} \\
&= \sum_{i=1}^{n} t_i + \sum_{i=1}^{n} \log \sum_{j=1}^{k} v_{ij} \\
&= \sum_{i=1}^{n} t_i + \sum_{i=1}^{n} \log s_i.
\end{aligned}$$

The last equation requires an invocation of secure logarithm for each observation. We can reduce the number of secure logarithms using an equivalent formula

$$\log L(\theta; X) = \sum_{i=1}^{n} t_i + \log \prod_{i=1}^{n} s_i.$$

Note that $1 \leq s_i < k$. We can incorporate the computation of the marginal log-likelihood function in the computation of the E step. I present the augmented privacy preserving E step in Protocol 5.14. We assume that the original privacy preserving protocol for the E step (protocol 5.13) also outputs (the shares of) s_i and t_i in addition to s_{ij}. In the protocol I use G to denote the marginal log-likelihood $\log L(\theta; X)$.

Protocol 5.14 Privacy preserving E step (augmented) in EM clustering

Input: $x_i \in R^q$ $(i = 1, \ldots, n)$, $\tau_j \in R$, $\mu_j \in R^q$ and $\Sigma_j \in R^{q \times q}$ for $j = 1, \ldots, k$, all of which are secrets split between Alice and Bob.

Output: $s_{ij} \in R$ $(i = 1, \ldots, n, j = 1, \ldots, k)$ and the marginal log-likelihood G; the results are secrets split between Alice and Bob.

1: Alice and Bob use Protocol 5.13 to privately compute s_{ij}, s_i and t_i for $i = 1, \ldots, n$ and $j = 1, \ldots, k$.
2: $s = 1$
3: $G = 0$
4: **for** $i = 1$ to n **do**
5: $\quad G = G + t_i$
6: $\quad s = ss_i$
7: **end for**
8: Alice and Bob use secure logarithm (Protocol 5.4) to privately compute $u = \log s$.
9: $G = G + u$

5. Final clustering.

The vector $(s_{i1}, \ldots, s_{ik}) \in R^k$ gives a soft clustering of the data point x_i. To get a hard clustering, we assign point x_i to cluster j such that s_{ij} is the largest among all s_{il} $(l = 1, \ldots, k)$. I present a secure protocol for this procedure in Protocol 5.15.

Protocol 5.15 Privacy preserving protocol for final clustering

Input: s_{ij} $(i = 1, \ldots, n; j = 1, \ldots, k)$, a soft clustering of the datasets, which are split between Alice and Bob.

Output: a vector of integers $c \in Z^n$ such that $c_i = j$ if s_{ij} is the largest among all s_{il} $(l = 1, \ldots, k)$. These results are secrets split between Alice and Bob.

1: **for** $i = 1$ to n **do**
2: $\quad c_i = 1$
3: $\quad m = s_{i1}$
4: $\quad$ **for** $l = 2$ to k **do**
5: $\quad\quad$ Alice and Bob use Protocol 3.1 to privately compare m and s_{il}. Let $\alpha = 1$ if $m \leq s_{il}$ and 0 otherwise.
6: $\quad\quad c_i = (1 - \alpha)c_i + \alpha l$
7: $\quad\quad m = (1 - \alpha)m + \alpha s_{il}$
8: $\quad$ **end for**
9: **end for**

Note that when $m \leq s_{il}$, $\alpha = 1$. Then in line 6 c_i is set to the cluster l and in line 7 m is assigned the value s_{il}; otherwise, we don't change c_i and m. In the implementation of this protocol, s_{ij} and m are real numbers and α and c_i are integers. We use secure multiplications of integers in lines 6 and 7. I summarize the privacy

preserving EM clustering protocol in Protocol 5.16.

Protocol 5.16 Privacy preserving two-party EM clustering

Input: a dataset $X = (x_1, \ldots, x_n)^{\mathrm{T}} \in R^{n \times q}$ is distributed between Alice and Bob, $X = X_1 + X_2$. Alice has input $(1^K, k, X_1)$ and Bob has input $(1^K, k, X_2)$, where k is the number of clusters and K is the security parameter.

Output: $c \in Z^n$ such that point x_i is assigned to cluster c_i.

1: Alice generates a pair of keys $(e, d) = G(1^K)$ and sends the public key e to Bob.
2: Alice and Bob use Protocol 5.7 to compute the initial clustering s_{ij}.
3: **for** $t = 1$ to MAX **do**
4: Alice and Bob use Protocol 5.8 to perform privacy preserving M step. The results $\theta^{(t+1)} = (\tau_1, \mu_1, \Sigma_1, \ldots, \tau_k, \mu_k, \Sigma_k)$ are secrets split between Alice and Bob.
5: Alice and Bob use Protocol 5.14 to perform privacy preserving E step. The results are s_{ij} and the marginal log-likelihood function $G^{(t+1)} = L(\theta^{(t+1)}; X)$. These results are secrets split between Alice and Bob.
6: **if** $|\frac{G^{(t)} - G^{(t+1)}}{G^{(t+1)}}| < \epsilon$ **then**
7: break
8: **end if**
9: **end for**
10: Alice and Bob use Protocol 5.15 to find the final clustering results $c \in Z^n$. The results are secrets split between Alice and Bob.
11: Alice and Bob exchange their shares of c_i $(i = 1, \ldots, n)$ so that both parties obtain the final clustering results.

We use secure comparison (Protocol 3.1) to implement line 6. Denote the comparison results by u_t and its two shares by $u_{t,A}$ and $u_{t,B}$. Alice and Bob exchange their shares of comparison results and determine whether they will stop the loop or not. This is the only place where Alice and Bob disclose extra information besides the final clustering results. It is equivalent to the disclosure of the number of iterations. In line 11, Alice and Bob exchange their shares of c and obtain the final clustering results.

The privacy of the EM clustering protocol is guaranteed by the following theorem.

Theorem 5.1. *The EM clustering protocol (Protocol 5.16) discloses only the number of iterations beside the final clustering results. In other words, if we add the number of iterations as part of the final results, the EM clustering protocol is secure in the semi-honest model.*

Proof. To prove that the privacy preserving two-party EM clustering protocol is secure in the semi-honest model, we need to show that:

(1) Given Alice's input, her output and the number of iterations T, we are able to simulate her view during the execution of the protocol.

(2) Given Bob's input, his output and the number of iterations T, we are able to simulate his view during the execution of the protocol.

Note that we implement Protocol 5.16 using secure multiplication, secure division, secure logarithm and secure exponential function. All these subprotocols are implemented using secure multiplication. So we actually implement Protocol 5.16 using secure multiplication only.

We first examine the message exchanges during the execution of Protocol 5.16.

(1) At the beginning (step 1), Alice generates a pair of keys $(e, d) = G(1^K)$ and sends the public key e to Bob. As usual, we denote the modulus associated with this key pair by N.

(2) At step 2, Alice randomly selects k indices $(d_1, \ldots, d_k)$ from $\{1, \ldots, n\}$ and sends these indices to Bob.

(3) During each invocation of secure multiplication, Alice sends two encrypted messages to Bob and Bob sends the encryption of a random number to Alice. Denote the total number of invocations of secure multiplication by v.

(4) At step 6, Alice and Bob exchange their shares of the comparison results u_t $(t = 1, \ldots, T)$. Alice sends $u_{t,A}$ to Bob and Bob sends $u_{t,B}$ to Alice.

(5) At the last step (step 11), Alice and Bob exchange their shares of c_i $(i = 1, \ldots, n)$. Alice sends $c_{i,A}$ to Bob and Bob sends $c_{i,B}$ to Alice.

To simulate Alice's view during the execution of Protocol 5.16, we first check the elements of Alice's view.

(1) Alice has input $(1^K, k, X_1)$.

(2) Alice uses a sequence of random coins $r_{1,A}$ to generate the key pair $(e, d) = G(1^K, r_{1,A})$ and uses another sequence of random coins $r_{2,A}$ to select k random indices according to some function f, $(d_1, \ldots, d_k) = f(r_{2,A})$. During the invocations of secure multiplication, Alice uses random numbers in Z_N^* to encrypt messages. Denote by $r_{3,A} = (r_{3,1,A}, \ldots, r_{3,2v,A})$ the sequences of random coins used to generate those random numbers. Denote all these sequences of random coins by $r_A = (r_{1,A}, r_{2,A}, r_{3,A})$.

(3) During each invocation of secure multiplication, Alice receives the encryption of a random number from Bob. Denote all these messages by $M_{1,A} = (m_1, \ldots, m_v)$.

(4) At step 6, Alice receives the shares of the comparison results $u_{t,B}$ from Bob. Denote these messages by $M_{2,A} = (u_{1,B}, \ldots, u_{T,B})$.

(5) Alice receives $c_{i,B}$ $(i = 1, \ldots, n)$ from Bob at step 11. Denote these messages by $M_{3,A} = (c_{1,B}, \ldots, c_{n,B})$.

So Alice's view is $VIEW_A = (1^K, k, X_1, r_A, M_{1,A}, M_{2,A}, M_{3,A})$. I now show how to simulate Alice's view based on her input $(1^K, k, X_1)$, her output (the clustering results c_i) and the number of iterations T. The simulator generates a number of sequences of independent random coins $r'_{1,A}, r'_{2,A}$ and $r'_{3,A} = (r'_{3,1,A}, \ldots, r'_{3,2v,A})$, which correspond to the sequences $r_A = (r_{1,A}, r_{2,A}, r_{3,A})$ that Alice uses in the real execution. Let $r'_A = (r'_{1,A}, r'_{2,A}, r'_{3,A})$. It is identically distributed with $r_A = (r_{1,A}, r_{2,A}, r_{3,A})$. The simulator generates $(e', d') = G(1^K, r'_{1,A})$, which is identically distributed with (e, d). Denote the modulus associated with the key pair (e', d') by N'.

I next show how to simulate the messages $M_1 = (m_1, \ldots, m_v)$. As I discuss in section 2.3, each message m_i that Alice receives during the invocation of secure multiplication is the encryption of a random number and is identically distributed with $E_e(b_1, b_2)$, where b_1 is uniformly random in Z_N and b_2 is uniformly random in Z_N^*. Here b_1 and b_2 correspond to r and s_3 in Protocol 2.1, respectively. The simulator generates a uniformly random number b'_1 in $Z_{N'}$ and another uniformly random number b'_2 in $Z_{N'}^*$ and computes $m'_i = E_{e'}(b'_1, b'_2)$. m'_i is identically distributed with m_i. Let $M'_{1,A} = (m'_1, \ldots, m'_v)$. $M'_{1,A}$ is identically distributed with $M_{1,A} = (m_1, \ldots, m_v)$.

I next show how to simulate the messages $M_2 = (u_{1,B}, \ldots, u_{T,B})$. Note that before Alice and Bob exchange their shares of the comparison results u_i, Alice has the share $u_{i,A}$ such that $(u_{i,A} + u_{i,B}) \bmod N = u_i$. Alice's share $u_{i,A}$ is computed from her input $(1^K, k, X_1)$, her random coins r_A, the messages in $M_{1,A}$ and the messages in $M_{2,A}$ that Alice received before the exchange of the shares of u_i according to some function $g_{i,A}$, $u_{i,A} = g_{i,A}(1^K, k, X_1, r_A, M_{1,A}, M_{2,A})$. The simulator computes $u'_{i,A} = g_{i,A}(1^K, k, X_1, r'_A, M'_{1,A}, M'_{2,A})$, which is identically distributed with $u_{i,A}$. Also note that we can infer the comparison result u_i of each iteration from the number of iterations T. The simulator computes $u'_{i,B} = (u_i - u'_{i,A}) \bmod N'$, which is identically distributed with $u_{i,B} = (u_i - u_{i,A}) \bmod N$. Let $M'_{2,A} = (u'_{1,B}, \ldots, u'_{T,B})$. It is identically distributed with $M_{2,A} = (u_{1,B}, \ldots, u_{T,B})$.

Similarly, we can simulate the messages $M_3 = (c_{1,B}, \ldots, c_{n,B})$. Before Alice and Bob exchange their shares of c_i, Alice has the share $c_{i,A}$ such that $(c_{i,A}+c_{i,B}) \bmod N = c_i$. $c_{i,A}$ is computed as $c_{i,A} = h_{i,A}(1^K, k, X_1, r_A, M_{1,A}, M_{2,A})$ for some function $h_{i,A}$. The simulator computes $c'_{i,A} = h_{i,A}(1^K, k, X_1, r'_A, M'_{1,A}, M'_{2,A})$, which is identically distributed with $c_{i,A}$. Since the simulator is given the final result c_i, it can compute $c'_{i,B} = (c_i - c'_{i,A}) \bmod N'$, which is identically distributed with $c_{i,B} = (c_i - c_{i,A}) \bmod N$. Let $M'_{3,A} = (c'_{1,B}, \ldots, c'_{n,B})$. It is identically distributed with $M_{3,A} = (c_{1,B}, \ldots, c_{n,B})$.

Let $M'_A = (1^K, k, X_1, r'_A, M'_{1,A}, M'_{2,A}, M'_{3,A})$. From the above discussion we know that M'_A is identically distributed with Alice's view $VIEW_A$ and thus it is computa-

tionally indistinguishable from $VEIW_A$. So we can simulate Alice's view using her input $(1^K, k, X_1)$, the clustering results c_i and the number of iterations T.

Now I show how to simulate Bob's view during the execution of Protocol 5.16 given Bob's input $(1^K, k, X_2)$, his output c_i and the number of iterations T. Bob's view is divided into seven parts.

(1) Bob has input $(1^K, k, X_2)$.

(2) During each invocation of secure multiplication, Bob uses a sequence of random coins $r_{3,i,B}$ to generate a uniformly random number b_1 in Z_N and another uniformly random number b_2 in Z_N^* and then computes $E_e(b_1, b_2)$. Denote these sequences of random coins by $r_B = (r_{3,1,B}, \ldots, r_{3,v,B})$. Here v is the total number of invocations of secure multiplication. We use the same subscript 3 here as we use in the simulation of Alice's view because they are both used in the invocations of secure multiplication.

(3) At the beginning (step 1), Bob receives the public key e from Alice.

(4) At step 2, Bob receives k indices that Alice randomly chooses from $\{1, \ldots, n\}$. Denote these messages by $M_{0,B} = (d_1, \ldots, d_k)$

(5) During each invocation of secure multiplication, Bob receives the encryptions of two numbers from Alice. Bob receives altogether $2v$ messages in the v invocations of secure multiplications. Denote the i-th message by $m_i = E_e(a_i, r_{3,i,A})$, where a_i is some number only known by Alice and $r_{3,i,A}$ is the sequence of random coins that Alice uses to encrypt a_i. We denote all these messages by $M_{1,B} = (m_1, \ldots, m_{2v}) = (E_e(a_1, r_{3,1,A}), \ldots, E_e(a_{2v}, r_{3,2v,A}))$.

(6) At step 6, Bob receives the shares of the comparison result $u_{t,A}$ from Alice. Denote these messages by $M_{2,B} = (u_{1,A}, \ldots, u_{T,A})$.

(7) At step 11, Bob receives $c_{i,A}$ $(i = 1, \ldots, n)$ from Alice. Denote these messages by $M_{3,B} = (c_{1,A}, \ldots, c_{n,A})$.

So Bob' view is $VIEW_B = (1^K, k, X_2, r_B, e, M_{0,B}, M_{1,B}, M_{2,B}, M_{3,B})$. I now show how to simulate Bob's view based on his input $(1^K, k, X_2)$, his output (the clustering results c_i) and the number of iterations T. The simulator generates a number of sequences of independent random coins $r'_B = (r'_{3,1,B}, \ldots, r'_{3,v,B})$, which correspond to the sequences of random coins $r_B = (r'_{3,1,B}, \ldots, r'_{3,v,B})$ that Bob uses in the real execution. $r'_{3,B}$ and $r_{3,B}$ are identically distributed.

The simulator generates a pair of keys $(e', d') = G(1^k, r'_{1,A})$, where $r'_{1,A}$ is a sequence of independent random coins which the simulator generates to simulate $r_{1,A}$ that Alice uses in the real execution. Because $r'_{1,A}$ is identically distributed with $r_{1,A}$, (e', d') is identically distributed with the key pair (e, d) that Alice generates in the real execution, and e' is also identically distributed with e. Denote the modulus

associated with the key pair (e', d') by N'.

I next show how to simulate the messages in $M_{0,B}$. The simulator computes $(d'_1, \ldots, d'_k) = f(r'_{2,A})$, where f is the function that Alice uses to select these random indices in the real execution and $r'_{2,A}$ is a sequence of independent random coins which the simulator generates to simulate $r_{2,A}$ that Alice uses to generate $(d_1, \ldots, d_k)$. $(d'_1, \ldots, d'_k)$ is identically distributed with $(d_1, \ldots, d_k)$.

I show how to simulate the messages $M_{1,B} = (E_e(a_1, r_{3,1,A}), \ldots, E_e(a_{2v}, r_{3,2v,A}))$. The simulator computes $M'_{1,B} = (E_{e'}(0, r'_{3,1,A}), \ldots, E_{e'}(0, r'_{3,2v,A}))$, where $r'_{3,i,A}$ is a sequence of independent random coins which the simulator generates to simulate $r_{3,i,A}$ that Alice uses in the real execution. To show that $M'_{1,B}$ is computationally indistinguishable from $M_{1,B}$, we define $M''_{1,B} = (E_e(0, r_{3,1,A}), \ldots, E_e(0, r_{e,2v,A}))$. Because e' and e are identically distributed and $r'_{3,i,A}$ is uniformly distributed in $Z^*_{N'}$ and $r_{3,i,A}$ is uniformly distributed in Z^*_N, we know that $(e, M''_{1,B})$ is identically distributed with $(e', M'_{1,B})$ and hence $(e, M''_{1,B})$ is computationally indistinguishable from $(e', M'_{1,B})$. Besides, because the Paillier cryptosystem is semantically secure, $(e, M''_{1,B})$ and $(e, M_{1,B})$ are computationally indistinguishable. So $(e', M'_{1,B})$ is also computationally indistinguishable from $(e, M_{1,B})$.

The simulations of the messages M_2 and the messages M_3 are similar to the simulations for Alice's view. Remember that $M_{2,B} = (u_{1,A}, \ldots, u_{T,A})$. Before Alice and Bob exchange their shares of the comparison result u_i, Bob has the share $u_{i,B}$ such that $(u_{i,A} + u_{i,B}) \bmod N = u_i$. Bob's share $u_{i,B}$ is computed as $u_{i,B} = g_{i,B}(1^K, k, X_2, r_B, e, M_{0,B}, M_{1,B}, M_{2,B})$ for some function $g_{i,B}$. The simulator computes $u'_{i,B} = g_{i,B}(1^K, k, X_2, r'_B, e', M'_{0,B}, M'_{1,B}, M'_{2,B})$, which is indistinguishable from $u_{i,B}$. Also note that we can infer the comparison result u_i of each iteration from the number of iterations. The simulator computes $u'_{i,A} = (u_i - u'_{i,B}) \bmod N'$, which is computationally indistinguishable from $u_{i,A} = (u_i - u_{i,B}) \bmod N$. Let $M'_{2,B} = (u'_{1,A}, \ldots, u'_{T,A})$. It is computationally indistinguishable from $M_{2,B}$.

To simulate the messages $M_{3,B} = (c_{1,A}, \ldots, c_{n,A})$, note that before Alice and Bob exchange their shares of c_i, Bob has the share $c_{i,B}$ such that $(c_{i,A} + c_{i,B}) \bmod N = c_i$. $c_{i,B}$ is computed as $c_{i,B} = h_{i,B}(1^K, k, X_2, r_B, e, M_{0,B}, M_{1,B}, M_{2,B})$ for some function $h_{i,B}$. The simulator computes $c'_{i,B} = h_{i,B}(1^K, k, X_2, r'_B, e', M'_{0,B}, M'_{1,B}, M'_{2,B})$, which is indistinguishable from $c_{i,B}$. Since the simulator is given the final result c_i, it can compute $c'_{i,A} = (c_i - c'_{i,B}) \bmod N'$, which is indistinguishable from $c_{i,A} = (c_i - c_{i,B} \bmod N$. Let $M'_{3,A} = (c'_{1,A}, \ldots, c'_{n,A})$. It is computationally indistinguishable from $M_{3,B}$.

Let $M'_B = (1^K, k, X_2, r'_B, e', M'_{0,B}, M'_{1,B}, M'_{2,B}, M'_{3,B})$. From the above discussion

we know that M'_B is computationally indistinguishable from Bob's view $VEIW_B$. So we can simulate Bob's view using his inputs $(1^K, k, X_2)$, the clustering results c_i and the number of iterations T.

□

5.6 Experimental Results

I have implemented the two-party privacy preserving EM clustering protocol in C++ based on the Paillier cryptosystem. I used the GMP library (Torbjorn Granlund et al.) for big integers. I tested the protocol on two separating computers. The settings of the computers and network system are the same as we used in section 4.8.

In Protocol 5.16, Alice generates a pair of keys for each running of the protocol. In the experiments, the pair of keys were fixed so that I could use the precomputation techniques presented in section 3.2. The key security parameter used in all the experiments was $K = 512$.

I tested the privacy preserving EM clustering protocol on three datasets: Iris, Zoo and Glass Identification, which are available in the UCI machine learning repository (Frank and Asuncion). The Glass Identification dataset contains 6 classes. I considered only two broad classes: window and non-window. I assume that the datasets are vertically partitioned between Alice and Bob. The results on other partitions are similar. I describe the datasets and their partitions in Table 5.1, in which n is the number of observations, q is the number of attributes and k is the number of clusters. Alice holds the first q_1 attributes and Bob holds the last q_2 attributes.

Table 5.1: Benchmark datasets for EM clustering

Dataset	n	q	q_1	q_2	k
Iris	150	4	2	2	3
Glass	214	9	4	5	2
Zoo	101	16	8	8	7

I report the execution time of the privacy preserving EM clustering protocol in Table 5.2. I used equation 5.15 as the stopping criterion and used $\epsilon = 10^{-6}$ as the threshold for all the experiments. The overall execution time equals the number of iterations times the running time per each iteration (The running time for initialization and the final clustering is relatively small compared to the iterations). I also report both the execution time per each iteration and the number of iterations. Note

that the number of iterations depends on the initial centers the algorithm randomly selects. A possible method to find a good initial clustering privately is to first run the privacy preserving k-means clustering algorithm on the datasets and use the results as the initial centers. Here I just gave a concept-of-proof implementation. I also implemented an ordinary EM clustering algorithm without privacy concerns in the Matlab software. In the experiments, I first ran the EM clustering algorithm in Matlab to select good initial centers. All these datasets have predefined clustering. The clustering accuracy is computed as the percentage of the correctly clustered observations. Using the same initial cluster centers as the secure protoocol, the Matlab program finishes in less than 0.1 second on all these three datasets. The secure protocol achieves the same clustering accuracies as the Matlab program.

Table 5.2: Experimental results of privacy preserving EM clustering on benchmark datasets

	Dataset		
Measure	Iris	Glass	zoo
Overall execution time	6643s	4287s	7234s
Number of iterations	13	6	4
Time per Iteration	500s	698s	1979s
Clustering Accuracy	96.67%	90.19%	76.24%

Because the number of iterations depends on the randomly selected initial centers and varies for each execution of the EM clustering protocol, in what follows I only report the execution time for each iteration. I first tested how the running time scales with the size of the dataset. The dataset of size n consists of the first n observations in the original dataset. I report the results on the Iris and Zoo datasets in Figure 5.1 and 5.2. In all the figures in this section, the execution time is measured in seconds. The figures show that the execution time per each iteration scales linearly with the number of observations in the datasets.

I then tested how the execution time scales with the number of attributes. I tested on the Zoo datasets. I assume that the attributes are evenly distributed between Alice and Bob. Note that when we don't use all the attributes, the clustering model may not be accurate. I report the results in Figure 5.3. In the figure, the line is superlinear in the number of attributes. It is expected to exhibit quadratic asymptotic behavior when the number of attributes is large.

I also tested how the execution time scales with the number of clusters. I tested on the Zoo dataset. The zoo dataset is supposed to have 7 clusters. I clustered it

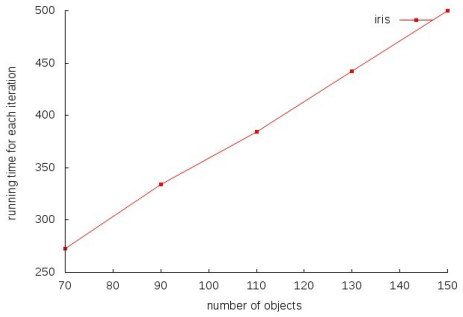

Figure 5.1: Scalability of privacy preserving *EM* clustering with respect to the number of observations (Iris).

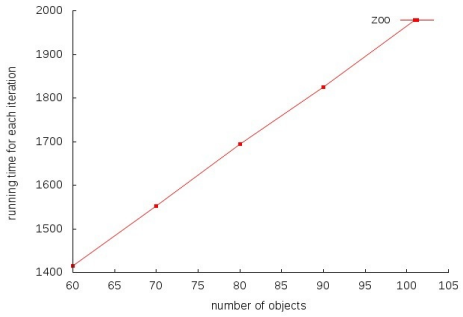

Figure 5.2: Scalability of privacy preserving *EM* clustering with respect to the number of observations (Zoo).

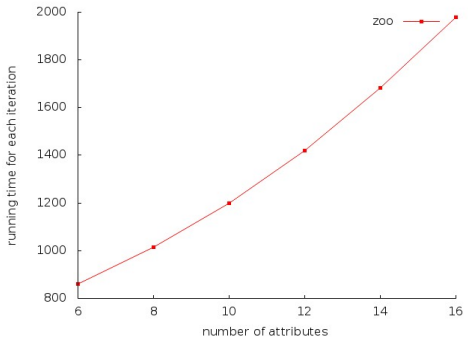

Figure 5.3: Scalability of privacy preserving *EM* clustering with respect to the number of attributes (Zoo)

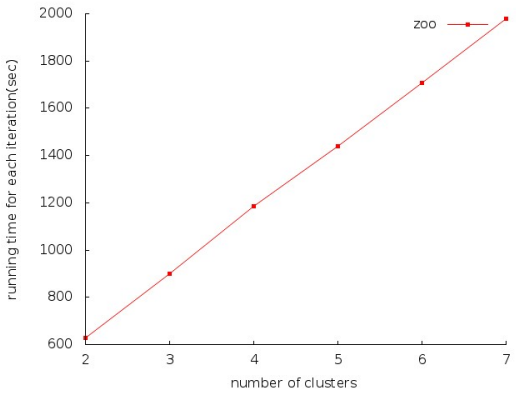

Figure 5.4: Scalability of privacy preserving *EM* clustering with respect to the number of clusters (Zoo)

into 2 to 7 clusters. This is only to test the execution time and the clustering model may not be useful. I report the execution time in Figure 5.4. We observe that the execution time per each iteration scales linearly with the number of clusters.

Chapter 6

Application of the Schur Complement

6.1 Introduction

A dataset is typically represented as a matrix. When data are partitioned among multiple parties, the data can be represented as a block matrix and each party holds a block of the matrix. For example, in the horizontal partitioned cases, we can write the data matrix as

$$X = \begin{pmatrix} X_1 \\ \vdots \\ X_m \end{pmatrix},$$

where party i holds the submatrix $X_i \in R^{n_i \times p}$ and $\sum_{i=1}^{k} n_i = n$. In the vertically partitioned cases, the data matrix can be written as

$$X = (X_1, \cdots, X_m),$$

where party i holds the submatrix $X_i \in R^{n \times p_i}$ and $\sum_{i=1}^{k} p_i = p$.

A natural question is: can we take advantage of the structure of the block matrix when we design privacy preserving data mining protocols? Motivated by this question, we study the potential applications of the Schur Complement in the design of efficient privacy preserving kernel ridge regression.

This chapter is organized as follows. I introduce the concept of the Schur Complement in section 6.2 and describe kernel regression in section 6.3. In section 6.4 I explore the potential application of the Schur Complement in the design of efficient privacy preserving kernel ridge regression.

6.2 Schur Complement

Consider a $(n_1 + n_2) \times (n_1 + n_2)$ matrix

$$W = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

where A is an $n_1 \times n_1$ squared matrix and D is an $n_1 \times n_1$ squared matrix.

We want to solve the following system of linear equations:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}, \tag{6.1}$$

namely

$$A\beta_1 + B\beta_2 = e_1 \tag{6.2}$$

$$C\beta_1 + D\beta_2 = e_2 \tag{6.3}$$

Assuming that D is invertible, we multiply equation 6.3 by D^{-1}

$$D^{-1}(C\beta_1 + D\beta_2) = D^{-1}e_2.$$

and get

$$\beta_2 = D^{-1}e_2 - D^{-1}C\beta_1. \tag{6.4}$$

We then substitute β_2 in equation 6.2 and we have

$$(A - BD^{-1}C)\beta_1 = e_1 - BD^{-1}e_2. \tag{6.5}$$

We can solve this equation to get β_1 and then substitute β_1 in equation 6.4 to get β_2.

The matrix $(A - BD^{-1}C)$ is called the Schur Complement of D. Similarly, the Schur Complement of A is $D - CA^{-1}B$. Using the technique of the Schur Complement, we can reduce the order of linear systems, however at the cost of inverting a smaller matrix and two matrix multiplications. When W is symmetric, it can be showed that W is symmetric positive if and only if D and its Schur Complement $(A - BD^{-1}C)$ are both symmetric and positive.

The concept of the Schur Complement has wide applications in numerical computation, for example in the domain decomposition method (Zhang, 2005). The Schur Complement is also employed in quadratic programming (Gill et al., 1987; Bartlett et al., 2006). In my research, I am studying how to apply the concept of the Schur

Complement to design efficient privacy preserving data mining protocols.

6.3 Kernel Regression

Suppose that we have a training set $X = (x_1, .., x_n)^{\mathrm{T}}$ and $Y = (y_1, .., y_n)^{\mathrm{T}}$, where $x_i \in R^p$ and $y_i \in R$ are the values of i-th observation for the predictor attributes and the response attribute respectively. We shall assume that $p \ll n$. The linear regression model assumes the linear relationship between the response attribute and the predictor attributes and tries to find parameters $\beta \in R^q$ which minimize the residual sum of squares (RSS)

$$L(\beta) = (Y - X\beta)^{\mathrm{T}}(Y - X\beta).$$

The minimizer is the solution of the following system:

$$(X^{\mathrm{T}}X)\beta = X^{\mathrm{T}}Y.$$

To explore the non-linear relationship between the response attribute and the predictor attributes, we can map each point x_i to some feature space $\phi(x_i)$ and find linear relationship in the feature space. The mapping ϕ corresponds to some nonlinear relationship in the original space. The kernel function $k(x_i, x_j)$ determines the inner product of any two points in the feature space. For example, the polynomial kernel k is a polynomial of the inner product of x_i and x_j and the Gaussian kernel is

$$k(x_i, x_j) = \frac{1}{(\delta\sqrt{2\pi})^p} \exp\left(-\frac{|x_i - x_j|^2}{2\delta^2}\right)$$

where δ is a parameter to control the width of the neighborhood.

The Gram (kernel) matrix G is an $n \times n$ matrix such that $G_{i,j} = k(x_i, x_j)$. Then we can solve the following system of equations:

$$G\alpha = Y. \tag{6.6}$$

The prediction of the response attribute for a new observation x is then computed as $\sum_i^n \alpha_i k(x_i, x)$. To overcome the over-fitting problem, we may add a regularization term to the above equation:

$$(G + \lambda I)\alpha = Y.$$

where I is the identity matrix and λ is a regularization parameter. For ease of presentation, I omit the regularization term in the following and assume that the Gram matrix is positive symmetric.

One direct method to solve equation 6.6 is to use Cholesky decomposition, which takes $O(n^3/3)$ multiplications. In the privacy preserving setting, secure multiplication is a costly operation. When the number of observations is large, it is not efficient to use Cholesky decomposition. One alternative approach is to use an iterative method, for example the conjugate gradient method, to find an approximate solution.

Conjugate gradient algorithm (CG) is an iterative method to solve linear systems of equations when the coefficient matrix is symmetric and positive. It is known that the solution to the linear system of equations 6.6 is the minimizer of the function

$$f(\alpha) = \frac{1}{2}\alpha^{\mathrm{T}} A\alpha + Y^{\mathrm{T}}\alpha.$$

We define the residual of some approximate solution α_i as $r_i = Y - G\alpha_i$ and two vectors v and u are called G-orthogonal if $v^t Gu = 0$. To find the minimizer of the function $f(\alpha)$, the conjugate gradient method starts with any initial point α_0 and searches along a set of G-orthogonal directions. These directions are constructed from the residuals step by step. It can be proved that the conjugate gradient method will find the exact solution in n steps. We may terminate the iteration when we find a satisfactory approximate solution. I present the conjugate gradient method to solve equation 6.6 in Algorithm 6.1. See (Jonathan, 1994) for more details.

Algorithm 6.1 Conjugate gradient method for kernel regression

Input: an $n \times n$ Gram matrix G, a vector $Y \in R^n$.
Output: an approximate solution of $G\alpha = Y$

$r_0 = Y - G\alpha_0; b_0 = r_0$
for $i = 0$ to T **do**
  $c_i = Gb_i$
  $\mu_i = \langle r_i, r_i\rangle / \langle c_i, b_i\rangle$
  $\alpha_{i+1} = \alpha_i + \mu_i b_i$
  $r_{i+1} = r_i - \mu_i c_i$
  if $r_{i+1}^2 < \epsilon$ **then**
    break;
  end if
  $\nu_i = \langle r_{i+1}, r_{i+1}\rangle / \langle r_i, r_i\rangle$
  $b_{i+1} = r_{i+1} + \nu_i b_i$
end for

Given the Gram matrix G, the conjugate gradient methods takes about kn^2 mul-

tiplications and $2k$ divisions if it terminates in k steps. We also need $n^2/2$ evaluations of the kernel function to form the Gram matrix.

6.4 Application of the Schur Complement in Privacy Preserving Kernel Regression

Consider the scenario where the dataset is horizontally partitioned between two parties $X = \binom{X_1}{X_2}$, $Y = \binom{Y_1}{Y_2}$, Alice holds the first n_1 observation $X_1 \in R^{n_1 \times p}$, $Y_1 \in R^{n_1}$ and Bob holds the last n_2 observations $X_2 \in R^{n_2 \times p}$, $Y_2 \in R^{n_2}$. Alice and Bob would like to apply kernel ridge regression on their joint dataset, but without disclosing their confidential data. They need to first form the Gram matrix G privately. We assume that we are using polynomial kernel function. When x_i and x_j are held by the same party, they can compute $k(x_i, x_j)$ locally. When they are held by different parties, Alice and Bob can use secure scalar product to compute $\langle x_i, x_j \rangle$ privately and then use secure multiplication to compute the polynomials. It requires $p + d$ invocations of secure multiplication to privately evaluate a polynomial of degree d.

Now let

$$G = \begin{pmatrix} G_1 & G_2 \\ G_2^{\mathrm{T}} & G_3 \end{pmatrix}$$

G_1 is an $n_1 \times n_1$ matrix whose entries are the inner products of the data points held by Alice. It can be computed by Alice locally. G_3 is an $n_2 \times n_2$ matrix whose entries are the inner products of the data points held by Bob and it can be computed by Bob locally. Alice and Bob jointly compute the $n_1 \times n_2$ matrix G_2 using the kernel polynomial function. So it takes $n_1 n_2 (p + d)$ secure multiplications to compute the Gram matrix G privately.

Kernel ridge regression needs to find the solution of $G\alpha = Y$. If we implement Cholesky decomposition method privately, it takes $O(n^3/3)$ secure multiplications. If we use the conjugate gradient method directly, it takes about $n^2 k$ secure multiplications if the iteration terminates in k steps. I now show how to use the technique of the Schur Complement to improve the efficiency. We write the equation $G\alpha = Y$ in block forms:

$$\begin{pmatrix} G_1 & G_2 \\ G_2^{\mathrm{T}} & G_3 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix}$$

G is symmetric positive, so both G_3 and its Schur Complement $G_1 - G_2 G_3^{-1} G_2^{\mathrm{T}}$

are symmetric and positive. Now equation 6.5 becomes:

$$(G_1 - G_2 G_3^{-1} \times G_2^{\mathrm{T}})\alpha_1 = Y_1 - G_2 G_3^{-1} Y_2.$$

We can apply the conjugate gradient method algorithm 6.1 to the above system, which is of order n_1. Clearly we can use secure multiplication, secure division and secure comparison to implement a privacy preserving conjugate gradient algorithm. The matrix-vector multiplication in algorithm 6.1 now becomes

$$(G_1 - G_2 G_3^{-1} G_2^{\mathrm{T}})b = G_1 b - G_2 G_3^{-1} G_2^{\mathrm{T}} b$$

The key observation is that Bob knows G_3. Although it takes $O(n_2^3)$ multiplications to invert G_3, Bob can invert it without any communication with the other party. So no secure multiplication is needed to invert G_3. The above matrix-vector multiplication requires $n_1^2 + n_1 n_2 + n_2^2 + n_1 n_2 = n^2$ secure multiplications. This is the same as if we apply the conjugate gradient method to solve $G\alpha = Y$ directly. However, now the order of the linear system is n_1. We may assume that $n_1 \leq n_2$ and we have $n_1 \leq n/2$. As we have far fewer unknowns, the conjugate gradient method converges faster and we can guarantee that it can get an exact solution in n_1 steps. If the iteration terminates in k steps, the number of secure multiplications is kn^2. Once we solve α_1, we can use equation 6.4 to compute α_2, which requires $n_1 n_2 + n_2^2 = n_2 n$ secure multiplications. Note that we need $n_1 n_2 (p + d)$ secure multiplications to form the Gram matrix. So the total number of secure multiplications is $n_1 n_2 (p + d) + kn^2 + n_2 n$.

The conjugate gradient method is sensitive to the accumulation of roundoff errors and is typically used with some form of preconditioners. If we apply the Schur Complement in the iterative method, we don't form the Schur Complement explicitly and it is not straightforward to construct a suitable preconditioner efficiently. This problem may be mitigated as we typically use long bits (for example, 1024 bits) to represent numbers in secure computation and we may allocate many bits (a few hundred bits) to represent the fractional parts. It remains to be solved to construct effective preconditioners efficiently in the privacy setting. It is also interesting to explore the possibility to apply the Schur Complement to other data mining algorithms.

Chapter 7

Conclusion

Privacy is a critically important concern in any application of computer technology and data mining in particular. The problem of prerserving privacy has attracted great interest in the communities of data mining, statistics and cryptography. My research focuses on how to protect privacy when several parties wish to conduct collaborative data mining. I am particularly interested in the design and implementation of privacy preserving distributed data mining protocols based on homomorphic encryption.

I have proposed a number of secure protocols for basic operations, including secure comparison, secure division with public divisor, secure inverse square root, secure square root, and the secure exponential function. All these protocols are implemented using secure multiplications. Using these protocols we are able to design a number of privacy preserving data mining protocols, for example, k-means and k-nearest neighbor.

In particular, we have designed and implemented privacy preserving protocols for two important data mining tasks: multiple linear regression and EM clustering. The privacy preserving multiple linear regression is based on the stable QR-decomposition method. The two-party linear regression protocol is provably secure in the semi-honest model. The two-party EM clustering protocol discloses only the number of iterations. I have implemented these protocols in C++, based on the Paillier cryptosystem. Experimental results on benchmark datasets show that privacy preserving data mining protocols are feasible for small datasets, although the computational costs are typically high, so we need to develop new techniques for them to be practical for larger datasets.

There are a number of interesting questions to be further researched. I have designed and implemented privacy preserving multiple linear regression protocol based on the Householder transformation. It would be informative to implement privacy

preserving multiple linear regression based on singular value decomposition (SVD) and the Givens rotation, respectively, and compare their performances. Another important task is to design and implement privacy preserving support vector machine (SVM). Existing protocols on privacy preserving SVM either consider only the polynomial kernel (Laur et al., 2006) or are not provably secure in the semi-honest model (Vaidya et al., 2008b; Mangasarian, 2009). It is an interesting question to design efficient provably secure SVM with various kernels.

I have explored the possibility of using the Schur Complement to design efficient privacy preserving kernel ridge regression protocol. However, it remains to be solved to construct efficient predicontioners in the privacy setting. The Schur Complement has potential applications in the design of privacy preserving protocols for other data mining tasks such as the cononical correlation analysis.

Bibliography

Rskesh Agrawal and Ramakrishnan Srikant. Privacy preserving data mining. In *Proceedings of the 19th ACM SIGMOD International Conference on Management of Data*, 2000.

James Demmel. Applied numberical linear algebra. SIAM, 1997.

Ankur Bansal, Tingting Chen and Sheng Zhong. Privacy preserving back-propagation neural network learning over arbitrarily partitioned data. In *Journal of Neural Computing and Applications*, Volume 20 Issue 1, 2011.

Roscoe A. Bartlett and Lorenz Biegler. QPSchur: A dual, active-set, Schur-complement method for large-scale and structure convex quadratic programming. In *Optimization Engineering*, Volume 7 Issue 1, 2006.

Paul S. Bradley and Usama M. Fayyad. Refining initial points for k-means clustering. In *Proceedings of the 13th International Conference on Machine Learning*, 1996.

Paul Bunn and Rafail Ostrovsky. Secure two-party k-means clustering. In *Proceedings of the 14th ACM conference on Computer and Communication Security*, 2007.

Keke Chen and Ling Liu. Privacy preserving data classification with rotation perturbation. In *Proceedings of the 5ht IEEE International Conference on on Data Mining*, 2005.

A.P Dempster, N.M Laird and D.B Rubin. Maximum likelihood from incomplete data via the EM algorithm. In *Journal of the Royal Statistical Society*, Volume 37, Issue 1, 1977.

Wenliang Du and Zhijun Zhan. Building decision tree classifier on private data. In *Proceedings of the IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, 2002.

Wenliang Du, Shigang Chen and Yunghsiang S. Han. Privacy preserving multivariate statistical analysis linear regression and classification. In *Proceedings of the 4th SIAM International Conference on Data Mining*, 2004.

Cynthia Dwork, Frank McSherry, Kobbi Nissim and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the 3rd Theory of Cryptography Conference*, 2006.

A. Frank and A Asuncion. UCI Machine Learning Repository [http://www.ics.uci.edu/~mlearn/MLRepository.html], 2007.

Strange L. From and Thomas Jakobsen. Secure Multi-Party Computation on Integers. Master thesis, University of Aarhus, Denmark, 2006.

Philip E. Gill, Walter Murray, Michael A. Saunders and Margaret H. Wright. A Schur-complement method for sparse quadratic programming. Technical Report, Stanford University CA System Optimization, 1987.

Oded Goldreich, Silvio Micali and Avi Wiigderson. How to play any mental games-a completeness theorem for protocols with honesty majority. In *Proceedings of the 19th ACM Symposium on the Theory of Computing*, 1987.

Oded Goldreich. Foundation of Cryptography, Volume 2. Cambridge University Press, 2004.

Bart Goethals, Sven Laur, Helger Lipmaa and Taneli Mielikänen. On private scalar product computation for privacy preserving data mining. In *Proceedings of the 7th International Conference on Information Security and Cryptology*, 2004.

Torbjörn Granlund et al. The GNU Multiple Precision Arithmetic Library [http://gmplib.org/].

Shuoguo Han and Wee Keong Ng. Privacy preserving linear fisher discriminant analysis. In *Proceedings of the 12th Pacific-Asia Conference on Advance in Knowledge Discovery and Data Ming*, 2008.

Shuguo Han, Wee Keong Ng and Philip S. Yu. Privacy preserving singular value decomposition. In *Proceedings of the 25th IEEE International Conference on Data Engineering*, 2009.

Rob Hall, Stephen E. Fienberg and Yuval Nardi. Secure multiple linear regression based on Homomorphic encryption. In *Journal of Official Statistics*, Volume 27, Issue 4, 2011.

Murat Kantarcioulu and Chris Clifton. Privacy preserving distributed mining of association rules on horizontally partitioned data. In *IEEE Transaction on Knowledge and Data Engiennering*, Volume 16, Issue 9, 2004

Eike Kiltz, Gregor Leander and John Malone Lee. Secure computation of the mean and related statistics. In *Proceedings of the 2nd International Conference on Theory of Cryptography*, 2005.

David Kincaid. Numerical analysis:mathematics of scientific computing. American Mathematical Society, 2002.

Sven Laur, Helger Lipmaa and Taneli Mielikaninen. Cryptographically private support vector machine. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

Ninghui Li, Tiancheng Li and Suresh Venkatasubramnian. t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In *Proceedings of the 23rd IEEE International Conference on Data Engineering*, 2007.

Hsiao-Ying Lin and Wen-Guey Tzeng. An efficient solution to the millionairs' problem based on homomorphic encryption. In *Proceedings of the 3rd International Conference on Applied Cryptography and Network Security*, 2005.

Xiaodong Lin, Chris Clifton and Michael Zhu. Privacy preserving clustering with distributed EM mixture modeling. In *Knowledge and Information System*, Volume 8 Issue 1, 2005.

Zhenmin Lin and Jerzy W. Jaromczyk. An efficient secure comparison protocol. In *Proceedings of the 12th IEEE International Conference on Intelligence and Security Informatics*, 2012.

Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Proceedings of the 20th International Cryptology Conference on Advances in Cryptology*, 2000.

Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy preserving data mining. In *Journal of Privacy and Confidentialiy*, Volume 1, Issue 1, 2009.

Kun Liu and Hillol Kargupta. Random perturbation-based multiplicative data perturbation for privacy preserving distributed data mining. In *IEEE Transaction on Knowledge and Data Engineering*, Volume 18, Issue 1, 2006.

Olvi Mangasarian. Privacy-preserving support vector machines via random kernel. In *Proceedings of the 9th IEEE International Conference on Data Mining*, 2009.

Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer and Muthuramsakrishnan Venkitasubramaniam. l-diversity: privacy beyond k-anonymity. In *Proceedings of the 22nd IEEE International Conference on on Data Engineering*, 2006.

Geoffrey Mclachlan and Thriyambakam Krishnan. The *EM* Algorithm and Extensions. Wiley, 2008.

McSherry F. Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th ACM Symposium on Foundation of Computer Science*, 2007.

Takashi Nishide and Kazuo Ohta. Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol. In *Proceedings of the 10th International Conference on Practice and Theory in Publik-key Cryptography*, 2007.

Rafail Ostrovsky, Yuval Rabani, Leonard Cschulman and Chaitanya Swamy. The effectiveness of LIoyd-Type methods for the k-means. In *Proceedings of the 47th IEEE Symposium on Foundation of Computer Science*, 2006.

Pascal Paillier. Public-Key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, 1999.

Yinian Qi and Maikhail J. Atallah. Efficient privacy preserving k-nearest neighbor search. In *Proceedings of the 28th IEEE Conference on Distributed Computing Systems*, 2008.

Adi Shamir. How to share a secret. In *Communications of the ACM*, Volume 22, Issue 11, 1979.

Ashish P. Sanil, Alan F. Karri, Xiaodong Lin and Jerome P. Reiter. Privacy preserving regression model via distributed computation. In *Proceedings of the 10th ACM SIGKDD Intenational Conference on on Knowledge Discovery and Data Mining*, 2004.

George Seber and Alan Lee, Linear Regression Analysis. Wiley, 2003.

Jonathan Richard Shewchuk. An Introduction to the conjugate gradient method without the agonizing pain, technical report. Carnegie Mellon University, 1994.

Latanya Sweeney. k-anonymity: a model for protecting privacy. In *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, Volume 10, Issue 5, 2002.

Latanya Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. In *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, Volume 10, Isuse 5, 2002.

Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining over vertically partitioned data. In *Proceedings of the 8th ACM SIGKDD International Conference On Knowledge Discovery and Data Ming*, 2002.

Jaideep Vaidya and Chris Clifton. Privacy preserving k-means clustering over vertically partitioned data sets. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

Jaideep Vaidya and Chris Clifton. Privacy preserving naive Bayesian classifier for vertically partitioned. In *Proceedings of the 4th SIAM International Conference on Data Mining*, 2004(a).

Jaideep Vaidya and Chris Clifton. Privacy preserving outliers detection. In *Proceedings of the 4th IEEE International Conference on Data Mining*, 2004(b).

Jaideep Vaidya and Chris Clifton. Secure set intersection cardinality with application to association rule mining. In *Journal of Cryptography*, Volume 13, Issue 4, 2005

Jaideep Vaidya and Chris Clifton. Privacy preserving decision tree over vertically partitioned data. In *ACM Transaction on Knowledge Discovery*, Volume 2, Issue 3, 2008(a).

Jaideep Vaidya, Hwanjo Yu and Xiaoqian Jiang. Privacy preserving SVM classification. In *Knowledge and Information System*, Volume 14, Issue 2, 2008(b).

Andrew Chi-Chih Yao. Protocols for secure computations. In *Proceedings of the 21st IEEE Symposium on Foundations of Computer Science*, 1982.

Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986.

Ningning Wu, Jing Zhang and Ning Li. Privacy preserving discovery of multivariate linear relationship. In *Journal Systemics, Cybernetics and Informatics*, Volume 4, Issue 5, 2006.

Zhiqiang Yang and Rebecca N.Wright. Privacy preserving computation of Bayesian network on vertically partitioned data. In *IEEE Transaction on Knowledge and Data Engineering*, Volume 18, Issue 9, 2006.

Zhiqiang Yang, Rebecca N. Wright and Hiranmayee Subtramaniam. Experimental Analysis of a Privacy-Preserving Scalar Product Protocol. In *International Journal of Computer System Science and Engineering*, Volume 21, Issue 1, 2006.

Fuzhen Zhang. The Schur Complement and its application, Springer, 2005.

Vita

Name: Zhenmin Lin

Education:

Master of Engineering in Computer Science, Xiamen Kentucky, China, 2002
Bachelor of Engineering in Computer Science, Xiamen University, China, 1998

Selected Publication:

1. Zhenmin Lin and Jerzy W. Jaromczyk. An efficient secure comparison protocol. In *Proceedings of the 12th IEEE International Conference on Intelligence and Security Informatics*, 2012.