A TOOLBOX FOR PRIVACY PRESERVING DISTRIBUTED DATA MINING

by

SELİM VOLKAN KAYA

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University
August 2007

A TOOLBOX FOR PRIVACY PRESERVING DISTRIBUTED DATA MINING

APPROVED BY

Assoc. Prof. Dr. Erkay Savaş ..... (Thesis Co-Supervisor)

Assist. Prof. Dr. Yücel Saygın ..... (Thesis Supervisor)

Assist. Prof. Dr. Albert Levi .....

Assist. Prof. Dr. Cem Güneri .....

Assist. Prof. Dr. Selim Balcısoy .....

DATE OF APPROVAL: .....

to My Family

&

Alkım

Acknowledgements

It is a pleasure to express my gratitude to all who made this thesis possible. I would like to thank my thesis advisors Assoc. Prof. Dr. Erkay Savaş and Assist. Prof. Dr. Yücel Saygın for their inspiration, guidance, patience, enthusiasm and motivation. I would especially like to thank Thomas B. Pedersen for being my mentor and my best friend for the last 2 years. Without their support, it would be impossible to complete this thesis. I am grateful to my family for the concern, caring, love and support they provided throughout my life.

A TOOLBOX FOR PRIVACY PRESERVING DISTRIBUTED DATA MINING

Selim Volkan Kaya

Computer Science and Engineering, MS Thesis, 2007

Supervisors: Assoc. Prof. Dr. Erkay Savaş and Assist. Prof. Dr. Yücel Saygın

Keywords: Data Mining, Cryptography, Secure Multi-party Computation, Distributed Computing, Algorithms

Abstract

Distributed structure of individual data makes it necessary for data holders to perform collaborative analysis over the collective database for better data mining results. However each site has to ensure the privacy of its individual data, which means no information is revealed about individual values. Privacy preserving distributed data mining is utilized for that purpose. In this study, we try to draw more attention to the topic of privacy preserving data mining by showing a model which is realistic for data mining, and allows for very efficient protocols. We give two protocols which are useful tools in data mining: a protocol for Yao's millionaires problem, and a protocol for numerical distance. Our solution to Yao's millionaires problem is of independent interest since it gives a solution which improves on known protocols with respect to both computation complexity and communication overhead. This protocol can be used for different purposes in privacy preserving data mining algorithms such as comparison and equality test of data records. Our numerical distance protocol is also applicable to variety of algorithms. In this study we applied our numerical distance protocol in a privacy preserving distributed clustering protocol for horizontally partitioned data. We show application of our protocol over different attribute types such as interval-scaled, binary, nominal, ordinal, ratio-scaled, and alphanumeric. We present proof of security of our protocol, and explain communication, and computation complexity analysis in detail.

MAHREMİYET KORUYUCU VERİ MADENCİLİĞİ İÇİN BİR KÜTÜPHANE GERÇEKLEMESİ

Selim Volkan Kaya

Bilgisayar Bilimi ve Mühendisliği, Yüksek Lisans Tezi, 2007

Tez Danışmanları: Doç. Dr. Erkay Savaş ve Yrd. Doç. Dr. Yücel Saygın

Anahtar sözcükler: Veri Madenciliği, Kriptografi, Güvenli Çoklu Hesaplama, Dağıtık Hesaplama, Algoritmalar

Özet

Günümüzde verilerin kurumlar arasındaki dağıtık yapısı, kurumların bu veriler üzerinde daha iyi raporlamar almaları için ortak hesaplama yapmalarını gerekli kılmıştır. Bununlar birlikte, ortak hesaplama evresinde herbir veri sahibi kurum kendi verisinin mahremiyetini sağlamalı ve hiçbir kişisel veriyi açığa çıkartmamalıdır. Mahremiyet koruyucu veri madenciliği işte bu noktada devreye girer. Bu çalışmamızda veri madenciliği için gerçekçi ve çok daha verimli işlem yapılmasına olanak sağlayacak protokoller önererek mahremiyet koruyucu veri madenciliğine dikkatleri daha fazla çekmek istedik. Bu amaçla veri madenciliği için yararlı iki farklı protokol önerisinde bulunduk. Bu protokoller Yao'nun milyonerler problemi ve sayısal fark protokolleridir. Yao'nun milyonerler problemi için önerdiğimiz method bugüne kadar aynı problem için önerilen methodlardan haberleşme ve işlem yükü açısından çok daha iyi sonuçlar vermiştir. Ayrıca bu methodun veri madenciliğinin pek çok alanında kullanımı vardır. Buna örnek olarak veri kayıtlarının karşılaştırılması ve eşitlik testi yapılması verilebilir. Önerdiğimiz ikinci method olan sayısal fark protokolünün de mahremiyet koruyucu veri madenciliğinde pek çok uygulaması vardır. Bu çalışmamızda, sayısal fark protokolümüzü yatay olarak dağıtılmış verinin mahremiyeti koruyarak gruplanması protokolüne uyguladık. Ayrıca sayısal fark protokolümüzün sıralı, sayısal, alfabetik, aralık-ölçekli ve oran-ölçekli veri tipleri üzerinde sorunsuz çalıştığını gösterdik. Buna ek olarak, sayısal fark protokolümüzün güvenli olduğunun ispatını, haberleşme ve işlem yükünü detayları ile açıkladık.

1

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Advances in data storage technologies make it possible to store and manage huge amounts of data. When combined with advanced access and processing capabilities, this provides new opportunities such as extracting new information from the stored data. Data mining techniques provide added value to data by extracting interesting and previously unknown patterns. The mined information is valuable but also sensitive from the privacy perspective since it may reveal confidential information about individuals. Therefore, data mining algorithms have to take privacy into consideration and they must guarantee that no sensitive information is retrieved without the consent of the data holder.

Privacy preserving distributed data mining is a new area of research which deserves more attention from the cryptology community. When personal data, spread out over several sites, is collected, and data mining or other analysis is performed on the joint data, the privacy of sensitive information is at risk. In recent years the data mining community has started to address these privacy issues, but no satisfactory protocols have been suggested so far.

Today personal data is spread out over several servers. Many governmental and private institutions collect data about their users and clients. In some cases it is fruitful to collect this data, and perform analysis on the union of all personal data available. In other words, many *data-holders* decide to join their data, and perform an analysis whose result is of mutual interest to the data-holders. On the other hand, each data-holder wants to protect the privacy of his clients, so he is not willing to reveal the data in his database.

Since the databases are often of considerable size, efficiency — especially in communication — is of paramount importance. Even a constant overhead of a hundred, say, is impractical if the databases contains terabytes of data.

1.1 Contributions of this Research

The goal of this study is to demonstrate that protocols with only a small constant communication and computation overhead can be made for privacy-preserving data mining. The main observation is that the use of semi-honest third parties is a realistic assumption for data mining applications. Our protocols use 2–3 semi-honest, non-colluding third parties, who receive secret shares of inputs. The data mining is performed on the secret shares as in many other multi party computation protocols. If we choose third parties who have an interest in the true result of the data mining, it is fair to assume that they behave according to the protocol. We can guarantee non-collusion by choosing third parties that have conflicting interests in the actual data. As an example one third party can be a consumer organisation who is interested in the privacy of consumers, while another third party is a representative of the industry — they both have interests in the right outcome, but will never collude. Another benefit of this model is that, while data-holders might only have limited computing power and bandwidth, third parties with high computing power and bandwidth can be chosen.

In this study, two protocols are proposed taking the assumptions above into consideration. The first protocol we propose is a numeric distance protocol. According to the protocol, taking two private numeric values as inputs, absolute value of the distance of these two numeric values is obtained without revealing none of the private inputs. As an application of our numeric distance protocol, we propose a privacy preserving distributed clustering algorithm.

The second protocol we propose is a greater-than-function protocol which answers the question 'Is X greater than Y?' without revealing private values X and Y. We show that our protocol is the most efficient approach among the other protocols proposed for the same problem. Our protocol can be applied in several privacy preserving data mining algorithms such as Yao's Millionaires' problem, equivalence test, and record matching.

CHAPTER 2

Privacy Preserving Clustering over Horizontally Partitioned Data

2.1 Introduction

Recent advances in data management technologies, especially in the directions of performance and storage capacity, cause a boost in database applications in the past decade. Every organization tries to manage their customers or members through database management systems. However plain data has no meaning in the analytical sense, and it has to be processed through some inference mechanisms. Data mining appears at that point with the promise of extracting non-trivial and sensitive information from large collections of data such as association rules, clusters and classification models. Valuable information extracted from plain data by means of data mining has a variety of application areas such as segmentation of customers for determining future marketing strategy, or analyzing associations among products with respect to buying behavior of customers for determining shelf arrangement in a supermarket.

Today individual data is distributed among several organizations, and organizations need to collaborate for better results by performing analysis on the union of all individual data available. However, privacy of individual data is important since migration of data to an organization other than the holder of that data could reveal sensitive information about each individual. Privacy preserving distributed data mining(PPDDM) is utilized for this purpose. Accordingly, PPDDM tries to produce global results from local databases without violating privacy of individuals.

Efficiency in communication and computation is crucial in PPDDM since databases are often of considerable size. Sample scenarios are sensor networks or RFID applications, where the sensor nodes or RFID readers that contain the data (data holders) have

very limited computation and communication capacity. In such scenarios, reducing the communication and computation costs is of utmost importance.

In this study we propose a new setting for privacy preserving clustering over horizontally partitioned data with only a small constant communication and computation overhead for data holders with no loss of accuracy. As stated by Inan et al.[11], we reduce privacy preserving clustering problem to privacy preserving dissimilarity matrix computation problem. After dissimilarity matrix is computed privately, it can be input to any hierarchical clustering algorithm. Our protocol uses two semi-honest, non-colluding third parties, who receive secret shares of inputs and compute intermediary results, while a data miner performs the actual clustering.

The main observation we make is that the use of semi-honest third parties is a realistic assumption for data mining applications. If we choose third parties who have an interest in the true result of the data mining, it is fair to assume that they behave according to the protocol. We can guarantee non-collusion by choosing third parties that have conflicting interests in the actual data. As an example one third party can be a consumer organization who is interested in the privacy of consumers, while another third party is a representative of the industry — they both have interests in the right outcome, but will never collude. Another benefit of this model is that, while data-holders might only have limited computing power and bandwidth, third parties with high computing power and bandwidth can be chosen. The most important benefit of our protocol is that the communication cost of all participants is linear in the size of the databases. Our protocol gives information theoretical security under the assumption that the two third parties follow the protocol, and do not collude to extract information.

2.2 Related Work and Background

The first protocols for PPDDM are proposed by Agrawal and Srikant[2],and Lindell and Pinkas[17] in 2000. In [2], Agrawal and Srikant use data perturbation for construction of a classification model privately. The basic idea is that original data values can be perturbed in such a way that original distribution of the aggregated data can be recovered but not the individual data values. Perturbation technique is efficient to implement however results in several side effects. First of all, even though the distribution of original values can be predicted with a certain confidence level, some accuracy is lost. Secondly, modification of data does not fully preserve privacy of individual

values, and may cause privacy breaches as shown in [6, 7]. Finally, perturbation has a predictable structure for certain cases and hence may not fully preserve privacy [13]. A different perturbation method is proposed by Saygin et al.[24] in 2001 for association rule hiding, where unknown values are introduced to hide sensitive association rules. As a consequence of unknown values, new association rules are created which causes computation overhead, and some insensitive rules present before perturbation process are lost which causes accuracy lost.

[17] employs cryptography as its main tool and implements a decision tree learning protocol. However oblivious transfer, which is the main building block of this protocol, causes huge computation and communication overhead due to exponentiation operations for each bit of private inputs and expansion of each bit of private data as a result of exponentiation respectively. [12] proposes a privacy preserving association rule mining protocol over horizontally partitioned data taking advantage of commutative encryption. Nevertheless the protocol requires encryption and decryption operations to be performed over each private input by all of the participants resulting in a large communication and computation cost.

Several protocols are proposed for privacy preserving clustering. Oliveira and Zaiane [19] introduce geometric data transformation methods(GDTMs) to distort confidential data values. The protocol tries to preserve main features of the confidential data for clustering while perturbing the data to meet privacy requirements. However, perturbation causes accuracy losses in clustering, and privacy of the data is not fully guaranteed. Consequently, Oliveira and Zaiane [20] introduce the notion of Rotation-Based Transformation(RBT). RBT provides confidentiality of attribute values while completely preserving the original clustering results. However RBT method has a computation overhead since attribute values are transformed pairwise, and selection of attribute pairs should be done in such a way that variance between the original and transformed attributes are maximum. In [21], Oliveira and Zaiane propose Object Similarity-Based Representation(OSBR) and Dimensionality Reduction-Based Representation(DRBT) methods for clustering over centralized and vertically partitioned databases. Therefore, OSBR has high computation cost since each data owner sends a dissimilarity matrix to a central party yielding a communication complexity of $O(n^2)$, while DRBT can cause loss of accuracy due to dimensionality reduction in the original data.

Merugu and Ghosh [18], and Klusch, Lodi and Moro [14] propose privacy preserving

clustering methods based on sharing models representing the original data instead of sharing the original data itself. Accordingly, clustering can be performed over the model without revealing the original data points. However clustering over low quality representatives of the original data causes loss of accuracy while efforts for high quality representatives means loss of privacy.

Vaidya and Clifton [26] propose a privacy preserving k-means clustering protocol based on secure multi-party(SMC) computation. Nevertheless there is a huge communication and computation cost due to iterative execution of several SMC protocols till a convergence point for the clusters is obtained. Jha, Kruger and McDaniel propose two privacy preserving k-means clustering protocols for horizontally partitioned data in [23]. The protocols use homomorphic encryption and oblivious polynomial evaluation as their building block which are inefficient to be applied over large databases due to cost of modular exponentiation and oblivious transfer respectively.

The most recent study for privacy preserving clustering is proposed by Inan et al. [11] over horizontally partitioned data and the problem is reduced to secure computation of dissimilarity matrix which will be input to any clustering algorithm but k-means. Each entry of the dissimilarity matrix is computed by a secure difference protocol where confidential data points are disguised by pseudo-random values and the disguise is removed by a trusted third party revealing the final difference. However secure difference protocol leads to privacy breaches because of the way pseudo-random values are used. According to the secure difference protocol, initiator of the protocol creates two disguise factors; one for follower of the protocol to disguise initiators value and the other for the trusted third party to disguise which participants input is subtracted from the other. Nevertheless the latter disguise factor is the same for each entry point within a row of dissimilarity matrix. In other words, trusted third party can guess which site's input is subtracted from the other with a probability of $\frac{1}{2}$ for each row. On the other hand, quadratic communication cost for dissimilarity matrix computation is a huge burden for data holders.

2.3 Preliminaries

In our scenario we have ℓ *data holders*: $DH_1, \ldots, DH_\ell$ where DH_i has a database with n_i objects: $o_1^i, \ldots, o_{n_i}^i$. The databases all have the same *[schema]* with m integer attributes (from a finite field). Since all databases have the same schema, we can write

the union of the databases as $o_1, o_2, \ldots, o_N$, where $N = \sum_{i=1}^{\ell} n_i$, and where object o_i has attributes $a_1^i, \ldots, a_m^i$. We say that the collective database is *horizontally partitioned* between the ℓ data holders.

The goal of our protocol is to compute the dissimilarity matrix of all objects in all databases, while keeping the actual values secret. Each entry of the dissimilarity matrix contains the weighted Manhattan distance between two elements from the collective database.

$$D_{ij} = \sum_{k=1}^{m} w_k |a_k^i - a_k^j|, \tag{2.1}$$

where $i, j = 1, \ldots, N$, and $w_1, \ldots, w_m$ are predefined weights. We introduce the notion of *partial dissimilarity matrices* which contains the numerical distance between a single attribute, so that the dissimilarity matrix can be written

$$D = \sum_{k=1}^{m} w_k D^k, \tag{2.2}$$

where D^k is the dissimilarity matrix with entries $D^k[i, j] = |a_k^i - a_k^j|$ which results from considering only the kth attribute.

2.3.1 Homomorphic Secret Sharing

Informally secret sharing is a way to share a secret among m players in a way that $t - 1$ or less colluding players cannot compute any information about the secret, but t arbitrary players can recover the secret. A player that wishes to share his secret s will create m *secret-shares* $s_1, \ldots, s_m$ and send one share to each player [3, 25].

The protocols we present in this study rely on additive secret sharing. To share a secret integer[1] s between two players, we choose a random integer r and give the share r to the first player and the share $s - r$ to the second player. Clearly both shares are random when observed alone, so no single player can compute any information about the secret. The secret is revealed by simply adding the two shares together, so the two players can recover the secret together.

A secret sharing scheme is said to be homomorphic with respect to a binary operation $\cdot$ if there is a binary operation $\star$ such that $c_i = a_i \star b_i, i = 1, \ldots, m$ are secret shares of the secret $a \cdot b$, when a_i, b_i are secret shares of a and b respectively.

Additive secret sharing is homomorphic with respect to addition: adding shares

[1] Or more precisely: to share an element from an additive group.

pairwise, gives an additive sharing of the sum of the secrets.

2.4 Our Protocol

There are two challenges for designing a protocol for computing Manhattan distance:(1)not to reveal private inputs, (2)to hide which input is the largest. We employ additive homomorphic secret sharing to fulfill the first challenge, with a very small communication and computation overhead for the data holders. The inputs are shared between two semi-honest non-colluding third parties, TP_1 and TP_2, who can compute a secret sharing of the difference between by using the homomorphic property. To avoid revealing the sign of the difference (which input is larger), TP_1 and TP_2 share a pseudo random number generator. Before the protocol starts TP_1 and TP_2 will each fill a $m \times N \times N$ table, PRNG, with one bit values(either 0 or 1) from the pseudo random number generator initialized with a shared seed.

Let a_k and b_k be the private values for the kth attribute of o_i^A and o_j^B held by DH_A and DH_B respectively. The (i, j)th entry in the D^k is $|a_k - b_k|$. To compute this Euclidean distance DH_A selects a random number α_k, and sends additive shares α_k and $a_k - \alpha_k$ to third party 1 (TP_1) and 2 (TP_2) respectively. Likewise DH_B creates additive sharing β_k and $b_k - \beta_k$ and sends them to TP_1 and TP_2 respectively. TP_1 computes $sh_1 = (-1)^{\text{PRNG(k,i,j)}}(\alpha_k - \beta_k)$ and TP_2 computes $sh_2 = (-1)^{\text{PRNG(k,i,j)}}((a_k - \alpha_k) - (b_k - \beta_k))$, and they send the results to the miner DM. When DM adds the two received values the result is

$$sh_1 + sh_2 = (-1)^{\text{PRNG(k,i,j)}}(a_k - b_k). \tag{2.3}$$

After receiving the numerical value the miner gets the results $|sh_1 + sh_2| = |a - b|$, which is the required (i, j)th entry of D^k. Overview of our Euclidean distance protocol is depicted in Figure 2.1.

To construct the dissimilarity matrix for the kth attribute, each data holder DH_i computes additive shares of their private values $a_k^1, a_k^2 \ldots a_k^{n_i}$. The resulting additive shares of each private value are distributed to secret share arrays $s_1^{i,k}$ and $s_2^{i,k}$. The resulting secret share arrays $s_1^{i,k}$ and $s_2^{i,k}$ are sent to TP_1 and TP_1 respectively. Steps of the protocol for data holders are demonstrated in Algorithm 1.

Receiving $s_{1(2)}^{1,k}, s_{1(2)}^{2,k}, \ldots, s_{1(2)}^{\ell,k}$ from all of the data holders, $TP_{1(2)}$ merges these arrays into $s_{1(2)}^k$. After merge operation, $s_{1(2)}^k$ contains additive shares of the collective

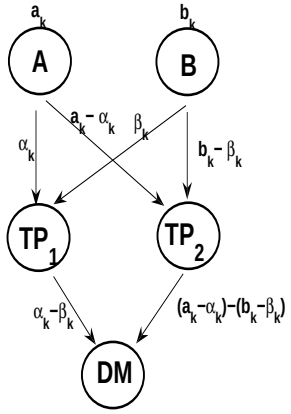

Figure 2.1: Overview of the numerical distance protocol

Algorithm 1 DH_i

Input: private values for attribute k: $a_k^1, a_k^2 \dots a_k^{n_i}$
Output: secret share arrays $s_1^{i,k}$ and $s_2^{i,k}$

1: Initialize secret share arrays $s_1^{i,k}$ and $s_2^{i,k}$ of size n_i
2: **for** $j = 1$ to n_i **do**
3: $\quad (s_1^{i,k}[j], s_2^{i,k}[j]) = secretshare(a_k^j)$
4: **end for**
5: Sends $s_1^{i,k}$ to TP_1
6: Sends $s_2^{i,k}$ to TP_2

database for the kth attribute. Then $TP_{1(2)}$ initializes an $N \times N$ matrix $D_{1(2)}^k$ and fills each entry (i,j) with value $(-1)^{\text{PRNG[k,i,j]}}(s_{1(2)}^k[a] - s_{1(2)}^k[b])$. The resulting matrix $D_{1(2)}^k$ is additive share of D^k. $TP_{1(2)}$ sends $D_{1(2)}^k$ to DM. The details of the protocol for TP_1 are depicted in Algorithm 2.

Algorithm 2 TP_1

Input: Secret share arrays $s_1^{1,k}, s_1^{2,k}, \dots, s_1^{\ell,k}$, matrix PRNG shared with TP_2
Output: Secret share matrix D_1^k

1: Initialize secret share array s_1^k of size $N = \sum_{i=1}^{\ell} n_i$
2: Initialize secret share matrix D_1^k of size $N \times N$
3: Merge $s_1^{1,k}, s_1^{2,k}, \dots, s_1^{\ell,k}$ into s_1^k
4: **for** $a = 1$ to N **do**
5: $\quad$ **for** $b = 1$ to N **do**
6: $\quad\quad D_1^k[a,b] = (-1)^{\text{PRNG[k,a,b]}}(s_1^k[a] - s_1^k[b])$
7: $\quad$ **end for**
8: **end for**
9: Sends D_1^k to **DM**

It is trivial for DM to construct D^k from matrices D_1^k and D_2^k by simply computing $D_1^k[i,j] + D_2^k[i,j]$ for each entry (i,j) of D^k. The protocol for DM is depicted in Algorithm 3.

When all m dissimilarity matrices have been computed, DM can compute the final

Algorithm 3 DM

Input: Secret share matrices D_1^k and D_2^k
Output: D^k

```
1: Initialize secret share matrix D^k of size N × N
2: for a = 1 to N do
3:    for b = 1 to N do
4:       D^k[a, b] = D_1^k[a, b] + D_2^k[a, b]
5:    end for
6: end for
```

dissimilarity matrix with the sum in Equation 2.2.

2.4.1 Application of Our Protocol to Different Data Types

As stated in [10], an object can be described by attributes of five different data types:(1) Interval-Scaled, (2) Binary, (3) Nominal, (4) Ordinal, and (5) Ratio-Scaled. In this section, we show how to apply our protocol for these data types.

1. Interval-Scaled attributes: These are attributes of continuous value from a linear scale like age, weight, and height. Our protocol can directly be applied to interval-scaled variables since this attribute type has numeric values.

2. Binary attributes: This attribute type has two values: 0 or 1. 0 means that attribute is absent, and 1 means that it is present. For example, attribute *married* is a binary attribute with values *Yes(1)*, and *No(0)*. We can easily adopt our protocol for a binary attribute k by treating values of k as 0 and 1. As a result, $D^k[x, y]$ will be 0 if $a_k^x = a_k^y$, and 1 otherwise.

3. Nominal attributes: Nominal attributes resemble binary attributes, however can take on more than two states. For instance, attribute *weather* is nominal with states *sunny*, *windy*, *cloudy*, and *rainy*. Application of our protocol to a nominal attribute is as follows: If number of all possible states for a nominal attribute k is m, then we can number each attribute value from range $1, 2, \ldots, m$. After computing D^k, non-zero entries of D^k are set to 1.

4. Ordinal attributes: Ordinal attributes are similar to nominal attributes, however states of ordinal attributes are ordered. Attribute *professional rank* has values ordered as *assistant*, *associate*, and *full*. To adopt a nominal attribute k with m states to our protocol, each state is numbered from range $1, 2, \ldots, m$, where states with higher rank get greater numbers. Then we can treat ordinal attribute as interval-scaled.

5. Ratio-Scaled attributes: These are attributes of continuous value from a nonlinear scale like exponential scale. Growth of a bacteria population is a typical example for ratio-scaled attributes. A Ratio-Scaled attribute k can easily be adopted to our protocol by employing logarithmic transformation such as each attribute value a_k^i is replaced with $\log a_k^i$. The updates attribute values are treated as interval-scaled attributes.

6. Alphanumeric attributes: These are sequences(strings) of characters from a given alphabet. Alphanumeric attributes are largely used by bioinformatics. For instance, DNA sequence data is an alphanumeric attribute where alphabet of the attribute is a,c,g,t. Edit distance[15] is a widely used notion to measure similarity of two strings with respect to insertions, deletions, and substitutions required to transform one string to another. For application of an alphanumeric attribute k to our protocol, each alphanumeric attribute value a_k^i needs to treated as an array of characters from a finite alphabet and each character is numbered like ordinal attributes. For instance; alphabet of a,c,g,t for DNA data is mapped to the values 0,1,2,3 respectively. Then secret sharing of characters for each attribute a_k^i is computed by data holders, and secret shares are sent to trusted third parties. Trusted third parties form matrices which includes secret shares of difference of each character of an attribute to other attributes' characters. DM forms the original difference matrix by simply adding these two matrices. At that point, as Inan et al. proposed in [11], Character Comparison Matrix(CCM) is utilized, where each entry (s,t,i,j) of CCM is filled as the the following: $CCM[s][t][i][j] = 0$ if ith character of a_k^s is equal to jth character of a_k^t, and $CCM[s][t][i][j] = 1$ otherwise. The final CCM is input to *editdistance* algorithm to form the final dissimilarity matrix. The details of the protocol for alphanumeric attributes are depicted in Algorithm 4,5, and 6 for DH_i, TP_1, and DM respectively.

2.5 Security of our Protocol

Our security definitions reflects that no (or at least not more than a negligible amount of) information is revealed about *any* object in the collective database during the data-mining protocol. Of course the final result of the protocol will on it's own reveal partial information, but information leakage is limited to whatever can be deduced from the final result. In our protocol the partial dissimilarity matrices are computed

Algorithm 4 DH_i for alphanumeric attributes

Input: private values for attribute k: $a_k^1, a_k^2 \dots a_k^{n_i}$
Output: secret share arrays $s_1^{i,k}$ and $s_2^{i,k}$

Initialize secret share matrices $s_1^{i,k}$ and $s_2^{i,k}$ of size $n_i \times len$ where $len = max(a_k^1.length, a_k^2.length \dots a_k^{n_i}.length)$
for $j = 1$ to n_i **do**
  for $l = 1$ to $a_k^j.length$ **do**
    $(s_1^{i,k}[j][l], s_2^{i,k}[j][l]) = secretshare(a_k^j[l])$
  end for
end for
Sends $s_1^{i,k}$ to TP_1
Sends $s_2^{i,k}$ to TP_2

Algorithm 5 TP_1 for alphanumeric attributes

Input: Secret share matrices $s_1^{1,k}, s_1^{2,k}, \dots, s_1^{\ell,k}$, matrix PRNG shared with TP_2
Output: Secret share matrix D_1^k

Initialize secret share matrix s_1^k of size $N \times len$ where $N = \sum_{i=1}^{\ell} n_i$ and $len = max(s_1^{1,k}[0].length, s_1^{2,k}[0].length, \dots, s_1^{\ell,k}[0].length)$
Initialize secret share matrix D_1^k of size $N \times N \times len \times len$
Merge $s_1^{1,k}, s_1^{2,k}, \dots, s_1^{\ell,k}$ into s_1^k
for $a = 1$ to N **do**
  for $b = 1$ to N **do**
    for $c = 1$ to $s_1^k[a].length$ **do**
      for $d = 1$ to $s_1^k[b].length$ **do**
        $D_1^k[a, b, c, d] = (-1)^{\text{PRNG[k,a,b,c,d]}}(s_1^k[a, c] - s_1^k[b, d])$
      end for
    end for
  end for
end for
Sends D_1^k to **DM**

Algorithm 6 DM for alphanumeric attributes

Input: Secret share matrices D_1^k and D_2^k
Output: D^k

Initialize matrix CCM of size $N \times N \times len \times len$
Initialize secret share matrix D^k of size $N \times N$
for $a = 1$ to N **do**
  for $b = 1$ to N **do**
    for $c = 1$ to $D_1^k[a][b].length$ **do**
      for $d = 1$ to $D_1^k[a][b][c].length$ **do**
        if $D_1^k[a, b, c, d] + D_2^k[a, b, c, d] == 0$ **then**
          $CCM[a, b, c, d] = 0$
        else
          $CCM[a, b, c, d] = 1$
        end if
      end for
    end for
    $D^k[a, b] = editdistance(CCM[a, b])$
  end for
end for

and revealed to a third party. In general even the information given in the partial dissimilarity matrices is too much, but we leave it for further improvements to complete the data-mining without revealing any intermediate results. *[See Section of future work for a discussion on how this can be done.]*

Definition A protocol for computing partial dissimilarity matrices is ϵ-*secure* if for all parties, and for all attributes A_j^i

$$\left|P[A_k^i = x|D^k, M] - P[A_k^i = x|D]\right| < \epsilon, \tag{2.4}$$

where M is a transcript of all messages send to a given party.

Theorem 2.5.1 *The protocol is private.*

Proof Since data holders never receive any information, Equation (2.4) is satisfied for these parties. Since blinding factors α_i are chosen randomly and independent, Equation (2.4) is also satisfied for TP_1. Since attributes a_i are chosen from finite fields $a_i - \alpha_i$ are also independent of the data, so Equation (2.4) is also satisfied for TP_2. The values received by DM enables DM to build D, where each entry has a random sign depending on RPNG. If RPNG is secure, no additional information can be computed. ∎

2.6 Complexity Analysis

In this section, we analyze computation and communication complexity of our protocol. Each analysis will be performed for DHs, TPs, and DM separately. We show effect of different data types to the complexity of our protocol. Since interval-scaled, binary, nominal, ordinal, and ratio-scaled attributes are treated as numbers as shown in Subsection 2.4.1, complexity of our protocol for these data types are the same. For that reason, we denote these data types as *numeric* attributes throughout our analysis. Therefore complexity analysis of our protocol is given with respect to alphanumeric and numeric attributes. We also show complexity analysis of the privacy preserving clustering protocol proposed by Inan et al. [11] to be able to make comparison with our protocol.

2.6.1 Computation Complexity

Since computation of secret shares of private inputs can be performed in parallel by each DH, computation complexity of our protocol for DHs is $O(n_{max})$ for numeric attributes, where $n_{max} = max(n_1, n_2, \ldots, n_\ell)$, and $O((n \times len)_{max})$ for alphanumeric attributes, where $(n \times len)_{max} = max(n_1 \times len_1, n_2 \times len_2, \ldots, n_\ell \times len_\ell)$ and len_i is the average length of alphanumeric attributes for DH_i. On the other hand, for [11] computation complexity of DHs is $O(N^2)$ for numeric attributes since data holders compute shared dissimilarity matrices pairwise which requires serial execution. For alphanumeric attributes, complexity of [11] is $O(N^2 \times length^2)$, where $length$ is the average length of alphanumeric attributes for the collective database.

In our protocol, for TPs, computation of secret share of D_k yields complexities of $O(N^2)$ for numeric, and $O(N^2 \times length^2)$ for alphanumeric attributes, which is due to computation of the global dissimilarity matrix, if we assume TPs operate in parallel and RPNG is generated in advance. In [11], there is only one TP and computation complexity of TP is $O(N^2)$ for numeric, and $O(N^2 \times length^2)$ for alphanumeric attributes.

Complexity of our protocol for DH is $O(N^2)$ for numeric attributes which is the cost of computing D^k. For alphanumeric attributes, DH computes CCM and D^k resulting in a complexity of $O(N^2 \times length^2)$. There is no DM in [11]. We summarize computation complexities of the protocols for each party in Table 3.1.

Table 2.1: Computation Complexities of our Protocol and [11]

Attribute Type	**DH**	**TP**	**DM**
Numeric	$O(n_{max})$	$O(N^2)$	$O(N^2)$
Numeric for [11]	$O(N^2)$	$O(N^2)$	–
Alphanumeric	$O((n \times len)_{max})$	$O(N^2 \times length^2)$	$O(N^2 \times length^2)$
Alphanumeric for [11]	$O(N^2 \times length^2)$	$O(N^2 \times length^2)$	–

2.6.2 Communication Complexity

In our protocol, each DH sends secret shares of their private inputs to TPs resulting in a total communication complexity of $O(N)$ and $O(N \times length)$ for numeric and alphanumeric attributes respectively. TPs send secret shares of D^k to DM and the total communication complexity is $O(N^2)$ for numeric attributes, and $O(N^2 \times length^2)$ for alphanumeric attributes. Since final clustering is done by DM, there is no further communication cost.

On the other hand, in [11], each DH sends local and shared dissimilarity matrices

to TP where global dissimilarity matrix is computed. Accordingly, communication complexity is $O(N^2)$ for numeric attributes, and $O(N^2 \times length^2)$ for alphanumeric attributes. Summary of the communication complexity analysis is depicted in Table 3.2.

Table 2.2: Communication Complexities of our Protocol and [11]

Attribute Type	**DH**	**TP**	**Total**
Numeric	$O(N)$	$O(N^2)$	$O(N^2)$
Numeric for [11]	$O(N^2)$	–	$O(N^2)$
Alphanumeric	$O(N \times length)$	$O(N^2 \times length^2)$	$O(N^2 \times length^2)$
Alphanumeric for [11]	$O(N^2 \times length^2)$	–	$O(N^2 \times length^2)$

2.7 Implementation and Performance Evaluation

In this chapter, performance evaluation of our protocol is explained and discussed in detail with comparison to the protocol proposed in [11]. Our distributed clustering protocol and [11] do not result in any loss of accuracy. Therefore, we perform only two tests: communication cost analysis and computation cost analysis. The experiments are conducted on an Intel Dual-Core Centrino PC with 2 MB cache, 2 GB RAM and 1.83 GHz clock speed. We used C# programming language to implement the algorithms.

2.7.1 Experimental Setup

To measure the performance of our protocol and [11], three test cases are identified. These test cases are for different values of the following entities:

1. Total number of entities (total database size)
2. Average length of alphanumeric attributes
3. Number of data holders

To show performance of our protocol over different attribute types, each test case is performed over numeric and alphanumeric attributes. For numeric attributes, we use two different data types: integer and double. Since test results for integer and double valued attribute values are similar, only test results for double data type are included due to space consideration.

For each experiment, we measure the communication and computation overhead of our protocol against the protocol proposed in [11], where each attribute value is blinded by random disguise factors which are removed at the end revealing the final result. [11] and our protocol only differ in the formation of the global dissimilarity matrix. After global dissimilarity matrix is formed, a clustering algorithm takes this matrix as input and the clustering is performed the same way in both protocols. Therefore, comparisons of these protocols in the experiments are done with respect to formation of the global dissimilarity matrices and clustering is not taken into consideration. For all the experiments, we denote our protocol as "Our protocol" and [11] as "protocol" in the figures. Except for the experiments on the number of data holders, we partitioned the generated datasets into four by distributing them into four datasets evenly so that each data holder has a balanced share.

For test case (1), we used total database sizes of 2K, 4K, 6K, 8K and 10K. Test case (2) shows the behavior of the baseline protocol and our protocol for varying average lengths of the alphanumeric attribute which are 5, 10, 15, 20, and 25. In test case (3), number of data holders, excluding the third party, is 2, 4, 6, 8, and 10.

For each test case, we first use synthetically generated datasets. Synthetic datasets are more appropriate for our experiments since we try to evaluate scalability and efficiency of our protocol for varying parameters, and synthetic datasets can be generated by controlling the number of entities, number of data holders, and average length of attributes. Data generator is developed in Eclipse Java environment. For the numeric attributes, each entity is chosen from the interval [0, 10000] uniformly, where precision for double data type is set to three. For alphanumeric attributes, we created sequence of characters whose length is chosen in accordance with normal distribution, the mean value being equal to average length of the attribute and alphabet size is equal to four. The reason for choosing alphabet size as four is to imitate behavior of DNA data in our experiments.

We also use KDD'99 Network Intrusion Detection stream dataset [1] to show the performance of our protocol over real datasets. We chose 'src_bytes' attribute of this dataset as our target attribute, which is numeric. To make tests over real numeric datasets compatible with tests over synthetic datasets, we divide real datasets into datasets of size 2K, 4K, 6K, 8K and 10K.

In our experiments, we use Advanced Encryption Standard (AES) cipher to generate pseudo-random numbers to hide data holders' inputs. We use Cryptography namespace

of MS .Net platform to perform AES encryption in the implementation of our protocol and [11]. For our protocol, keys and initialization vectors (IVs) are chosen by each data holder independent of the others, while for [11], seeds for pseudo-random number generator shared between data holders are used as keys and an initialization vector (IV) globally known to every data holder is used. Ciphertext as a result of encryption of IV by AES key is used as the pseudo-random number. For the next random number generation, random number (ciphertext) generated in the previous step is used as the message (plaintext) to be encrypted which yields the next random number as a result. In our implementation, we use 128 bits AES encryption. We preferred AES for sake of simplicity and safety. Nevertheless, a faster PRNG based on a stream cipher (such as SEAL) can also be used to decrease the overhead in the computation complexity of the proposed protocol.

2.7.2 Computation Cost Analysis

Comparison of computation costs for our protocol and [11] for varying database sizes from $2K$ to $10K$ is depicted in Figure 2.2. For numeric attributes, Figures 2.2(a) and 2.2(b) show that both our protocol and [11] behave quadratically which is due to formation of global dissimilarity matrix. However our protocol performs better than [11] since data holders operate in parallel in our protocol and the overall computation cost for each data holder is n AES encryption for computing secret shares of the data, where n is database size of each data holder. However, [11] performs n AES encryptions at each data holder to disguise data values and n AES encryptions at TP to remove these disguise factors. As a result, [11] performs $k * n$ AES encryptions more than our protocol where there are k data holders. As Figures 2.2(a) and 2.2(b) show, execution time difference between our protocol and [11] gets larger as database size increases since number of AES encryptions performed at TP also increases. On the other hand, in our protocol no encryption is performed by any party other than data holders. Comparing performance results of numeric attributes for real and synthetic datasets, execution time for real dataset is slightly greater than execution time of synthetic dataset due to implementation of our protocol since dissimilarity matrices are formed in double data type which requires conversion of real datasets from type integer to type double.For alphanumeric attributes, the situation is similar to numeric attributes. However for alphanumeric attributes the difference in execution times are larger than numeric attributes since extra number of AES encryptions that have to be

performed is $n * l$ where l is the average length of alphanumeric attribute.

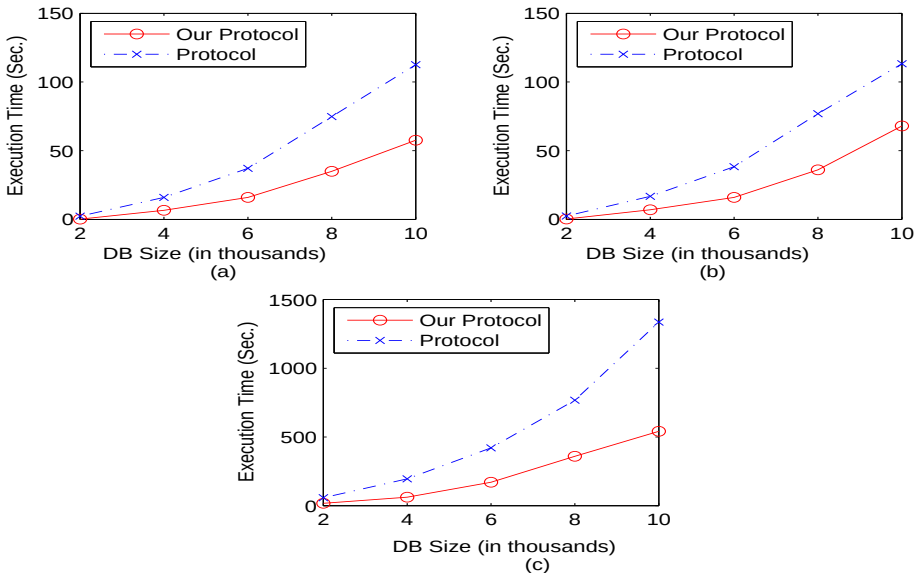

Figure 2.2: Computation cost for different database sizes: (a) For numeric attribute from synth. dataset. (b) For numeric attribute from real dataset. (c) For alphanumeric attributes.

Our protocol outperforms [11] when number of data holders increases. For this experiment, we generate a dataset of 10K entities and then horizontally partitioned this dataset by distributing the complete dataset over the data holders so that each party holds the same number of entities. As depicted in Figure 2.3, execution time for [11] increases as number of data holders increases. This is due to the fact that, C_2^k number of pairwise computation between data holders have to be performed to compute shared dissimilarity matrices where k is total number of data holders. However, increase in total execution time for [11] gets smaller as number of data holders increases since amount of data owned by each data holder gets smaller. On the other hand, increase in number of data holders reduces total execution time for our protocol since share of each data holder gets smaller which means less encryption is performed by data holders in parallel. In our protocol, the computation cost of TPs and DM does not effect from the change in number of data holders.

To measure the relation between the length of alphanumeric attributes and the execution time, we generate 6K alphanumeric entities with varying average lengths. The total execution times of the protocols are depicted in Figure 2.4. Accordingly, execution times of the protocols with increasing average attribute length increase quadratically as expected for our protocol and [11]. The execution time difference between our protocol and [11] can be explained with the same reasoning as Figure 2.2(c).

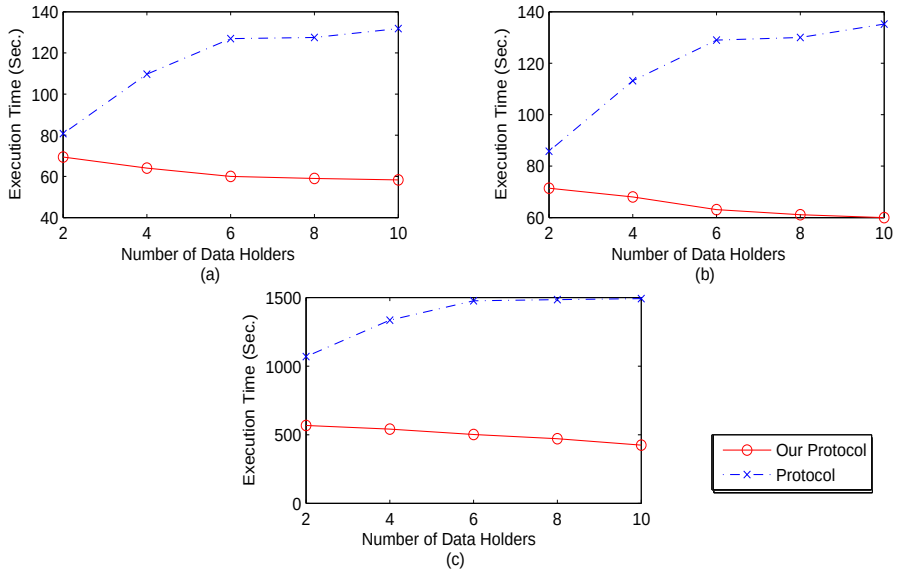

Figure 2.3: Computation cost for different number of data holders: (a) For numeric attribute from synth. dataset. (b) For numeric attribute from real dataset. (c) For alphanumeric attributes.

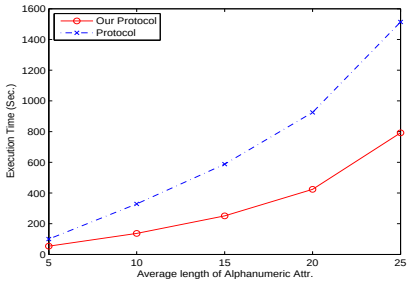

Figure 2.4: Computation cost for different average alphanumeric attribute lengths

2.7.3 Communication Cost Analysis

Overall communication costs of our protocol and [11] for various database sizes are depicted in Figure 2.5. As seen in the figures, overall communication cost of our protocol is almost twice of [11] for both numeric and alphanumeric attributes. This is due to secret sharing employed in our protocol where two shares are created for each entity. Both our protocol and [11] behave quadratically since overall communication cost is dominated by communication cost of dissimilarity matrices. Figures 2.5(a) and 2.5(b) also show that communication cost for synthetic dataset is larger than real dataset since synthetic dataset is in double data format stored in 64 bits while real dataset is in integer data format stored in 32 bits. On the other hand, apart from the overall communication cost, communication cost of data holders for our protocol and [11] is depicted in Figure 2.6. As shown in Figure 2.6, communication cost of our protocol for data holders is linear in the size of each data holders dataset while [11] requires quadratic communication for data holders. For that reason, communication cost of our protocol for data holders is negligible compared to [11]. Accordingly, our protocol puts the communication burden over trusted third parties and requires negligible amount of communication from data holders which are resource limited. On the other hand [11] requires all the communication performed by data holders. However our protocol is more appropriate for the real life situation as seen from the example given in Section 3.1.

Analysis of overall communication costs for different number of data holders is depicted in Figure 2.7. A dataset containing 10K entities is evenly distributed among data holders in these tests. As the figures show, communication cost of our protocol remains the same for different number of data holders since collective database size is the same. On the other hand, [11] shows an increase in communication cost when number of data holders increase. However the amount of increase in communication cost gets smaller as number of data holders increase, due to the same reasoning as in Figure 2.3. As the figure shows, overall communication cost of our protocol is more than [11]. However when communication costs of data holders are compared, our protocol outperforms [11] as shown in Figure 2.8. The same reasoning as in Figure 2.6 is also applicable to Figure 2.8.

Figure 2.9 depicts the relation between communication cost and average length of alphanumeric attributes for both protocols. As seen in the figure, both protocols behave quadratically with respect to increase in the average length of alphanumeric attributes.

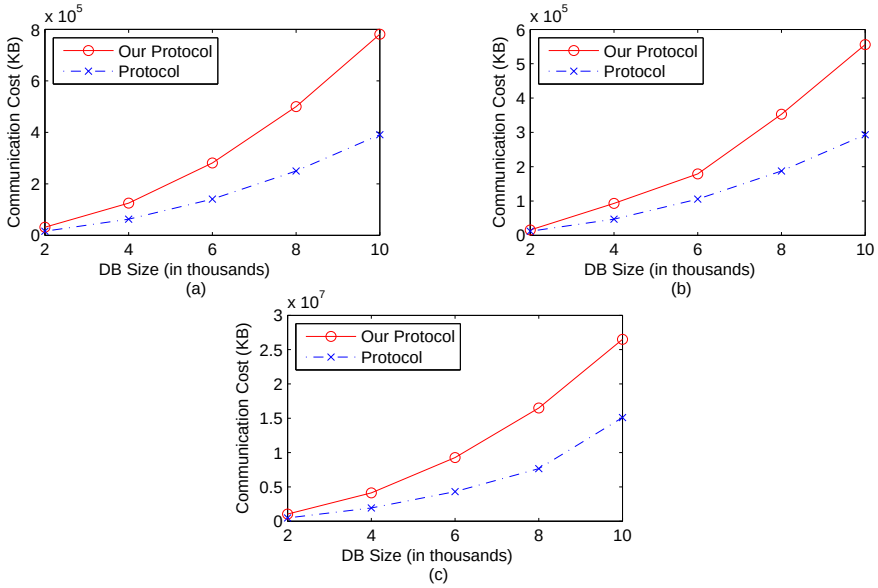

Figure 2.5: Overall communication cost for different database sizes: (a) For numeric attribute from synth. dataset. (b) For numeric attribute from real dataset. (c) For alphanumeric attributes.

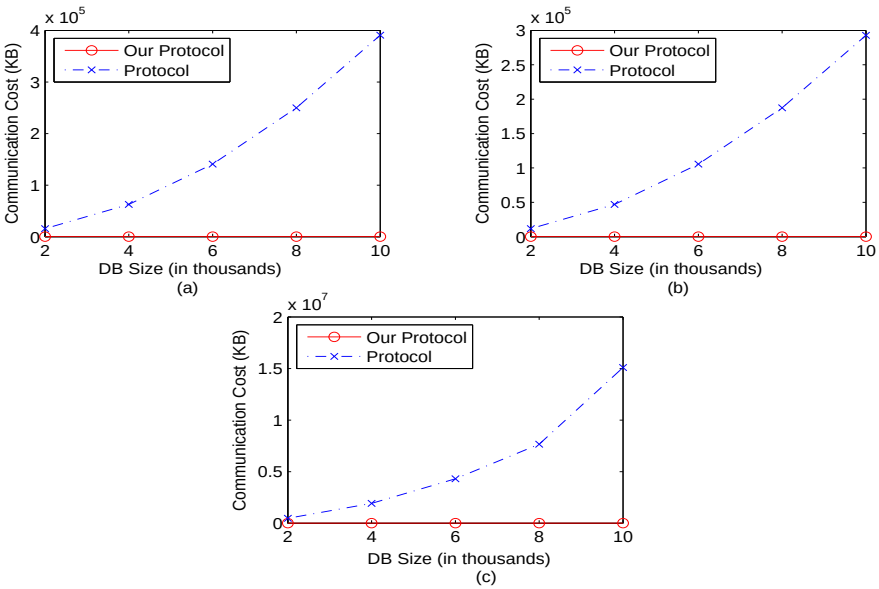

Figure 2.6: Communication cost of data holders for different database sizes: (a) For numeric attribute from synth. dataset. (b) For numeric attribute from real dataset. (c) For alphanumeric attributes.

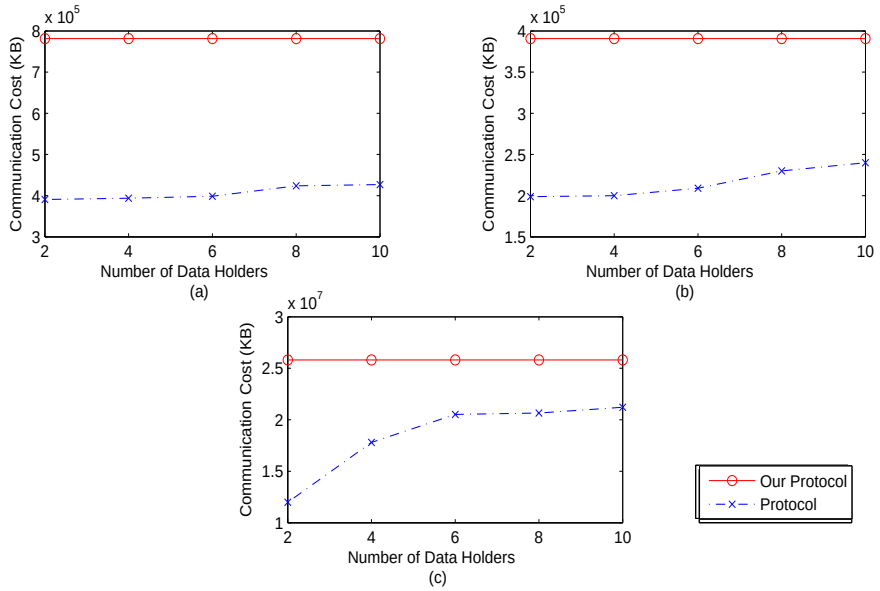

Figure 2.7: Overall communication cost for different numbers of data holders: (a) For numeric attribute from synth. dataset. (b) For numeric attribute from real dataset. (c) For alphanumeric attributes.

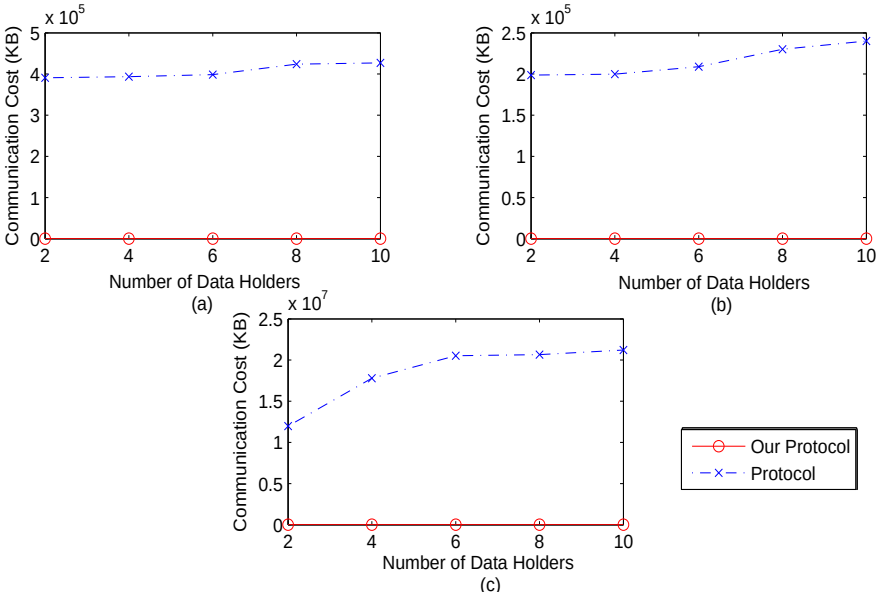

Figure 2.8: Communication cost of data holders for different numbers of data holders: (a) For numeric attribute from synth. dataset. (b) For numeric attribute from real dataset. (c) For alphanumeric attributes.

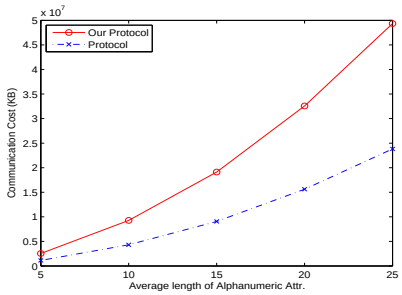

Figure 2.9: Overall communication cost for different average alphanumeric attribute lengths

However, the amount of increase in the communication cost is higher for our protocol than [11] due to redundant communication caused by secret sharing.

2.8 Discussion

In this section, we discuss the advantages of our protocol against [11] proposed by Inan et al. There are two reasons why we choose [11] for comparison. Firstly, [11] is the most recently proposed privacy preserving clustering protocol. Secondly, [11] has the most similar structure to our protocol which provides fairness throughout our discussion.

[11] separates data holders into two: initiators and followers. Initiator i starts secure difference protocol by sending its disguised inputs. Follower receives the disguised values and computes difference of each of its input to i's input. The main problem with this setting is synchronization of data holders. In other words, follower has to be idle and ready for computation when an initiator sends its input, which is hard to manage in large distributed systems. On the other hand, our protocol requires no interaction between data holders which means no synchronization requirements.

Dissimilarity matrices are computed in terms of local and shared dissimilarity matrices in [11], where DM computes the final dissimilarity matrix by merging these local and shared dissimilarity matrices received from data holders. This structure causes quadratic computation and communication complexities for data holders with respect to size of data holders' local database. However it is more realistic to assume that data holders have limited computation capabilities and to leave computation of dissimilarity matrix to trusted third parties with high computation power, which is the case in our protocol. Accordingly, computation and communication complexities of our protocol for data holders are linear with the size of data holders' databases.

Another problem with [11] is that, each initiator-follower and initiator-DM pairs has to share a pseudo-random number generator seed which brings forward the problem of seed generation and deployment in large distributed environments. However in our protocol, merely one pseudo-random number generator seed between the two-trusted third parties is shared.

In [11], the pseudo-random numbers used to disguise which attribute is subtracted from the other is the same within each columns of shared dissimilarity matrix. If we assume a shared dissimilarity matrix D of size $m \times n$ for DH_x and DH_y, then DM can easily say that it is always $D[j,k] = a_j^x - a_k^y$ or $D[j,k] = a_k^y - a_j^x$ for $j = 1, 2, \ldots, m$. Based on this observation, DM can find out the maximum and minimum attributes in $DH_x \cup DH_y$ by simply checking signs(positive, negative, zero) of the entries of D since only for minimum and maximum attribute values, all entries of a column of D have the same sign. If there is no such column in D, this means minimum and maximum attribute values reside in DH_x. However even if one such column exist in D, DM can figure out with a certain confidence all the elements in $DH_x \cup DH_y$ if the domain of possible values for that attribute is small. However this is not the case in our protocol since for each entry of the dissimilarity matrix, a different pseudo-random number is used.

CHAPTER 3

An Efficient Solution to Millionaires' Problem

3.1 Introduction

Secure evaluation of the greater-than function GT(x, y) tries to answer the question "Is x greater than y?" without revealing inputs of the function (a.k.a Yao's Millionaires' problem). Several studies have been done on that issue; however most of these are inefficient in the sense that communication and computation costs are very large. In this study, we propose a more efficient solution to this problem taking [1] as our main reference point. Fischlin [8] proposed a protocol based on quadratic-residuosity bit-encryption of Goldwasser and Micali [9]. Goldwasser-Micali system (GM) uses modular exponentiation of each bit of data for an RSA modulus N which results in expansion of one bit to $log(N)$ bits. There are several approaches [4, 16] to the same problem based on encryption of each bit of data which are considered to be the most efficient ones. Nevertheless expansion of bits due to modular exponentiation increases communication cost. Also Modular exponentiation is costly to encrypt one bit of data.

For that reason, we adopt additive secret sharing to GM and merely use $\oplus$ operation to evaluate $GT(x, y)$. Using $\oplus$ operation in this sense reduces communication cost drastically since one bit of data is encoded into 2 bits of data. Additive secret sharing also reduces computation cost due to one $\oplus$ operation instead of modular exponentiation for encryption and decryption of the data.

However our protocol settings are a bit different from the protocols [8, 4, 16] for Yao's Millionaires' problem. Our protocol requires existence of two semi-trusted third parties. Semi-honest third parties obey the rules of the protocol without any collision or collaboration with any other parties, however they could record whatever they re-

ceive throughout the whole protocol, they could try to infer useful information out of the recorded data. Another difference of our protocol with respect to the previously proposed protocols is that result of the evaluation for $GT(x, y)$ could be delivered to any party other than owners of the secrets x and y without loss of privacy and security. This property makes our protocol appropriate for the applications where evaluation result is required not to be known by the participants of the protocol while a third party should know the result to form some global rules. Such applications exist for data mining where a global data miner gets input from different data holders and try to infer global information out of these different data sources such as clustering, classification, association rule mining while learning no information about single data sources. In other words, our protocol is flexible and adaptable for several applications with its requirements and settings.

3.1.1 Related Work

The first solution to Yao's Millionaires' problem is proposed in [27]. However this approach is far away from being efficient. All of the recent approaches considered to efficient for solving secure greater-than function evaluation employ encryption as their main tool to provide security. [16] tries to compare secrets of Alice and Bob, x and y respectively, based on $0-encoding$ (S_s^0) and $1-encoding$ (S_s^1) of these numbers. S_s^0 and S_s^1 of a number are the sets of binary strings such as :

$$S_s^0 = s_n s_{n-1} \dots s_{i+1} 1 | s_i = 0, 1 \leq i \leq n$$
$$S_s^1 = s_n s_{n-1} \dots s_i | s_i = 1, 1 \leq i \leq n$$

Accordingly, if $x > y := 1$, then $1-encoding$ of x (S_x^1) has a common element with $0-encoding$ of y (S_y^0). After Alice and Bob has formed S_x^1 and S_y^0 of their secrets respectively, the problem of evaluating the greater-than function is reduced to finding intersection of S_x^1 and S_y^0. If $S_x^1 \cap S_y^0 \neq \emptyset$, then $x > y := 1$. [16] utilizes ElGamal Homomorphic Encryption scheme [5] to find intersection of S_x^1 and S_y^0 by taking advantage of multiplicative homomorphic property of ElGamal Encryption scheme. Multiplicative homomorphic property of ElGamal Encryption scheme provides ability to perform multiplication over encrypted data, and reveal the multiplication result when data is decrypted. This way multiplication operation is performed without revealing the

operands of the multiplication which are secrets. This property can be summarized as:

$$E(x) * E(y) = E(x * y)$$

Accordingly, Alice forms a $2 \times n$ table T, where $T[x_j, j] = E(1)$ and $T[\neg x_j, j] = E(r)$ where r is a random number for j=1,2, ...,n. Alice sends table T to Bob, where Bob calculates $c_t = T[t_n, n] * T[t_{n-1}, n-1] * \ldots * T[t_i, i]$ for each $t = t_n t_{n-1} \ldots t_i \in S_y^0$. Bob randomizes each c_t and sends them to Alice, where Alice decides $x > y := 1$ if $D(c_t) = 1$. Computation costs for Alice and Bob are $3n \log p$ and $2n \log p + 2n$ modular multiplications respectively, where p is the modulus for ElGamal Encryption and n is length of the private input. Communication costs for Alice and Bob are $4n \log p$ and $2n \log p$ respectively. As seen from communication and computation cost of [16], the dominant factor is $n \log p$ modular multiplications which is due to encryption of each bit.

[8] proposes a protocol for secure greater-than function evaluation based on GM-encryption scheme. GM- encryption is firstly proposed in [9] which uses quadratic-residuosity to encrypt a bit. Encryption of a bit b with GM-encryption scheme is performed as the following:

$$E(b) = z^b r^2 \, mod \, N$$

where $N = p * q$ is an RSA modulus, z is a quadratic non-residue of Z_n^* with Jacobi symbol +1, r is a random number such as $r \in Z_n^*$. The public key is (N, z) and the private key is (p, q). GM-encryption scheme shows that $E(b)$ is a quadratic non-residue if and only if b is 1. Quadratic residuosity of bit b can be recognized efficiently only if factorization p and q of N are known. GM-encryption scheme has three properties that [8] uses, which are:

- XOR-property: $E(a) * E(b) = E(a \oplus b)$
- NOT-property: $E(a) * z = E(a \oplus 1) = E(\neg a)$
- Re-randomization: $Rand(E(a)) := E(a) * E(0)$

[8] uses a modified version of GM-encryption scheme to implement an AND-homomorphic encryption scheme (E^{AND}). Accordingly, a bit value b is encrypted as a sequence of λ encrypted values which are either $zE(b)$ or $E(0)$. With respect to the modified version

of GM-encryption scheme, $E^{AND}(a) * E^{AND}(b)$, which is pairwise $*$ operation of λ encrypted values, is equivalent to the evaluation of $a \wedge b$ for bit values a and b, and the result is 1 if $E^{AND}(a) * E^{AND}(b)$ is a sequence of λ quadratic residues and 0 otherwise. Greater-than function is formulated as:

$$x > y := \vee_{i=1}^{n} \left(x_i \wedge \neg y_i \wedge_{j=i+1}^{n} \neg \left(x_j \oplus y_j \right) \right) \tag{3.1}$$

where Equation (3.1) can be computed taking advantage of AND-homomorphic encryption scheme. The protocol starts when Alice encrypts her own secret $x = x_n x_{n-1} \ldots x_1$ as $E(x_i)$ for $i = 1, 2, \ldots, n$ and sends encrypted values to Bob. Bob encrypts his own secret $y = y_n y_{n-1} \ldots x_1$ as $E(y_i)$ for $i = 1, 2, \ldots, n$ and compute encrypted values for evaluation of $\neg \left(x_j \oplus y_j \right)$ as $E^{\neg(x_i \oplus y_i)} = NOT(E(x_i) * E(y_i))$. Then Bob calculates $E_{x_i}^{AND} = E^{AND}(E(x_i))$ and $E_{y_i}^{AND} = E^{AND}(NOT(E(y_i)))$, and evaluation of Equations (3.1) is done by $GT_i = E_{x_i}^{AND} * E_{y_i}^{AND} * \prod_{j=i+1}^{n} E^{\neg(x_j \oplus y_j)}$. Bob sends GT_i's to Alice, and Alice concludes that $x > y := 1$, if one of GT_i's is a sequence of λ quadratic residues. Computation costs are $n\lambda + 2n$ and $6n\lambda$ modular multiplications, and computation costs are $n \log N$ and $n\lambda \log N$ for Alice and Bob respectively where N is an RSA modulus. Similar to [16], the dominant factor is $n\lambda$ modular multiplications due to encryption of each bit.

Another cryptographic protocol for secure greater-than function evaluation is proposed by [4]. Additive homomorphic property of Paillier Cryptosystem [22] is used to implement this protocol. Additive homomorphism has the following property:

$$E(x) * E(y) = E(x + y)$$

Using Paillier Cryptosystem (E), Alice initializes the protocol by encrypting each bit of her secret $x = x_n x_{n-1} \ldots x_1$ as $E(x_i)$ for $i = 1, 2, \ldots, n$ and sends encrypted values to Bob. Then Bob follows a five step process:(1) Computes $E(d_i) = E(x_i - y_i)$, (2) Computes $E(f_i) = E(x_i \oplus y_i) = E((x_i - y_i)^2) = E(x_i - 2x_i y_i + y_i)$, (3) Computes $E(\gamma_i) = E(2\gamma_{i-1} + f_i)$ where $\gamma_0 = 0$, (4) Computes $E(\delta_i) = E(d_i + r_i(\gamma_i - 1))$ where $r_i \in_R Z_N$, (5) Permutes $E(\delta_i)$ and sends to Alice. Alice decrypts $E(\delta_i)$, and concludes that $x > y := 1$ if there exists a decrypted value $+1$. Computation costs are $12n \log N$ and $4n \log N + 28n$ modular multiplications while communication costs are $2n \log N$ and $2n \log N$ for Alice and Bob respectively. The dominant factor is caused by encryption again which is $n \log N$ modular multiplications.

3.2 Preliminaries

Throughout this chapter, encryption and decryption operations are performed on single bits. In order to compute a function over a number x, x is represented in binary notation as $x = x_n x_{n-1} \dots x_1$ which can also be denoted as $\sum x_n * 2^{n-1}$. We write $S(x, r) = c$ for secret sharing of bit x with random bit r and the corresponding shares of the secret are assigned to bit r and $c = r \oplus x$. Revealing the secret bit x from shares works in similar fashion as $c \oplus r = x$, which means a simple $\oplus$ operation over shares c and r results in the original bit x.

3.2.1 XOR Homomorphic Secret Sharing Scheme

We use XOR operation for secret sharing of a bit x of information in the existence of a random bit r. Basically our secret sharing scheme is:

$$S^{\oplus}(x, r) = c \text{ where } x \oplus r = c;$$

For restoring the secret x from c and random bit r, the corresponding procedure is as the following:

$$D^{\oplus}(c, r) = x \text{ where } c \oplus r = x;$$

This additive secret sharing scheme has three properties which will be of great use in the following sections. These properties are;

1. XOR homomorphism: $S^{\oplus}(a, r_a) \oplus S^{\oplus}(b, r_b) = S^{\oplus}(a \oplus b, r_a \oplus r_b)$
2. NOT property: $S^{\oplus}(\neg a, r_a) = 1 \oplus S^{\oplus}(a, r_a)$
3. Rerandomization: $Rand(S^{\oplus}(a, r_a)) := 0 \oplus S^{\oplus}(a, r_a) = S^{\oplus}(r, r) \oplus S^{\oplus}(a, r_a) = S^{\oplus}(r \oplus a, r \oplus r_a)$

3.2.2 AND Homomorphic Secret Sharing Scheme

XOR homomorphic secret sharing scheme explained in Section 3.2.1 can be used to generate an AND homomorphic secret sharing scheme in the following manner: AND homomorphic secret sharing ($S^{\wedge}$) of a bit can be encoded as a sequence of XOR homomorphic secret sharings ($S^{\oplus}$). Encoding of a bit is a sequence of λ elements each

of which will be either $S^{\oplus}$ of NOT x ($S^{\oplus}(\neg x, r)$) or $S^{\oplus}$ of bit 0 ($S^{\oplus}(0, r)$). In other words;

$$S^{\wedge}(x) = S_1^{\oplus}(x)||\ldots||S_\lambda^{\oplus}(x) \tag{3.2}$$

$$S_i^{\oplus}(x) = S^{\oplus}(\neg x_i, r_i) \text{ or } S^{\oplus}(0, r_i) \tag{3.3}$$

In Equation (3.3), choice of the value of $S_i^{\oplus}$ is made by a fair coin flip. For secret sharing of bit *1*, we encode it as a sequence of λ random secret sharings of *0*. For secret sharing of bit *0*, we encode it as a sequence of λ random secret sharings of *0* and *1*. Accordingly, computation of $(x \wedge y)$ can be done as the following;

$$\begin{aligned} x \wedge y &= S^{\wedge}(x) \oplus S^{\wedge}(y) \\ &= S_1^{\oplus}(x) \oplus S_1^{\oplus}(y)||\ldots||S_\lambda^{\oplus}(x) \oplus S_\lambda^{\oplus}(y) \end{aligned} \tag{3.4}$$

$S_i^{\oplus}(x)$ and $S_i^{\oplus}(y)$ uses the same randomness to encrypt bits x_i and y_i in Equation (3.4). If result of $(S^{\wedge}(x) \oplus S^{\wedge}(y))$ is a sequence of all *0*'s, this means $x \wedge y = 1$. On the other hand, even if one of the elements in the sequence is *1*, then $x \wedge y = 0$. However, since value of $S_i^{\oplus}$ is selected among $S^{\oplus}(1, r_i)$ or $S^{\oplus}(0, r_i)$ during the encoding phase of bit *0*, there can be failure in the encoding phase with probability of $2^{-\lambda}$ if all the elements in the sequence are chosen as $S^{\oplus}(0, r_i)$. Another failure scenario is that while calculating $S^{\wedge}(x) \oplus S^{\wedge}(y)$, a piecewise $\oplus$ operation of encodings of 1 and 1 results in encryption of 0 ($S^{\oplus}(1, r_i) \oplus S^{\oplus}(1, r_i) = S^{\oplus}(0, r_i)$). This kind of cancellation failures occur with probability of $2^{-\lambda}$. For that reason, λ has to be chosen appropriately to minimize failure probability.

3.3 Evaluating Greater Than (GT) function

Evaluation of the boolean function $f(X, Y)$ such as: $f(X, Y) = 1$ if $X > Y$, and $f(X, Y) = 0$ otherwise, can be done by comparison of each number bitwise such as if $X > Y$, then $x_i = 1$, and $y_i = 0$, and $x_j = y_j$ (which can be computed by $\neg(x_j \oplus y_j)$) for all more significant bits [1] which is evaluated by Equation (3.1). Accordingly, GT function can easily be implemented since the term $\neg(x_j \oplus y_j)$ can be computed by XOR homomorphic scheme explained in Section 3.2.1, and the term $x_i \wedge \neg y_i \wedge_{j=i+1}^{n} \neg\,(x_j \oplus y_j)$

can be computed by AND homomorphic scheme explained in Section 3.2.2.

Evaluation of GT function based on our XOR and AND homomorphic secret sharing scheme in Sections 3.2.1, 3.2.2 requires presence of two semi-honest third parties(SHTP) since Equation (3.1) has to be computed securely without revealing any bits of x and y. Evaluation process starts at Parties X and Y where shares of secret bits x_i and y_i are formed by $S^{\oplus}$. Shares of each secret bit are distributed among SHTP. From that point on, SHTP start computation to form shares of the evaluation for Equation (3.1). SHTP share a seed to initialize a pseudo-random number generator (PRNG). Based on this PRNG, SHTP form two components both: (1) Share of secret $x_i \wedge \neg y_i$, (2)Share of secret $\wedge_{j=i+1}^{n} \neg (x_j \oplus y_j)$. SHTP compute the first component by employing an encoding function based on shared PRNG. The encoding function is as the following;

$$S_{encode}^{c_i}(c_i, PRNG, inverse) = (r_{i,j} \text{ OR } c_i \oplus inverse \oplus r_{i,j}) = e_{i,j}^{c} \tag{3.5}$$

$$S_{encode}^{x_i \wedge \neg y_i} = e_{i,1}^{x} \oplus e_{i,1}^{y} || \ldots || e_{i,\lambda}^{x} \oplus e_{i,\lambda}^{y} \tag{3.6}$$

Encoding function in Equation (3.5) takes three parameters. First parameter c_i is share of the secret for either party X or Party Y. Second parameter is PRNG which is used to randomize output of the encoding function in Equation (3.5). The third parameter is the bit value *inverse* which determines whether to inverse the output or not. There are two possible outputs of the function in Equation (3.5). To chose among these possible outputs, shared PRNG is used. If random number created is odd, then the first output, otherwise the second output is chosen to be the actual output. After deciding on what is going to be the output, value of the output is calculated. For that purpose, another random number rnd is created and value of the $r_{i,j}$ is set as rnd mod 2. Equation (3.5) is called λ times for each bit of the share of the secret.

The second component is computed in a similar fashion by SHTP. The functions used are;

$$S_{encode}^{x_i, y_i}(x_i, y_i, PRNG) = (r_{i,j} \text{ OR } x_i \oplus y_i) = e_{i,j}^{x \oplus y} \tag{3.7}$$

$$S_{encode}^{\wedge_{j=i+1}^{n} \neg (x_j \oplus y_j)} = e_{j,1}^{x \oplus y} \oplus \ldots \oplus e_{n,1}^{x \oplus y} || \ldots || e_{j,\lambda}^{x \oplus y} \oplus \ldots \oplus e_{n,\lambda}^{x \oplus y} \tag{3.8}$$

Function in Equation (3.7) takes shares of the secrets for Party X and Y, and PRNG as parameters. The encoding procedure is same as Equation (3.5). Output of

Equation (3.7) is used as input for Equation (3.8) and the resulting output is share of each semi-trusted parties for component (2).

$$S_{encode}^{x_i \wedge \neg y_i \wedge_{m=i+1}^{n} \neg(x_j \oplus y_j)} = S_{encode}^{x_i \wedge \neg y_i} \oplus S_{encode}^{\wedge_{m=i+1}^{n} \neg(x_j \oplus y_j)} \tag{3.9}$$

Using Equations (3.5),(3.6),(3.7),(3.8), the shares of the evaluation Equation (3.1) is calculated. The final share of each SHTP for evaluation of the Equation (3.1) is calculated by Equation 3.9. However share of the Equation (3.1) can reveal information about which bit of Secret X and Y fulfill Equation (3.1) when combined in Computation Site since resulting value for this bit will be a sequence of λ 0's. Based on this fact, Computation Site knows that value corresponding to that bit is 1 for Secret X and 0 for Secret Y, and the other more significant bits are equal. To prevent this information leakage, SHTP will permute their final result with respect to a permutation scheme they share, before sending their shares of the result to the Computation Site. The details of the steps, each party follows is given in detail in Section 3.4.

3.4 Our Protocol

Algorithm 7 Party X

Input: Secret value $x = x_n x_{n-1} \dots x_1$
Output: $c_1^x, c_2^x, \dots, c_n^x$ and $r_1^x, r_2^x, \dots, r_n^x$

1: **for** $i = 0$ to n **do**
2: $\quad c_i^x = S^{\oplus}(x_i, r_i^x)$
3: **end for**
4: Sends $c_1^x, c_2^x, \dots, c_n^x$ to **TRUSTED PARTY 1**
5: Sends $r_1^x, r_2^x, \dots, r_n^x$ to **TRUSTED PARTY 2**

Algorithm 8 Party Y

Input: Secret value $y = y_n y_{n-1} \dots y_1$
Output: $c_1^y, c_2^y, \dots, c_n^y$ and $r_1^y, r_2^y, \dots, r_n^y$

1: **for** $i = 0$ to n **do**
2: $\quad c_i^y = S^{\oplus}(y_i, r_i^y)$
3: **end for**
4: Sends $c_1^y, c_2^y, \dots, c_n^y$ to **Trusted party 2**
5: Sends $r_1^y, r_2^y, \dots, r_n^y$ to **Trusted party 1**

Algorithm 9 Trusted Party 1

Input: $c_1^x, c_2^x, \ldots, c_n^x$ and $r_1^y, r_2^y, \ldots, r_n^y$
Output: Permuted Matrix GT^1

Initialize PRNG with seed shared with Trusted party 2
Share permutation scheme with Trusted party 2
for $i = 1$ to n **do**
  for $j = 1$ to λ **do**
    $e_{i,j}^x = S_{encode}^{c_i^x}(c_i^x, PRNG, 0)$
    $e_{i,j}^y = S_{encode}^{r_i^y}(r_i^y, PRNG, 1)$
  end for
end for
for $i = 1$ to n **do**
  for $j = 1$ to λ **do**
    $S_{encode[j]}^{x_i \wedge \neg y_i} = e_{i,j}^x \oplus e_{i,j}^y$
  end for
end for
for $i = 1$ to n **do**
  for $j = 1$ to λ **do**
    $e_{i,j}^{x \oplus y} = S_{encode}^{c_i^x, r_i^y}(c_i^x, r_i^y, PRNG)$
  end for
end for
for $i = 1$ to $n - 1$ **do**
  for $j = 1$ to λ **do**
    $S_{encode[j]}^{\wedge_{m=i+1}^{n} \neg(x_j \oplus y_j)} = e_{m,j}^{x \oplus y} \oplus \ldots \oplus e_{n,j}^{x \oplus y}$
  end for
end for
for $i = 1$ to $n - 1$ **do**
  for $j = 1$ to λ **do**
    $S_{encode[j]}^{x_i \wedge \neg y_i \wedge_{m=i+1}^{n} \neg(x_j \oplus y_j)} = S_{encode[j]}^{x_i \wedge \neg y_i} \oplus S_{encode[j]}^{\wedge_{m=i+1}^{n} \neg(x_j \oplus y_j)}$
  end for
end for
Permute Matrix $S^{x_i \wedge \neg y_i \wedge_{j=i+1}^{n} \neg(x_j \oplus y_j)}$ with respect to column i's resulting in GT^1
Sends Matrix GT^1 to **Computation Site**

Algorithm 10 Trusted Party 2

Input: $c_1^y, c_2^y, \ldots, c_n^y$ and $r_1^x, r_2^x, \ldots, r_n^x$
Output: Permuted Matrix GT^2

Initialize PRNG with seed shared with Trusted party 1
Share permutation scheme with Trusted party 1
Negate each Share of Party Y as $\neg c_i^y = 1 \oplus c_i^y$
for $i = 1$ to n **do**
  for $j = 1$ to λ **do**
    $e_{i,j}^x = S_{encode}^{r_i^x}(r_i^x, PRNG, 1)$
    $e_{i,j}^y = S_{encode}^{c_i^y}(\neg c_i^y, PRNG, 0)$
  end for
end for
for $i = 1$ to n **do**
  for $j = 1$ to λ **do**
    $S_{encode[j]}^{x_i \wedge \neg y_i} = e_{i,j}^x \oplus e_{i,j}^y$
  end for
end for
for $i = 1$ to n **do**
  for $j = 1$ to λ **do**
    $e_{i,j}^{x \oplus y} = S_{encode}^{r_i^x, c_i^y}(r_i^x, c_i^y, PRNG)$
  end for
end for
for $i = 1$ to $n - 1$ **do**
  for $j = 1$ to λ **do**
    $S_{encode[j]}^{\wedge_{m=i+1}^{n} \neg(x_j \oplus y_j)} = e_{m,j}^{x \oplus y} \oplus \ldots \oplus e_{n,j}^{x \oplus y}$
  end for
end for
for $i = 1$ to $n - 1$ **do**
  for $j = 1$ to λ **do**
    $S_{encode[j]}^{x_i \wedge \neg y_i \wedge_{m=i+1}^{n} \neg(x_j \oplus y_j)} = S_{encode[j]}^{x_i \wedge \neg y_i} \oplus S_{encode[j]}^{\wedge_{m=i+1}^{n} \neg(x_j \oplus y_j)}$
  end for
end for
Permute Matrix $S^{x_i \wedge \neg y_i \wedge_{j=i+1}^{n} \neg(x_j \oplus y_j)}$ with respect to column i's resulting in GT^2
Sends Matrix GT^2 to **Computation Site**

Algorithm 11 Computation Site

Input: GT^1 and GT^2
Output: **True** if $(X > Y)$, **false** otherwise

for $i = 1$ to n **do**
  for $j = 1$ to λ **do**
    $GT[i][j] = GT^1[i][j] \oplus GT^2[i][j]$
  end for
end for
for $i = 1$ to n **do**
  if $GT[i]$ is a sequence of λ 0's **then**
    return true
  end if
end for
return false

3.5 Complexity Analysis of Our Protocol

In this section, we show computation and communication complexity of our protocol with comparison to three protocols [8, 4, 16] proposed for greater-than function evaluation which are explained in detail in Subsection 3.1.1. The reason why we choose these three protocols is that they are the most recently proposed protocols which are considered to be the most efficient solutions to the secure greater-than function evaluation.

3.5.1 Computation Complexity

The summary of computation costs for each protocol is depicted in Table 3.1. Computation cost of our protocols can be explained in terms of Party X, Party Y, Trusted Party 1, Trusted Party 2, and Computation Site. Party X and Y has the same computation cost which is composed of n fair coin flip and n $\oplus$ operations for secret sharing of bits of secrets x and y. Computational cost of our protocol for Trusted Party 1 is as the following: $6n\lambda$ $(= 3 * 2n\lambda)$ PRNG for lines 5, 6, and 16, $6n\lambda$ $(= 6 * n\lambda)$ $\oplus$ operations for lines 5, 6, 11, 16, 21, and 26. Trusted Party 2 has the same computation cost as Trusted Party 1 if we ignore n $\oplus$ operations at line 3. Computation cost of Computation Site is $n\lambda$ $\oplus$ operations for line 3. Accordingly computational complexity of our protocol is dominated by $n\lambda$ PRNG. If we assume that 128 bits AES is used for pseudo-random number generation, then complexity of our protocol is $n\lambda \log S$ where S is an AES modulus. As explained in detail in Subsection 3.1.1, computational complexity of [16],[4], and [8] are dominated by $n \log p$, $n \log N$, and $n\lambda \log N$ modular multiplications respectively. Assuming a conservative security level for each protocol , we can say that it is possible to take $\log p = 512$, $\log N = 512$ and $\log S = 128$.

Table 3.1: Comparison of Computation Cost for Different Protocols

Protocol	Party X	Party Y	STTP1	STTP2
[8]	$(n\lambda + 2n)m$	$(6n\lambda)m$	-	-
[4]	$(12n \log N)m$	$(4n \log N + 28n)m$	-	-
[16]	$(3n \log p)m$	$(2n \log p + 2n)m$	-	-
Our Protocol	$nc + nx$	$nc + nx$	$6n\lambda x + 6n\lambda r$	$6n\lambda x + 6n\lambda r + nx$

*m, c, x, r stand for modular multiplication, fair coin flip, bitwise $\oplus$ operation, and pseudo-random number generation respectively. *p is ElGamal modulus *N is RSA modolus *n is length of private inputs in bits

3.5.2 Communication Complexity

Communication costs of our protocol are $2n$ for Party X and Party Y , and $n\lambda$ for Trusted Party 1 and 2. The summary of communication costs for all protocols are depicted in Table 3.2. According to Table 3.2, our protocol causes a dramatic reduction in communication cost compared to previously proposed protocols.

Table 3.2: Comparison of Communication Cost for Different Protocols

Protocol	Party X	Party Y	STTP1	STTP2	Total
[8]	$n \log N$	$n\lambda \log N$	-	-	$n \log N(\lambda + 1)$
[4]	$2n \log N$	$2n \log N$	-	-	$4n \log N$
[16]	$4n \log p$	$2n \log p$	-	-	$6n \log p$
Our Protocol	$2n$	$2n$	$n\lambda$	$n\lambda$	$4n + 2n\lambda$

CHAPTER 4

Conclusion and Future Work

In this thesis, we have shown two very efficient protocols which are useful for data mining. The efficiency was achieved by observing that the use of two (or three) semi-honest non-colluding third parties is realistic in a data mining setting.

Our solution to Yao's millionaires problem is of independent interest, since it improves over existing protocols in both communication complexity and computation complexity. It is, to the best of our knowledge, the best solution so far. Our solution is based on the solution by Fischlin, but whereas Fischlin uses only one semi-honest third party, we use two. Compared to Fischlin, our protocol has communication overhead of 2λ, whereas practical implementations of Fischlin will have communication overhead several orders of magnitudes more. Fischlin's protocol uses exponentiation of cipher-texts in the evaluation of each Boolean circuit, wheres our protocol executed the Boolean circuit directly on bits (2λ bits in parallel).

Secondly by utilizing our numerical distance protocol, we proposed a privacy preserving distributed clustering protocol for horizontally partitioned data using secret sharing scheme, which is homomorphic with respect to addition operation. The model that we adopted is unprecedented in the sense that it uses two non-colluding third parties. The idea of using two third parties that greatly alleviates the computation and communication burden of the data holders is especially useful in applications such as sensor networks and RFID where data holders are resource-limited sensor nodes and RFID readers. When compared to the most efficient former techniques, which exclusively rely on the computation and communication capabilities of the data holders, our protocol can run even on the most simple data holders, such sensor nodes or RFID readers. One can even think that there is no need for the data holders to store the actual data. It is true that the communication overhead between the two third parties is greater than the other protocols. Nevertheless, third parties can easily be equipped

with high computation capability and bandwidth.

The use of two third parties and non-collusion property are realistic when they are chosen to have conflicting interests in the data mining results of the actual data. As an example, one third party can be a consumer organization who is interested in the privacy of consumers, while the other is representative of the industry - they both have interests in the right outcome, but will never collude.

We proved information theoretically that the third parties cannot gather any information about the data under the non-colluding assumption. Therefore, our protocol adopts a stronger security model than computational infeasibility, which is used by the majority of other privacy preserving data mining algorithms.

Other two benefits of the proposed protocol are that the data holders do not need to be synchronized or to share keys. Almost all previously proposed methods rely on synchronous and interactive protocols, where data holders must always be on-line during the protocol execution, while the data holders in our protocol can send the shares of their data asynchronously at their convenience. Since there is no communication between the data holders, there is no need for them to share keys; therefore there is no key distribution problem.

And finally, the model based on the use of two third parties and homomorphic secret sharing can be extended to other data types and different dissimilarity metrics.

Bibliography

[1] Referenced at kdd'99 classifier learning contest: http://www-cse.ucsd.edu/users/elkan/clresults.htm (2006).

[2] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 439–450. ACM Press, 2000.

[3] Charles Asmuth and John Bloom. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, 29(2), 1983.

[4] I. Blake and V. Kolesnikov. Strong conditional oblivious transfer and computing on intervals. *Advances in Cryptology*, 3329:515–529, 2004.

[5] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31:469–472, 1985.

[6] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 211–222, New York, NY, USA, 2003. ACM Press.

[7] Alexandre Evfimievski, Ramakrishnan Srikant, Rakesh Agarwal, and Johannes Gehrke. Privacy preserving mining of association rules. *Inf. Syst.*, 29(4):343–364, 2004.

[8] M. Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. *Lecture Notes in Computer Science*, 2020:457–472, 2001.

[9] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, 1984.

[10] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2000.

[11] Ali Inan, Yücel Saygın, Erkay Savaş, Ayça Azgın Hintoğlu, and Albert Levi. Privacy preserving clustering on horizontally partitioned data. In *Privacy Preserving Clustering on Horizontally Partitioned Data*, page 95. IEEE Computer Society, 2006.

[12] Murat Kantarcıoğlu and Chris Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1026–1037, 2004.

[13] Hillol Kargupta, Souptik Datta, Qi Wang, and Krishnamoorthy Sivakumar. Random-data perturbation techniques and privacy-preserving data mining. *Knowl. Inf. Syst.*, 7(4):387–414, 2005.

[14] Matthias Klusch, Stefano Lodi, and Gianluca Moro. Distributed clustering based on sampling local density estimates. In *IJCAI '03:Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 485–490. AAAI Press, 2003.

[15] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.

[16] H.Y. Lin and W.G. Tzeng. An efficient solution to the millionaires' problem based on homomorphic encryption. *Lecture Notes in Computer Science*, 3531:456–466, 2005.

[17] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 36–54. Springer, 2000.

[18] Srujana Merugu and Joydeep Ghosh. Privacy-preserving distributed clustering using generative models. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, pages 211–218, Washington, DC, USA, 2003. IEEE Computer Society.

[19] Stanley Oliveira and Osmar R. Zaiane. Privacy preserving clustering by data transformation. In *18th Brazilian Symposium on Databasesn*, pages 304–318, 2003.

[20] Stanley Oliveira and Osmar R. Zaiane. Achieving privacy preservation when sharing data for clustering. In *International Workshop on Secure Data Management*

in a Connected World (SDM'04) in conjunction with VLDB 2004, pages 67–82. Springer Verlag, 2004.

[21] Stanley Oliveira and Osmar R. Zaiane. Privacy-preserving clustering by object similarity-based representation and dimensionality reduction transformation. In *Workshop on Privacy and Security Aspects of Data Mining (PSDM'04) in conjunction with the Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 21–30, 2004.

[22] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Lecture Notes in Computer Science*, 1592:223–238, 1999.

[23] P. McDaniel S. Jha, L. Kruger. Privacy preserving clustering. In *ESORICS'05:10th European Symposium On Research In Computer Security*, pages 397–417, 2005.

[24] Yucel Saygin, Vassilios S. Verykios, and Chris Clifton. Using unknowns to prevent discovery of association rules. *SIGMOD Rec.*, 30(4):45–54, 2001.

[25] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[26] Jaideep Vaidya and Chris Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *KDD'03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215, New York, NY, USA, 2003. ACM Press.

[27] Andrew C. Yao. Protocols for secure computation. In *23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.