CERIAS Tech Report 2005-51

PRIVACY-PRESERVING DISTRIBUTED DATA MINING AND PROCESSING ON HORIZONTALLY PARTITIONED DATA

by Murat Kantarcioglu

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

PRIVACY-PRESERVING DISTRIBUTED DATA MINING AND PROCESSING ON HORIZONTALLY PARTITIONED DATA

A Thesis

Submitted to the Faculty

of

Purdue University

by

Murat Kantarcıoğlu

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2005

To my parents, Fatma and Özkan.
To my brother, İsmail.

Sevgili Ailem,
Sizin desteğiniz ve sevginiz olmasa idi,
bugün olduğum yere asla gelemez idim.
Bu yüzden doktora tezimi size adıyorum.
Sizin hakkınızı asla ödeyemem.
Umarim size lăyik bir evlat olabilmişimdir.

ACKNOWLEDGMENTS

I feel extremely lucky to have been advised by Prof. Chris Clifton. He has been a great mentor and a good role model. He has always kept his door open to me and patiently answered all my questions. He has helped me in my research and in many practical issues like how to fix my car. Without him, I would not have been able to complete my degree.

I also would like to acknowledge the huge contributions of my other committee members. Prof. Mike Atallah has made many useful suggestions related to my research. Discussions with him have been enlightening and inspiring. Prof. Elisa Bertino has been very supportive during my job hunt. Her wonderful database security course has helped me in seeing the big picture. Our discussions with her have given me many useful ideas.

Prof. Wojciech Szpankowski and his "curious minds" meetings have been very helpful. I have learned many interesting things through those meetings. If I had not attended the "curious minds" meetings, I would not have read the parts of the Knuth's new volume or learned about the Johnson-Lindenstrauss theorem.

During my graduate years, I have done two great internships with Dr. Wen-Syan Li. It has been enticing to do research with him. His contributions have been invaluable in improving my research skills.

It has been a great honor to do an internship with Dr. Rakesh Agrawal. His vision and his creativity have been truly inspiring. His determination to solve even the hardest problem and his "I will solve it" attitude have shown me what it really means to be a great researcher.

I have been blessed with many good friends during my graduate years. Jaideep Vaidya, Wei Jiang, Amit J. Shirsat and Radu Sion have been my fellow companions in my journey to finish my dissertation. Their help and support have ensured that I

have made progress on my thesis. Also my interactions with “ddm2” group members: Abhilasha Bhargav, Mummoorthy Murugesan, Ercan Nergiz and Yang Wang have been a useful source of information. I have many other friends who have made the life in Lafayette bearable. I would like to thank Tolga Açıkalın, “2in1” gang and friends from Turkish student association for all the joy we have shared together.

I would like to thank Mrs. Patricia Clifton for treating me like a family member and for all the great parties she has organized for us.

Burcu Korkut has entered my life during the stressful job hunt period. Her support has made me believe in myself. Her love has made me happy. Her devotion has made me strong. Her smile has given me a hope. I am so lucky to have her in my life.

For my family, no words can suffice. Thanks for your love and support. I dedicate this thesis to you.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

DDM	Distributed Data Mining
KDD	Knowledge Discovery in Databases
SMC	Secure Multiparty Computation
PPDM	Privacy-preserving Data Mining
PPDDM	Privacy-preserving Distributed Data Mining

ABSTRACT

Kantarcıoğlu, Murat. Ph.D., Purdue University, August, 2005. Privacy-Preserving Distributed Data Mining and Processing on Horizontally Partitioned Data. Major Professor: Christopher W. Clifton.

Data mining can extract important knowledge from large data collections, but sometimes these collections are split among various parties. Data warehousing, bringing data from multiple sources under a single authority, increases risk of privacy violations. Furthermore, privacy concerns may prevent the parties from directly sharing even some meta-data.

Distributed data mining and processing provide a means to address this issue, particularly if queries are processed in a way that avoids the disclosure of any information beyond the final result. This thesis presents methods to mine horizontally partitioned data without violating privacy and shows how to use the data mining results in a privacy-preserving way. The methods incorporate cryptographic techniques to minimize the information shared, while adding as little as possible overhead to the mining and processing task.

1 INTRODUCTION

Data mining technology has emerged as a means of identifying patterns and trends from large quantities of data. Recently, there has been growing concern over the privacy implications of data mining. Some of this is public perception: The "Data Mining Moratorium Act of 2003" introduced in the U.S. Senate [36] was based on a fear of government searches of private data for individual information, rather than what the technical community views as Data Mining. However, concerns remain. While data mining is generally aimed at producing general models rather than learning about specific individuals, the *process* of data mining creates integrated data warehouses that pose real privacy issues. Data that is of limited sensitivity by itself becomes highly sensitive when integrated, and gathering the data under a single roof greatly increases the opportunity for misuse. Even though some of the distributed data mining tasks protect individual data privacy, they still require that each site reveals some partial information about the local data. What if even this information is sensitive?

For example, suppose the Centers for Disease Control (CDC), a public agency, would like to mine health records to try to find ways to reduce the proliferation of antibiotic resistant bacteria. Insurance companies have data on patient diseases and prescriptions. CDC may try to mine association rules of the form $X \Rightarrow Y$ such that the $Pr(X\&Y)$ and $Pr(Y|X)$ are above some certain thresholds. Mining this data for association rules would allow the discovery of rules such as $Augmentin\&Summer \Rightarrow Infection\&Fall$, i.e., people taking Augmentin in the summer seem to have recurring infections.

The problem is that insurance companies will be concerned about sharing this data. Not only must the privacy of patient records be maintained, but insurers will be unwilling to release rules pertaining only to them. Imagine a rule indicating a

high rate of complications with a particular medical procedure. If this rule doesn't hold globally, the insurer would like to know this; they can then try to pinpoint the problem with their policies and improve patient care. If the fact that the insurer's data supports this rule is revealed (say, under a Freedom of Information Act request to the CDC), the insurer could be exposed to significant public relations or liability problems. This potential risk could exceed their own perception of the benefit of participating in the CDC study.

Our solution to this problem is to avoid disclosing data beyond its source, while still constructing data mining models equivalent to those that would have been learned on an integrated data set. Since we prove that data is not disclosed beyond its original source, the opportunity for misuse is not increased by the process of data mining.

The definition of privacy followed in this line of research is conceptually simple: no site should learn anything new from the *process* of data mining. Specifically, anything learned during the data mining process must be derivable given one's own data and the final result. In other words, nothing is learned about any other site's data that isn't inherently obvious from the data mining result. The approach followed in this research has been to select a type of data mining model to be learned and develop a protocol to learn the model while meeting this definition of privacy.

In addition to the type of data mining model to be learned, the different types of data distribution result in a need for different protocols. For example, the first paper in this area proposed a solution for learning decision trees on horizontally partitioned data: each site has complete information on a distinct set of entities, and an integrated dataset consists of the union of these datasets. In contrast, vertically partitioned data has different types of information at each site; each has partial information on the same set of entities. In this case an integrated dataset would be produced by *joining* the data from the sites. While [53] showed how to generate ID3 decision trees on horizontally partitioned data, a completely new method was needed for vertically partitioned data [28].

This thesis presents solutions such that the parties learn (almost) nothing beyond the global results. We assume homogeneous databases and horizontally partitioned data: All sites have the same schema, but each site has information on different entities. Given solutions are relatively efficient and proved to preserve privacy under some reasonable assumptions. Specifically, in Chapter 4, we show how to mine distributed association rules without revealing anything other than the rules. We address the privacy issues in distributed k-nearest neighbor classification methods in Chapter 5. Privacy preserving distributed Naïve Bayes classifier is discussed in Chapter 6.

The techniques mentioned above reveal almost nothing other than the final data mining result. This may not be enough to guarantee privacy. The data mining results may reveal some sensitive information about the original data sources. In Chapter 7, we provide a framework to analyze whether the resulting data mining models inherently violate privacy. We also provide techniques for computing a lower bound on the privacy loss due to data mining results.

Even if the mining process and the resulting model does not violate privacy, usage of the mined model may be a threat to privacy. Consider the following example:

A U.S. government initiative plans to classify each airline passenger with a green, yellow or red risk level. Passengers classified as green will be subject to only normal checks, while yellow will get extra screening and red won't fly. [13] Although government agencies guarantee that no discriminatory rules will be used in the classification and that the privacy of the data will be maintained, this is not enough for many privacy advocates. In Chapter 8 we show that if such a system *must* exist, it is possible to do so while achieving significant levels of privacy and guaranteeing that rules meet certain constraints without disclosing them.

There exists a false assumption that we have to sacrifice privacy to perform data mining. In this thesis, we show that many data mining tasks on horizontally partitioned data can be performed without sacrificing the privacy. Even more, we show that those mined models can be used in a privacy preserving manner.

2 PRIVACY-PRESERVING DATA MINING: STATE-OF-THE-ART AND RELATED ISSUES

2.1 Introduction

Cheap data storage and abundant network capacity have revolutionized the data collection and data dissemination. At the same time, advances in data mining have made it possible to learn previously unknown and interesting knowledge from the collected data [41]. These developments have caused serious concerns related to privacy implications of data mining. In [22], Clifton et al. identify two possible privacy drawbacks:

- Inference problems due to mining published data,
- Privacy issues in sharing data for data mining.

The first drawback refers to the fact that some private information can be inferred from mining public data. Follow up work described in [21] analyzes the disclosure due to building a classifier from the released data. Similar work has been done to limit disclosure due to association rule mining. The basic idea in that work is to modify the released data so that the association rules deemed to be "private" may not be learned by using data mining. It is shown that finding the best data modification for limiting the sensitive association rules is a NP-Hard problem [5]. The work given in [74] describes heuristics to modify the 0/1 binary database to hide the "sensitive" association rules. The site effect of this approach is the introduction of "ghost" association rules which are not supported by the original database. In order to solve the above problem and to prevent the problems due to modified data in some fields (i.e, health care), blocking some entries of the binary database by replacing the 0/1 value with "?" is suggested [68]. The authors of that work also gave a modified

association rule mining algorithm to handle unknown values. A detailed survey of association rule hiding can be found in [75].

The second drawback refers to privacy issues arise in data collection and data sharing for data mining. Due to privacy concerns, individuals and organizations may not be willing to share their data. For example, individuals may not be willing to tell their incomes or companies may not be willing to reveal statistics related to their core businesses. Fortunately, in many cases, the information aggregated over many individuals may not be considered a privacy problem. Therefore, it can be assumed that the data mining result itself are not a privacy violation. The goal is to learn the data mining results by disclosing as little as possible about the data sources. Many different solutions suggested for addressing this privacy challenge. Since most of the work in this dissertation fall under this category, we describe these solutions in detail.

Some of the techniques used for data mining have their roots in the statistical database security work. We briefly give an overview of the field in Chapter 2.2. In Chapter 2.3, we describe the data perturbation techniques used in data mining. In Chapter 2.4, we focus on the privacy-preserving distributed data mining. We conclude this chapter by discussing the contributions of this thesis to privacy-preserving data mining.

2.2 Statistical Database Security

The database community has long been concerned with privacy issues. An early example is work in statistical queries: Given queries that return aggregate values, can we be certain that the individual values used to compute the aggregates are not revealed? The work of Dobkin, Jones, and Lipton in query set overlap control [26], as well as work by Denning [24], showed that we can determine when a set of queries do not reveal individual values. For a survey of this area see [1]. The database security community has also addressed inference problems on a broader

scale: how do we guarantee that a value in a collection of "high" data cannot be inferred from a collection of "low" data? [42, 43, 55, 78] One common feature of this work is the emphasis on learning private values *exactly*; the ability to obtain probabilistic estimates on private data has received less work.

2.3 Data Perturbation Approach for Privacy-preserving Data Mining

The key idea of data perturbation is to modify the individual values such that the real values cannot be reconstructed, while the statistical summaries needed for data mining can be deduced. Since the individual values are modified, the privacy can be preserved to some extent. Another advantage of this approach is that it enables individuals to modify their own private data according to some guidelines before it is revealed. Data perturbation approach have been used extensively by the U.S. Census Bureau for protecting public use microdata sets. Data perturbation techniques can be categorized into two broad categories:

- Data Swapping: Data values are swapped between records without changing some certain statistics. For example, the age field of two patient records can be swapped without changing the average age statistics [57].

- Randomization: Data values are modified by adding some kind of random noise. For example, all the age values can be modified by adding a random value drawn from a normal distribution with mean 0, variance 1 [70].

The challenge is to obtain valid data mining results from the perturbed data while giving reasonable guarantees about the privacy.

In [4], an initial solution for privacy-preserving data mining is given. The key result is that the perturbed data, and information on the distribution of the random data used to perturb the data, can be used to generate an approximation to the original data *distribution*, without revealing the original data *values*. The distribution is used to improve mining results over mining the perturbed data directly, primarily

through selection of split points to "bin" continuous data. Later refinement of this approach tightened the bounds on what private information is disclosed, by showing that the ability to reconstruct the distribution can be used to tighten estimates of original values based on the distorted data [2]. In [62], similar techniques were used for doing collaborative filtering.

More recently, the data perturbation approach has been applied to boolean association rules [34, 67]. Again, the idea is to modify data values such that reconstruction of the values for any individual transaction is difficult, but the rules learned on the distorted data are still valid. One interesting feature of this work is a flexible definition of privacy; e.g., the ability to correctly guess a value of '1' from the distorted data can be considered a greater threat to privacy than correctly learning a '0'. Methods for limiting privacy breaches for perturbed boolean attributes are given in [33].

Although data perturbation approaches are simple to implement, the privacy guarantees provided by these approaches are not clear. For example, In [49], it is shown that some of the suggested perturbation approaches do not protect privacy as intended.

The data distortion approach addresses a different problem from our work. The assumption with perturbation is that the values must be kept private from whoever is doing the mining. We instead assume that *some* parties are allowed to see *some* of the data, just that nobody is allowed to see *all* the data. In return, we are able to get exact, rather than approximate, results with provable security properties.

2.4 Privacy-preserving Distributed Data Mining

Imagine the case where different drug companies want to combine their clinical trial databases for doing data mining to discover new drugs. Although each company can access its own database, due to privacy concerns and regulations, it cannot access the data in other drug companies. Even more, they may not want to reveal

some certain statistics for competition reasons. Privacy-preserving distributed data mining (PPDDM) tries to address this problem. In this line of work, the goal is to learn the data mining results without revealing anything other than the result itself. The solutions suggested for this problem treat privacy-preserving distributed data mining as a special case of secure multi-party computation (Chapter 3.2) and not only aims for preserving individual privacy but also tries to preserve leakage of any information other than the final result. Recently, a different framework for PPDDM has been suggested [29]. In that work, perturbation techniques are combined with query restrictions to give very limited but provably secure solutions.

Usually PPDDM solutions are given under two different data partition assumptions:

- Vertical Partitioning: This model assumes that different sites collect information about the same set of entities but they collect different feature sets. For example, both a university and a hospital may collect information about a student. The university may keep some information related to the student's academic success and the hospital may keep some information related to the student's physical condition. The hospital and the university may want to do data mining to reveal the relationship between physical health and academic success.

- Horizontal Partitioning: This model assumes that different sites collect the same set of information about different entities. For example, different credit card companies may collect credit card transactions of different individuals.

In relational terms, with horizontal partitioning the relation to be mined is the union of the relations at the sites. In vertical partitioning, the relations at the individual sites must be joined to get the relation to be mined. The change in the way the data is partitioned makes it a much different problem and requires a very different set of solutions. We describe the solutions provided for these two different data partition models below.

2.4.1 Privacy-preserving Distributed Data Mining on Vertically Partitioned Data

Many solutions are suggested for PPDDM on vertically partitioned data. The problem of privately computing association rules on *vertically* partitioned distributed data has been addressed in [72]. A solution is given for ID3 tree based classification in [28]. In [71], a detailed survey of PPDDM over vertically partitioned data is given.

2.4.2 Privacy-preserving Distributed Data Mining on Horizontally Partitioned Data

In the first work on PPDDM [53], the goal is to securely build an ID3 decision tree where the training set is distributed between two parties (i.e, data is horizontally partitioned). The basic idea is that finding the attribute that maximizes information gain is equivalent to finding the attribute that minimizes the conditional entropy. The conditional entropy for an attribute for two parties can be written as a sum of the expression of the form $(v_1 + v_2) \times \log(v_1 + v_2)$. The authors give a way to securely calculate the expression $(v_1 + v_2) \times \log(v_1 + v_2)$ and show how to use this function for building the ID3 securely. (Details are given in Chapter 3.5.4) In [52], secure clustering using the expectation maximization method is given for horizontally partitioned data.

2.5 Our Contribution

This thesis significantly expands the available algorithms for privacy-preserving data mining on horizontally partitioned data. We provide privacy-preserving solutions for:

- association rule mining (Chapter 4),
- k-nn methods (Chapter 5), and
- Naïve Bayes classification (Chapter 6).

We also addressed the issues in privacy-preserving data mining that were not considered at all prior to our work. We provide solutions so that the use of data mining results do not violate privacy (Chapter 7 and Chapter 8).

3 GENERAL SECURE MULTI-PARTY COMPUTATION AND CRYPTOGRAPHIC TOOLS

3.1 Introduction

One of the goals of this dissertation is to provide efficient solutions for mining horizontally partitioned data without disclosing anything other than the result itself. A closely related question that comes to mind is: what kind of functions can be evaluated without revealing anything other than the function result? The answer shown by Yao [77] is: any function that can be represented as a polynomial-size circuit. This result indicates that almost any data mining task can be done without violating privacy. Although this generic result is too inefficient to be used for data mining, it provides a framework and a general methodology to prove security of the proposed solutions. In this dissertation, we use the above framework to provide provably secure (i.e., nothing is revealed other than the result itself) and efficient solutions.

Strong theoretical foundations for secure multi-party computation and many cryptographic protocols already exist. To enhance the understanding of this dissertation, we review these concepts in this chapter.

3.2 Secure Multi-party Computation

Imagine the case, where two millionaires want to learn who is richer. (Yao's Millionaire problem [77]) Also, due to privacy reasons, they do not want to disclose their net worth to each other. We can easily solve this problem using a trusted third party. Each millionaire can send his or her input to the trusted party. Later, the trusted party can send the final result back to the millionaires. Assuming the

communication between the trusted party and the millionaires is secure, millionaires only learn who is richer. The obvious question is what happens if the millionaires do not trust any one. The goal of secure multi-party computation is to come up with solutions where we can reach the privacy level of having a third trusted party without actually having one.

Substantial work has been done on secure multi-party computation (SMC). The key result is that a wide class of computations can be computed securely under reasonable assumptions. We give a brief overview of this work, concentrating on material that is used later in the thesis. The definitions given here are from Goldreich [39]. For simplicity, we concentrate on the two party case. Extending the definitions to the multi-party case is straightforward.

3.2.1 Security in Semi-honest model

A semi-honest party follows the rules of the protocol using its correct input, but is free to later use what it sees during execution of the protocol to compromise security. This is somewhat realistic in the real world because parties who want to mine data for their mutual benefit will follow the protocol to get correct results. In some cases, parties may only want to learn the results but not the data itself. For example, recently one credit card transaction processing company was hacked and the credit card transactions that involve 40 million credit card numbers were stolen. Apparently, the transaction data was stored only to do data mining. On the other hand, the loss of the data made the company vulnerable to potential law suits [11]. Similar examples can be also given for health care data. Clearly such companies may follow the protocol to just learn the data mining results and nothing else. Also a protocol that is buried in a complex software may be difficult to alter.

A formal definition of private two party computation in the semi-honest model is given below. Computing a function privately is equivalent to computing it in the ideal model where we can use a third trusted party [39].

Definition 3.2.1 *(privacy w.r.t. semi-honest behavior): [39]*

Let $f : \{0,1\}^* \times \{0,1\}^* \longmapsto \{0,1\}^* \times \{0,1\}^*$ *be probabilistic, polynomial-time functionality, where* $f_1(x,y)$*(resp.,* $f_2(x,y)$*) denotes the first (resp., second) element of* $f(x,y)$*) and let* Π *be two-party protocol for computing* f*.*

Let the view of the first (resp. second) party during an execution of Π *on* (x,y)*, denoted* $view_1^{\Pi}(x,y)$ *(resp.,* $view_2^{\Pi}(x,y)$*) is* $(x, r_1, m_1, \ldots, m_t)$ *(resp.,* $(y, r_2, m_1, \ldots, m_t)$*) where* r_1 *represents the outcome of the first (resp.,* r_2 *second) party's internal coin tosses, and* m_i *represents the* i^{th} *message it has received.*

The output of the first (resp., second) party during an execution of Π *on* (x,y) *is denoted* $output_1^{\Pi}(x,y)$ *(resp.,* $output_2^{\Pi}(x,y)$*) and is implicit in the party's view of the execution.*

Π *privately computes* f *if there exist probabilistic polynomial time algorithms, denoted* S_1, S_2 *such that*

$$\{(S_1(x, f_1(x,y)), f_2(x,y))\}_{x,y\in\{0,1\}^*} \equiv^C \left\{\left(view_1^{\Pi}(x,y), output_2^{\Pi}(x,y)\right)\right\}_{x,y\in\{0,1\}^*} \tag{3.1}$$

$$\{(f_1(x,y), S_2(x, f_1(x,y)))\}_{x,y\in\{0,1\}^*} \equiv^C \left\{\left(output_1^{\Pi}(x,y), view_2^{\Pi}(x,y)\right)\right\}_{x,y\in\{0,1\}^*} \tag{3.2}$$

where $\equiv^C$ *denotes computational indistinguishability.*

The above definition says that a computation is secure if the view of each party during the execution of the protocol can be effectively simulated by the input and the output of the party. This is not quite the same as saying that private information is protected. For example, if two parties use a secure protocol to mine distributed association rules, a secure protocol still reveals that if a particular rule is not supported by particular site and that rule appears in the globally supported rule set then it must be supported by the other site. A site can deduce this information by solely looking at its locally supported rules and the globally supported rules. On the other hand, there is no way to deduce the exact support count of some itemset by looking at the globally supported rules. With three or more parties, knowing

a rule holds globally reveals that at least one site supports it, but no site knows which site (other than, obviously, itself). In summary, secure multi-party protocol will not reveal more information to a particular party than the information that can be induced by looking at that party's input and the output.

We also use a key result from the Secure Multi-party Computation literature, the composition theorem. We state it for the semi-honest model. A detailed discussion of this theorem, as well as the proof, can be found in [39].

Theorem 3.2.1 *(Composition Theorem for the semi-honest model) [39]: Suppose that g is privately reducible to f and that there exists a protocol for privately computing f. Then there exists a protocol for privately computing g.*

This allows us to use existing secure protocols as components in a black-box fashion.

3.2.2 Yao's General Two Party Secure Function Evaluation

Yao's general secure two party evaluation is based on expressing the function $f(x, y)$ as a circuit and encrypting the gates for secure evaluation [77]. With this protocol any two party function can be evaluated securely in semi-honest model but the functions that can be efficiently evaluated must have small circuit representation. We will not give details of this generic method here. In some of our protocols, we will use this generic result for finding out whether $a \geq b$ for arbitrary a, b (Yao's millionaire problem). For comparing any two integers securely, Yao's generic method is one of the most efficient methods known, although other asymptotically equivalent but practically more efficient algorithms could be used as well [45].

3.2.3 Commutative Encryption

Commutative encryption is an important tool that can be used in many privacy-preserving protocols. An encryption algorithm is commutative if the following two

equations hold for any given feasible encryption keys $K_1, \ldots, K_n \in K$, any m in items domain M, and any permutations of i, j.

$$E_{K_{i_1}}(\ldots E_{K_{i_n}}(M)\ldots) = E_{K_{j_1}}(\ldots E_{K_{j_n}}(M)\ldots) \tag{3.3}$$

$\forall M_1, M_2 \in M$ such that $M_1 \neq M_2$ and for given k, $\epsilon < \frac{1}{2^k}$

$$Pr(E_{K_{i_1}}(\ldots E_{K_{i_n}}(M_1)\ldots) = E_{K_{j_1}}(\ldots E_{K_{j_n}}(M_2)\ldots)) < \epsilon \tag{3.4}$$

These properties of commutative encryption can be used to check whether two items are equal without revealing them. For example, assume that party A has item i_A and party B has item i_B. To check if the items are equal, each party encrypts its item and sends it to the other party: Party A sends $E_{K_A}(i_A)$ to B and party B sends $E_{K_B}(i_B)$ to A. Each party encrypts the received item with its own key, giving party A $E_{K_A}(E_{K_B}(i_B))$ and party B $E_{K_B}(E_{K_A}(i_A))$. At this point, they can compare the encrypted data. If the original items are the same, Equation 3.3 ensures that they have the same encrypted value. If they are different, Equation 3.4 ensure that with high probability they do not have the same encrypted value. During this comparison, each site sees only the other site's values in encrypted form.

In addition to meeting the above requirements, we require that the encryption be secure. Specifically, the encrypted values of a set of items should reveal no information about the items themselves. Consider the following experiment. For any two sets of items, we encrypt each item of one randomly chosen set with the same key and present the resulting encrypted set and the initial two sets to a polynomial-time adversary. Loosely speaking, our security assumption implies that this polynomial-time adversary will not be able to predict which of the two sets were encrypted with a probability better than a random guess. Under this security assumption, it can be shown that the resulting encrypted set is indistinguishable by a polynomial adversary from a set of items that are randomly chosen from the domain of the encryption; this fact is used in the proof of the privacy-preserving properties of our protocol. The formal definition of multiple-message semantic security can be found in [40].

There are several examples of commutative encryption, perhaps the most famous being RSA [66] (if keys are not shared). The following part describes a how Pohlig-Hellman encryption [61] can be used to fulfill our requirements, as well as further discussion of relevant cryptographic details.

Pohlig-Hellman Encryption Scheme

The Pohlig-Hellman encryption scheme [61] can be used for a commutative encryption scheme meeting the requirements of Chapter 3.2.3. Pohlig-Hellman works as follows. Given a large prime p with no small factors of $p-1$, each party chooses a random e, d pair such that $e*d = 1 \pmod{p-1}$. The encryption of a given message M is $M^e \pmod{p}$. Decryption of a given ciphertext C is done by evaluating $C^d \pmod{p}$. $C^d = M^{ed} \pmod{p}$, and due to Fermat's little theorem, $M^{ed} = M^{1+k(p-1)} = M \pmod{p}$.

It is easy to see that Pohlig-Hellman with shared p satisfies equation 3.3. Let us assume that there are n different encryption and decryption pairs $((e_1, d_1), \ldots, (e_n, d_n))$. For any permutation function i, j and $E = e_1 * e_2 * \cdots * e_n = e_{i_1} * e_{i_2} \ldots e_{i_n} = e_{i_1} * e_{i_2} \ldots e_{i_n} \pmod{p-1}$:

$$\begin{aligned} E_{e_{i_1}}(\ldots E_{e_{i_n}}(M)\ldots) &= (\ldots((M^{e_{i_n}} \pmod{p})^{e_{i_{n-1}}} \pmod{p})\ldots)^{e_{i_1}} \pmod{p}) \\ &= M^{e_{i_n} * e_{i_{n-1}} \cdots * e_{i_1}} \pmod{p} \\ &= M^E \pmod{p} \\ &= M^{e_{j_n} * e_{j_{n-1}} \cdots * e_{j_1}} \pmod{p} \\ &= E_{e_{j_1}}(\ldots E_{e_{j_n}}(M)\ldots) \end{aligned}$$

Equation 3.4 is also satisfied by the Pohlig-Hellman encryption scheme. Let $M_1, M_2 \in GF(p)$ such that $M_1 \neq M_2$. Any order of encryption by all parties is equal to evaluating E^{th} power mod p of the plain text. Let us assume that after encryptions M_1 and M_2 are mapped to the same value. This implies that $M_1^E = M_2^E \pmod{p}$. By exponentiating both sides with $D = d_1 * d_2 * \cdots * d_n \pmod{p-1}$, we get

$M_1 = M_2 \pmod{p}$, a contradiction. (Note that $E*D = e_1*e_2*\cdots*e_n*d_1*d_2*\ldots d_n = e_1 * d_1 \ldots e_n * d_n = 1 \pmod{p-1}$.) Therefore, the probability that two different elements map to the same value is zero.

Direct implementation of Pohlig-Hellman is not secure. Consider the following example, encrypting two values a and b, where $b = a^2$. $E_e(b) = E_e(a^2) = (a^2)^e \pmod{p} = (a^e)^2 \pmod{p} = (E_e(a))^2 \pmod{p}$. This shows that given two encrypted values, it is possible to determine if one is the square of the other (even though the base values are not revealed.) This violates the security requirement of Chapter3.2.3.

Huberman et al. provide a solution [44]. Rather than encrypting items directly, a hash of the items is encrypted. The hash occurs only at the initial site, the second and later encryption of items can use Pohlig-Hellman directly. Under the random oracle assumption (i.e., the output of the hash function looks like random), the hash breaks the relationship revealed by the encryption (e.g., $a = b^2$). After decryption, the hashed values must be mapped back to the original values. This can be done by hashing the original items to build a lookup table.

We present the approach from [44] as an example; any secure encryption scheme that satisfies Equations 3.3 and 3.4 can be used in our protocols. Other approaches, and further definitions and discussion of their security, can be found in [8, 25, 31, 69].

3.3 Third Untrusted Non-colluding Site

In order to achieve solutions that are both secure and efficient, we sometimes make use of an untrusted, non-colluding site. Use of such a site was first suggested in [35]. Application to privacy-preserving data mining was discussed in [48]. The key to such a site is that it learns nothing, however by colluding with other sites it may be able to obtain information that should not be revealed. Thus, the only trust placed in the site is that it will not collude with any of the other sites to violate privacy. Although this seems like a strong assumption, this non-colluding assumption often occurs in real life. For example, bidders or sellers on e-bay assume that e-bay is not

colluding with other bidders or sellers against them. We stress that the untrusted site learns nothing except the information already available publicly available. On the other hand, the site described in Chapter 3.2 is trusted with all the information.

3.4 Probabilistic Public-Key Encryption

Although RSA style public key encryption is an useful tool for many cryptographic protocols, the problem with deterministic public key algorithm is that two items corresponding to the same plaintext map to the same ciphertext. This may reveal the entropy of the input of each site as well as some other information related to the site. Even if no site may learn what the result is, but something of the distribution is revealed.

Fortunately, the cryptography community has developed a solution: *probabilistic* public-key encryption. The idea behind probabilistic public-key encryption is that the same plaintext may map to different ciphertexts, but these will all map back to the same plaintext when decrypted. We will briefly describe the Blum-Goldwasser [10] probabilistic encryption scheme as an example.

3.4.1 Blum-Goldwasser Probabilistic Encryption Scheme

Let $n = pq$ where p and q are the same size and $p, q \equiv 7 \bmod 8$. n is the public key and p, q are the secret key. Given the message m and key n encryption proceeds as follows:

1. Choose r randomly such that $r = x^2 \pmod{n}$ for some x
2. Compute $r^2, r^4, \ldots, r^{2^{|m|}} \pmod{n}$
3. Generate an $|m|$ bit number t such that the i^{th} bit of t $(0 \leq i \leq |m| - 1)$ is the least significant bit of $r^{2^i} \pmod{n}$
4. Set ciphertext $c = m \oplus t, r^{2^{|m|}} \pmod{n}$ where $\oplus$ is the xor operation

Given ciphertext $c = (m', r')$, decryption is performed by:

1. Compute r such that $r^{2^{|m'|}} = r' \pmod{n}$ (This can be done easily if the factorization of n is known.)
2. Compute $r, r^2, \ldots, r^{2^{|m'|-1}} \pmod{n}$
3. Generate an $|m'|$ bit number t' such that the i^{th} bit of t' $(0 \leq i \leq |m'| - 1)$ is the least significant bit of $r^{2^i} \pmod{n}$
4. Set $m = m' \oplus t'$ where $\oplus$ is the xor operation

Under the assumption that factoring is hard, it can be shown that the above encryption is secure and that the output of the encryption is computationally indistinguishable from a randomly generated string. The details of this method can be found in [10].

3.5 Secure Algebraic Calculations

Some of the privacy-preserving algorithms developed later are based on the several secure two-party computation protocols. While most have either been previously published, or are straightforward given previously published work, we summarize them here for completeness.

3.5.1 Secure Sum

One building block frequently required is a way to securely calculate the sum of values from individual sites. Assuming three or more parties and no collusion, the following method from [46] securely computes such a sum.

Assume that the value $v = \sum_{i=1}^{k} v_i$ to be computed is known to lie in the range $[0..n-1]$ where v_i denotes the share of the i^{th}.

One site is designated the *master* site, numbered 1. The remaining sites are numbered $2..k$. Site 1 generates a random number R, uniformly chosen from $[0..n-1]$.

Site 1 adds this to its local value v_1, and sends the sum $R+v_1 \bmod n$ to site 2. Since the value R is chosen uniformly from $[0..n-1]$, the number $R + v_1 \bmod n$ is also distributed uniformly across this region, so site 2 learns nothing about the actual value of v_1.

For the remaining sites $i = 2..k-1$, the algorithm is as follows. Site i receives

$$V = R + \sum_{j=1}^{i-1} v_j \bmod n.$$

Since this value is uniformly distributed across $[0..n-1]$, i learns nothing. Site i then computes

$$R + \sum_{j=1}^{i} v_i \bmod n = (v_i + V) \bmod n$$

and passes it to site $i+1$.

Site k performs the above step, and sends the result to site 1. Site 1, knowing R, can subtract R to get the actual result. Note that site 1 can also determine $\sum_{i=2}^{k} v_i$ by subtracting v_1. This is possible from the global result *regardless of how it is computed*, so site 1 has not learned anything from the computation.

This method faces an obvious problem if sites collude. Sites $i-1$ and $i+1$ can compare the values they send/receive to determine the exact value for v_i. The method can be extended to work for an honest majority. Each site divides v_i into shares. The sum for each share is computed individually. However, the path used is permuted for each share, such that no site has the same neighbor twice. To compute v_i, the neighbors of i from each iteration would have to collude. Varying the number of shares varies the number of dishonest (colluding) parties required to violate security.

3.5.2 1-out-of-N Oblivious Transfer

The 1-out-of-N Oblivious Transfer protocol involves two parties, Alice and Bob. Alice has an input $\sigma, 1 \leq \sigma \leq N$, while Bob has N inputs $X_1, \ldots, X_n$. At the end of the protocol, Alice learns only X_σ and nothing else while Bob learns nothing at all.

The 1-out-of-2 Oblivious Transfer (OT_1^2) was suggested by Even, Goldreich and Lempel [32] as a generalization of Rabin's "oblivious transfer" [64]. Naor and Pinkas [59] provide efficient protocols for 1-out-of-N Oblivious Transfer. For completeness, we now describe a very simple (though inefficient) method for doing Oblivious Transfer for semi-honest parties.

Bob generates N public key pairs $E_1, D_1, \ldots, E_N, D_N$

Bob sends $E_1, \ldots, E_N$ to Alice.

Alice generates an asymmetric key K.

Alice forms the vector $\vec{V}$: if $i = \sigma$, $V_i = E_i(K)$, otherwise $V_i =$ (a random) R_j.

Alice sends the N-dimensional vector $\vec{V}$ to Bob

Bob decrypts $\vec{V}$ to form the vector $\vec{K}$ where $K_i = D_i(V_i)$

Bob encrypts his data items with the keys in $\vec{K}$ ands sends them to Alice (i.e. Bob sends $K_i(X_i), i = 1 \ldots N$ to Alice)

Since $K_\sigma = D_\sigma(E_\sigma(K)) = K$, Alice decrypts the σ row with K to get X_σ

Clearly this protocol reveals nothing to Bob [39]. In the semi-honest model, as long as Alice acts exactly according to the protocol, she too does not learn anything since all the other values are encrypted with random keys unknown to her. Though it is easy to break this protocol when parties are allowed to be malicious, better protocols (more secure and efficient) can easily be found in the literature.

3.5.3 Oblivious Evaluation of Polynomials

Alice has a polynomial P of degree k over some finite field $\mathcal{F}$. Bob has an element $x \in \mathcal{F}$ and also knows k. Alice would like to let Bob compute the value $P(x)$ in such a way that Alice does not learn x and Bob does not gain any additional information about P (except $P(x)$). This problem was first investigated by [58]. Subsequently, there have been more protocols improving the communication and computation efficiency [23] as well as extending the problem to floating point numbers [16]. For our protocols, we use the protocol given in [23] since it requires only $O(k)$ exponen-

tiations in order to evaluate a polynomial of degree k (where the constant is very small). This works well since we only require evaluation of low-degree polynomials.

We now briefly describe the protocol used for oblivious polynomial evaluation. This description is excerpted from [53]: Let $P(y) = \sum_{i=0}^{k} a_i y^i$ be Alice's input and x be Bob's input. The following protocol enables Bob to compute $g^{P(x)}$, where g is a generator of a group in which the Decisional Diffie-Hellman assumption holds. The protocol can be converted to one computing $P(x)$ using the methods of Paillier [60], who presented a trapdoor for computing discrete logs. The protocol is quite simple since the parties are assumed to be semi-honest. Bit-commitment and zero knowledge proofs can be used to achieve security against malicious parties. The protocol consists of the following steps:

Bob chooses a secret key s, and sends g^s to Alice.

for $i = 0 \ldots k$ **do**

Bob generates a random r_i.

Bob computes $c_i = (g^{r_i}, g^{sr_i} g^{x^i})$.

end for

Bob sends $c_0, \ldots, c_k$ to Alice.

Alice computes $C = \prod_{i=0}^{k} (c_i)^{a_i} = (g^R, g^{sR} g^{P(x)})$, where $R = \sum_{i=0}^{k} r_i a_i$.

Alice chooses a random value r and computes $C' = (g^R g^r, g^{sR} g^{P(x)} g^{sr})$.

Alice sends C' to Bob.

Bob divides the second element of C' by the first element of C' raised to the power of s, and obtains $g^{P(x)}$.

By the DDH assumption, Alice learns nothing of x^i from the messages $c_0, \ldots, c_k$ sent by Bob to her. On the other hand, Bob learns nothing of P from C'.

3.5.4 Privately Computing $\ln x$

In classifying an instance, we need to be able to privately compute $\ln x$, where $x = x_1 + x_2$ with x_1 known to P_1 and x_2 known to P_2. Thus, P_1 should get y_1 and P_2

should get y_2 such that $y_1 + y_2 = \ln x = \ln(x_1 + x_2)$. One of the key results presented in [54] was a cryptographic protocol for this computation. We now describe the protocol in brief: Note that $\ln x$ is *Real* while general cryptographic tools work over finite fields. We multiply the $\ln x$ with a known constant to make it integral.

The basic idea behind computing random shares of $\ln(x_1+x_2)$ is to use the Taylor approximation for $\ln x$. Remember that the Taylor approximation gives us:

$$\begin{aligned}
&\ln(1+\epsilon) \\
&= \sum_{i=1}^{\infty} \frac{(-1)^{i-1}\epsilon^i}{i} \\
&= \epsilon - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3} - \frac{\epsilon^4}{4} + \ldots \text{for } -1 < \epsilon < 1
\end{aligned}$$

For an input x, let $n = \lfloor \log_2 x \rfloor$. Then 2^n represents the closest power of 2 to x. Therefore, $x = x_1 + x_2 = 2^n(1+\epsilon)$ where $-1/2 \leq \epsilon \leq 1/2$. Consequently,

$$\begin{aligned}
\ln(x) &= \ln(2^n(1+\epsilon)) \\
&= \ln 2^n + \ln(1+\epsilon) \\
&\approx \ln 2^n + \sum_{i=1\ldots k} (-1)^{i-1}\epsilon^i/i \\
&= \ln 2^n + T(\epsilon)
\end{aligned}$$

where $T(\epsilon)$ is a polynomial of degree k. This error is exponentially small in k.

There are two phases to the protocol. Phase 1 finds an appropriate n and ϵ. Let N be a predetermined (public) upper-bound on the value of n. First, Yao's circuit evaluation is applied to the following small circuit which takes x_1 and x_2 as input and outputs random shares of $\epsilon 2^N$ and $2^N n \ln 2$. Note that $\epsilon 2^n = x - 2^n$, where n can be determined by simply looking at the two most significant bits of x and $\epsilon 2^N$ is obtained simply by shifting the result by $N - n$ bits to the left. Thus, the circuit outputs random α_1 and α_2 such that $\alpha_1 + \alpha_2 = \epsilon 2^N$, and also outputs random β_1 and β_2 such that $\beta_1 + \beta_2 = 2^N n \ln 2$. This circuit can be easily constructed. Random shares are obtained by having one of the parties input random values $\alpha_1, \beta_1 \in_R \mathcal{F}$ into the circuit and having the circuit output $\alpha_2 = \epsilon 2^N - \alpha_1$ and $\beta_2 = 2^N n \ln 2 - \beta_1$ to the other party.

Phase 2 of the protocol involves computing shares of the Taylor series approximation, $T(\epsilon)$. This is done as follows: P_1 chooses a random $w_1 \in \mathcal{F}$ and defines a polynomial $Q(x)$ such that $w_1 + Q(\alpha_2) = T(\epsilon)$. Thus $Q(\cdot)$ is defined as

$$Q(x) = \text{lcm}(2, \ldots, k) \sum_{i=1}^{k} \frac{(-1)^{i-1}}{2^{N(i-1)}} \frac{(\alpha_1 + x)^i}{i} - w_1$$

P_1 and P_2 then execute an oblivious polynomial evaluation with P_1 inputting $Q(\cdot)$ and P_2 inputting α_2, in which P_2 obtains $w_2 = Q(\alpha_2)$. P_1 and P_2 define $u_1 = lcm(2, \ldots, k)\beta_1 + w_1$ and $u_2 = lcm(2, \ldots, k)\beta_2 + w_2$. We have that $u_1 + u_2 \approx 2^N lcm(2, \ldots, k) \ln x$

Further detail on the protocol, as well as the proof of security, can be found in [54].

3.5.5 Secure Set Intersection

There are many different secure set intersection protocols. [37, 73]. Here, we give a brief description of a protocol based on [37].

Assume that each site has n elements, where Site S_1 has $a_1 \ldots a_n$ and Site S_2 has $b_1 \ldots b_n$. Using a homomorphic public key encryption (i.e., secure encryption that satisfies $E(a).E(b) = E(a+b), (E(a))^c = E(ca)$), S_1 can generate a polynomial $P(y) = \Pi_{i=1}^{n}(a_i - y) = \sum_{i=0}^{n}(c_i * y^i)$ with n+1 coefficient and encrypt each coefficient with homomorphic public key encryption. Note that if any of the b_i is in the intersection then $P(b_i)$ will be zero. Since the coefficients are encrypted, there is no way that site S_2 can learn the polynomial P but can calculate the encrypted polynomial result multiplied with a random number using the above mentioned properties of the homomorphic encryption. Therefore S_2 computes $E(rP(b_i) + r_{b_i})$ for every element b_i, where r_{b_i} is a unique random number for b_i. If b_i is in the intersection set then S_1 will get r_{b_i}. Now S_1 and S_2 can use a small secure circuit evaluation to check whether the intersection size is bigger than the threshold. The circuit basically checks whether the i^{th} result retrieved by the S_1 after decryption is equal to r_{b_i} and

then checks the threshold. At the end of the protocol each site (or even only one of the sites) learns only if the threshold is exceeded or not.

3.5.6 Connections Between Secure Computation Methods

We have utilized many different secure computation subroutines in our protocols. In an actual implementation, a few well implemented secure protocols suffice. The rest can be easily built from those basic few protocols: secure summation and polynomial evaluation. Secure $\ln(x)$ (Chapter 3.5.4) is already defined using secure polynomial evaluation. We can also give a simple protocol for secure dot product using polynomial evaluation. Assume Alice has $A = a_1, a_2, \ldots, a_k$ and Bob has $B = b_1, b_2, \ldots, b_k$.

Alice chooses k randoms $r_1, r_2, \ldots, r_k$.

Alice forms k degree 1 polynomials Bob forms

$$\vec{U} = \begin{bmatrix} P_1(y) = a_1 y + r_1 \\ P_2(y) = a_2 y + r_2 \\ \vdots \\ P_k(y) = a_k y + r_k \end{bmatrix} \qquad \vec{V} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}$$

Alice and Bob engage in k degree 1-polynomial evaluation in parallel so that (only) Bob gets $P_k(b_k)$ for all k:

$$\vec{X} = \begin{bmatrix} P_1(b_1) \\ P_2(b_2) \\ \vdots \\ P_k(b_k) \end{bmatrix}$$

Alice forms Bob forms

$R_a = \sum_{i=1}^{k} r_i$ $R_b = \sum_{i=1}^{k} \vec{X}(i)$

Clearly $R_b + (-R_a)$ is the desired dot product result. Also given a dot product protocol, we can easily create a protocol for polynomial evaluation. Let $P(y) = \sum_{i=0}^{k} a_i y^i$

be Alice's input and x be Bob's input, using secure dot product, Bob can evaluate the $P(x)$ as follows

Alice forms Bob forms

$$\vec{U} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} \quad \vec{V} = \begin{bmatrix} 1 \\ x \\ \vdots \\ x^k \end{bmatrix}$$

Alice and Bob engage in secure dot product so that (only) Bob gets $r = \vec{U}.\vec{V}$ Clearly $r = \sum_{i=0}^{k} a_i x^i = P(x)$.

The above simple and efficient conversions indicate that a toolbox with implementation of few protocols may be sufficient for implementing a complex privacy-preserving data mining algorithms.

4 PRIVACY-PRESERVING DISTRIBUTED ASSOCIATION RULE MINING

4.1 Introduction

This chapter addresses the problem of efficiently computing association rules when the goal is to reveal only the association rules and nothing else. We assume homogeneous databases: All sites have the same schema, but each site has information on different entities. The goal is to produce association rules that hold globally, while limiting the information shared about each site.

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

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

Note that this doesn't require sharing any individual transactions. We can easily extend an algorithm such as a-priori [3] to the distributed case using the following lemma: If a rule has $support > k\%$ globally, it must have $support > k\%$ on at least one of the individual sites. A distributed algorithm for this would work as follows: Request that each site send all rules with support at least k. For each rule returned, request that all sites send the count of their transactions that support the rule, and the total count of all transactions at the site. From this, we can compute the global support of each rule, and (from the above lemma) be certain that all

rules with support at least k have been found. More thorough studies of distributed association rule mining can be found in [17, 18].

The above approach protects individual data privacy, but it does require that each site disclose what rules it supports, and how much it supports each potential global rule. What if this information is sensitive? For example, suppose that a trade association of steel mini-mills would like to help members improve their ability to compete with large integrated mills in producing high-grade steel. The members have tried to produce high-grade steels, but the products produced have a much higher defect rate than those from the integrated mills. It appears that certain combinations of raw materials are not suitable for producing the high-grade steels, but the number of types of materials (recycled steel, purchased slab, etc.) make it cost-prohibitive for an individual mill to experiment sufficiently to determine the offending combinations. Instead, they agree to a common representation for raw materials and defect rates, and to share information to produce globally valid association rules linking raw materials and defects. From this, they produce a rule $scrap\_iron\&high\_carbon\_steel \Rightarrow Stress\_fractures$.

Suppose that one mini-mill has developed a process that allows them to produce high-grade steel using a combination of scrap iron and high carbon steel but still wants to improve its process by using others experience. This new process is a trade secret, allowing them to be more profitable than their competitors. Sharing their support for the above rule and the antecedent gives the other mini-mills the knowledge that such a process is possible – and they will increase research (or corporate spying), erasing advantage of the secret.

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

The method presented here assumes three or more parties. In the two party case, knowing a rule is supported globally and not supported at one's own site reveals that

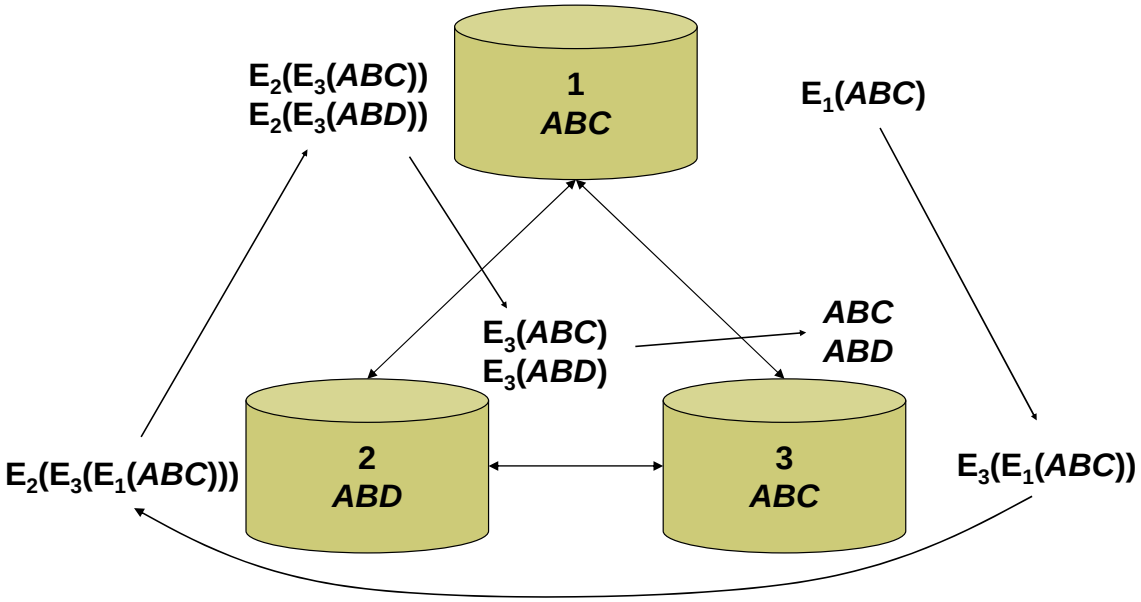

Figure 4.1. Determining global candidate itemsets

the other site supports the rule. Thus, much of the knowledge we try to protect is revealed even with a completely secure method for computing the global results. We discuss the two party case further in Chapter 4.5. By the same argument, we assume no collusion, as colluding parties can reduce this to the two party case.

4.1.1 Private Association Rule Mining Overview

Our method follows the basic approach outlined on Page 27 except that values are passed between the local data mining sites rather than to a centralized combiner. The two phases are discovering candidate itemsets (those that are frequent on one or more sites), and determining which of the candidate itemsets meet the global support/confidence thresholds.

The first phase (Figure 4.1) uses commutative encryption. Each party encrypts its own itemsets, then the (already encrypted) itemsets of every other party. These are passed around, with each site decrypting, to obtain the complete set.

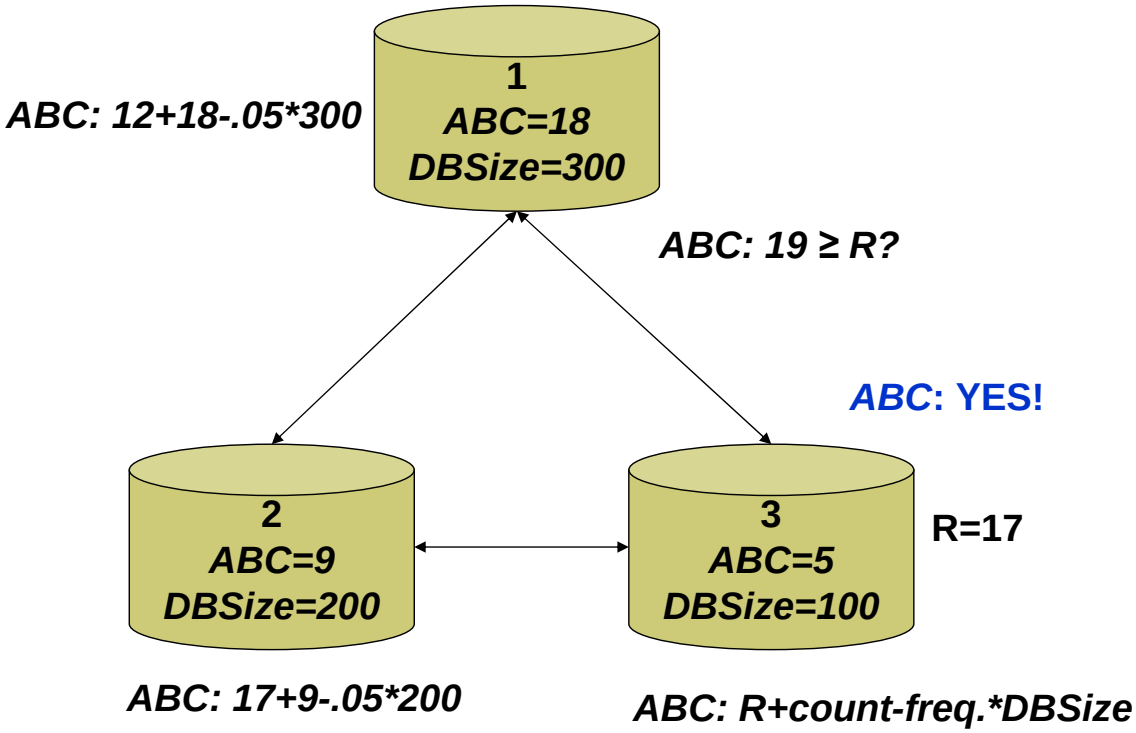

Figure 4.2. Determining if itemset support exceeds threshold

In the second phase (Figure 4.2), an initiating party passes its support count, plus a random value, to its neighbor. The neighbor adds its support count and passes it on. The final party then engages in a secure comparison with the initiating party to determine if the final result is greater than the threshold plus the random value.

This gives a brief, oversimplified idea of how the method works. Chapter 4.3 gives full details. Before going into the details, we give relevant background and definitions.

4.2 Background

There are several bodies of work that serve as a basis for our work. This section summarizes association rule mining and related concepts used in our solution.

4.2.1 Mining of Association Rules

The association rules mining problem can be defined as follows: [3] Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. Let DB be a set of transactions, where each transaction T is an itemset such that $T \subseteq I$. Given an itemset $X \subseteq I$, a transaction T *contains* X if and only if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$ where $X \subseteq I, Y \subseteq I$ and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ has *support* s in the transaction database DB if $s\%$ of transactions in DB contain $X \cup Y$. The association rule holds in the transaction database DB with *confidence* c if $c\%$ of transactions in DB that contain X also contains Y. An itemset X with k items called k-itemset. The problem of mining association rules is to find all rules whose support and confidence are higher than certain user specified minimum support and confidence.

In this simplified definition of association rules, missing items, negatives and quantities are not considered. In this respect, transaction database DB can be seen as a 0/1 matrix where each column is an item and each row is a transaction. In this work, we use this view of association rules.

Distributed Mining of Association Rules

The above problem of mining association rules can be extended to distributed environments. Let us assume that a transaction database DB is horizontally partitioned among n sites (namely $S_1, S_2, \ldots, S_n$) where $DB = DB_1 \cup DB_2 \cup \cdots \cup DB_n$ and DB_i resides at site S_i $(1 \leq i \leq n)$. The itemset X has *local* support count of $X.sup_i$ at site S_i if $X.sup_i$ of the transactions contains X. The *global* support count of X is given as $X.sup = \sum_{i=1}^{n} X.sup_i$. An itemset X is *globally supported* if $X.sup \geq s \times (\sum_{i=1}^{n} |DB_i|)$. Global confidence of a rule $X \Rightarrow Y$ can be given as $\{X \cup Y\}.sup/X.sup$.

The set of large itemsets $L_{(k)}$ consists of all k-itemsets that are globally supported. The set of locally large itemsets $LL_{i(k)}$ consists of all k-itemsets supported locally at

site S_i. $GL_{i(k)} = L_{(k)} \cap LL_{i(k)}$ is the set of globally large k-itemsets locally supported at site S_i. The aim of distributed association rule mining is to find the sets $L_{(k)}$ for all $k > 1$ and the support counts for these itemsets, and from this compute association rules with the specified minimum support and confidence.

A fast algorithm for distributed association rule mining is given in Cheung et. al. [17]. Their procedure for fast distributed mining of association rules (FDM) is summarized below.

1. **Candidate Sets Generation**: Generate candidate sets $CG_{i(k)}$ based on $GL_{i(k-1)}$, itemsets that are supported by the S_i at the (k-1)-th iteration, using the classic a-priori candidate generation algorithm. Each site generates candidates based on the intersection of globally large (k-1) itemsets and locally large (k-1) itemsets.

2. **Local Pruning**: For each $X \in CG_{i(k)}$, scan the database DB_i at S_i to compute $X.sup_i$. If X is locally large S_i, it is included in the $LL_{i(k)}$ set. It is clear that if X is supported globally, it will be supported in one site.

3. **Support Count Exchange**: $LL_{i(k)}$ are broadcast, and each site computes the local support for the items in $\cup_i LL_{i(k)}$.

4. **Broadcast Mining Results**: Each site broadcasts the local support for itemsets in $\cup_i LL_{i(k)}$. From this, each site is able to compute $L_{(k)}$.

The details of the above algorithm can be found in [17].

4.3 Secure Association Rule Mining

We will now use the tools described in Chapter 3 to construct a distributed association rule mining algorithm that preserves the privacy of individual site results. The algorithm given is for three or more parties – the difficulty with the two party case is discussed in Chapter 4.5.

4.3.1 Problem Definition

Let $i \geq 3$ be the number of sites. Each site has a private transaction database DB_i. We are given support threshold s and confidence c as percentages. The goal is to discover all association rules satisfying the thresholds, as defined in Chapter 4.2.1. We further desire that disclosure be limited: No site should be able to learn contents of a transaction at any other site, what rules are supported by any other site, or the specific value of support/confidence for any rule at any other site, unless that information is revealed by knowledge of one's own data and the final result. (e.g., if a rule is supported globally but not at one's own site, we can deduce that at least one other site support the rule.) Here we assume no collusion (this is discussed further in Chapter 4.4.)

4.3.2 Method

Our method follows the general approach of the FDM algorithm [17], with special protocols replacing the broadcasts of $LL_{i(k)}$ and the support count of items in $LL_{(k)}$. We first give a method for finding the union of locally supported itemsets without revealing the originator of the particular itemset. We then provide a method for securely testing if the support count exceeds the threshold.

Secure union of locally large itemsets

In the FDM algorithm (Chapter 4.2.1), step 3 reveals the large itemsets supported by each site. To accomplish this without revealing what each site supports, we instead exchange locally large itemsets in a way that obscures the source of each itemset. We assume a commutative encryption algorithm with negligible collision probability (Chapter 3.2.3).

The main idea is that each site encrypts the locally supported itemsets, along with enough "fake" itemsets to hide the actual number supported. Each site then

encrypts the itemsets from other sites. Since Equation 3.3 holds, duplicates in the locally supported itemsets will be duplicates in the encrypted itemsets, and can be deleted. In addition, the decryption can occur in any order, so by permuting the encrypted itemsets we prevent sites from tracking the source of each itemset. The algorithm is given in Protocol 4.1. In the protocol F represents the data that can be used as fake itemsets. $|LLe_{i(k)}|$ represents the set of the encrypted k itemsets at site i. E_i is the encryption and D_i is the decryption by site i. Clearly, Protocol 4.1 finds the union without revealing which itemset belongs to which site. It is not, however, secure under the definitions of secure multi-party computation. It reveals the number of itemsets having common support between sites, e.g., sites 3, 5, and 9 all support some itemset. It does not reveal *which* itemsets these are, but a truly secure computation (as good as each site giving its input to a "trusted party") could not reveal even this count. Allowing innocuous information leakage (the number of itemsets having common support) allows an algorithm that is sufficiently secure with much lower cost than a fully secure approach. (Though this leakage is not acceptable for steel mill example given in the introduction)

Even the number of jointly supported itemsets can be masked by allowing sites to inject itemsets that aren't really supported locally (i.e, steel mills may include fake large itemsets to prevent disclosures); if not globally supported they will be filtered from the final result when global support is calculated. The commonly supported itemsets "leak" then becomes an upper bound rather than exact, at an increased cost in the number of candidates that must be checked for global support. This is still not truly zero-knowledge, but does limit confidence in the common support information leaked.

If we deem leakage of the number of commonly supported itemsets as acceptable, we can prove that this method is secure under the definitions of secure multi-party computation. We do this by augmenting the result with the leaked information, and proving that everything else seen during the protocol can be simulated. This technique can be quite powerful for generating reasonably secure and efficient protocols.

Protocol 4.1 Finding secure union of large itemsets of size k

Require: $N \geq 3$ sites numbered $0..N-1$, set F of non-itemsets.

Phase 0: Encryption of all the rules by all sites
for each site i **do**
  generate $LL_{i(k)}$ as in steps 1 and 2 of the FDM algorithm
  $LLe_{i(k)} = \emptyset$
  for each $X \in LL_{i(k)}$ **do**
    $LLe_{i(k)} = LLe_{i(k)} \cup \{E_i(X)\}$
  end for
  for $j = |LLe_{i(k)}| + 1$ to $\lfloor \frac{1}{minimum\ support} \rfloor$ **do**
    $LLe_{i(k)} = LLe_{i(k)} \cup \{E_i(\text{random selection from } F)\}$
  end for
end for

Phase 1: Encryption by all sites
for Round $j = 0$ to $N-1$ **do**
  if Round j= 0 **then**
    Each site i sends $LLe_{i(k)}$ to site $(i+1) \bmod N$
  else
    Each site i encrypts all items in $LLe_{(i-j \bmod N)(k)}$ with E_i and sends it to site $(i+1) \bmod N$
  end if
end for{At the end of Phase 1, site i has the itemsets of site $(i+1) \bmod N$ encrypted by every site}

Phase 2: Merge odd/even itemsets
Each site i sends $LLe_{i+1 \bmod N}$ to site $1 - (((i+1) \bmod N) \bmod 2)$
Site 0 sets $RuleSet_1 = \cup_{j=1}^{\lceil (N-1)/2 \rceil} LLe_{(2j-1)(k)}$
Site 1 sets $RuleSet_0 = \cup_{j=0}^{\lfloor (N-1)/2 \rfloor} LLe_{(2j)(k)}$

Phase 3: Merge all itemsets
Site 1 sends $RuleSet_1$ to site 0
Site 0 sets $RuleSet = RuleSet_0 \cup RuleSet_1$

Phase 4: Decryption
for $i = 0$ to $N-1$ **do**
  Site i decrypts items in $RuleSet$ using D_i
  Site i sends $RuleSet$ to site $i+1 \bmod N$
end for
Site $N-1$ decrypts items in $RuleSet$ using D_{N-1}
$RuleSet_{(k)} = RuleSet - F$
Site $N-1$ broadcasts $RuleSet_{(k)}$ to sites $0..N-2$

A protocol that is proved not to reveal anything other than the required result and information deemed not privacy threatening could be sufficient for many practical purposes. We use this approach to prove that Protocol 4.1 reveals only the union of locally large itemsets and a clearly bounded set of innocuous information.

Theorem 4.3.1 *Protocol 4.1 privately computes the union of the locally large itemsets assuming no collusion, revealing at most the result* $\cup_{i=1}^{N} LL_{i(k)}$ *and:*

1. *Size of intersection of locally supported itemsets between any subset of odd numbered sites,*

2. *Size of intersection of locally supported itemsets between any subset of even numbered sites, and*

3. *Number of itemsets supported by at least one odd and one even site.*

Proof *Phase 0:* Since no communication occurs in Phase 0, each site can simulate its view by running the algorithm on its own input.

Phase 1: At the first step, each site sees $LLe_{i-1(k)}$. The size of this set is $\lfloor 1/minimum\ support \rfloor$. Assuming the security of encryption, each item in this set is computationally indistinguishable from a number chosen from a uniform distribution. A site can therefore simulate the set using a uniform random number generator. This same argument holds for each subsequent round.

Phase 2: In Phase 2, site 0 gets the fully encrypted sets of itemsets from the other even sites. Assuming that each site knows the source of a received message, site 0 will know which fully encrypted set $LLe_{(k)}$ contains encrypted itemsets from which (odd) site. Equal itemsets will now be equal in encrypted form. Thus, site 0 learns if any odd sites had locally supported itemsets in common. We can still build a simulator for this view, using the information in point 1 above. If there are k itemsets known to be common among all $\lfloor N/2 \rfloor$ odd sites (from point 1), generate k random numbers and put them into the simulated $LLe_{i(k)}$. Repeat for each $\lfloor N/2 \rfloor - 1$ subset, etc., down to 2 subsets of the odd sites. Then fill each $LLe_{i(k)}$ with randomly

chosen values until it reaches size $\lfloor 1/minimum\ support \rfloor$. The generated sets will have exactly the same combinations of common items as the real sets, and since the *values* of the items in the real sets are computationally indistinguishable from a uniform distribution, they simulation matches the real values.

The same argument holds for site 1, using information from point 2 to generate the simulator.

Phase 3: Site 1 eliminates duplicates from the $LLe_{i(k)}$ to generate $RuleSet_1$. We now demonstrate that Site 0 can simulate $RuleSet_1$. First, the size of $RuleSet_1$ can be simulated knowing point 2. There may be itemsets in common between $RuleSet_0$ and $RuleSet_1$. These can be simulated using point 3: If there are k items in common between even and odd sites, site 0 selects k random items from $RuleSet_0$ and inserts them into $RuleSet_1$. $RuleSet_1$ is then filled with randomly generated values. Since the encryption guarantees that the values are computationally indistinguishable from a uniform distribution, and the set sizes $|RuleSet_0|$, $|RuleSet_1|$, and $|RuleSet_0 \cap RuleSet_1|$ (and thus $|RuleSet|$) are identical in the simulation and real execution, this phase is secure.

Phase 4: Each site sees only the encrypted items after decryption by the preceding site. Some of these may be identical to items seen in Phase 2, but since all items must be in the union, this reveals nothing. The simulator for site i is built as follows: take the values generated in Phase 2 step $N-1-i$, and place them in the $RuleSet$. Then insert random values in $RuleSet$ up to the proper size (calculated as in the simulator for Phase 3). The values we have not seen before are computationally indistinguishable from data from a uniform distribution, and the simulator includes the values we have seen (and knew would be there), so the simulated view is computationally indistinguishable from the real values.

The simulator for site $N-1$ is different, since it learns $RuleSet_{(k)}$. To simulate what it sees in Phase 4, site $N-1$ takes each item in $RuleSet_{(k)}$, the final result, and encrypts it with E_{N-1}. These are placed in $RuleSet$. $RuleSet$ is then filled with items chosen from F, also encrypted with E_{N-1}. Since the choice of items from

F is random in both the real and simulated execution, and the real items exactly match in the real and simulation, the *RuleSet* site $N-1$ receives in Phase 4 is computationally indistinguishable from the real execution.

Therefore, we can conclude that above protocol is privacy preserving in the semi-honest model with the stated assumptions. ■

The information disclosed by points 1-3 could be relaxed to the number of itemsets support by 1 site, 2 sites, ..., N sites if we assume anonymous message transmission (e.g., Crowds [65]), however this raises other security issues and increases the communication cost for little increased privacy.

Testing support threshold without revealing support count

Protocol 4.1 gives the full set of locally large itemsets $LL_{(k)}$. We still need to determine which of these itemsets are supported globally. Step 4 of the FDM algorithm forces each site to reveal its own support count for every itemset in $LL_{(k)}$. All we need to know is for each itemset $X \in LL_{(k)}$, is $X.sup \geq s\% \times |DB|$? The following allows us to reduce this to a comparison against a sum of local values (the *excess support* at each site):

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

Therefore, checking for support is equivalent to checking if $\sum_{i=1}^{n}(X.sup_i - s * |DB_i|) \geq 0$. The challenge is to do this without revealing $X.sup_i$ or $|DB_i|$. An algorithm for this is given in Protocol 4.2. The first site generates a random number x_r for each itemset X, adds that number to its $(X.sup_i - s*|DB_i|)$, and sends it to the next site. (All arithmetic is $\pmod m$ where $m \geq 2 * |DB|$, for reasons

Protocol 4.2 Finding the global support counts securely

Require: $N \geq 3$ sites numbered $0..N-1$, $m \geq 2 * |DB|$

$rule\_set = \emptyset$

at site 0:

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

choose random integer x_r from a uniform distribution over $0..m-1$;

$t = r.sup_i - s * |DB_i| + x_r \pmod{m}$;

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

end for

send $rule\_set$ to site 1 ;

for $i = 1$ to $N - 2$ **do**

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

$\bar{t} = r.sup_i - s * |DB_i| + t \pmod{m}$;

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

end for

send $rule\_set$ to site $i + 1$;

end for

at site N-1:

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

$\bar{t} = r.sup_i - s * |DB_i| + t \pmod{m}$;

securely compute if $(\bar{t} - x_r) \pmod{m} < m/2$ with the site 0; { Site 0 knows x_r }

if $(\bar{t} - x_r) \pmod{m} < m/2$ **then**

multi-cast r as a globally large itemset.

end if

end for

that will become apparent later.) The random number masks the actual excess support, so the second site learns nothing about the first site's actual database size or support. The second site adds its excess support and sends the value on. The random value now hides both support counts. The last site in the change now has $\sum_{i=1}^{n}(X.sup_i - s * |DB_i|) + x_r \pmod m$. Since the total database size $|DB| \leq m/2$, negative summation will be mapped to some number that is bigger then or equal to $m/2$. ($-k = m-k \bmod m$.) The last site needs to test if this sum minus $x_r \pmod m$ is less then $m/2$. This can be done securely using Yao's generic method [77]. Clearly this algorithm is secure as long as there is no collusion, as no site can distinguish what it receives from a random number. Alternatively, the first site can simply send x_r to the last site. The last site learns the actual excess support, but does not learn the support values for any single site. In addition, if we consider the excess support to be a valid part of the global result, this method is still secure.

Theorem 4.3.2 *Protocol 4.2 privately computes globally supported itemsets in the semi-honest model.*

Proof To show that Protocol 4.2 is secure under the semi-honest model, we have to show that a polynomial time simulator can simulate the view of the parties during the execution of the protocol, based on their local inputs and the global result. We also need to use the general composition theorem for semi-honest computation. [39] The theorem says that if g securely reduces to f and f is computed f securely then the computation of $f(g)$ is secure. In our context, f is the secure comparison of two integers, and g is Protocol 4.2. First, we show that the view of any site during the addition phase can efficiently simulated given the input of that site and the global output. Site i uniformly chooses a random integer s_r, $0 \leq s_r < m$. Next, we show that view and the output of the simulator are computationally indistinguishable by showing that the probability of seeing a given x in both is equal. In the following equations, x_r is the random number added at the beginning of Protocol 4.2, $0 \leq X_r < m$. The arithmetic is assumed to be $\pmod m$. Also note that $X.sup_i$ is fixed for each site.

$$\begin{aligned} Pr\left[VIEW_i^{Protocol4.2} = x\right] &= Pr\left[x_r = x - \sum_{k=1}^{k=i-1} X.sup_i\right] \\ &= \frac{1}{m} \\ &= Pr\left[s_r = x\right] \\ &= Pr\left[Simulator_i = x\right] \end{aligned}$$

Therefore, what each site sees during the addition phase is indistinguishable from that simulated with a random number generator. During the comparison phase we can use the generic secure method, so from the composition theorem we conclude that Protocol 4.2 is secure in the semi-honest model. ■

4.3.3 Securely Finding Confidence of a Rule

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

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

Since each site knows $XY.sup_i$ and $X.sup_i$, we can easily use Protocol 4.2 to securely calculate the confidence of a rule.

4.4 Security Against Collusion

Collusion in Protocol 4.1 could allow a site to know its own frequent itemsets after encryption by all parties. Using this it can learn the size of the intersection between its own itemsets and those of another party. Specifically, if site i colludes with site

$i-1$, it can learn the size of its intersection with site $i+1$. Collusion between sites 0 and 1 exacerbates the problem, as they know encrypted values of itemsets for all odd (even) sites. This may reveal the actual itemsets; if $|LL_{i(k)} \cap LL_{i+1(k)}| = |LL_{i(k)}|$, then site i has learned a subset of the itemsets at site $i+1$. Noise addition (fake itemsets) limits the damage.

Collusion can be a problem for our second protocol, because site $i+1$ and site $i-1$ can collude to reveal site i's excess support value. This protocol can be made resilient against collusions using a straightforward technique from the cryptographic community. The basic idea is each party divides its input into n parts, and send the $n-1$ pieces to different sites. To reveal any parties input, $n-1$ party must collude. The following is a brief summary of the protocol, details can be found in [7]. (A slightly more efficient version can be found in [19].)

1. Each site i randomly chooses n elements such that $x_i = \sum_{j=1}^{n} z_{i,j} \bmod m$ where x_i is the input of site i. Site i sends $z_{i,j}$ to site j.

2. Every site i computes $w_i = \sum_{j=1}^{n} z_{j,i} \bmod m$ and sends w_i to site n.

3. Site n computes the final result $\sum_{i=1}^{n} w_i \bmod m$

The above protocol can be easily used to improve our second protocol. Assume site 0 is the starting site in our protocol and site $N-1$ is the last site. Choose m such that $2 * |DB| \leq m$. Set $x_1 = X.sup_1 - s * d_1 + x_r \bmod m$ and $x_i = X.sup_i - s * d_i \bmod m, i \neq 1$. After this point, the above protocol can be used to find $\sum_{i=1}^{n}(X.sup_i - s * d_i) + x_r \bmod m$. At the end, one secure addition and comparison is done as in Protocol 4.2 to check if itemset X is globally supported.

4.5 Two Party Case

The two party case is problematic. First, globally supported itemsets that are not supported at one site are known to be supported at the other site – this is an artifact of the result. Protocol 4.1 is worse yet, as itemsets that are supported at one site but

not supported globally will become known to the other site. To retain any privacy, we must dispense with local pruning entirely (steps 1 and 2 of the FDM algorithm) and compute support for all candidates in $CG_{(k)}$ (as computed from $L_{(k-1)}$). Second, the secure comparison phase at the end of the protocol 4.2 cannot be removed, as otherwise the support of one site is disclosed to the other. It is difficult to improve on this, as evidenced by the following theorem.

Theorem 4.5.1 *For itemset X, to check whether* $\frac{X.sup_1+X.sup_2}{d_1+d_2} \geq k$ *can be securely computed if and only if Yao's millionaire problem can be securely solved for arbitrary* a *and* b.

Proof Checking $\frac{X.sup_1+X.sup_2}{d_1+d_2} \geq k$ is equivalent to checking $(X.sup_1 - k * d_1) \geq (k * d_2 - X.sup_2)$. If we have $a = X.sup_1 - k * d_1$ and $b = k * d_2 - X.sup_2$, we have an instance of Yao's millionaire problem for a and b. Assume we have a secure protocol that computes whether X is supported globally or not for arbitrary $X.sup_1$, $X.sup_2$, d_1, d_2 and k. Take $X.sup_1 = 3a, d_1 = 4a, X.sup_2 = b, d_2 = 4 * b$ and $k = 0.5$. This is equivalent to checking whether $a \geq b$. ■

The above theorem implies that if we develop a method that can check securely if an itemset is globally supported for the two party case in semi-honest model, it is equivalent to finding a new solution to Yao's millionaire problem. This problem is well studied in cryptography and to our knowledge, there is no significantly faster way for arbitrary a and b than using the generic circuit evaluation solution.

It is worth noting that eliminating local pruning and using Protocol 4.2 to compute the global support of all candidates in $CG_{(k)}$ is secure under the definitions of secure multi-party computation, for two or more parties. The problem with the two-party case is that knowing a rule is supported globally that is not supported at one's own site reveals that the other site supports that rule. This is true no matter how secure computation, it is an artifact of the result. Thus, extending to secure computation in the two party case is unlikely to be of use.

4.6 Communication and Computation Costs

We now give cost estimates for association rule mining using the method we have presented. The number of sites is N. Let the total number of locally large candidate itemsets be $|CG_{i(k)}|$, and the number of candidates that can be directly generated by the globally large (k-1) itemsets be $|CG_{(k)}|$ (= $apriori\_gen(L_{(k-1)})$). The excess support $X.sup_i - |DB_i|$ of an itemset X can be represented in $m = \lceil \log_2(2 * |DB|) \rceil$ bits. Let t be the number of bits in the output of the encryption of an itemset. A lower bound on t is $log_2(|CG_{(k)}|)$; based on current encryption standards $t = 512$ is an appropriate value.[1]

The total bit-communication cost for Protocol 4.1 is $O(t * |CG_{(k)}| * N^2)$, however, as much of this happens in parallel we can divide by N to get an estimate of the communication time. For comparison, the FDM algorithm requires $O(t * |\cup_i LL_{i(k)}| * N)$ for the corresponding steps, with effectively the same reduction in time due to parallelism (achieved through broadcast as opposed to simultaneous point-to-point transmissions). The added cost of Protocol 4.1 is due to padding $LLe_{i(k)}$ to hide the actual number of local itemsets supported, and the increase in bits required to represent encrypted itemsets. The worst-case value for $|CG_{(k)}|$ is $\binom{item\ domain\ size}{k}$, however, the optimizations that make the a-priori algorithm effective in practice would fail for such large $|CG_{(k)}|$. In practice, only in the first round ($k = 1$) will this padding pose a high cost; $|CG_{(1)}|$ = the size of the domain of items. In later iterations, the size of $|CG_{(k)}|$ will be much closer to $|LLe_{i(k)}|$. The computation cost increase due to encryption is $O(t^3 * |CG_{(k)}| * N^2)$, where t is the number of bits in the encryption key. Here t^3 represents the bit-wise cost of modular exponentiation.

Protocol 4.2 requires $O(m * | \cup_i LL_{i(k)}| * (N + t))$ bits of communication. The t factor is for the secure circuit evaluations between sites $N - 1$ and 0 required to determine if each itemset is supported. FDM actually requires an additional factor of N due to the broadcast of local support instead of point-to-point communication.

[1]The worst-case bound on $|CG_{(k)}|$ is $\binom{item\ domain\ size}{k}$. $t = 512$ can represent such worst-case itemsets for 50 million possible items and $k = 20$, adequate for most practical cases.

However, the broadcast results in a single round instead of N rounds of our method. The final secure comparison requires a computation cost of $O(|\cup_i LL_{i(k)}| * m * t^3)$.

As discussed in Chapter 4.5, using only Protocol 4.2 directly on $CG_{(k)}$ is fully secure assuming the desired result includes all globally large itemsets. The communication costs becomes $O(m*|CG_{(k)}|*N)$, but because the communication in Protocol 4.2 is sequential the communication time is roughly the same as the full protocol. The encryption portion of the computation cost becomes $O(|CG_{(k)}| * m * t^3)$ for the secure comparison at the end of the protocol. However, there is a substantial added cost in computing the support, as we must compute support for all $|CG_{(k)}|$ itemsets. This is generally much greater than the $|CG_{i(k)} \cup (\cup_i LL_{i(k)})|$ required under the full algorithm (or FDM), as shown in [18]. It is reasonable to expect that this cost will dominate the other costs, as it is linear in $|DB|$.

4.6.1 Optimizations and Further Discussion

The cost of "padding" $LLe_{i(k)}$ from F to avoid disclosing the number of local itemsets supported can add significantly to the communication and encryption costs. In practice, for $k > 1$, $|CG_{(k)}|$ is likely to be of reasonable size. However, $|CG_{(1)}|$ could be very large, as it is dependent only on the size of the domain of items, and is not limited by already discovered frequent itemsets. If the participants can agree on an upper bound on the number of frequent items supported at any one site that is tighter than "every item may be frequent" without inspecting the data, we can achieve a corresponding decrease in the costs with no loss of security. This is likely to be feasible in practice; the very success of the a-priori algorithm is based on the assumption that relatively few items are frequent. Alternatively, if we are willing to leak an upper bound on the number of itemsets supported at each site, each site can set its own upper bound and pad only to that bound. This can be done for every round, not just $k = 1$. As a practical matter, such an approach would achieve acceptable security and would change the $|CG_{(k)}|$ factor in the communication and

encryption costs of Protocol 4.1 to $O(|\cup_i LL_{i(k)}|)$, equivalent to FDM.

Another way to limit the encryption cost of padding is to pad randomly from the domain of the encryption output rather than encrypting items from F. Assuming |domain ofE_i| >> |domain of itemsets|, the probability of padding with a value that decrypts to a real itemset is small, and even if this occurs it will only result in additional itemset being tested for support in Protocol 4.2. When the support count is tested, such "false hits" will be filtered out, and the final result will be correct.

The comparison phase at the end of protocol 4.2 can be also removed, eliminating the $O(m*|\cup_i LL_{i(k)}|*t)$ bits of communication and $O(|\cup_i LL_{i(k)}|*m*t^3)$ encryption cost. This reveals the excess support for each itemset. Practical applications may demand this count as part of the result for globally supported itemsets, so the only information leaked is the support counts for itemsets in $\cup_i LL_{i(k)} - L_{(k)}$. As these cannot be traced to an individual site, this will generally be acceptable in practice.

The cost estimates are based on the assumption that all frequent itemsets (even 1-itemsets) are part of the result. If exposing the globally frequent 1-itemsets is a problem, the algorithm could easily begin with 2-itemsets (or larger). While the worst-case cost would be unchanged, there would be an impact in practical terms. Eliminating the pruning of globally infrequent 1-itemsets would increase the size of $CG_{i(2)}$ and thus $LL_{i(2)}$, however, local pruning of infrequent 1-itemsets should make the sizes manageable. More critical is the impact on $|CG_{(2)}|$, and thus the cost of padding to hide the number of locally large itemsets. In practice, the size of $CG_{(2)}$ will rarely be the theoretical limit of $\binom{item\ domain\ size}{2}$, but this worst-case bound would need to be used if the algorithm began with finding 2-itemsets (the problem is worse for $k > 2$). A practical solution would again be to have sites agree on a reasonable upper bound for the number of locally supported k-itemsets for the initial k, revealing some information to substantially decrease the amount of padding needed.

4.6.2 Practical Cost of Encryption

While achieving privacy comes at a reasonable increase in communication cost, what about the cost of encryption? As a test of this, we implemented Pohlig-Hellman, described in Chapter 3.2.3. The encryption time per itemset represented with $t = 512$ bits was 0.00428 seconds on a 700MHz Pentium 3 under Linux. Using this, and the results for distributed association rule mining reported in [18], we can estimate the cost of privacy-preserving association rule mining on the tests in [18].

The first set of experiments described in [18] contain sufficient detail for us to estimate the cost of encryption. These experiments used three sites, an item domain size of 1000, and a total database size of 500k transactions.

The encryption cost for the initial round ($k = 1$) would be 4.28 seconds at each site, as the padding need only be to the domain size of 1000. While finding two-itemsets could potentially be much worse ($\binom{1000}{2} = 499500$), in practice $|CG_{(2)}|$ is much smaller. The experiment in [18] reports a total number of candidate sets ($\sum_{k>1} |CG_{(k)}|$) of just over 100,000 at 1% support. This gives a total encryption cost of around 430 seconds per site, with all sites encrypting simultaneously. This assumes none of the optimizations of Chapter 4.6.1; if the encryption cost at each site could be cut to $|LL_{i(k)}|$ by eliminating the cost of encrypting the padding items, the encryption cost would be cut to 5% to 35% of the above on the datasets used in [18].

There is also the encryption cost of the secure comparison at the end of Protocol 4.2. Although the data reported in [18] does not give us the exact size of $\cup_i LL_{i(k)}$, it appears to be on the order of 2000. Based on this, the cost of the secure comparison, $O(|\cup_i LL_{i(k)}| * m * t^3)$, would be about 170 seconds.

The total execution time for the experiment reported in [18] was approximately 800 seconds. Similar numbers hold at different support levels; the added cost of encryption would at worst increase the total run time by roughly 75%.

4.7 Conclusions

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

5 PRIVACY-PRESERVING DISTRIBUTED K-NEAREST NEIGHBOR CLASSIFICATION

This chapter presents a method for privately computing k-nn classification from distributed sources without revealing any information about the sources or their data, other than that revealed by the final classification result.

Consider the case of a physician who wants to learn the most likely diagnosis for a patient by looking at diagnoses of similar symptoms at other hospitals. Specifically, the physician wants to use a *k-nearest neighbor* (k-nn) classifier to predict the disease of the patient. Revealing the patient's particular test results may not be a threat to privacy (if only the physician knows the identity of the patient) but privacy of the different hospitals may be at risk. If this procedure is implemented naïvely, the researcher may learn that two patients with the same medical test results are diagnosed with different diseases in different hospitals. This could damage the reputations of the hospitals. The possibility of such incidents may prevent hospitals from participating in such a diagnostic tool. The obvious question is, can this be done without revealing anything other than the final classification? The answer is yes: Here we present an efficient method with provable privacy properties for k-nn classification.

We assume that each database is able to construct its own k-nearest neighbors independently (This is possible because of the horizontally partitioned data assumption). The *distributed* problems are determining which of the local results are the closest globally, and finding the majority class of the global k-nearest neighbors. We assume that attributes of the instance that needs to be classified are not private (i.e., we do not try to protect the privacy of the query issuer); we want to protect the privacy of the data sources. The approach makes use of an untrusted, non-colluding party (Chapter 3.3): a party that is not allowed to learn anything about any of the

data, but is trusted not to collude with other parties to reveal information about the data.

The basic idea is that each site finds its own k-nearest neighbors, and encrypts the class with the public key of the site that sent the instance for classification (querying site). The parties compare their k-nearest neighbors with those of all other sites – except that the comparison gives each site a random share of the result, so no party learns the result of the comparison. The results from all sites are combined, scrambled, and given to the untrusted, non-colluding site. This site combines the random shares to get a comparison result for each pair, enabling it to sort and select the global k-nearest neighbors (but without learning the source or values of the items). The querying site and the untrusted, non-colluding site then engage in a protocol to find the class value. Each site learns nothing about other sites. (the comparison results appears to be randomly chosen bits.) The untrusted site sees $k*n$ encrypted results. It is able to totally order the results, but since it knows nothing about what each means or where it comes from, it learns nothing. The querying site only sees the final result.

Details of the algorithm are given in Chapter 5.2, along with a discussion of the privacy of the method. Computation and communication costs are given in Chapter 5.3. First, we discuss related work and relevant background.

5.1 Related Work

Finding the k-nearest neighbors of a multidimensional data point q [50] and building k-nn classifiers [38] have been well studied.

There has also been work in evaluating distributed top-k queries over remote sources [12] and continuous top-k queries over distributed data streams [6]. As far as we know, none of the previous work dealt with security issues.

Another closely related problem is private information retrieval [20]. The key difference between this work and private information retrieval is that we do not

view the query as private information, only the data. This enables solutions where the communication cost is independent of the size of the database; a difficulty with private information retrieval is that any solution that does not transfer the entire database cannot be secure in an information theoretic sense. This can be improved somewhat under computationally secure private information retrieval (*cPIR*); the best known *cPIR* method for database of n bits needs at least $O(n^c)$ bit communication for any given $c > 0$ [51]. For many applications including homeland security, we believe maintaining secrecy of the query point is not critical, it is the data that must be protected.

5.2 Secure k-nn Classification

We first formally define the problem. Let R be the domain of the attributes and C be the domain of the class values. Let D_i denote the database of instances at site S_i. Let (x, d, k) be the query originated by site O, where $x \in R$ is the instance to be classified, and $d : R \times R \rightarrow [0, 1]$ is a distance function used to determine which k items are closest to x (e.g., Euclidean distance, although any metric could be used provided each site can compute $d(x, x_j)$ for every x_j in its database.) Given the data instance x, our goal is to find the k nearest neighbors of x in the union of the databases and return the class of the majority of those neighbors as the predicted class of x:

$$C_x = \text{Maj}\left(\prod_c \left(\underset{(x_i,c_i)\in D_1\cup D_2\cdots\cup D_n}{\text{argmin}_k} (d(x_i, x))\right)\right)$$

where $\prod$ is the projection function and Maj is the majority function.

The security/privacy goal is to find C_x while meeting the following criteria:

- No site except O will be able to predict C_x better than looking at (x, d, k) and its own database D_i (E.g., if Site S_i has k points x_i such that $d(x, x_i) = 0$, it is likely that the majority class of the x_i will be the result); and

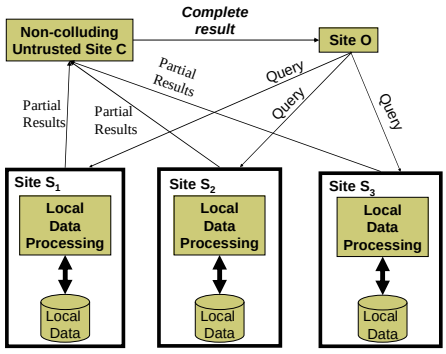

Figure 5.1. Information flow in secure k-nn classification

- No site learns anything about the source of x_i except its own.

Theorem 5.2.1 shows this by proving what *is* disclosed. Assuming the number of sites n and the query (x, d, k) are public, site O learns only the result C_x. The other sites learn only what they can infer from their own data and the query.

We emphasize that the untrusted site learns nothing except the public values k and n. A diagram showing the information flow is given in Figure 5.1.

5.2.1 The Algorithm

Given the query, each site can find its own closest k items without exchanging information. These $n * k$ candidate items must contain the k nearest neighbors of x; what remains is to find the closest k among the candidates and return the majority class of those k instances to site O. This poses two security challenges:

1. Determining which of the $n * k$ items are the k nearest neighbors without revealing anything about the items, where they come from, or their distance to the instance that needs to be classified; and

2. Learning the class value while disclosing it to only the originating site.

At first glance, this appears simple – have each site send their local k-nn distances and classes to a third party C. C learns the result, but also learns distances and the sites that contributes to the global k-nn result. A slightly more sophisticated approach is for C to publish a candidate distance value, and have local sites return the number of items within that distance, refining the value until k items are within the distance. Then the class value can be computed. This still reveals the sites that contribute to the global result, the distances to the queried instance, the final classification result to C, and more. To ensure privacy concerns are met, we provide a solution that (under the non-collusion assumption) reveals **nothing** to any site that cannot be inferred by looking at its own data and the instance, except that O learns the final result.

The use of an untrusted third party, along with public-key encryption, makes it easy to solve challenge 2. Each site encrypts the class value with the public key of O before passing it to the non-colluding site C. The data source sites are then left out of the loop – since they never see the data again, they can learn nothing after passing their data to C (e.g., they cannot test to see if their own encrypted values match those selected for the result.) C and O will participate in a special protocol to reveal only the majority class (explained later.)

Meeting challenge 1 is more difficult. Sending the distance $d(x_i, x) = d_i$ to C, even with encrypted results, reveals information about the location of the points. Instead site C is sent an $n * k - 1$ length vector for each point x_i, containing the results of comparing x_i with all other points. This enables C to order the points and select the k nearest neighbors. Since the existence of a distance metric implies a total ordering exists, and the number of points is fixed, C learns nothing.

Two problems remain. The first is that we must prevent C from learning which point comes from which site, or it can learn the source of the k nearest neighbors (among other things.) This is easily addressed – all points and their associated comparison vectors are sent to one of the data sites S_s, which combines them and scrambles the order before passing them on to C. Public key encryption, using C's

public key, prevents S_s from learning anything. The second issue is more challenging: How do we build the comparison vectors at site S_i without S_i learning about values at other sites? If two sites just compare items, they both learn something about the data at the other site (e.g., if S_i's items are all closer to x than S_j's items, both learn that S_j does not have an item that contributes to the result.) For this we use the share-splitting idea from secure multi-party computation. Instead of containing the results of comparing x_i with other items, the vector contains a *random share* of the comparison result. E.g., if d_i for x_i is smaller than d_j for x_j, then the comparison $d_i < d_j$ should return 0. Either the element of the d_i vector corresponding to d_j and the element of the d_j vector corresponding to d_i both contain 0, or they both contain $1 - 0 \oplus 0 = 1 \oplus 1 = 0$. However, knowing only one share tells nothing: a share 0 could mean either $0 \oplus 0 = 0$ or $0 \oplus 1 = 1$. From d_i's view, the share has equal probability of being 0 or 1 (a random choice), so it learns nothing.

To generate random shares of the comparison, we return to secure multi-party computation. We stop the generic circuit comparison method before combining shares to learn the final result. In other words, given two integers a and b, secure comparison of a, b ($f : \{0,1\}^* \times \{0,1\}^* \longmapsto \{0,1\} \times \{0,1\}$) is defined as follows:

$$f(a,b) = \begin{cases} (1 \oplus r, r) & \text{if } a > b \\ (0 \oplus r, r) & \text{if } a < b \end{cases}$$

where each site sees only one component of the function output. This states that if $a > b$ the xor of the shares of the participating sites will be 1, otherwise the xor of the shares will be 0. Using this function, each site can compare its elements with all other elements and learn nothing about the result of the comparisons.

Two additional details. First, the identifiers used to track the d_j in the comparison share vector for d_i must not disclose anything. One option would be for the combining site S_s to assign identifiers, but this would require independent encryption of the single bits of the comparison shares, and single bit encryption is problematic. Instead, each site generates pseudo-random unique identifiers site C cannot distinguish from random. One simple and secure way to handle this problem is using a

pseudo-random permutation function. With a fixed key, DES is assumed to be such a function. The sites agree on a key K, and each site S_i generates k identifiers by evaluating $E_K(ik), E_K(ik+1), \ldots, E_K(ik+k-1)$. Since encryption is assumed to be secure and a permutation, each site will get non-intersecting identifiers that appear random to any (polynomial time) adversary not knowing K (in particular, C). The identifiers are also used to determine which of a comparison pair will be the left-hand side in a comparison: The item with the smaller identifier corresponds to x in the comparison function $f(x, y)$.

The second detail is equality. Identifying that two items are at equal distances reveals information. We must disambiguate consistently, without giving even a probabilistic estimate on the likelihood of equality. The solution is to add extra low-order bits to each distance, based on a unique mapping from the identifier that appears random to C - the same trick used to generate identifiers. The distance used for comparison is actually $d||E_u(ik + j)$, where the encryption function E is as above, but with a different key. This ensures that distances are unique, guaranteeing a total ordering of distances.

Protocol 5.1 gives the algorithm details. Note that at the end of the k-nearest neighbor selection phase, C has the class of the k-nearest neighbors encrypted with E_o. Assuming the usage of Blum-Goldwasser encryption, described in Chapter 3.4, each class value $class_i$ will have ciphertext of the form $(\acute{r}, class_i \oplus r)$, where O has enough information to determine r given $\acute{r}$, enabling decryption to get $class_i$. Instead of sending these values to O, C will xor each of these values with a random value r_i. C then sends $(\acute{r}, class_i \oplus r \oplus r_i)$ to O. O decrypts to get $class'_i = class_i \oplus r_i$, indistinguishable (to O) from a random value. O and C now use the generic secure circuit evaluation approach to evaluate the majority function:

$$\text{Maj}(class'_1 \oplus r_1, \ldots, class'_k \oplus r_k).$$

This is a simple circuit with size complexity dependent on k and the number of distinct classes. The cost is dominated by the k-nearest neighbor selection phase.

To clarify, we give an example for $k = 1$ and $n = 3$.

Protocol 5.1 Privacy-preserving k-nn classification algorithm

Require: n sites S_i, $1 \leq i \leq n$, each with a database D_i; permuting site S_s (where s may be in $1, \ldots, n$); and untrusted non-colluding site C (distinct from S_i or S_s). Query (x, d, k) generated by originating site O. Public encryption keys E_c for site C and E_o for site O, key generation function E_K and E_u known only to the S_i.

for all sites S_i, in parallel **do**
  {Build vector of random key, distance, and result for local closest k}
  Select k items $(d(x_i, x), \prod_{c_i}(x_i, c_i))$ with smallest $d(x_i, x)$ from D_i into N_i
  $R_i = \emptyset$
  for $j = 0..k-1$ {Compute identifiers and "extended" local distances} **do**
    $R_i = R_i \cup \{(E_K(ik+j), N_i[j].d || E_u(ik+j),\ E_o(N_i[j].result))\}$ {|| is string concatenation}
  end for
  $ER_i = \emptyset$
  for each $(id, d, E_o(c)) \in R_i$ {Comparison phase} **do**
    $v = \emptyset$
    for each site $h = i \ldots n$ {If $i = h$, just generate values locally.} **do**
      for $j = 0 \ldots k-1$ **do**
        if $id < R_h[j].id$ **then**
          $v = v \cup$ $\{(R_h[j].id, S_i$'s share of $f(d, R_h[j].d)\}$
          $v_{hj} = v_{hj} \cup \{(id, S_j$'s share of $f(d, R_h[j].d)\}$
        else if $id > R_h[j].id$ **then**
          $v = v \cup$ $\{(R_h[j].id, S_i$'s share of $f(R_h[j].d, d)\}$
          $v_{hj} = v_{hj} \cup \{(id, S_j$'s share of $f(R_h[j].d, d)\}$
        end if
      end for
    end for
    $ER_i = ER_i \cup (id, E_c(v), E_o(c))$
  end for
  send ER_i to S_s
end for
{At site S_s: Permutation phase}
set $ER = \cup_{i=1}^{n}(ER_i)$, permute it and send it to C
{At site C: k nearest neighbor selection phase}
Decrypt the encrypted shares of the comparison results
Use the pairwise comparisons to find the global k nearest neighbors
Let R be the set of encrypted class values$(E_o(c))$ of the global k nearest neighbor
for all $R.E_o(c_i)$ {encrypted as $(\acute{r}, c_i \oplus r)$} **do**
  $NR[i] = R.E_o(c_i) \oplus$ random r_i
end for
Site C sends NR to O
{At site O:}
for all $NR[i]$ $\{= (\acute{r}, c_i \oplus r \oplus r_i)\}$ **do**
  Find r using $\acute{r}$ and the private key, set $NR_d[i] = c_i \oplus r \oplus r_i \oplus r$
end for
Find Maj from the random shares of C and NR_d using secure circuit evaluation.

Example 5.2.1 Given $(x, d, 1)$, each site finds its 1-nearest neighbor. Assume that site S_1 has $(d(x, x_1), \prod_c(x_1, c_1) = (0.1, c_1)$, site S_2 has (x_2, c_2) at distance 0.2, and site S_3 has (x_3, c_3) at 0.15. After generating random identifiers, S_1 has $(3, 0.1, c_1)$, S_2 has $(1, 0.2, c_2)$, and S_3 has $(2, 0.15, c_3)$. (For simplicity we omit the low-order bit disambiguation.) In generating the comparison vector for c_1 and c_2, S_1 notes that $c_1.id = 3 > 1$, so it has the right argument of $f(a, b)$ and generates a random bit (say 0) as its share of the output. Since $0.2 \geq 0.1$, S_2 learns that its share should be $1 \oplus 0 = 1$. (Neither S_2 or S_1 learns the other's share, or the comparison result.) Secure comparisons with the other sites are performed, giving each site tuples containing its share of the comparison with all other items. These are encrypted with C's public key to give:

$$\begin{aligned} S_1 &: (3, E_c((1,1),(2,0)), E_o(c_1)) \\ S_2 &: (1, E_c((2,1),(3,0)), E_o(c_2)) \\ S_3 &: (2, E_c((1,1),(3,1)), E_o(c_3)) \end{aligned}$$

The above are sent to S_3, which permutes the set, strips source information, and sends it to C. Site C decrypts the comparison share vectors to get:

$$\begin{aligned} &(2, ((1,1),(3,1)), E_o(c_3)) \\ &(3, ((1,1),(2,0)), E_o(c_1)) \\ &(1, ((2,1),(3,0)), E_o(c_2)) \end{aligned}$$

C now compares the items to find the nearest neighbor. As an example, to compare items 2 and 3, we take the pair $(3, 1)$ from the first (2) row and the pair $(2, 0)$ from the second (3) row. Combining the share portions of these pairs gives $1 \oplus 0 = 1$, so $d(x, x_2) \geq d(x, x_3)$. Likewise, comparing 1 and 3 gives $0 \oplus 1 = 1$, so $d(x, x_1) \geq d(x, x_3)$. Therefore, x_1 is closest to x. C sends $E_o(c_1)$ to O, which decrypts to get c_1, the correct result. (With $k > 1$, C and O would engage in a protocol to determine which c_i was in the majority, and send $E_o(c_i)$ to O.)

5.2.2 Security of the Protocol

We now prove that Protocol 5.1 is secure. We assume that sites O and C are not among the S_i. We have discussed the need for C being a separate site. O cannot be a data source, as it would be able to recognize its own $E_o(x)$ among the results, thus knowing if it was the source of some of the k nearest neighbors.

To define security we use definitions from the Secure Multi-party Computation community (discussed in Chapter 3.2).

We need one additional tool to prove the security of the protocol. The encrypted items seen by S_s and C during execution of the protocol may disclose some information. The problem is that two items corresponding to the same plaintext map to the same ciphertext. If multiple items are of the same class (as would be expected in k-nn classification), the permuting site S_s would learn the class entropy in the k-nn of each site as well as the number of identical results between sites. The comparison site C would learn this for the data as a whole. Neither learns the result, but something of the distribution is revealed.

Fortunately, the cryptography community has a solution: *probabilistic* public-key encryption (discussed in Chapter 3.4). The idea is that the same plaintext may map to different ciphertexts, but these will all map back to the same plaintext when decrypted. Using probabilistic public-key encryption for E_o allows us to show Protocol 5.1 is secure. The Blum-Goldwasser probabilistic encryption scheme [10], with a cipher text of the form $(\acute{r}, M \oplus r)$ for message M, is one example. In this, given $\acute{r}$ and the private key, it is possible to compute r to recover the original message.

Theorem 5.2.1 *Protocol 5.1 privately computes the k-nn classification in the semi-honest model where there is no collusion; only site O learns the result.*

Proof To show that Protocol 5.1 is secure under the semi-honest model, we must demonstrate that what each site sees during the execution of the protocol can be simulated in polynomial time using only its own input and output. Specifically,

the output of the simulation and the view seen during the execution must be computationally indistinguishable. We also use the general composition theorem for semi-honest computation: if g securely reduces to f and there is a way to compute f securely, then there is a way to compute g securely. In our context, f is the secure comparison of distances, and g is Protocol 5.1. We show that our protocol uses comparison in a way that reveals nothing.

We first define the simulator for the view of site S_i. Before the comparison phase, S_i can compute its view from its own input. The comparison phase involves communication, so simulation is more difficult. If we look at a single comparison, S_i sees several things. First, it sees the identifier $R_h[j].id$. Since S_i knows E_K, h, and j; the simulator can generate the exact identifier, so the simulator view is identical to the actual view. It also sees a share of the comparison result. If $i = h$, S_i generates the values locally, and the simulator does the same. If not local, there are two possibilities. If $id > R_h[j].id$, it holds the second argument, and generates a random bit as its share of the comparison result. The simulator does the same. Otherwise, the secure comparison will generate S_i's share of the comparison. Assume $d < R_h[j].d$: S_i's share is $0 \oplus r$, where r is S_h's randomly chosen share. Assuming S_h is equally likely to generate a 1 or 0, the probability that S_i's share is 1 is 0.5. This is independent of the input – thus, a simulator that generates a random bit has the same likelihood of generating a 1 as S_i's view in the real protocol. The composition theorem (and prior work on secure comparison) shows the algorithm so far is privacy preserving.

We can extend this argument to the entire set of comparisons seen by S_i during execution of the protocol. The probability that the simulator will output a particular binary string x for a given sequence of comparisons is $\frac{1}{2^{nk-1}}$. Since actual shares of the comparison result are chosen randomly from a uniform distribution, the same probability holds for seeing x during actual execution:

$$\begin{aligned} Pr\left[VIEW_{S_i}^{v_j} = x\right] &= \frac{1}{2^{nk-1}} \\ &= Pr\left[Simulator_i = x\right] \end{aligned}$$

Therefore, the distribution of the simulator and the view is the same for the entire result vectors. Everything else is simulated exactly, so the views are computationally indistinguishable. Nothing is learned during the comparison phase.

The sites S_i now encrypt the result vectors; again the simulator mimics the actual protocol. Since the sources were indistinguishable, the results are as well.

The next step is to show that S_s learns nothing from receiving ER_i. Site S_s can generate the identifiers $ER_i[j].id$ it will receive, as in simulating the comparison phase. By the security definitions of encryption, the $E_c(v)$ must be computationally indistinguishable from randomly generated strings of the same length as the encrypted values, provided no two v are equal (which they cannot be, as discussed above.) Likewise, the definition of probabilistic encryption ensures that the $E_o(x)$ are computationally indistinguishable from randomly generated strings of the same length. Since E_c and E_o are public, S_s knows the length of the generated strings. The simulator chooses a random string from the domain of E_c and E_o; the result is computationally indistinguishable from the view seen during execution of the protocol. (If S_s is one of the S_i, the simulator must reuse the ER_s generated during the comparison simulation instead of generating a new one.)

C receives $n * k$ tuples consisting of an identifier, an encrypted comparison set v, and encrypted class value $E_o(c)$. Since the identifiers are created with an encryption key unknown to C, the values are computationally indistinguishable from random values. The simulator for C randomly selects $k * n$ identifiers from a uniform distribution on the domain of ER_i. The outcomes $E_o(c)$ are simulated the same as by S_s above. The hardest part to simulate is the comparison set. Since the comparison produces a total ordering, C cannot simply generate random comparison results. Instead, the simulator for C picks an identifier i_1 to be the closest, and generates a comparison set consisting of all the other identifiers and randomly chosen bits corresponding to the result shares. It then inserts into the comparison set for each other identifier i_k the tuple consisting of i_1 and the appropriate bit so that the comparison of i_1 with i_k will show i_1 as closest to q. For example, if $i_1 \geq i_k$, then $f(i_k, i_1)$ should

be 1. If the bit for i_1's share is chosen to be 0, the tuple $(i_1, 1)$ is placed in i_k's comparison set. By the same argument used in the comparison phase, this simulator generates comparison values that are computationally indistinguishable from the view seen by C. Since the actual identifiers are computationally indistinguishable, and the value of the comparison is independent of the identifier value, the order of identifiers generated by C is computationally indistinguishable from the order in the real execution. The simulator encrypts these sets with E_c to simulate the data received.

In the final stage, O sees the $NR[i]$. The simulator for O starts with $NR_d[i] = c_i \oplus r_i$. The one-time pad r_i (unknown to O) ensures NR_d can be simulated by random strings of the length of $NR_d[i]$. Xor-ing the $NR_d[i]$ with r simulates NR_d. The final step reveals $E_o(c)$ to O, where c is the majority class. Since O knows the result c, the simulator generates $E_o(c)$ directly. Applying the composition theorem shows that the combination of the above simulation with the secure circuit evaluation is secure.

We have shown that there is a simulator for each site whose output is computationally indistinguishable from the view seen by that site during execution of the protocol. Therefore, the protocol is secure in the semi-honest model. ■

The algorithm actually protects privacy in the presence of malicious parties, providing O and C do not collude. Note that since each site holds a random share of each of the comparison of its own items, and that random share is not disclosed to any site but C, even if all the other S_i collude against a site S_1 they can still simulate their comparison shares for S_1. Collusion of the S_i and O could reveal S_1's k nearest neighbors, as the other S_i could simply state their k-nn were farther from x that S_1's, giving O S_1's k nearest neighbor. The same argument allows O and any subset of the S_i to collude against the honest subset of S_i as a group. However, assuming no collusion involving O and C, no data traceable to any honest S_i can be revealed even in the presence of malicious parties.

5.3 Communication and Computation Cost Analysis

Privacy is not free. Assume m is the size required to represent the distance, and q bits are required to represent the result. A simple insecure distributed k-nn protocol would have the S_i send their k nearest neighbor distances/results to O, for $O(nk(m+q))$ bit communication cost. The computation by O could easily be done in $O(nk\log(k))$ comparisons. (One pass through the data, inserting each item into the appropriate place in the running list of the k nearest neighbors.) Although we do not claim this is optimal, it makes an interesting reference point.

In the secure protocol 5.1, each site performs k^2 comparisons with every other site. Since there are $\frac{(n-1)n}{2}$ different site combinations, this gives $O(n^2k^2)$ comparisons. Each m bit secure comparison has communication cost $O(mt)$, where t is a security parameter, i.e., the key size used for encryption. Therefore, the total communication cost of the comparison phase in protocol 5.1 is $O(n^2k^2mt)$ bits. Assuming the Blum-Goldwasser encryption scheme, each site then sends $O(nk+t)$ bits of encrypted comparison shares for each item, plus the $O(q+t)$ result, to S_s and on to C. This gives $O(n^2k^2+nkq+nkt)$ bits, which sends $O(k(q+t))$ bits of encrypted result to O. The dominating factor is the secure comparisons, $O(n^2k^2mt)$; asymptotically greater than the simple method.

To evaluate the computation cost, we count the number of oblivious transfers (the dominating factor of the cost for secure circuit evaluation) and the number of encryptions. There are total $O(nk)$ encryptions of the query results, each of size q, and $O(nk)$ encryptions of comparison sets of size $O(nk)$. The dominating factor is again the $O(n^2k^2)$ secure comparisons. Each of these requires $O(m)$ 1 out of 2 oblivious transfers. An oblivious transfer requires a constant number of encryptions, giving $O(n^2k^2m)$ encryptions as the dominating computation cost. Assuming RSA public-key encryption for the oblivious transfer, the bitwise computation cost is $O(n^2k^2mt^3)$.

The parallelism inherent in a distributed system has a strong impact on the

execution time. Since the secure comparisons may proceed in parallel, the time complexity is $O(nk^2mt^3)$. Batching the comparisons between each pair of sites allows all comparisons to be done in a constant number of rounds. Thus, the dominating *time* factor would appear to be decryption of the nk comparison sets, each of size $O(nk)$. Note that m must be greater than $\log(nk)$ to ensure no equality in distances, so unless n is large relative to the other values the comparisons are still likely to dominate. Once decrypted, efficient indexing of the comparison vectors allows the same $O(nk\log(k))$ cost to determine the k nearest neighbor as in the simple insecure protocol described above.

A more interesting comparison is with a fully secure k-nn algorithm based directly on secure circuit evaluation. The generic method for evaluating a circuit with n parties requires $O(n^2C)$ 1 out of 2 oblivious transfers, where C is the size of the circuit. This gives a communication complexity $O(n^2Ct)$. To compare our result with the generic method, we would need a lower bound on the size of a circuit that evaluates k-nn classification on nk $(m+q)$-bit inputs. An obvious lower bound for the circuit size is $\Omega(nk(m+q))$: the circuit must (at least) be capable of processing all data. The generic method has a bit complexity of at least $O(n^2nk(m+q)t)$. Our method clearly wins if $n > k$ and is asymptotically superior for fixed k; for $n \leq k$ the question rests on the complexity of an optimal circuit for k-nn classification.

5.4 Conclusions

We have presented a provably secure algorithm for computing k-nn classification from distributed sources. The method we have presented is not cheap – $O(n^2k^2)$ where n is the number of sites – but when the alternative is not performing the task at all due to privacy concerns, this cost is probably acceptable.

6 PRIVACY-PRESERVING DISTRIBUTED NAïVE BAYES CLASSIFIER

.

6.1 Introduction

Naïve Bayes is a simple but highly effective classifier. This combination of simplicity and effectiveness has led to its use as a baseline standard by which other classifiers are measured. With various enhancements it is highly effective, and receives practical use in many applications (e.g., text classification [56]). This chapter extends the portfolio of privacy-preserving distributed data mining to include this standard classifier.

In Chapter 6.2, we briefly describe the Naïve Bayes classifier. We then present the model, algorithm and proof of security for horizontally partitioned data in Chapter 6.3.

6.2 The Naïve Bayes Classifier

The *Naïve Bayes classifier* is a highly practical Bayesian learning method. The following description is based on the discussion in Mitchell [56]. The Naïve Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and the target function $f(x)$ can take on any value from some finite set C. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values $< a_1, a_2, \ldots, a_n >$. The learner is asked to predict the target value, or classification, for this new instance.

The Bayesian approach to classifying the new instance is to assign the most probable target value, c_{MAP}, given the attribute values $< a_1, a_2, \ldots, a_n >$ that describe the instance.

$$c_{MAP} = \underset{c_j \in C}{argmax} \left(P(c_j | a_1, a_2, \ldots, a_n) \right) \tag{6.1}$$

Using Bayes theorem,

$$\begin{aligned} c_{MAP} &= \underset{c_j \in C}{argmax} \left(\frac{P(a_1, a_2, \ldots, a_n | c_j) P(c_j)}{P(a_1, a_2, \ldots, a_n)} \right) \\ &= \underset{c_j \in C}{argmax} \left(P(a_1, a_2, \ldots, a_n | c_j) P(c_j) \right) \end{aligned} \tag{6.2}$$

The Naïve Bayes classifier makes the further simplifying assumption that the attribute values are conditionally independent given the target value. Therefore,

$$c_{NB} = \underset{c_j \in C}{argmax} \left(P(c_j) \prod_i P(a_i | c_j) \right) \tag{6.3}$$

where c_{NB} denotes the target value output by the Naïve Bayes classifier.

The conditional probabilities $P(a_i|c_j)$ need to be estimated from the training set. The prior probabilities $P(c_j)$ also need to be fixed in some fashion (typically by simply counting the frequencies from the training set). The probabilities for differing hypotheses (classes) can also be computed by normalizing the values received for each hypothesis (class).

Probabilities are computed differently for nominal and numeric attributes.

6.2.1 Nominal Attributes

For a nominal attribute X with r possible attributes values $x_1, \ldots, x_r$, the probability $P(X = x_k | c_j) = \frac{n_j}{n}$ where n is the total number of training examples for which $C = c_j$, and n_j is the number of those training examples that also have $X = x_k$.

6.2.2 Numeric Attributes

In the simplest case, numeric attributes are assumed to have a "normal" or "Gaussian" probability distribution.

The probability density function for a normal distribution with mean μ and variance σ^2 is given by

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{6.4}$$

The mean μ and variance σ^2 are calculated for each class and each numeric attribute from the training set. Now the required probability that the instance is of the class c_j, $P(X = x'|c_j)$, can be estimated by substituting $x = x'$ in equation 6.4.

6.3 Privacy-preserving Distributed Naïve Bayes Classification

In this Chapter, we will focus on securely learning a Naive Bayesian Classifier on horizontally partitioned data. Records of different patients that are treated by different hospitals can be seen as an example of horizontally partitioned data. All of the information for a given patient is contained at one hospital, but different hospitals have different patients.

In order to see how a privacy-preserving Naive Bayesian classifier is constructed, we need to address two issues: How to select the model parameters and how to classify a new instance. The following discussion provide details on both issues. We first give a simple and very efficient protocol that compromise slightly on security. At the end of the protocol, all parties learn the total number of instances. In effect, they learn the numerator and denominator for all the fractions computed. For multiple parties, this may not be a serious privacy concern. However, we also present a technical solution to this problem. Thus, in Chapter 6.3.3, we present methods which do not reveal anything except the class of an instance to be classified.

6.3.1 Building the classifier model

The procedures for calculating the parameters are different for nominal attributes and numeric attributes. They are described below.

Nominal attributes

For a nominal attribute, the conditional probability that an instance belongs to class c given that the instance has an attribute value $A = a$, $P(C = c|A = a)$, is given by

$$P(C = c|A = a) \quad = \quad \frac{P(C = c \cap A = a)}{P(A = a)} = \frac{n_{ac}}{n_a}. \tag{6.5}$$

n_{ac} is the number of instances in the (global) training set that have the class value c and an attribute value of a, while n_a is the (global) number of instances which simply have an attribute value of a. The necessary parameters are simply the counts of instances, n_{ac} and n_a. Due to horizontal partitioning of data, each party has partial information about every attribute. Each party can locally compute the local count of instances. The global count is given by the sum of the local counts. Securely computing a global count is straight forward. (See secure summation in Chapter 3.5.1.) Assuming that the total number of instances is public, the required probability can be computed by dividing the appropriate global sums. Note that the local number of instances is not revealed. Protocol 6.1 formally defines the protocol.

For an attribute a with l different attribute values, and a total of r distinct classes, $l * r$ different counts need to be computed for each combination of attribute value and class value. For each attribute value a total instance count also needs to be computed, which gives l additional counts.

Numeric attributes

For a numeric attribute, the necessary parameters are the mean μ and variance σ^2 for each class. Again, the necessary information is split between the parties. To compute the mean, each party needs to sum the attribute values of the appropriate instances having the same class value. These local sums are added together and divided by the total number of instances having that same class to get the mean for that class value. Once all of the means μ_y are known, it is quite easy to compute

Protocol 6.1 Nominal attributes

Require: k parties, r class values, l attribute values

1: $\{c^x_{yz}$ represents #instances with party P_x having class y and attribute value $z\}$
2: $\{n^x_y$ represents #instances with party P_x having class $y\}$
3: $\{p_{yz}$ represents the probability of an instance having class y and attribute value $z\}$
4: **for all** class values y **do**
5: **for** $i = 1 \dots k$ **do**
6: $\forall z$, Party P_i locally computes c^i_{yz}
7: Party P_i locally computes n^i_y
8: **end for**
9: **end for**
10: $\forall (y, z)$, All parties calculate using the secure sum protocol (see chapter 3.5.1), $c_{yz} = \sum_{i=1}^{k} c^i_{yz}$
11: $\forall y$, All parties calculate using secure sum protocol, $n_y = \sum_{i=1}^{k} n^i_y$
12: All parties calculate $p_{yz} = c_{yz}/n_y$

the variance σ_y^2, for all class values. Since each party knows the classification of the training instances it has, it can subtract the appropriate mean μ_y from an instance having class value y, square the value, and sum all such values together. The global sum divided by the global number of instances having the same class y gives the required variance σ_y^2. Protocol 6.2 formally describes the protocol.

Protocol 6.2 Numeric attributes

1: $\{x_{iyj}$ represents the value of instance j from party i having class value $y\}$
2: $\{s_y^i$ represents the sum of instances from party i having class value $y\}$
3: $\{n_y^i$ represents #instances with party P_i having class value $y\}$
4: **for all** class values y **do**
5: **for** $i = 1 \ldots k$ **do**
6: Party P_i locally computes $s_y^i = \sum_j x_{iyj}$
7: Party P_i locally computes n_y^i
8: **end for**
9: All parties calculate using the secure sum protocol (see Chapter 3.5.1), $s_y = \sum_{i=1}^{k} s_y^i$
10: All parties calculate using secure sum protocol, $n_y = \sum_{i=1}^{k} n_y^i$
11: All parties calculate $\mu_y = s_y / n_y$
12: **end for**
13: {Create $\vec{V} = (\vec{X} - \mu)^2$}
14: **for** $i = 1 \ldots k$ **do**
15: $\forall j, v_{iyj} = x_{iyj} - \mu_y$
16: $\forall j, v_{iy} = \sum_j (v_{iyj}^2)$
17: **end for**
18: $\forall y$, All parties calculate using secure sum protocol,
$v_y = \sum_{i=1}^{k} v_{iy}$
19: All parties calculate $\sigma_y^2 = \frac{1}{n_y - 1} * v_y$

6.3.2 Proof of Security

Clearly above protocols reveal more than the Naïve Bayes model. For example, for nominal attributes, Protocol 6.1 reveals n_y (number of instances with class value y). The obvious question is: Do they reveal more? In the following proofs, we precisely state what is revealed and show that nothing else is revealed by utilizing the definitions given in Chapter 3.2. (A protocol that reveals nothing other than the model itself is given in Chapter 6.3.3)

Theorem 6.3.1 *Protocol 6.1 securely computes the probabilities p_{yz} without revealing anything except the probability p_{yz}, the global count c_{yz} or the global number of instances n_y.*

Proof The only communication taking place is at steps 11 and 12. During these steps, the secure sum algorithm is invoked to compute the global counts c_{yz} and n_y. We apply the Theorem 3.2.1, with g being the nominal attribute computation algorithm and f being the secure sum algorithm. ■

Theorem 6.3.2 *Protocol 6.2 securely computes the means μ_y and variance σ_y^2 without revealing anything except μ_y, σ_y^2, the global sum of instance values for each class s_y and the global number of instances n_y, as also the sum v_y.*

Proof The only communication takes place at steps 9, 10 and 18. At all three of these steps the secure sum algorithm is invoked to compute s_y, n_y and v_y. Thus, again, we simply apply the Theorem 3.2.1, with g being the numeric attribute computation algorithm and f being the secure sum algorithm. ■

6.3.3 Enhancing Security

The protocols given above are not completely secure in the sense that something more than just the model parameters are revealed. The true numerators and the denominators making up the actual parameter values are revealed. For three or more parties, this allows upper bounds on the number of instances with a party and upper bounds on the composition of those instances (i.e., upper bound on the number belonging to a particular class, etc.). Privacy of individual instances is always preserved. With an increasing number of parties, it is more difficult to get accurate estimates of the remaining parties. However, with just two parties, this does reveal quite a bit of extra information. In general, the problem is to calculate the value of the fraction without knowing the shared numerator and/or shared denominator. For two parties, Du and Atallah solve exactly this problem under the term of the Division

protocol [27]. This is based on a secure scalar product protocol for 2 parties. The protocol is easily extendible to the general case of multiple parties assuming that a general scalar product protocol for multiple parties is available. However, no such protocol has yet been developed.

Note that the amount of information revealed for multiple parties is not much more than what the parameters themselves reveal. However technical solutions (even with increased cost) are more satisfying as they allow an individual decision of whether to trade off security for efficiency. In the following subsection, we now present a secure protocol based on computing the logarithm securely.

6.3.4 Secure Logarithm Based Approach

As mentioned above, to make our algorithm fully secure (i.e., reveal nothing), we need to evaluate $(\sum_{i=1}^{k} c_i / \sum_{i=1}^{k} n_i)$ securely. Here evaluating the division becomes the main problem. In order to overcome this problem, we can rewrite the above expression as follows:

$$\exp\left[\ln\big(\sum_{i=1}^{k} c_i\big) - \ln\big(\sum_{i=1}^{k} n_i\big)\right]$$

Then evaluating $\ln(\sum_{i=1}^{k} c_i) - \ln(\sum_{i=1}^{k} n_i)$ securely is sufficient. Clearly, this requires secure evaluation of $\ln(\sum_{i=1}^{k} x_i)$ function. In our work, we will use the secure $\ln(x)$ evaluation method given in Chapter 3.5.4. The one important restriction of their method is that it only works for two parties. In our case, it is easy to reduce the k party problem to the two-party case. Note that the last step in the semi-honest version of the secure summation protocol has the first party subtracting the random number from the result. So just before this subtraction occurs, no party has the summation and nothing is revealed. At this point, instead of subtracting the random number, both parties can use the secure approximate $\ln(x_1 + x_2)$ protocol given in [54]. Using their protocol, it is easy to get random v_1, v_2 such that $v_1 + v_2 = C \cdot \ln(x_1 + x_2) \bmod p$

Protocol 6.3 Fully secure approach for nominal attributes

Require: k parties, r class values, l attribute values

1: $\{c^x_{yz}, n^x_y, p_{yz}$ are defined as in Protocol 6.1$\}$
2: **for all** class values y **do**
3: **for** $i = 1 \ldots k$ **do**
4: $\forall z$, Party P_i locally computes c^i_{yz}
5: Party P_i locally computes n^i_y
6: **end for**
7: **end for**
8: $\forall (y, z)$, All parties, use secure sum protocol (see Chapter 3.5.1) until last step for finding $c_{yz} = \sum_{i=1}^{k} c^i_{yz}$ and $n_y = \sum_{i=1}^{k} n^i_y$
9: Let party 1 has R_c and R_n
10: Let party k has $R_c + c_{yz} \bmod p$ and $R_n + n_y \bmod p$
11: {Note that last step of the summation has not been executed}
12: Using secure $\ln(x)$ protocol, party 1 and k gets random v_1,v_k s.t., $v_1 + v_k = C \cdot \ln(R_c + c_{yz} - R_c \bmod p) \bmod p$
13: Using secure $\ln(x)$ protocol, party 1 and k gets random u_1,u_k s.t., $u_1 + u_k = C \cdot \ln(R_n + n_y - R_n \bmod p) \bmod p$
14: Party k calculates $s_k = v_k - u_k \bmod p$ and sends it to party 1
15: Party 1 calculates the $s_1 = s_k + v_1 - u_1 \bmod p$
16: All parties calculate $p_{yz} = \exp(s_1/C)$

One important fact about the secure $\ln(x)$ evaluation algorithm is that there is a public constant C used to make all elements integral. The method for determining C is given in [54]. Also, operations are executed in a field with size p that is capable of containing the actual results multiplied by the constant. Our reduction requires us to slightly change the protocol. In [54], protocol $x_1 + x_2$ is directly added using a small addition circuit. In our case we use modular addition. (This does not change the asymptotic performance of the method.) After using secure logarithm, it is easy to evaluate our desired function securely. Protocol 6.3 describes how these ideas can be applied to our problem. Here, we only give the protocol for nominal attributes, it is straightforward to extend this to continuous attributes.

Theorem 6.3.3 *Protocol 6.3 securely evaluates p_{yz} in the semi-honest model.*

Proof To show that the above protocol is secure in the semi-honest model, we will show that each party's view of the protocol can be simulated based on its input and

its output. Again, we will use the secure composition theorem to prove the entire protocol secure since our method securely reduces to the logarithm function.

Parties $2, \ldots, k-2$ only see a summation added to some random number. Therefore, as earlier, the simulator for these parties will be a uniform number generator. Note that the probability that they will see some number x during the execution is $\frac{1}{p}$. The simulator will generate the number with the same probability.

For parties 1 and k, there is the additional step of computing the logarithm. We have to show that this does not reveal anything either. Assume that logarithm protocol returns random shares of the result.

Now let us define the simulator for the party k. Clearly, before the logarithm protocol has started, party k has $R_n + n_y \bmod p$ and $R_c + c_{yz} \bmod p$. These are indistinguishable from a random number drawn from an uniform distribution. The execution of the logarithm protocol can be simulated by using the simulator for the logarithm protocol. The details for this simulator can be found in [54]. After the protocol, party k only sees u_2, v_2 which are also indistinguishable from uniform distribution. Therefore the messages it sees during the protocol can be easily generated by an uniform random number generator.

If we look at the messages received by party 1, one set of messages come from the execution of logarithm, then it receives random shares of u_1, v_1. Also it receives $(u_2 - v_2) \bmod p$. We can define the simulator for k as follows: First it runs the simulator of the logarithm function, then it generates three random numbers uniformly chosen between 0 and $p-1$. Note that u_2, v_2 are independent and $u_2 - v_2 \bmod p$ is also uniformly distributed, as:

$$\begin{aligned}
\Pr(u_2 - v_2 = k \bmod p) &= \sum_{v=0}^{p-1} \Pr(u_2 = k + v \bmod p | v_2 = v) \cdot \Pr(v_2 = v) \\
&= \sum_{v=0}^{p-1} \Pr(u_2 = k + v \bmod p) \cdot \Pr(v_2 = v) \\
&= \sum_{v=0}^{p-1} \frac{1}{p^2} = \frac{1}{p}
\end{aligned}$$

This concludes the proof. ■

Communication and Computation Cost

Privacy is not free. In order to evaluate the secure logarithm, we need to make $O(\log(p))$ oblivious transfer and total $O(\log(p) \cdot t)$ bits must be transferred. (p is the size of the field used and depends on the range of the variables and the precision required in calculating the logarithm; t is the security parameter.) Therefore, total number of bits transferred will be $O(\log(p) \cdot (t + k))$, where k is the number of parties. Since oblivious transfer is much more expensive then addition, the $O(\log(p))$ oblivious transfers will dominate the computation cost.

6.4 Conclusions

In this chapter, we showed how to create a privacy-preserving distributed Naïve Bayes Classifier. We gave a new secure division protocol based on the existing secure logarithm protocol. We analyzed the computation and the communication cost of the stated protocols.

7 WHEN DO DATA MINING RESULTS VIOLATE PRIVACY?

7.1 Introduction

In the previous chapters, we gave provably secure distributed data mining protocols that reveal nothing but the resulting data mining model. This work still leaves a privacy question open: Do the resulting data mining models inherently violate privacy?

This chapter presents a start on methods and metrics for evaluating the privacy impact of data mining models. While the methods provide results only for classification, they provide a cross-section of what needs to be done, and a demonstration of techniques to analyze privacy impact. Work in privacy-preserving data mining has shown how to build models when the training data is kept from view; the full impact of privacy-preserving data mining will only be realized when we can guarantee that the resulting models do not violate privacy.

To make this clear, we present a "medical diagnosis" scenario. Suppose we want to create a "medical diagnosis" model for public use: a classifier that predicts the likelihood of an individual getting a terminal illness. Most individuals would consider the classifier output to be sensitive – for example, when applying for life insurance. The classifier takes some public information (age, address, cause of death of ancestors), together with some private information (eating habits, lifestyle), and gives a probability that the individual will contract the disease at a young age. Since the classifier requires some information that the insurer is presumed not to know, can we state that the classifier does not violate privacy?

The answer is not as simple as it seems. Since the classifier uses some public information as input, it would appear that the insurer could *improve* an estimate of the disease probability by repeatedly probing the classifier with the known public

information and "guesses" for the unknown information. At first glance, this appears to be a privacy violation. Surprisingly, as we show in Chapter 7.1.1, given reasonable assumptions on the external knowledge available to an adversary we can *prove* the adversary learns nothing new.

In this chapter, we assume that data falls into three classes:

- **Public Data:**(P) This data is accessible to every one including the adversary.
- **Private/Sensitive Data:**(S) We assume that this kind of data must be protected: The values should remain unknown to the adversary.
- **Unknown Data:**(U) This is the data that is not known to the adversary, and is not *inherently* sensitive. However, before disclosing this data to an adversary (or enabling an adversary to estimate it, such as by publishing a data mining model) we must show that it does not enable the adversary to discover sensitive data.

7.1.1 Example: Classifier Predicting Sensitive Data

The following example shows that for the "medical diagnosis" scenario above, it is reasonable to expect that publishing the classifier will not cause a privacy violation. Individuals can use the classifier to predict their own likelihood of disease, but the adversary (insurer) does not gain any *additional* ability to estimate the likelihood of the disease.

To simplify the problem, we assume that the classifier is a "black-box": the adversary may probe (use the classifier), but cannot see inside. An individual can use the classifier without any risk of disclosing either their private data or their private result.[1] This represents a best-case scenario: If this classifier violates privacy, then no approach (short of limiting the adversary's access to the classifier) will provide privacy protection.

[1]This is feasible as shown in Chapter 8

Formally, suppose $X = (P, U)^T$ is distributed as $N(0, \Sigma)$ with

$$\Sigma = \begin{pmatrix} 1 & r \\ r & 1 \end{pmatrix}, \tag{7.1}$$

where $-1 < r < 1$ is the correlation between P and U. Assume that for n independent samples $(x_1, x_2, \ldots, x_n)$ from $N(0, \Sigma)$, the sensitive data $S = (s_1, s_2, \ldots, s_n)$ can be discovered by a classifier C_0 that compares the public data p_i and the unknown data u_i:

$$s_i = C_0(x_i) = \begin{cases} 1 & \text{if } p_i \geq u_i \\ 0 & \text{otherwise;} \end{cases}, \text{ where:} \tag{7.2}$$

- each p_i is a public data item that everyone can access,
- the data items denoted by u_i are unknown to the adversary; u_i is only know to the i-th individual,
- each s_i is sensitive data we need to protect, and
- The adversary knows that $X \sim N(0, \Sigma)$, it may or may not know r.

We now study whether publishing the classifier C_0 violates privacy, or equivalently, whether the adversary can get a better estimate of any s_i by probing C_0.

Given the public data p_i for an individual i, the adversary could try to probe the classifier C_0 to get an estimate of s_i as follows. It is reasonable to assume that the adversary has knowledge of the (marginal) distribution that the u_i are sampled from; we can even assume that the adversary knows the joint distribution that $(p_i, u_i)^T$ are sampled from, or equivalently Σ or r. (We will see soon that though the adversary seems to know a lot, he doesn't know anything more about the s_i – this makes our example more surprising). Thus for each individual or for each p_i, the adversary could sample $\tilde{u}_i$ from the conditional distribution of $(U|P)$, he then can use the pairs $(p_i, \tilde{u}_i)^T$ to probe C_0 and get an estimate $\tilde{s}_i \stackrel{\triangle}{=} C_0(p_i, \tilde{u}_i)$. Assuming that the information P was correlated with S, this will give the adversary a better estimate than simply taking the most likely result in S.

However, this assumes the adversary has no prior knowledge. In our medical example, it is likely that the adversary has some knowledge of the relationship between P and S. For example, cause of death is generally public information, giving the adversary a training set (Likely as complete as that used to generate C_0, as for some diseases – Creutzfeldt-Jakob, Alzheimer's until recently – an accurate diagnosis required post-mortem examination, so the training data for C_0 would likely be deceased individuals.)

Given that the adversary has this knowledge, what does the adversary know if we do not publish C_0? Notice that

$$Pr\{S = 1|P = p\} = \Phi(\frac{1-r}{\sqrt{1-r^2}}p) \tag{7.3}$$

$$= \begin{cases} \geq 1/2, & \text{if } p \geq 0, \\ < 1/2, & \text{otherwise,} \end{cases} \tag{7.4}$$

where $\Phi(\cdot)$ is the cdf of $N(0,1)$. According to (7.3), (or even just based on symmetry), the best classifier the adversary can choose in this situation is:

$$s_i = \begin{cases} 1 & \text{if } p_i > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{7.5}$$

Let C_1 denote this classifier.

Next, we study what the adversary knows if we publish the classifier C_0. We even allow the adversary to know r. In this situation, the best classifier the adversary can use is the Bayesian estimator C_2, which is based on the probability of $Pr\{U \leq P|P = p_i\}$:

$$s_i = \begin{cases} 1 & \text{if } Pr\{U \leq P|P = p_i\} > \frac{1}{2}\ , \\ 0 & \text{otherwise.} \end{cases} \tag{7.6}$$

However, notice that

$$\Phi(\frac{1-r}{\sqrt{1-r^2}}p_i) = Pr\{U \leq P|P = p_i\}$$

compare this to (7.3), we conclude that $C_1 \equiv C_2$.

Thus in this situation, publishing C_0 or even the key parameter r doesn't give the adversary any additional capability, as long as the adversary has no access to the u_i. This enables us to argue that even though C_0 apparently reveals sensitive information, it does *not* actually violate privacy.

As the above example demonstrates, determining if a data mining model violates privacy requires knowing many things: What information is sensitive? To whom is it sensitive? What else is known? Whose privacy is at risk? What is an acceptable tradeoff between privacy and the benefit of the data mining result, and how do we measure this tradeoff?

Specifically, in Chapter 7.2, we present a model that enables us to discuss these issues in the context of classification. Chapter 7.3 presents a metric for privacy loss for one such situation, including examples of when the metric would be appropriate and how the metric could be calculated (analytically or empirically) in specific situations.

7.2 The Model for Privacy Implications of Data Mining Results

To understand the privacy implications of data mining results, we first need to understand how data mining results can be used (and misused). As described previously, we assume data is either *P*ublic, *U*nknown, or *S*ensitive. We now discuss additional background leading toward a model for understanding the impact of data mining results on privacy.

We assume an adversary with access to *P*ublic data, and polynomial-time computational power. The adversary may have some additional knowledge, possibly including *U*nknown and *S*ensitive data for some individuals. We want to analyze the effect of giving the adversary access to a classifier C; specifically if it will improve the ability of the adversary to accurately deduce *S*ensitive data values for individuals that it doesn't already have such data for.

7.2.1 Access to Data Mining Models

If the classifier model C is completely open (e.g., a decision tree, or weights in a neural network), the model description may reveal sensitive information. This is highly dependent on the model.

Instead, we model C as a "black box": The adversary can request that an instance be classified, and obtain the class, but can obtain no other information on the classifier. This is a reasonable model: We are providing the adversary with *access* to C, not C itself. For example, nothing is revealed by the solution provided in Chapter 8 but the class of an instance [47]. (As shown before, the party holding the classifier need not even learn attribute values.)

Here, we will only consider the data mining results in the form of classification models.

7.2.2 Basic Metric for Privacy Loss

While it is nice to show that an adversary gains no privacy-violating information, in many cases we will not be able to say this. Privacy is not absolute; most privacy laws provide for cost/benefit tradeoffs when using private information. For example, many privacy laws include provisions for use of private information "in the public interest" [30]. To tradeoff the benefit vs. the cost of privacy loss, we need a metric for privacy loss.

One possible way to define such a metric for classifier accuracy is using the Bayesian classification error. Suppose for data $(x_1, x_2, \ldots, x_n)$, we have classification problems in which we try to classify x_i's into m classes which we labeled as $\{0, 1, \ldots, m-1\}$. For any classifier C:

$$x_i \mapsto C(x_i) \in \{0, 1, \ldots, m-1\}, \qquad i = 1, 2, \ldots, n,$$

we define the classifier accuracy for C as:

$$\sum_{i=0}^{m-1} Pr\{C(x) \neq i | z = i\} Pr\{z = i\}. \tag{7.7}$$

Does this protect the individual? The problem is that *some* individuals will be classified correctly: If the adversary can predict which individuals are likely to be classified correctly, then the privacy loss for those individuals is worse than expected. Tightening such bounds requires that the adversary have training data, i.e., individuals for which it knows the sensitive value.

7.2.3 Possible Ways to Compromise Privacy

The most obvious way a classifier can compromise privacy is by taking *P*ublic data and predicting *S*ensitive values. However, there are many other ways a classifier can be misused to violate privacy. We break down the possible forms a classifier that could be (mis)used by the adversary can take.

1. $P \rightarrow S$: Classifier that produces sensitive data given public data. Metric based on accuracy of classification.

$$\sup_i \left(Pr\{C(X) \neq Y | Y = i\} - \frac{1}{n_i} \right) \tag{7.8}$$

2. $PU \rightarrow S$: Classifier taking public and unknown data into sensitive data. Metric same as above.

3. $PS \rightarrow P$: Classifier taking public and sensitive data into public data. Can adversary determine value of sensitive data. (May also involve unknown data, but this is a straightforward extension.)

4. The adversary has access to *S*ensitive data for some individuals. What is the effect on privacy of other individuals of classifiers as follows.

   (a) $P \rightarrow S$: Can the adversary do better with such a classifier because of their knowledge, beating the expectations of the metric for 1.

   (b) $P \rightarrow U$: Can giving the adversary a predictor for *U*nknown data improve its ability to build a classifier for *S*ensitive data?

We gave a brief example of how we can analyze problem 2 in Chapter 7.1.1. The rest of the paper looks at item 4b above, giving both analytical and empirical methods to evaluate the privacy impact of a classifier that enables estimation of unknown values.

7.3 Classifier Revealing Unknowns

A classifier reveals a relationship between the inputs and the predicted class. Unfortunately, even if the class value is not sensitive, such a classifier can be used to create unintended inference channels. Assuming the adversary has t samples from a distribution (P, S), it can build a classifier C_1 using those t samples. Let a_1 be the prediction accuracy of the classifier C_1. Assume a "non-sensitive" classifier $C : P \rightarrow U$ is made available to the adversary. Using C, and the t samples, the adversary can build a classifier $C_2 : P, C(P) \rightarrow S$. Let a_2 be the accuracy of the C_2. If a_2 is better than a_1, then C compromises the privacy of S.

7.3.1 Formal Definition

Given a distribution (P, U, S), with P being public data that everyone including the adversary can access, S sensitive data we are trying to protect (but known for some individuals), and U is data not known by the adversary. A "black-box" classifier C is available to the adversary that can be used to predict U given P. Assume that t samples $((p_1, s_1), \ldots, (p_t, s_t))$ are already available to adversary, our goal is to test whether revealing C increases the ability of the adversary to predict the S values for unseen instances.

First, assume attributes P and U are independent, or more generally, though P and U are dependent, C only contains the marginal information of P. In such cases, classifier C would not be much help to the adversary: as C contains no valuable information of U, we expect that C would not be much more accurate than random guess, and as a result, we expect that the adversary is unable to improve his estimate

about S by using C, or formally, the Bayes error for all classifiers using P only should be the same as the Bayes error for all classifiers using $(P, C(P))$.

However, it is expected that C contains information on the joint distribution of P and U (or equivalently the conditional information of $(U|P)$), otherwise C would be uninteresting (no better than a random guess.) The adversary can thus combine C or $C(P)$ with already known information of P to create an inference channel for S, and the prediction accuracy of the newly learned classifier violates privacy.

Formally, given C and t samples from P, S, letting

$$\rho(t) = \rho_{\{t;P,S\}}, \qquad \rho(t;C) = \rho_{\{t;P,C(P),S\}}$$

be the Bayes error for classifiers using P only and using $P, C(P)$ respectively; also, letting

$$\bar{\rho} = \lim_{t\to\infty} \rho(t), \qquad \bar{\rho}(C) = \lim_{t\to\infty} \rho(t;C),$$

we have the following definition:

Definition 7.3.1 *For $0 < p < 1$, we call the classifier C (t,p)-privacy violating if $\rho(t;C) \leq \rho(t) - p$, and the classifier C is (∞,p)-privacy violating if $\bar{\rho}(C) \leq \bar{\rho} - p$.*

The important thing to notice about the above definition is that we measure the privacy violation with respect to number of available samples t. An adversary with many training instances will probably learn a better classifier than one with few training instances.

In this case, the release of the C_1 has created a privacy threat. The main difference between this example and the one given in the Chapter 7.1 is that we put a limitation on the number of available examples to the adversary.

7.3.2 Analysis for Mixture of Gaussians

We now give a formal analysis of such an inference in the case of Gaussian mixtures. Although we gave our definitions for a classifier C, in the case of the Gaussian mixtures, the sensible way to model C is the conditional distribution of

some particular attribute based on the other attributes. Note that C can also be viewed as a "black box".

Suppose $X = (P, U)^T$ is distributed as a n-dimensional 2-point mixture $(1 - \epsilon)N(0, \Sigma) + \epsilon N(\mu, \Sigma)$, where

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \qquad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{12}' & \Sigma_{22} \end{pmatrix}. \tag{7.9}$$

For a set of t realizations $X = (x_1, x_2, \ldots, x_t)$ (here $x_i = (p_i, u_i)^T$), t sensitive data $S = (s_1, s_2, \ldots, s_t)$ are generated according to the rule:

$$s_i = \begin{cases} 1, & \text{if } x_i \text{ is generated from } N(0, \Sigma), \\ 0, & \text{if } x_i \text{ is generated from } N(\mu, \Sigma). \end{cases} \tag{7.10}$$

Assume:

- The adversary has access to p_i, and knows the marginal distribution of P in detail (this is possible for example for sufficiently large sample size t),
- The adversary has no access to u_i,
- The adversary knows that x_i are from the above 2-point mixture, he knows n, ϵ, μ_1, and Σ_{11}, which can be obtained through the marginal of P, but not μ_2 or any other entries in Σ that can not be obtained through the marginal of P.

We are concerned with the following two questions.

1. What is the privacy loss by releasing u_i? In other word, what is the Bayes error when we limit the adversary's to the knowledge to the above assumption.
2. What is the privacy loss by allowing the adversary to know the conditional distribution of $(U|P)$?

Before answering these questions, we work out the Bayes error when only p_i are available and when both p_i and u_i are available. Notice here that, by symmetry, the Bayes error for t samples is the same of univariate Bayes error.

By direct calculation, the Bayes error with only p_i's is:

$$\rho(\epsilon, \mu_1, \Sigma_{11}) = (1-\epsilon)Pr\{C_B(p_i) = 1 | s_i = 0\} \tag{7.11}$$

$$+ \epsilon Pr\{C_B(p_i) = 0 | s_i = 1\} \tag{7.12}$$

where C_B is the Bayesian classifier. The Bayes error can be rewritten as:

$$\rho(\epsilon, \mu_1, \Sigma_{11}) \tag{7.13}$$

$$= (1-\epsilon)\bar{\Phi}\left(\frac{a + \mu_1^T \Sigma_{11}^{-1}\mu_1}{\sqrt{\mu_1^T \Sigma_{11}^{-1}\mu_1}}\right) + \epsilon\bar{\Phi}\left(\frac{a - \mu_1^T \Sigma_{11}^{-1}\mu_1}{\sqrt{\mu_1^T \Sigma_{11}^{-1}\mu_1}}\right) \tag{7.14}$$

where $a = \log(\frac{1-\epsilon}{\epsilon})$ and $\bar{\Phi}(\cdot)$ is the survival function of $N(0,1)$.

In comparison, the Bayes error with both p_i's and u_i's is:

$$\rho(\epsilon, \mu, \Sigma) = (1-\epsilon)Pr\{C_B(p_i, u_i) = 1 | s_i = 0\}$$

$$+ \epsilon Pr\{C_B(p_i, u_i) = 0 | s_i = 1\}.$$

This can be rewritten as:

$$(1-\epsilon)\bar{\Phi}\left(\frac{a + \mu'\Sigma^{-1}\mu}{\sqrt{\mu'\Sigma^{-1}\mu}}\right) + \epsilon\bar{\Phi}\left(\frac{a - \mu'\Sigma^{-1}\mu}{\sqrt{\mu'\Sigma^{-1}\mu}}\right).$$

We can now answer question 1:

Lemma 7.3.1 *Let* $\psi(z) \stackrel{\triangle}{=} (1-\epsilon)\bar{\Phi}(\frac{a+z}{\sqrt{z}}) + \epsilon\bar{\Phi}(\frac{a-z}{\sqrt{z}})$. *Then*

1. $\psi(z)$ *strictly decreases in* z.

2. $\mu_1^T \Sigma_{11}^{-1}\mu_1 \leq \mu^T\Sigma^{-1}\mu$ *with equality if and only if* $\mu_2 = \Sigma_{12}^T \Sigma_{11}^{-1}\mu_1$.

3. *As a result,* $\rho(\epsilon, \mu, \Sigma) \leq \rho(\epsilon, \mu_1, \Sigma_{11})$, *with equality if and only if* $\mu_2 = \Sigma_{12}^T \Sigma_{11}^{-1}\mu_1$.

The proof of Lemma 7.3.1 is omitted. Lemma 7.3.1 tells us that, in general, releasing u_i's or any classifier that predicts u_i's will compromise privacy. This loss of privacy can be measured by Bayes error, which has an explicit formula and can be easily evaluated through the function $\psi(z)$.

Next, for question 2, we claim that from the privacy point of view, telling the adversary the detailed conditional distribution of $(U|P)$ is equivalent to telling the

adversary all the u_i, in other words, the privacy loss for either situation are exactly the same. To see this, notice that when the adversary knows the conditional distribution of $(U|P)$, he knows the distribution of S in detail since he already knew the marginal distribution of P. Furthermore, he can use this conditional distribution to sample u_i based on each p_i, the resulting data $s_i = (p_i, \tilde{u}_i)^T$ is distributed as $(1-\epsilon)N(0,\Sigma) + \epsilon N(\mu,\Sigma)$; though s_i's are not the data on our hand, but in essence the adversary has successfully constructed an independent copy of our data. In fact, the best classifier for either case is the Bayesian rule, which classifies s_i's to 1 or 0 according to

$$\epsilon f(x;\mu,\Sigma) \geq (1-\epsilon) f(x;0,\Sigma), \tag{7.15}$$

here we use $f(x;\mu,\Sigma)$ to denote the density function of $N(\mu,\Sigma)$. Thus there won't be any difference if the adversary know any u_i's of our data set, or just know the conditional distribution of $(U|P)$. This suggests that when S is highly correlated with U, revealing any good method to predict U may be problematic.

7.3.3 Practical Use

For most distributions it is difficult to analytically evaluate the impact of a classifier on creating an inference channel. An alternative heuristic method to test the impact of a classifier is described in Algorithm 7.1. We now give experiments demonstrating the use, and results, of this approach.

Algorithm 7.1 Testing a classifier for inference channels

1: Assume that S depends on only P, U, and the adversary has at most t data samples of the form (p_i, s_i).
2: Build a classifier C_1 on t samples (p_i, s_i).
3: To evaluate the impact of releasing C, build a classifier C_2 on t samples $(p_i, C(p_i), s_i)$.
4: If the accuracy of the classifier C_2 is significantly higher than C_1, conclude that revealing C creates a inference channel for S.

We tested this approach on several of the UCI datasets [9]. We assumed that the class variable of each data set is private, treat one attribute as unknown, and

simulate the effect of access to a classifier for the unknown. For each nominal valued attribute of each data set, we ran six experiments. In the first experiment, a classifier was built without using the attribute in question. We then build a classifier with the unknown attribute correctly revealed with probability 0.6, 0.7, 0.8, 0.9, and 1.0. For example, for each instance, if 0.8 is used, the attribute value is kept the same with probability 0.8, otherwise it is randomly assigned to an incorrect value. The other attributes are unchanged.

In each experiment, we used C4.5 with default options given in the Weka package [76]. Before running the experiments, we filtered the instances with unknown attributes from the training data set. Ten-fold cross validation was used in reporting each result.

Most of the experiments look like the one shown in Figure 7.1 (the credit-g dataset). Giving an adversary the ability to predict unknown attributes does not significantly alter classification accuracy (at most 2%). In such situations, access to the public data may be enough to build a good classifier for the secret attribute; disclosing the unknown values to the adversary (e.g., by providing a "black box" classifier to predict unknowns) does not really increase the accuracy of the inference channel.

In a few data sets (credit-a, kr-vs-kp, primary-tumor, splice, and vote) the effect of providing a classifier on some attribute increased the prediction accuracy significantly. We discuss the "credit-a" data set as an example of these. If the adversary does not have an access to the 9^{th}(A9) attribute (a binary attribute), it can build a decision tree that infers the secret (class) attribute with 72% accuracy – versus 86% if given all data. This holds even if the adversary is given a classifier (C) that predicts $A9$ with 60% accuracy. However, as shown in Figure 7.2, if C has accuracy 80% or greater, the adversary can do a significantly better job of predicting the secret (class) attribute.

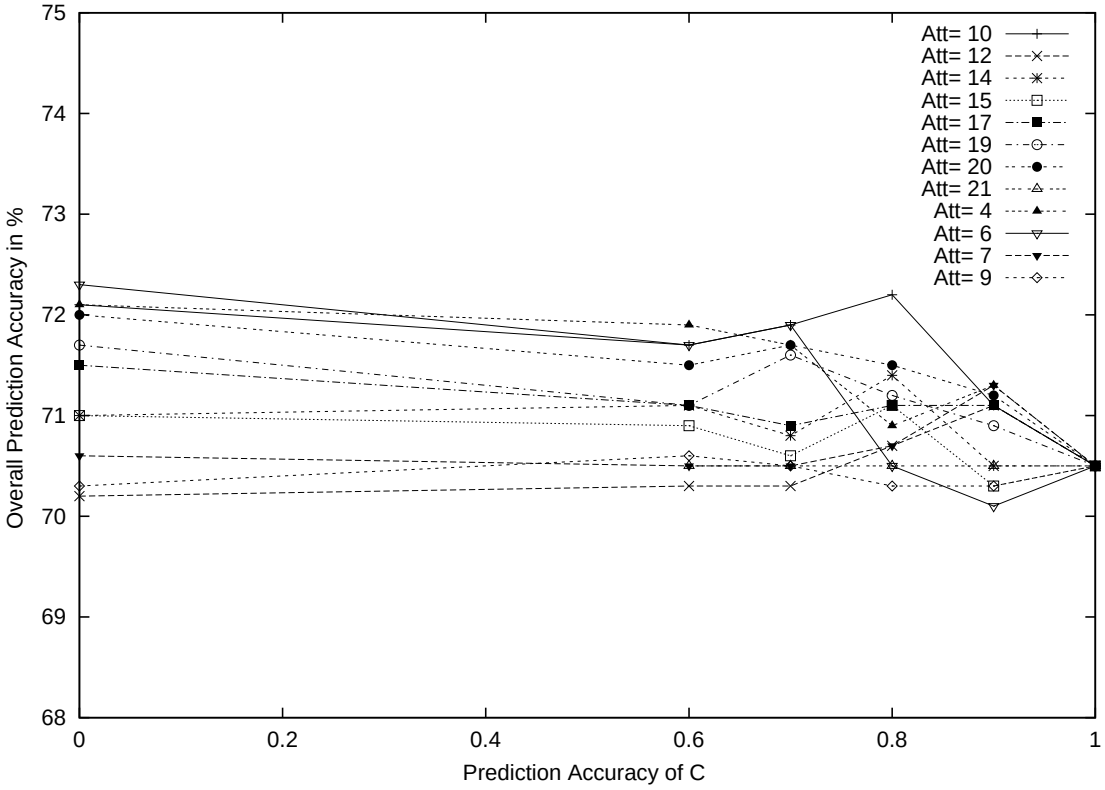

Figure 7.1. Effect of classification with varying quality estimate of one attribute on "credit-g" data (representative of most UCI data sets.)

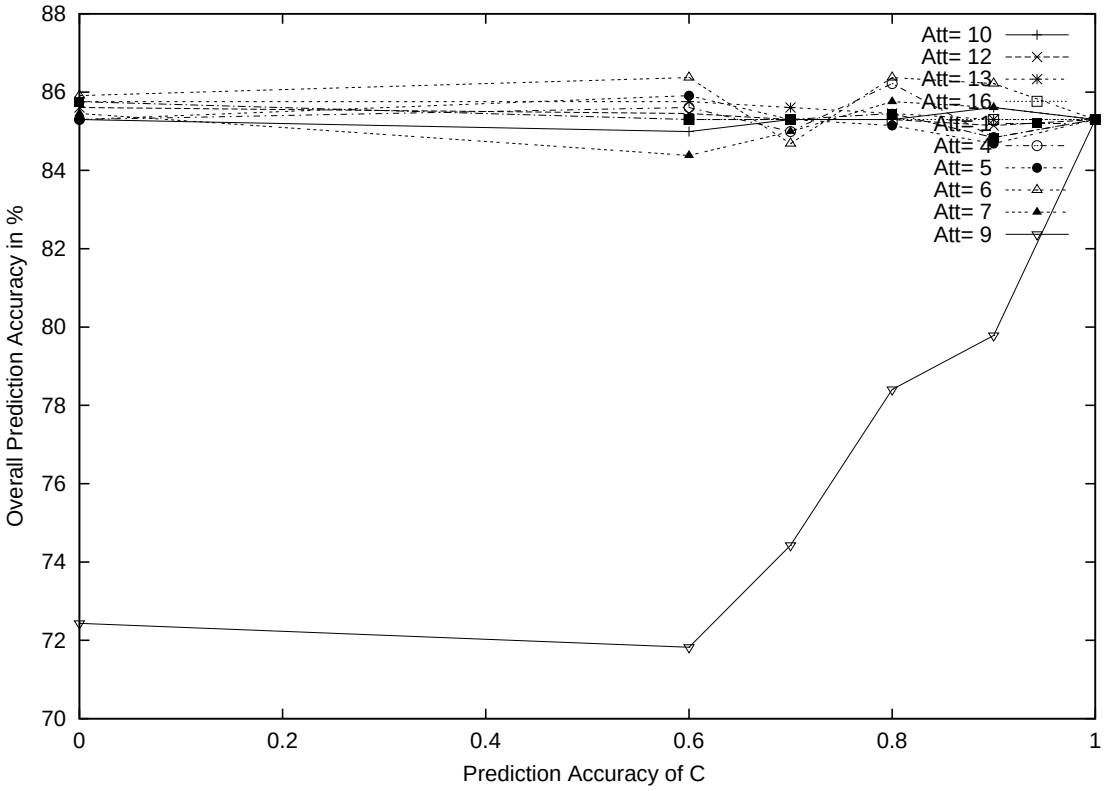

Figure 7.2. Effect of classification with varying quality estimate of one attribute on "credit-a" data (representative of five UCI data sets.)

7.4 Conclusions

Increases in the power and ubiquity of computing resources pose a constant threat to individual privacy. Tools from privacy-preserving data mining and secure multi-party computation make it possible to process the data without disclosure, but do not address the privacy implication of the results. We define this problem and explored ways that data mining results can be used to compromise privacy. We give definitions to model the effect of the data mining results on privacy, analyze our definitions for a Mixture of Gaussians for two class problems, and give a heuristic example that can be applied to more general scenarios.

8 USING DECISION RULES FOR PRIVATE CLASSIFICATION

8.1 Introduction

The new U.S. government CAPPS II initiative plans to classify each airline passenger with a green, yellow or red risk level. Passengers classified as green will be subject to only normal checks, while yellow will get extra screening and red won't fly. [13, 14] Although government agencies promise that no discriminatory rules will be used in the classification and that privacy of the data will be maintained, this does not satisfy privacy advocates. [15] Many similar profiling examples exist: Money laundering discovery from financial transaction records, smuggling in import shipments, etc.

One solution is to have the government send the classification rules to the owners of the data (e.g., credit reporting agencies). The data owners would then verify that the rules are not discriminatory, and return the classification for each passenger. The problem with this approach is that revealing the profiling rules gives an advantage to terrorists. For example, knowing that there is a rule "A one-way ticket paid in cash implies yellow risk", no real terrorist will buy one way tickets with cash.

This appears to give three conflicting privacy/security requirements. The data must not be revealed, the classifier must not be revealed, and the classifier must be checked for validity. Although these seem contradictory, we show that if such a system *must* exist, it can be done while achieving significant levels of privacy. We prove that under reasonable assumptions (i.e., the existence of one way functions, non-colluding parties) it is possible to do classification similar to the above example that provably satisfies the following conditions:

- No one learns the classification result other than designated party.

- No information other than the classification result will be revealed to designated party.
- Rules used for classification can be checked for the presence of certain conditions without revealing the rules.

While such a system still raises privacy issues (particularly for anyone classified as red), the privacy risks are significantly reduced relative to a "give all information to the government" (or anyone else) model.

We formally define the problem in Chapter 8.2. Chapter 8.3 presents the solution and a discussion of security. Chapter 8.4 shows how the parties can verify that no forbidden clauses or combinations of attributes are being used for profiling. We analyze the cost, in both order of magnitude and practical terms, in Chapter 8.5. The result is a protocol that fulfills practical requirements for secure and private profiling.

8.2 Private Controlled Classification

We first formally define the problem of classifying items using decision rules, when no party is allowed to know both the rules and the data, and the rules must be checked for "forbidden" tests. This problem could be solved using the generic method described in Chapter 3.2. While we have a construction and proof of such a circuit, the cost is prohibitive (this is discussed in Chapter 8.5). Instead, we present a comparatively efficient approach based on the notion of an untrusted third party that processes streams of encrypted data from the data and rule sources. Before describing the problem statement, we will first give a description of the decision trees and rules.

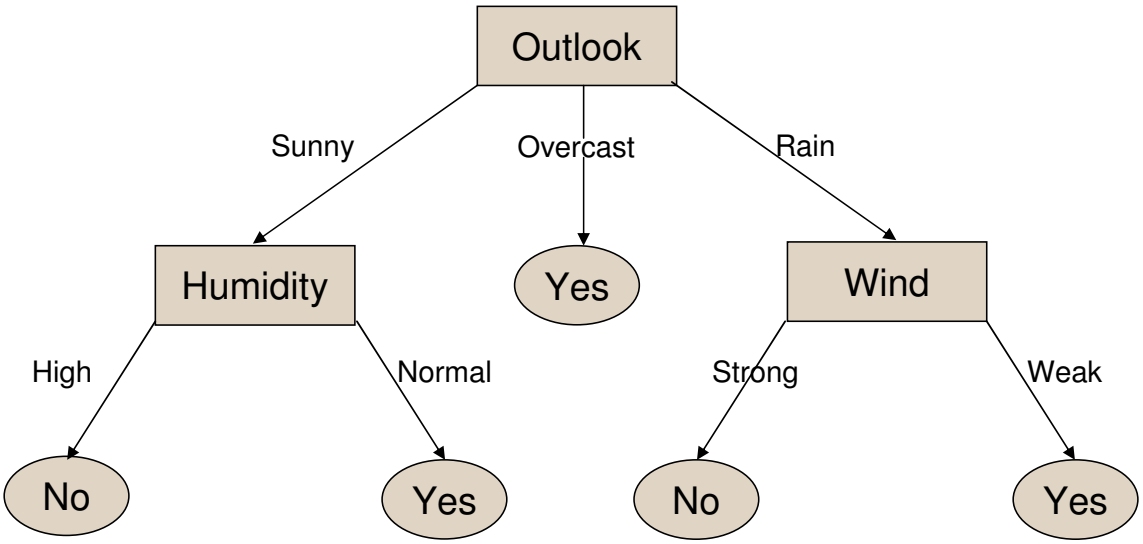

Figure 8.1. An example of a decision tree created using ID3 classifier [63].

8.2.1 Decision Trees and Rules

Decision Tree Methods are one of the most popular machine learning classification tools. The main advantage for this machine learning technique is that the model can be easily represented as human understandable rules.

Decision Trees are composed of two kinds of nodes. Internal nodes represent a particular attribute and leaf nodes have the class values. Each instance is classified by following a path from the root node to a leaf node.

Figure 8.1 illustrates an example of a decision tree (from Quinlan [63]). The above decision tree predicts whether the weather is suitable for playing tennis or not. Let us assume that on a given day, the outlook is rainy and there is a weak wind, then following the path from root node to a leaf node, we learn that the prediction is yes. Also decision tree can be represented as a set of decision rules. Each path in the decision tree corresponds to a rule. Set of the rules constructed from the paths can represent a decision tree. For example we can convert one of the paths in the tree to a rule as follows: (Outlook=Rain $\wedge$ Wind=Weak) $\Rightarrow$ Yes.

We believe that in profiling systems like CAPPS-II, government may be required to explain the reasoning behind a prediction. (Private systems may as well: The European Community privacy directive requires the logic behind automated decisions to be available to those affected [30].) Therefore, the data mining models used must be human understandable. Decision rules effectively capture a path through a decision tree: the left hand side is a set of constraints on visible attributes; the right hand side is the output class. Decision rules can also be helpful in incorporating the domain knowledge. Rules learned through the data mining process can be combined with the rules implied by the domain knowledge. As a result, we will focus on private classification using decision rules.

8.2.2 Problem Statement

Given an instance x from site Data with v attributes, we want to classify x according to a rule set R provided by site Government. Let us assume that each attribute of x has n bits, and let x_i denotes the i^{th} attribute of x. We assume that each given classification rule $r \in R$ is of the form $(L_1 \wedge L_2 \wedge \cdots \wedge L_v) \rightarrow C$ where C is the predicted class if $(L_1 \wedge L_2 \wedge \cdots \wedge L_v)$ evaluates to true. Each L_i is either $x_i = a$, $x_i \neq a$ or $x_i =?$ (a don't care; always true). While the don't care clauses are redundant in the problem definition, they will need to be included explicitly in the protocol to mask the number of clauses in each rule. By using don't cares, G can define rules with an arbitrary number of clauses; the other parties gain no information about the number of "real" clauses in the rule.

We assume that for any given x only one rule will be satisfied. Given a set of ordered rules, the use of negated clauses enables construction of an equivalent set of unordered rules where each instance can satisfy at most one rule.

In addition, D has a set F of rules that are not allowed to be used in classification. In other words, $\forall f, r \in F, R, f \not\subseteq r$. The goal is to find the class value of x according to R while satisfying the following conditions:

- D will not be able to learn any rules in R,
- D will be convinced that no clauses in F are used, and
- G will only learn the class value of x and what is implied by that class value.

To achieve a solution that is both secure and efficient, we make use of an untrusted, non-colluding site (Chapter 3.3). In our approach, the untrusted site learns only upper bounds on the number of literals v, the number of rules $|R|$, and how many literals of a given rule are not satisfied. It does not learn what those literals are, what the class is, how they relate to literals satisfied by other rules or other data items, or anything else except what is explicitly stated above.

8.3 Protocol for Private Controlled Classification

We now show how to solve the problem presented in Chapter 8.2.2 between sites D, G, and an untrusted, non-colluding site C, where C learns only

1. an upper bound on the number of attributes v,
2. an upper bound on the number of rules $|R|$, and
3. the number (and type - positive or negative) of literals that *fail* to be satisfied by each rule for a given instance x.

The first two items are innocuous. The only potential issue is with item 3. C knows exactly one rule will be satisfied for each instance (from the problem definition). However, the degree to which unsatisfied rules fail gives some information about each instance: C could cluster instances, for example. In our envisioned scenario of high-throughput profiling C does not know which instance or rule is which, or the content of the rules. Even knowing that an instance comes close to satisfying several rules does not provide damaging information; the "nearly satisfied" rules could as easily be for a harmless ("green") case rather than an indication of "near warning". While

not cryptographically secure, this does meet the standard of "protecting individually identifiable information" required under most privacy regulations.

The basic idea behind the protocol is that sites D and G send synchronized streams of encrypted data and rule clauses to site C. The order of attributes are scrambled in a way known to D and G, but not C. Each attribute is given two values, one corresponding to don't care, the other to its true value. Each clause also has two values for each attribute. One is simply an "invalid" value (masking the real value). The other is the desired result, either the a (for a clause $x_j = a$), or the agreed upon "don't care" value. C compares to see if either the first or second values match, if so then either the attribute is a match or the clause is a don't care. When the literal has a negation, site C negates the comparison result. If there is a match for every clause in a rule, then the rule is true.

To prevent revealing the number of negations, we add a "fake" literal corresponding to each real attribute. For every positive clause, there is a corresponding fake negative clause; and vice-versa. The fake literals are chosen jointly by D and G so that they always match.

The key to hiding the values is that the don't care, true, invalid, and fake values are encrypted differently for each data/rule pair in the stream, in a way shared by D and G but unknown to C. The order (is the first attribute the value, or the don't care value) also changes, again in a way known only to D and G. Since all values are encrypted (again, with a key unknown to C), the non-colluding site C learns nothing except which rule matches and the number of items that fail for rules that don't match. Since the rule identifier is also encrypted, this is useless to C.

The protocol operates in two phases: encryption and prediction. There is a third *verification* phase that can be run optionally to check for forbidden rules (described in Chapter 8.4.) Two methods are used for encryption, a one-time pad based on a pseudo-random number generator shared by D and G, and a standard deterministic encryption method E. To aid in understanding the discussion, a summary of the functions / symbols used is given in Table 8.1.

Table 8.1

Symbol	Description
E_K	Encryption with key K
R_g	Common pseudo-random generator shared by site D, G
e	Data item to be evaluated, a vector of attributes
x	Data item to be evaluated with fake attributes added
A	Rules $\times$ attributes matrix of encrypted instances created by site D
$R[i]$	Rule i, consisting of clauses corresponding to attributes of x
$F[i]$	Forbidden Rule i
B	Rules $\times$ attributes matrix of encrypted rules created by site G
$n_j \ldots n_{j+t}$	Values outside the domain of j^{th} attribute
i	Index for rules
j	Index for attributes
σ	Index for "match" and "invalid" pairs

Protocol 8.1 Private classification: encryption phase for D

Require: D and G share a pseudo-random generator R_g, $R_g(k)$ represents the k^{th} number generated by the pseudo-random generators. e is the vector of attributes for the entity to be tested. n_{j+t} are values outside the domain of e_j.

$cnt \leftarrow 0$ {Initialize on first call to protocol.}
$K_r \leftarrow R_g(cnt++)$
for $i \leftarrow 1, \ldots, |R|$ **do**
  $x[1..|e|] \leftarrow e$; $xf[1..|e|] \leftarrow 0$; $xf[|e|+1..2|e|] \leftarrow 1$
  randomly permute x and xf with $\pi(cnt++)$
  for all $x_j \in x$ **do**
    $\sigma \leftarrow (R_g(cnt++) \bmod 2)$
    if xf[j]=0 **then**
      $A[i][j][\sigma] \leftarrow E_{K_r}(x_j \oplus R_g(cnt++))$ {Real value}
    else
      $A[i][j][\sigma] \leftarrow E_{K_r}(n_{j+1} \oplus R_g(cnt++))$ {Fake value}
    end if
    $A[i][j][(1-\sigma) \bmod 2] \leftarrow E_{K_r}(n_j \oplus R_g(cnt++))$ {Don't care value}
  end for
end for
send A to site C

Protocol 8.2 Private classification: encryption phase for G

Require: D and G share a pseudo-random generator R_g, G has a private generator Rp_g.
$R_g(k)$ ($Rp_g(k)$) represents the k^{th} number generated by the pseudo-random generators.
$|e|$ is the number of literals. n_{j+t} are values outside the domain of e_j.
$cnt \leftarrow cntp \leftarrow 0$ {Initialize on first call to protocol.}
randomly permute rules in R
$K_r \leftarrow R_g(cnt++)$
{$R[i][j]$ is the i^{th} rule's j^{th} literal, $R[i].result$ is the predicted class for the i^{th} rule}
for $i \leftarrow 1, \ldots, |R|$ **do**
  $x \leftarrow R[i]$; $xf[1..|e|] \leftarrow 0$; $xf[|e|+1..2|e|] \leftarrow 1$
  randomly permute x and xf with $\pi(cnt++)$
  $vi \leftarrow$ a binary vector of length $|e|$ with $|e| - ng_i$ random 1's {ng_i is the number of literals with negation}
  $cntn \leftarrow 1$
  for $j \leftarrow 1, \ldots, |x|$ **do**
    $\sigma \leftarrow (R_g(cnt++) \bmod 2)$
    if $xf[j] = 0$ {Real value} **then**
      if $R[i][j]$ is of the form $X_j = a_j$ or $\neg(X_j = a_j)$ **then**
        $B[i][j][\sigma] \leftarrow E_{K_r}(a_j \oplus R_g(cnt++))$
        $B[i][j][(1-\sigma) \bmod 2] \leftarrow E_{K_r}((n_{j+1}) \oplus R_g(cnt++))$
        if $R[i][j]$ is of the form $X_j = a_j$ **then**
          $B[i][j][2] \leftarrow 0$
        else
          $B[i][j][2] \leftarrow 1$ {Negated value}
        end if
      else
        $B[i][j][\sigma] \leftarrow E_{K_r}((n_{j+1}) \oplus R_g(cnt++))$
        $B[i][j][(1-\sigma) \bmod 2] \leftarrow E_{K_r}(n_j \oplus R_g(cnt++))$
        $B[i][j][2] \leftarrow 0$ {Don't Care - always positive}
      end if
    else
      if $vi[cntn++] = 1$ {Fake negation} **then**
        $B[i][j][\sigma] \leftarrow E_{K_r}((n_{j+2}) \oplus R_g(cnt++))$
        $B[i][j][(1-\sigma) \bmod 2] \leftarrow E_{K_r}(n_{j+1} \oplus R_g(cnt++))$
        $B[i][j][2] \leftarrow 1$ {Fake negative}
      else
        $B[i][j][\sigma] \leftarrow E_{K_r}((n_{j+2}) \oplus R_g(cnt++))$
        $B[i][j][(1-\sigma) \bmod 2] \leftarrow E_{K_r}(n_j \oplus R_g(cnt++))$
        $B[i][j][2] \leftarrow 0$ {Fake positive}
      end if
    end if
  end for
  $r_c \leftarrow Rp_g(cntp++)$, $\bar{B}[i] \leftarrow (r_c, E_{k_r}(r_c) \oplus R[i].result)$
end for
send $B, \bar{B}$ to site C

In the encryption phase, site D first checks whether x_j is valid or a fake attribute. If it is a fake attribute it encrypts two values n_j, n_{j+1}, otherwise it creates an encrypted version of the original item and n_j. An additional attribute is added corresponding to the don't care condition using the "illegal" value n_j not in the domain of x_j. Every item is xored with a random pad before encryption.

Site G not only needs to hide the data values, it also needs to hide the number of literals with negations. For a rule R_i with ng_i literals, it must add $|e| - ng_i$ fake negations from the $|e|$ fake attributes. If the attribute is not fake, it creates two encrypted values based on the actual literal. $B[i][j][2]$ is set to 1 if the literal is a negation. G applies a different cryptographic scheme for class values (again padding with a newly generated random each time), ensuring C doesn't see that different rules may have the same class value. This ensures that C doesn't learn the class distribution of different instances over multiple runs of the protocol. This is repeated for every rule (giving multiple encryptions of the same x, but with different encryption each time.) The encryption phase is described in detail in Protocols 8.1 and 8.2.

Protocol 8.3 Private classification: prediction phase

Require: $A, B, \bar{B}$ be generated and sent to site C.

At site C:
for $i \leftarrow 1, \ldots, |R|$ **do**
  if $\forall j, 1 \leq j \leq |A[i]|, ((A[i][j][0] = B[i][j][0] \vee A[i][j][1] = B[i][j][1]) \oplus B[i][j][2])$ **then**
    $(r_c, c_e) \leftarrow \bar{B}[i]$;
    break;
  end if
end for
randomly generate r
send $(r_c, c_e \oplus r)$ to D and send r to G;

At site D:
receive (r_c, c_e) from site C;
send $c_e \oplus E_{k_r}(r_c)$ to site G

At site G:
receive r from site C and c from site D
output $c \oplus r$ as the classification result

Site C compares the vectors to find which rule is satisfied in its entirety. Note that if the literal is negation then xoring with $B[i][j][2]$ effectively negates the literal. C does not send the prediction result to site G as this would reveal which rule is satisfied, since this is the encrypted (and distinct for each rule) value rather than the actual class. C instead sends the result xored with a random number to site D. D decrypts this to get the class, but the true class value is masked by the random generated by C. Finally G can combine the information it gets from site C and site D to learn the classification. This process is fully described in Protocol 8.3.

8.3.1 Security of the Protocol

To prove that this protocol reveals nothing but the number of positive and negative literals that fail to match, and upper bounds on the total number of literals and rules, we use the secure multi-party computation paradigm of Chapter 3.2. This reduces to showing that the received messages can be simulated; the algorithm itself generates the rest of the view. During the encryption phase, D and G receive no messages.

Site C sees the matrixes A, B, and $\bar{B}$. The size of each is known. Assuming encryption is secure, output of the encryption is computationally indistinguishable from a randomly chosen string over the domain of the encryption output.

The simulator randomly generates values from the range of the encryption function and assigns them to A_s. The location of negations is simulated by randomly populating $B_s[i][j][2]$ with 1/2 0's and 1/2 1's. Please note that the original location of the negations were chosen randomly by site G. For the rest of B_s, first a number of locations corresponding to the number of non-matching positive/negative literals are chosen at random from values with the appropriate negation flag in $B_s[i][j][2]$. For these locations, a random value is used for $B_s[i][j][0]$. For the remaining locations, one of $B_s[i][j][0]$ or $B_s[i][j][0]$ is chosen to receive a random value, the other receives $A_s[i][j]$. Due to the random padding and pseudo-random functions, the

relative true/don't care mapping of $B[i][j][0]$ and $B[i][j][1]$ will be independent of $B[i][j][2]$.

Are A_s and B_s computationally indistinguishable from A and B? Assume that the output of the simulator's A_s, B_s is **not** computationally indistinguishable from the A, B seen during the protocol. Let M be the distinguisher. Then M must be able to distinguish between some $A[i][j][\sigma]$ and $A_s[i][j][\sigma]$. This means M can distinguish between a random number and $E_{K_r}(X \oplus R)$, contradicting our secure encryption assumption. For $\bar{B}$ a similar argument applies.

For the classification phase site D only sees r, d. Since both of them are random numbers, a simple random number generator can be used for the simulator. The message G receives can be simulated using a random number generator (simulating the message from C) and the actual result $(result \oplus r)$.

8.4 Checking for Forbidden Rules

Protocols 8.1, 8.2, and 8.3 generate the correct classification without revealing rules or data. Protocol 8.4 shows how C and D can test if $\forall f \in Ff \not\subset r$ (possibly before D returns the masked result to G in Protocol 8.3.) The basic idea is that D generates a set of "forbidden rules" in the same manner as by G in Protocol 8.2. D and C perform a secure set intersection to see if all of the clauses in a forbidden rule are used, if so the rule is deemed invalid.

To improve the efficiency of verification we only check that the rule used for classification is not violating privacy (i.e., it is not part of the forbidden rule set F). The index of the matching rule is given to D to seed the random number generation, but since the rules are randomly ordered this conveys no information to D. The verification phase also needs to keep C and D in the dark on the contents of rules (unless they are in violation of the forbidden rule set.) For example, we do not want to reveal the exact location of the literals with negations, also we do not want D to learn the number of matches with forbidden rule set. This is satisfied by using a

Protocol 8.4 Private classification: verification phase

Require: Let B be the rule set received from G in Protocol 8.2.

At site C:

let r_f be the rule that matched.

Send the index of r_f to D.

for $j = 1; j \leq v; j++$ **do**

$TC = TC \cup \{H(j||B[r_f][j][0]||B[r_f][j][2])\} \cup \{H(j||B[r_f][j][1]||B[r_f][j][2])\}$

{|| is string concatenation.}

end for

At site D:

Receive r_f and use it to appropriately seed the random number generator.

Generate F_e, the matrix of encrypted forbidden rules, from F using the method used by G to generate B from R in Protocol 8.2, except that it uses n_{j+2} instead of n_{j+1} for the "filler" in a true forbidden rule (to ensure that it doesn't match).

for $i = 1; i \leq |F|; i++$ **do**

for $j = 1; j \leq v; j++$ **do**

if j^{th} attribute is not "fake" or not "do not care" **then**

$TF[i] = TF[i] \cup \{H(j||F_e[i][j][\sigma]||F_e[i][j][2])\}$ {Note that $F_e[i][j][\sigma]$ is the encrypted form of the actual literal}

end if

end for

end for

{Cooperatively between C and D:}

{let l_i be the number of actual literals in F_i. Using a secure set intersection size threshold protocol, Site D (or site C, or both) learns if $\mid TC \cap TF[i] \mid = l_i$.}

if $\forall i, \mid TC[i] \cap TF[i] \mid < l_i$ **then**

no forbidden clauses are used.

end if

secure set intersection size protocol that only reveals if the intersection size is above a threshold; specifically the number of literals in the forbidden rule.

Notice that if two rules are the same, then the literals for the true attributes must be the same. By calculating the hash value of the form $H(j||B[i][j][\sigma]||B[i][j][2])$, only the same literals for the same attributes will match. Since D knows all the random numbers, it can encrypt the forbidden rules in correct form. Assume that a forbidden rule F_i has l_i literals then $|TC \cap TF[i]|$ must be l_i if F_i and the rule used for classification are the same. Since D only calculates the $H(j||F_e[i][j][\sigma]||F_e[i][j][2])$ for the genuine attributes and correct literal, the intersection size can be at most l_i. Using the secure intersection protocol (Secure set intersection is described in Chapter 3.5.5) this can be easily checked. Each secure intersection protocol execution will only reveal whether the rule used for classification and the forbidden rule is the same or not. Since the only data exchange is done during set intersection protocol, if the protocol used is secure then the check protocol will be secure.

8.4.1 Security of Verification

Each party constructs its own sets; the only information exchanged is the index of the rule to be checked. This is governed by a random permutation unknown to D (from the beginning of Protocol 8.2), so it can be simulated by choosing from a uniform distribution on $1, \ldots, |R|$. The only other communication is part of the set intersection, for which the only result is if the intersection size equals the threshold. Any protocol that can check if the size of the intersection is bigger than some threshold can be used for our purposes (Chapter 3.5.5). The final result, if this number of clauses equals the total in the forbidden rule, is a secure comparison and reveals only the result – easily simulated knowing the result.

In practice the site C would operate in a streaming mode, with D sending each new instance x and G re-encrypting the rules R. To C these would appear to be continuous streams – the repetition in R is masked by the random pad. This avoids

C learning anything based on results over multiple instances (e.g., giving C a single encrypted rule set for all instances would enable C to learn what rule was most common, even if it didn't know what that rule was.)

One interesting advantage to this method over the generic method is that by colluding (e.g., under a court order), any two sites could reveal what the third site has. For example if G does use a forbidden rule, C and D can collaborate to expose what that rule is. This is not directly possible with generic circuit evaluation, since a malfeasant G could delete the keys used in the protocol to permanently hide the forbidden rule.

8.5 Cost Analysis

Privacy is not free. Keeping the necessary information private requires many encryptions for each classification. Given v literals in each rule both sites G and D perform $O(|R| \cdot v)$ encryptions in the encryption phase. The prediction phase requires a single encryption.

In a real life execution, we need to be able to process and classify each instance very quickly. To measure the time needed for encrypting the rules and data, we implemented the encryption phase (Protocols 8.1 and 8.2) using the Java crypto library cryptix. We used AES with 128 blocks and a 128 bit key. Our experiments on a Sun Blade 1000 shows that an instance with 25 attributes can be classified using 500 rules in an average of 275 milliseconds (excluding the network transfer time). A system which uses only 100 rules with 15 attributes could classify an instance in 34 milliseconds. Hardware-accelerated cryptography would be much faster; our initial implementation suggests that using off-the-shelf hardware our secure classification throughput could easily support real life requirements.

Verification is more expensive. We need to run F set intersection protocols. Each execution of the set intersection protocol requires $O(v \ln \ln v)$ homomorphic encryptions. Therefore total computation cost will be $O(F \cdot v \ln \ln v)$ homomorphic

encryptions. Each homomorphic encryption is considerably more expensive than standard private key encryptions. Although the execution of the verification protocol will be slower in practice, it will be used rarely: simply the *credible threat* of verification is sufficient to prevent use of forbidden rules.

The communication cost of the protocol is similar. Assume the size of each encrypted value is t bits. The encryption phase sends $O(|R| \cdot v \cdot t)$ bits, and the prediction phase sends $3t$ bits. In the verification phase the set F is transmitted in encrypted form, $O(|F| \cdot v \cdot t)$ bits. The total communication cost is $O((|R|+|F|) \cdot v \cdot t)$ bits.

Assume that we have 500 rules and 25 literals, using AES with 128 bit key and 128 bit block size. We need to transfer close to 5Mb per classification. With current network speeds, this is relatively small compared to the encryption cost, particularly since it can operate in a streaming mode. While the communication cost may seem high, particularly since this is the cost per instance to be evaluated, it is likely to work well in practice. Bandwidth is growing rapidly, it is generally latency that limits performance. This protocol adapts well to streaming - the small number of messages, and the fact that each site sends only one per phase, means a continuous stream of instances could be fed through the system. The throughput of the system could approach the bandwidth of the communication network.

Note that the generic circuit evaluation is likely to be significantly more expensive. We have a circuit construction that solves the problem with $O((|R| + |F|) \cdot v \cdot n))$ encryptions where n is the number of bits to represent a literal. The circuit size is $O(|R| \cdot |F| \cdot v \cdot n)$ gates, giving a bit transmission cost of $O(|R| \cdot |F| \cdot v \cdot n \cdot t)$. Due to the necessity of representing all possible input in the construction of the circuit, a significantly more efficient approach based on generic circuit evaluation is unlikely.

8.6 Conclusions

As the usage of information technology for homeland security and other potentially intrusive purposes increases, privately *using* this information will become more important. We have shown that it is possible to ensure privacy while profiling without compromising the ultimate goal of achieving security.

9 SUMMARY

This thesis presents efficient solutions for many privacy preserving data mining tasks on horizontally partitioned data. Another result of this thesis is that many privacy preserving distributed data mining protocols on horizontally partitioned data can be efficiently implemented by securely reducing them to few basic secure building blocks. Also this thesis suggests some initial solutions on how to use the data mining results without violating privacy.

We believe the need for mining of data where access is restricted by privacy concerns will increase. Examples include knowledge discovery among intelligence services of different countries and collaboration among corporations without revealing trade secrets. Even within a single multi-national company, privacy laws in different jurisdictions may prevent sharing individual data. Many more examples can be imagined. Even more, new technologies such as a RFID tags, biometrics and DNA sequencing will enable us to collect more data that is potentially useful at the same highly privacy sensitive. We are hoping that some of the work presented in this thesis will be helpful in exploiting those useful technologies without sacrificing our privacy.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Nabil R. Adam and John C. Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys*, 21(4):515–556, December 1989.

[2] Dakshi Agrawal and Charu C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 247–255, Santa Barbara, California, May 21-23, 2001. ACM.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, Santiago, Chile, September 12-15, 1994. VLDB.

[4] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD Conference on Management of Data*, pages 439–450, Dallas, TX, May 14-19, 2000. ACM.

[5] M. Atallah, E. Bertino, A. Elmagarmid, M. Ibrahim, and V. Verykios. Disclosure limitation of sensitive rules. In *Knowledge and Data Engineering Exchange Workshop (KDEX'99)*, pages 25–32, Chicago, Illinois, November 8, 1999.

[6] Brian Babcock and Chris Olston. Distributed top-k monitoring. In *Proceedings of the 2003 ACM SIGMOD International Conference on on Management of Data*, San Diego, California, June 9-12, 2003.

[7] Josh Cohen Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In A.M. Odlyzko, editor, *Advances in Cryptography, CRYPTO86: Proceedings*, volume 263, pages 251–260. Springer-Verlag, Lecture Notes in Computer Science, 1986.

[8] Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Advances in Cryptology – EUROCRYPT'93, Workshop on the Theory and Application of Cryptographic Techniques*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285, Lofthus, Norway, May 1993. Springer-Verlag.

[9] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[10] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption that hides all partial information. In R. Blakely, editor, *Advances in Cryptology – Crypto 84 Proceedings*. Springer-Verlag, 1984.

[11] Hiawatha Bray and Sasha Talcott. Firm says up to 40m credit card files stolen. *Boston Globe*, June 18, 2005.

[12] Nicolas Bruno, Luis Gravano, and Amelie Marian. Evaluating top-k queries over web-accessible databases. In *Proceedings of the 18th International Conference on Data Engineering*, pages 369–380, San Jose, California, February 26 - March 1, 2002.

[13] Computer assisted passenger pre-screening II program. URL `http://www.fcw.com/fcw/articles/2003/0310/news-tsa-03-10-03.asp`.

[14] Fact sheet: CAPPS II at a glance, February 12 2004. URL `http://www.dhs.gov/dhspublic/display?theme=20\&content=3161`.

[15] CAPPS II: Government surveillance via passenger profiling, February 2004. http://www.eff.org/Privacy/cappsii/concern.php.

[16] Yan-Cheng Chang and Chi-Jen Lu. Oblivious polynomial evaluation and oblivious neural learning. *Lecture Notes in Computer Science*, 2248:369+, 2001.

[17] David Wai-Lok Cheung, Jiawei Han, Vincent Ng, Ada Wai-Chee Fu, and Yongjian Fu. A fast distributed algorithm for mining association rules. In *Proceedings of the 1996 International Conference on Parallel and Distributed Information Systems (PDIS'96)*, pages 31–42, Miami Beach, Florida, USA, December 1996. IEEE.

[18] David Wai-Lok Cheung, Vincent Ng, Ada Wai-Chee Fu, and Yongjian Fu. Efficient mining of association rules in distributed databases. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):911–922, December 1996.

[19] Benny Chor and Eyal Kushilevitz. A communication-privacy tradeoff for modular addition. *Information Processing Letters*, 45(4):205–210, 1993.

[20] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, 1998.

[21] Chris Clifton. Using sample size to limit exposure to data mining. *Journal of Computer Security*, 8(4):281–307, November 2000.

[22] Chris Clifton and Don Marks. Security and privacy implications of data mining. In *Workshop on Data Mining and Knowledge Discovery*, pages 15–19, Montreal, Canada, June 1996. ACM SIGMOD.

[23] R. Cramer, Niv Gilboa, Moni Naor, Benny Pinkas, and G. Poupard. Oblivious Polynomial Evaluation. Can be found in the Privacy Preserving Data Mining paper by Naor and Pinkas, 2000.

[24] Dorothy E. Denning. Secure statistical databases with random sample queries. *ACM Transactions on Database Systems*, 5(3):291–315, September 1980.

[25] Whit Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[26] David Dobkin, Anita K. Jones, and Richard J. Lipton. Secure databases: Protection against user influence. *ACM Transactions on Database Systems*, 4(1):97–106, March 1979.

[27] Wenliang Du and Mikhail J. Atallah. Privacy-preserving statistical analysis. In *Proceeding of the 17th Annual Computer Security Applications Conference*, New Orleans, Louisiana, USA, December 10-14, 2001.

[28] Wenliang Du and Zhijun Zhan. Building decision tree classifier on private data. In Chris Clifton and Vladimir Estivill-Castro, editors, *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, volume 14, pages 1–8, Maebashi City, Japan, December 9, 2002. Australian Computer Society.

[29] Cynthia Dwork and Kobbi Nissim. Privacy-preserving data mining on vertically partitioned databases. In *To appear in Proc. of CRYPTO 2004*.

[30] Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of the European Communities*, No I.(281):31–50, October 24, 1995.

[31] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, July 1985.

[32] S. Even, Oded Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[33] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2003)*, pages 211–222, San Diego, CA, June 9-12, 2003.

[34] Alexandre Evfimievski, Ramakrishnan Srikant, Rakesh Agrawal, and Johannes Gehrke. Privacy preserving mining of association rules. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–228, Edmonton, Alberta, Canada, July 23-26, 2002.

[35] Uri Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation. In *26th ACM Symposium on the Theory of Computing (STOC)*, pages 554–563, 1994.

[36] Mr. Feingold, Mr. Corzine, Mr. Wyden, and Mr. Nelson. Data Mining Moratorium Act of 2003. U.S. Senate Bill (proposed), January 16, 2003.

[37] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Eurocrypt 2004*, Interlaken, Switzerland, May 2-6, 2004. International Association for Cryptologic Research (IACR).

[38] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press Professional, Inc., 1990.

[39] Oded Goldreich. *The Foundations of Cryptography*, volume 2, chapter General Cryptographic Protocols. Cambridge University Press, 2004.

[40] Oded Goldreich. *The Foundations of Cryptography*, volume 2, chapter Encryption Schemes. Cambridge University Press, 2004.

[41] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, California, April 2000.

[42] Thomas H. Hinke and Harry S. Delugach. Aerie: An inference modeling and detection approach for databases. In Bhavani Thuraisingham and Carl Landwehr, editors, *Database Security, VI, Status and Prospects: Proceedings of the IFIP WG 11.3 Workshop on Database Security*, pages 179–193, Vancouver, Canada, August 19-21, 1992. IFIP, Elsevier Science Publishers B.V. (North-Holland).

[43] Thomas H. Hinke, Harry S. Delugach, and Randall P. Wolf. Protecting databases from inference attacks. *Computers and Security*, 16(8):687–708, 1997.

[44] Bernardo A. Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of the First ACM Conference on Electronic Commerce (EC99)*, pages 78–86, Denver, Colorado, November 3–5, 1999. ACM Press.

[45] Ioannis Ioannidis and Ananth Grama. An efficient protocol for Yao's millionaires' problem. In *Hawaii International Conference on System Sciences (HICSS-36)*, pages 205–210, Waikoloa Village, Hawaii, January 6-9, 2003.

[46] Murat Kantarcıoğlu and Chris Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *The ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02)*, pages 24–31, Madison, Wisconsin, June 2, 2002.

[47] Murat Kantarcıoğlu and Chris Clifton. Assuring privacy when big brother is watching. In *The 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'2003)*, pages 88–93, San Diego, California, June 13, 2003.

[48] Murat Kantarcıoğlu and Jaideep Vaidya. An architecture for privacy-preserving mining of client information. In Chris Clifton and Vladimir Estivill-Castro, editors, *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, volume 14, pages 37–42, Maebashi City, Japan, December 9, 2002. Australian Computer Society.

[49] Hillol Kargupta, Souptik Datta, Qi Wang, and Krishnamoorthy Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*, Melbourne, Florida, November 19-22, 2003.

[50] Flip Korn, Nikolaos Sidiropoulos, Christos Faloutsos, Eliot Siegel, and Zenon Protopapas. Fast nearest neighbor search in medical image databases. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *Proceedings of 22th International Conference on Very Large Data Bases*, pages 215–226, Mumbai (Bombay), India, September 3-6, 1996. VLDB, Morgan Kaufmann.

[51] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science*, pages 364–373. IEEE, October 20-22, 1997.

[52] Xiaodong Lin, Chris Clifton, and Michael Zhu. Privacy preserving clustering with distributed EM mixture modeling. *Knowledge and Information Systems*, to appear 2004.

[53] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO 2000*, pages 36–54. Springer-Verlag, August 20-24, 2000.

[54] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[55] Donald G. Marks. Inference in MLS database systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(1), February 1996.

[56] Tom Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1st edition, 1997.

[57] Richard A. Moore, Jr. Controlled data-swapping techniques for masking public use microdata sets. Statistical Research Division Report Series RR 96-04, U.S. Bureau of the Census, Washington, DC., 1996.

[58] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, pages 245–254, Atlanta, Georgia, 1999. ACM Press.

[59] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of SODA 2001 (SIAM Symposium on Discrete Algorithms)*, Washington, D.C., January 7-9, 2001.

[60] P. Paillier. Public key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - Eurocrypt '99 Proceedings, LNCS 1592*, pages 223–238. Springer-Verlag, 1999.

[61] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, IT-24(1):106–110, 1978.

[62] Huseyin Polat and Wenliang Du. Privacy-preserving collaborative filtering using randomized perturbation techniques. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*, pages 625–628, Melbourne, Florida, November 19-22, 2003.

[63] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[64] Michael Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[65] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[66] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[67] Shariq J. Rizvi and Jayant R. Haritsa. Maintaining data privacy in association rule mining. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 682–693, Hong Kong, August 20-23, 2002. VLDB.

[68] Yücel Saygın, Vassilios S. Verykios, and Chris Clifton. Using unknowns to prevent discovery of association rules. *SIGMOD Record*, 30(4):45–54, December 2001.

[69] Adi Shamir, Ronald L. Rivest, and Leonard M. Adleman. Mental poker. Technical Memo MIT-LCS-TM-125, Laboratory for Computer Science, MIT, February 1979.

[70] Subcommittee on Disclosure Limitation Methodology, Federal Committee on Statistical Methodology. Report on statistical disclosure limitation methodology. Statistical Policy Working Paper 22 (NTIS PB94-16530), Statistical Policy Office, Office of Information and Regulatory Affairs, Office of Management and Budget, Washington, DC., May 1994.

[71] Jaideep Vaidya. *Privacy Preserving Data Mining over Vertically Partitioned Data*. PhD thesis, Purdue University, West Lafayette, Indiana, 2004.

[72] Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 639–644, Edmonton, Alberta, Canada, July 23-26, 2002.

[73] Jaideep Vaidya and Chris Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Security*, to appear.

[74] V. S. Verykios, Ahmed K. Elmagarmid, Bertino Elisa, Yucel Saygin, and Dasseni Elana. Association rule hiding. *IEEE Transactions on Knowledge and Data Engineering*, 16(4), 2004.

[75] Vassilios S. Verykios, Elisa Bertino, Igor Nai Fovino, Loredana Parasiliti Provenza, Yücel Saygın, and Yannis Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Record*, 33(1), March 2003.

[76] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Fransisco, October 1999.

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

[78] R. Yip and K. Levitt. The design and implementation of a data level database inference detection system. In *Proceedings of the Twelfth Annual IFIP WG 11.3 Working Conference on Database Security*, Chalkidiki, Greece, July 15–17, 1998.

VITA

VITA

Murat Kantarcıoğlu received the Ph.D. degree from Purdue University in 2005 under the supervision of Prof. Christopher W. Clifton. He received his master's in computer science from Purdue University in 2002 and his bachelor degree in computer engineering from Middle East Technical University, Ankara, Turkey in 2000. During his graduate years, he worked as a summer intern at IBM Almaden Research Center and at NEC Labs.

His research interests lie at the intersection of privacy, security, data mining and databases: security and privacy issues raised by data mining; distributed data mining techniques; security issues in databases; applied cryptography and secure multi-party computation techniques; use of data mining for intrusion detection.