CERIAS Tech Report 2004-40

PRIVACY PRESERVING DATA MINING OVER VERTICALLY PARTITIONED DATA

by Jaideep Vaidya

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

PRIVACY PRESERVING DATA MINING

OVER

VERTICALLY PARTITIONED DATA

A Thesis

Submitted to the Faculty

of

Purdue University

by

Jaideep Shrikant Vaidya

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2004

To my parents, with sincere love and respect.

ACKNOWLEDGMENTS

I would like to acknowledge the overwhelming contribution and endless hours invested in me ("the biggest time sink") by my advisor, Prof. Chris Clifton. Without him, I would not even be close to being done. Apart from all his advisory help, I also have to thank this self proclaimed "information junkie", for all the useful trivia I have harvested over the past couple of years from him.

I would also like to acknowledge the innumerable contributions of my prior advisor and current committee member, Prof. Mike Atallah. Without his constant support and confidence in my abilities, I could not have continued in the Ph.D. program. Prof. Sonia Fahmy also helped me out when the going seemed tough and I was floundering in the search of a new research direction. Thank you. I would also like to thank my committee members for their careful reading of the dissertation and useful comments.

For me, the heart and soul of the Computer Science Department at Purdue has always been epitomized by Dr. William Gorman (or good ol' wjg). He has remained the best example of everything that is good about the CS department at Purdue. I thank him for simply being there.

Amy Ingram, the graduate secretary has been the ever smiling face who has made life at Purdue that much more fun. She is the single most person (apart from Dr. Gorman) whom I have bugged the most (often for no reason), but has been a loyal friend all through.

I would like to thank all of lunchbunch who saved me asphyxiation due to lack of social life. Sundar, specifically, has often been my first sounding board, as well as the source of all the midnight coffee breaks that kept me sane and ensured that I made progress on my thesis. Murat and Wei, my co-advisees have suffered with me through this past couple of years. I really appreciate their support. Interaction

with many members of CERIAS and ICDS, the groups I was affiliated with, has always been a useful source of information. Rajesh and Ramesh Venugopal have been true mentors to me since I came to the US. I thank them for the innumerable intangibles. Finally, I would also like to thank my sister Shilpa and her family for their encouragement throughout.

A special vote of thanks goes to Mrs. Patricia Clifton for bearing with me, according me the privilege of being treated as a family member, and allowing me to abuse those privileges at will.

For my parents, no words can suffice. You have and always will be my source of inspiration. I offer you my best work with the hope that it stands up to the high standards you have always expected of me. To you, I dedicate this thesis.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

PPDM	Privacy Preserving Data Mining
SMC	Secure Multiparty Computation
KDD	Knowledge Discovery in Databases
DDM	Distributed Data Mining

ABSTRACT

Vaidya, Jaideep Shrikant. Ph.D., Purdue University, August, 2004. Privacy Preserving Data Mining over Vertically Partitioned Data. Major Professor: Chris Clifton.

The goal of data mining is to extract or "mine" knowledge from large amounts of data. However, data is often collected by several different sites. Privacy, legal and commercial concerns restrict centralized access to this data. Theoretical results from the area of secure multiparty computation in cryptography prove that assuming the existence of trapdoor permutations, one may provide secure protocols for *any* two-party computation as well as for *any* multiparty computation with honest majority. However, the general methods are far too inefficient and impractical for computing complex functions on inputs consisting of large sets of data. What remains open is to come up with a set of techniques to achieve this efficiently within a quantifiable security framework. The distributed data model considered is the heterogeneous database scenario with different features of the same set of data being collected by different sites. This thesis argues that it is indeed possible to have *efficient* and *practical* techniques for useful privacy-preserving mining of knowledge from large amounts of data. The dissertation presents several privacy preserving data mining algorithms operating over vertically partitioned data. The set of underlying techniques solving independent sub-problems are also presented. Together, these enable the secure "mining" of knowledge.

1 INTRODUCTION

It is possible to efficiently extract or "mine" knowledge from large amounts of vertically partitioned data within quantifiable security restrictions. Knowledge Discovery in Databases (KDD) is the term used to denote the process of extracting knowledge from large quantities of data. The KDD process assumes that all the data is easily accessible at a central location or through centralized access mechanisms such as federated databases and virtual warehouses. Moreover, advances in information technology and the ubiquity of networked computers have made personal information much more available. Privacy advocates have been challenging attempts to bring more and more information into integrated collections. Attempts to combine data have even resulted in public protest, witness Japan's creation of a national registry containing information previously held by the prefectures [87]. Data mining in particular has come under siege, such as the introduction of U.S. Senate Bill 188, the "Data-Mining Moratorium Act of 2003" [35]. While aimed specifically at the Total Information Awareness program [88], the bill as introduced would forbid data-mining (including research and development) by the entire U.S. Department of Defense, except for searches of public information or searches based on particular suspicion of an individual. In addition, all U.S. government agencies would be required to report to congress on how their data-mining activities protect individual privacy.

The irony is that data mining *results* rarely violate privacy. The objective of data mining is to generalize across populations, rather than reveal information about individuals. The hitch is that data mining works by evaluating individual data that is subject to privacy concerns. Thus, the *true* problem is not data mining, but the way data mining is done.

However, the concern among privacy advocates is well founded, as bringing data together to support data mining makes misuse easier. Much of this information has already been collected, however it is held by various organizations. Separation of control and individual safeguards prevent correlation of this information, providing acceptable privacy in practice. However, this separation also makes it difficult to use the information for purposes that would benefit society, such as identifying criminal activity. Proposals to share information across agencies, most recently to combat terrorism, would eliminate the safeguards imposed by separation of the information.

Imagine the following scenario. A law enforcement agency wants to cluster individuals based on their financial transactions, and study the differences between the clusters and known money laundering operations. Knowing the differences and similarities between normal individuals and known money launderers would enable better direction of investigations. Currently, an individual's financial transactions may be divided between banks, credit card companies, tax collection agencies, etc. Each of these (presumably) has effective controls governing release of the information. These controls are not perfect, but violating them (either technologically or through insider misuse) reveals only a subset of an individual's financial records. The law enforcement agency could promise to provide effective controls, but now overcoming them gives access to an individual's entire financial history. This raises justifiable concerns among privacy advocates.

Similarly, application of data mining in other domains is also increasing. Recent emphasis on bioinformatics and in the medical domain try to leverage the power of data mining in finding interesting patterns, disease causes, effectiveness of drugs and so on. In the healthcare domain, especially with patient databases, legal issues are prominent obstacles in jointly utilizing information.

Privacy and data mining *can* coexist. The problem with the above scenario is not the data mining results, but how they are obtained. If the results could be obtained without sharing information between the data sources, and the results were truly summary and could not be used to deduce private information, there would be no

loss of privacy through data mining. While obtaining globally meaningful results without sharing information may seem impossible, it *can* be done.

The goal of this dissertation is to develop and evaluate new algorithms to efficiently solve several types of distributed computations over large data sets in a secure manner.

Chapter 2 provides an overview of the state of the art in privacy, security and data mining. Chapters 3, 4, and 5 constitute the main work in the thesis. Chapter 3 describes the solutions developed for some of the major data mining problems. Chapter 4 presents solutions for the underlying secure primitives used in the work of the prior chapter. The essential focus of this thesis/dissertation? has been to propose efficient solutions for several data mining problems and to prove them secure. Chapter 5 serves to experimentally validate the claims of efficiency as well as to place them in context. Chapter 6 summarizes the thesis.

Apart from original work, several protocols developed by others are also used in support of the work here. The Appendix provides a brief listing of these protocols for completeness.

2 STATE OF THE ART IN PRIVACY, SECURITY AND DATA MINING

This chapter provides the background material required to give an appropriate perspective for the work done in this thesis. The chapter begins with a short summarization of prevalent data mining algorithms. Section 2.2 covers the state of the art in distributed data mining and also gives some detail on the different data partitioning models. Section 2.3 provides an overview of Secure Multiparty Computation, the theoretical framework we use for proof of security. The final section presents the contemporary work done within Privacy Preserving Data Mining.

2.1 State of the Art in Data Mining Techniques

Data Mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the owner [44]. There are many different data mining functionalities. A brief definition of each of these functionalities is now presented. The definitions are directly collated from [43]. *Data characterization* is the summarization of the general characteristics or features of a target class of data. *Data Discrimination*, on the other hand, is a comparison of the general features of target class data objects with the general features of objects from one or a set of contrasting classes. *Association analysis* is the discovery of association rules showing attribute-value conditions that occur frequently together in a given set of data. *Classification* is the process of finding a set of models (or functions) that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown. The derived model can be represented in various forms, such as *classification rules, decision trees, mathematical formulae*, or *neural networks*. Unlike classification and prediction, which analyze class-labeled

data objects, *clustering* analyzes data objects without consulting a known class label. *Outlier Analysis* attempts to find outliers or anomalies in data. A detailed discussion of these various functionalities can be found in [43]. Even an overview of the representative algorithms developed for knowledge discovery is beyond the scope of this dissertation. The interested person is directed to the many books which amply cover this in detail [29, 43, 44].

2.2 Distributed Data Mining

In contrast to the centralized model, the Distributed Data Mining (DDM) model assumes that the data sources are distributed across multiple sites. Algorithms developed within this field address the problem of efficiently getting the mining results from all the data across these distributed sources. Since the primary (if not only) focus is on efficiency, most of the algorithms developed to date do not take security consideration into account. However, they are still useful in framing the context of the thesis.

A simple approach to data mining over multiple sources that will not share data is to run existing data mining tools at each site independently and combine the results [17, 18, 75]. However, this will often fail to give globally valid results. Issues that cause a disparity between local and global results include:

- Values for a single entity may be split across sources. Data mining at individual sites will be unable to detect cross-site correlations.
- The same item may be duplicated at different sites, and will be over-weighted in the results.
- Data at a single site is likely to be from a homogeneous population. Important geographic or demographic distinctions between that population and others cannot be seen on a single site.

Cheung et al. proposed a method for horizontally partitioned data [21]. Distributed classification has also been addressed. A meta-learning approach has been developed that uses classifiers trained at different sites to develop a global classifier [17,18,75]. This *could* protect the individual entities, but it remains to be shown that the individual classifiers do not disclose private information. Recent work has addressed classification using Bayesian Networks in vertically partitioned data [20], and situations where the distribution is itself interesting with respect to what is learned [90]. Shenoy et al. [85] propose an efficient algorithm for vertically mining association rules.

Data mining algorithms that partition the data into subsets have been developed [82]. Although the goal of parallelizing data mining algorithms is performance, the communication cost between nodes is an issue. Parallel data mining algorithms may also serve as a starting point [51, 93]. However, none of this work directly addresses *privacy* concerns.

With distributed data, the way the data is distributed also plays an important role in defining the problem. Data could be partitioned into many parts either vertically or horizontally.

2.2.1 Vertical Partitioning

Vertical partitioning (a.k.a. heterogeneous distribution) of data implies that though different sites gather information about the same set of entities, they collect different feature sets. For example, financial transaction information is collected by banks, while the IRS collects tax information for everyone. An illustrative example of vertical partitioning and the kind of useful knowledge we can hope to extract is given in Figure 2.1. The figure describes two databases, one contains medical records of people while another contains cell phone information for the same set of people. Mining the joint global database might reveal information like "Cell phones with Li/Ion batteries lead to brain tumors in diabetics."

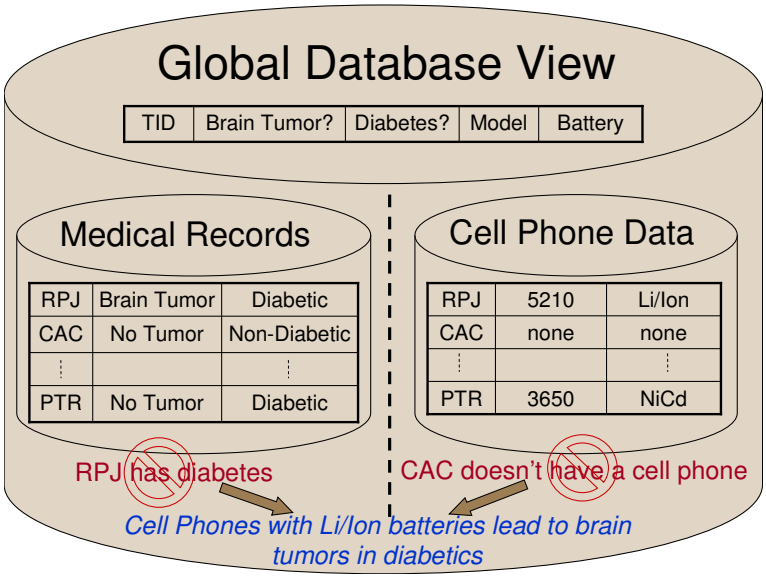


Figure 2.1. Vertically partitioned database

Unless otherwise stated, the model assumed is as follows: There are k parties, $P_0, \ldots, P_{k-1}$. There are a total of n transactions for whom information is collected. Party P_i collects information about m_i attributes, such that $m = \sum_{i=0}^{k-1} m_i$ is the total number of attributes/features. This thesis only considers privacy-preserving data mining in the case of vertical partitioning of data. For the sake of completeness, the following section gives some detail on horizontal partitioning of data.

2.2.2 Horizontal Partitioning

In horizontal partitioning (a.k.a. homogeneous distribution), different sites collect the same set of information, but about different entities. An example of that would be grocery shopping data collected by different supermarkets (also known as market-basket data in the data mining literature). Figure 2.2 illustrates horizontal partitioning and shows the credit card databases of two different (local) credit unions. Taken together, one may find that fraudulent customers often have similar transaction histories, etc.

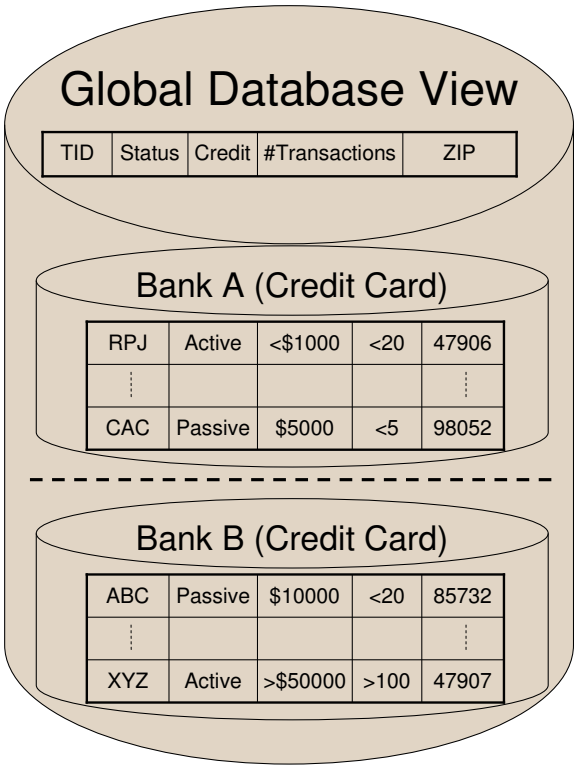


Figure 2.2. Horizontally partitioned database

These different partitionings pose different problems, leading to different algorithms for privacy-preserving data mining.

2.3 State of the Art in Secure Multiparty Computation

Consider a set of parties who do not trust each other, nor the channels by which they communicate. Still, the parties wish to correctly compute some common function of their local inputs, while keeping their local data as private as possible. This, in a nutshell, is the problem of Secure Multiparty Computation (SMC). It is clear that the problem we wish to solve, privacy-preserving data mining, is a special case of the secure multi-party computation problem.

Before proposing algorithms that preserve privacy, it is important to define the notion of privacy. The framework of secure multiparty computation provides a solid

theoretical underpinning for privacy. The key notion is to show that a protocol reveals nothing except the results. This is done by showing how everything seen during the protocol can be simulated from knowing the input and the output of the protocol. Yao first postulated the two-party comparison problem (Yao's Millionaire Protocol) and developed a provably secure solution [92]. This was extended to multiparty computations (for any computable functionality) by Goldreich et al. [40] and to the malicious model of computation by Ben-Or et al. [11]. Overall, a framework was developed for secure multiparty computation. Goldreich [39] shows that computing a function privately is equivalent to computing it securely.

We now cover some of the different models of computation in SMC.

2.3.1 Trusted Third Party Model

The gold standard for security is the assumption that we have a trusted third party to whom we can give all data. The third party performs the computation and delivers only the results – except for the third party, it is clear that nobody learns anything not inferable from its own input and the results. The goal of secure protocols is to reach this same level of privacy preservation, without the (potentially insoluble) problem of finding a third party that everyone trusts.

2.3.2 Semi-honest Model

The Semi-honest model is also known in the literature as the honest-but-curious model. A semi-honest party follows the rules of the protocol using its correct input, but after the protocol is free to use whatever it sees during execution of the protocol to compromise security / privacy.

Two party computation

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 securely. The formal proof of this can be found in [39].

Definition 2.3.1 *(privacy w.r.t. semi-honest behavior) [39]:*

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

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

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

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

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

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

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

Privacy by Simulation 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 given the input and the output of that party. Thus, in all of our proofs of security, we only need to show the existence of a simulator for each party that satisfies the above equations.

This does not quite guarantee that private information is protected. Whatever information can be deduced from the final result obviously cannot be kept private.

For example, consider a secure sum functionality which simply outputs the sum of the local input of the participants. With two parties, the output reveals the input of the other party. This is an artifact of the functionality itself, not of the process of computing it. Thus, this breach of privacy cannot be avoided as long as computing the result is deemed necessary. The key to the definition of privacy is that nothing is learned *beyond* what is inherent in the result.

A key result we use is 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 2.3.1 *(Composition Theorem for the semi-honest model): 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.*

PROOF. Refer to [39]. □

In summary, a truly secure multi-party protocol should not reveal more information to a particular party than the information that can be induced by looking at that party's input and the final output.

Multiparty computation

The above definitions are easily extended to more than two parties.. Details can be found in [39].

2.3.3 Malicious Model

In the malicious model, no restrictions are placed on any of the participants. Thus any party is completely free to indulge in whatever actions it pleases.

Similar definitions of privacy/security exist for both two-party and multi-party computation in this model. The key result of Goldreich is valid even in the case of malicious adversaries. Details can be found in [39].

In general, it is quite difficult to develop *efficient* protocols that are still valid under the malicious model. However, the semi-honest model does not provide sufficient protection for many application. As an intermediate step, it is possible to develop protocols valid under a weakened malicious model – for example, still assuming no collusion between parties, no guarantees on the results if a party is malicious, but a guarantee that there is no disclosure to a malicious party beyond that the party could achieve in the trusted third party model. Other such models are also possible requiring successively stricter conditions.

Despite all of such models, it is still possible for parties to actually modify their inputs to the protocol to begin with. Since this problem exists even with the trusted third party model itself, it is not addressed by cryptography. Other models are needed to address this problem. We discuss one such notion below, which should be used in conjunction with cryptographic approaches.

2.3.4 Other (Partial) Models – Incentive Compatibility

While the semi-honest and malicious models have been well researched in the cryptographic community, other models outside the purview of cryptography are possible. One example is the interesting economic notion of incentive compatibility. A protocol is incentive compatible if it can be shown that a cheating party is either caught or else suffers an economic loss. Under the rational model of economics, this would serve to ensure that parties do not have any advantage by cheating. Of course, in an irrational model (for example – in the case of a monopoly where one party is willing to suffer losses to ensure the loss/elimination of another party), this would not work. Incentive compatibility can be used with methods presented in this dissertation to ensure that complete data mining applications provide adequate privacy protection.

2.4 State of the Art in Privacy Preserving Data Mining Algorithms

The solutions proposed in this thesis rely mostly on cryptographic techniques. However, other techniques have also been used with some success. We now give a brief overview of other techniques as well as note other solutions that have been developed independent of this thesis.

2.4.1 Data Perturbation Techniques

The basic idea of data perturbation is to alter the data so that real individual data values cannot be recovered, while preserving the utility of the data for statistical summaries. Since the data doesn't reflect the real values of private data, even if a data item is linked to an individual that individual's privacy is not violated. (It is important that such data sets are known to be perturbed, so anyone attempting to misuse the data knows the data cannot be trusted.) This approach has been brought to a high art by the U.S. Census Bureau with the Public Use Microdata sets. A primary perturbation technique used is data swapping: exchanging data values between records in ways that preserve certain statistics, but destroy real values [64]. An alternative is randomization: Adding noise to data to prevent discovery of the real values. Since the data no longer reflects real-world values, it cannot be (mis)used to violate individual privacy. The challenge is obtaining valid data mining results from the perturbed data.

In [7], Agrawal and Srikant presented the first solution to this problem. Given the distribution of the noise added to the data, and the randomized data set, they were able to reconstruct the *distribution* (but not actual data values) of the data set. This enabled a data mining algorithm to construct a much more accurate decision tree than mining the randomized data alone, approaching the accuracy of a decision tree constructed on the real data.

Other methods for distribution reconstruction have also been developed. Agrawal and Aggarwal [2] developed an approach based on Expectation Maximization that

also gave a better definition of privacy, and an improved algorithm. Evfimievski et al. [33] applied a similar technique to mine association rules. Rizvi and Haritsa [80] consider the case where different item values (0 and 1) have differing privacy requirements. Polat and Du [74] propose a technique for doing collaborative filtering using randomized perturbation techniques. Solutions for other data mining tasks are certainly feasible. While one will not get the exact same data mining results post-randomization as pre-randomization, the results have been experimentally shown to be accurate enough in the case of both classification [7] and association rule mining [33].

One concern with randomization approaches it that the very techniques that allow us to reconstruct distributions also give information about the original data values. For example, consider the case of perturbing age. It is clear from the general distribution of age that there are no drivers under 16. Assume that it is known that randomization was done by adding noise randomly chosen from the range [-15,15]. Though the reconstructed distribution does not appear to tell us the age of any individual – a driver who is 40 years old is equally likely to have their age given as anything from 25 to 55 – but what about an individual whose age is shown as 1 in the noisy data? We know (from the reconstructed distribution) that no drivers are under the age of 16 – so the driver whose age is given as 1 in the noisy data must be 16 years old! Work has been done to quantify the privacy provided by randomization techniques; they must be used carefully to ensure that the desired privacy is really achieved. Kargupta et al. [52] formally analyze the security of randomization techniques and show that in many cases it falls short of the desired minimum. Evfimievski et al. [32] show how to limit privacy breaches while using randomization for privacy preserving data mining.

2.4.2 Secure Multiparty Computation Based Solutions

There have been some cryptography based algorithms as well. Lindell and Pinkas [59] first introduced a secure multi-party computation technique for classification using the *ID3* algorithm, over horizontally partitioned data. Du and Zhan [28] propose a cryptographic protocol for making the *ID3* algorithm privacy preserving over vertically partitioned data. Lin and Clifton [58] propose a secure way for clustering using the *EM* algorithm [25] over horizontally partitioned data. Kantarcioglu and Clifton describe protocols for privacy preserving distributed data mining of association rules on horizontally partitioned data [48, 50], privately computing distributed top-k queries [49]. Kantarcioglu and Vaidya [47] present an architecture for privacy preserving mining of client information. Agrawal et al. [3] present a technique for computing set intersection, union, and equi-joins for two parties. Clifton et al. provide a good overview of tools for privacy preserving distributed data mining [22], while Clifton and Marks [23] present an early position paper on the privacy implications of data mining.

2.5 Other Work

There has been some other work that does not properly fall into either the perturbation or cryptographic categories. Atallah et. al [8] explore the disclosure limitation of sensitive rules. Saygin et al. [83] present a way of using special values, known as "unknowns", to prevent the discovery of association rules. Oliveira and Zaiane [69–72] develop several different methods for association rule mining, clustering and access control for privacy preserving data mining. There has also been extensive work done in statistical databases. This work is outside the scope of this thesis, however, Adam and Wortmann [1] provide a good starting point. There has also been extensive work in cryptography creating building blocks, which is also outside the scope of this thesis. Many examples can be found in [26].

3 PROBLEMS ADDRESSED

This chapter covers the various data mining problems solved in this thesis. Solutions to the primitives used are presented in the following chapter.

3.1 Two-party Association Rule Mining

This section presents a way to mine association rules from vertically partitioned data. This section presents an algorithm for two parties while the following section presents a general solution for multiple parties. Informally, the problem is to mine association rules across two heterogeneous data sets. One database is designated the *primary*, and is the initiator of the protocol. The other database is the *responder*. There is a *join key* present in both databases. The remaining attributes are present in one database or the other, but not both. The goal is to find association rules involving attributes other than the join key.

3.1.1 Problem Definition

The association rule mining problem can be formally stated as follows [4]: Let $\mathcal{I} = \{i_1, i_2, \cdots, i_m\}$ be a set of literals, called items. Let $\mathcal{D}$ be a set of transactions, where each transaction T is a set of items such that $T \subseteq \mathcal{I}$. Associated with each transaction is a unique identifier, called its *TID*. We say that a transaction T *contains* X, a set of some items in $\mathcal{I}$, if $X \subseteq T$. An *association rule* is an implication of the form, $X \Rightarrow Y$, where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, and $X \cap Y = \phi$. The rule $X \Rightarrow Y$ holds in the transaction set $\mathcal{D}$ with *confidence* c if c% of transactions in $\mathcal{D}$ that contain X also contain Y. The rule $X \Rightarrow Y$ has *support* s in the transaction set $\mathcal{D}$ if s% of transactions in $\mathcal{D}$ contain $X \cup Y$.

Within this framework, we consider mining boolean association rules. The absence or presence of an attribute is represented as a 0 or 1. Transactions are strings of 0 and 1; the database can be represented as a matrix of {0,1}.

3.1.2 Algorithm

The algorithm is based on the classic Apriori algorithm of Agrawal and Srikant [5]. The key issue is computing the support of an itemset. To find out if a particular itemset is frequent, we count the number of records where the values for all the attributes in the itemset are 1. This translates into a simple mathematical problem, given the following definitions:

Let the total number of attributes be $l+m$, where A has l attributes A_1 through A_l, and B has the remaining m attributes B_1 through B_m. Transactions/records are a sequence of $l+m$ 1s or 0s. Let k be the support threshold required, and n be the total number of transaction/records.

Let $\vec{X}$ and $\vec{Y}$ represent columns in the database, i.e., $x_i = 1$ iff row i has value 1 for attribute X. The scalar (or dot) product of two cardinality n vectors $\vec{X}$ and $\vec{Y}$ is defined as

$$\vec{X} \cdot \vec{Y} = \sum_{i=1}^{n} x_i * y_i$$

Determining if the two-itemset $\langle XY \rangle$ is frequent thus reduces to testing if $\vec{X} \cdot \vec{Y} \geq k$.

In Section 4.3 we present an efficient way to compute scalar product $\vec{X} \cdot \vec{Y}$ without either side disclosing its vector. First we will show how to generalize the above protocol from two-itemsets to general association rules without sharing information other than through scalar product computation.

The generalization of this protocol to a w-itemset is straightforward. Assume A has p attributes $a_1 \ldots a_p$ and B has q attributes $b_1 \ldots b_q$, and we want to compute the frequency of the $w = p+q$-itemset $\langle a_1, \ldots, a_p, b_1, \ldots, b_q \rangle$. Each item in $\vec{X}$ ($\vec{Y}$) is composed of the product of the corresponding individual elements, i.e., $x_i = \prod_{j=1}^{p} a_j$ and $y_i = \prod_{j=1}^{q} b_j$. This computes $\vec{X}$ and $\vec{Y}$ without sharing information between

A and B. The scalar product protocol then securely computes the frequency of the entire w-itemset.

For example, suppose we want to compute if a particular 5-itemset is frequent, with A having 2 of the attributes, and B having the remaining 3 attributes. I.e., A and B want to know if the itemset $l = \langle A_a, A_b, B_a, B_b, B_c \rangle$ is frequent. A creates a new vector $\vec{X}$ of cardinality n where $\vec{X} = \vec{A_a} * \vec{A_b}$ (component multiplication) and B creates a new vector $\vec{Y}$ of cardinality n where $\vec{Y} = \vec{B_a} * \vec{B_b} * \vec{B_c}$. Now the scalar product of $\vec{X}$ and $\vec{Y}$ provides the (in)frequency of the itemset.

The complete algorithm to find frequent itemsets is:

1. $L_1 = \{$large 1-itemsets$\}$
2. for (k=2; $L_{k-1} \neq \phi$; k++) do begin
3. C_k = apriori-gen(L_{k-1});
4. for all candidates $c \in C_k$ do begin
5. if all the attributes in c are entirely at A or B
6. that party independently calculates $c.count$
7. else
8. let A have l of the attributes and B have the remaining m attributes
9. construct $\vec{X}$ on A's side and $\vec{Y}$ on B's side where $\vec{X} = \prod_{i=1}^{l} \vec{A_i}$ and $\vec{Y} = \prod_{i=1}^{m} \vec{B_i}$
10. compute $c.count = \vec{X} \cdot \vec{Y} = \sum_{i=1}^{n} x_i * y_i$
11. endif
12. $L_k = L_k \cup c | c.count \geq minsup$
13. end
14. end
15. Answer = $\cup_k L_k$

In step 3, the function apriori-gen takes the set of large itemsets L_{k-1} found in the $(k-1)th$ pass as an argument and generates the set of candidate itemsets C_k. This is done by generating a superset of possible candidate itemsets and pruning this set. [5] discusses the function in detail.

Given the counts and frequent itemsets, we can compute all association rules with $support \geq minsup$.

Only the steps 1, 3, 10 and 12 require sharing information. Since the final result $\cup_k L_k$ is known to both parties, steps 1, 3 and 12 reveal no extra information to either party. Section 4.3 shows how to compute step 10 without revealing information.

3.1.3 Security Analysis

For a complete security analysis of the process, we must first analyze the security of the component scalar product protocol, and then analyze the security of the entire association rule mining algorithm.

The security of the scalar product protocol is based on the inability of either side to solve k equations in more than k unknowns. More details are given in Section 4.3.4.

Therefore, the disclosure risk in this method is based on the number of data values that the other party might know from some external source. The scalar product protocol is used once for every candidate item set. Multiple w-itemsets in the candidate set may be split as 1, $w-1$ on each side. Consider two possible candidate sets A_1, B_1, B_2, B_5 and A_1, B_2, B_3. If A uses new/different equations for each candidate set, it imperils the security of A_1. However, B can reuse the values sent the first time. The equations sent by B can be reused for the same combinations of B_i, only a new sum must be sent. This reveals an additional equation, limiting the number of times B can run the protocol.

3.1.4 Computation and Communication Analysis

The overall computation and communication analysis for the entire association rule mining algorithm hinges on the computation and communication cost of the scalar product protocol. The computation cost of the scalar product protocol is $O(n^2)$ arithmetic operations, which is extremely low. There is no communication

cost for any itemset wholly contained on either side. For every itemset split between the 2 parties, we engage in the scalar product protocol once. As noted earlier, values that have already been sent can be reused. The entire algorithm extends the apriori algorithm. Essentially we provide a means of determining if a candidate itemset is frequent. The communication cost can be expressed in terms of the i/o cost of the apriori algorithm, in fact a constant multiple of the i/o cost of the apriori algorithm.

3.2 Three or More Party Association Rule Mining

An association rule is a simple probabilistic statement about the co-occurrence of certain events in a database, and is particularly applicable to sparse transaction data sets. The general idea of finding association rules originated in applications involving "market-basket data". These data are usually recorded in a database such that each observation consists of an actual basket of items (such as grocery items), and the variables indicate whether or not a particular item was purchased. Association rules were invented as a way to find simple patterns in such data in a relatively efficient computational manner.

We now formally define the association rule mining problem for heterogeneous distribution of data over multiple parties.

3.2.1 Problem Definition

Let there be k (> 2) parties $P_1, P_2, \ldots, P_k$. The database is vertically partitioned between the k parties. The association rule mining problem has already been formally stated in Section 3.1.1. The goal is to find association rules over the attributes across all of the parties.

3.2.2 Algorithm

A transaction database can be represented in several ways. One is to describe it as a $m \times n$ boolean matrix, where m represents the number of items/attributes/features and n represents the number of transactions. A 1 denotes the presence of an item in the transaction, a 0 represents its absence.

A second representation is the transaction identifier (TID) list approach. Every attribute/feature has a transaction identifier (TID) list associated with it. This TID-list contains the identifiers of transactions that contain the attribute. This is a more compact representation in practice, as most transactions are sparse (contain few items). It also lends itself well to secure processing with our size of set intersection protocol.

A clearer explanation of these representations is given by the following example. Consider a database having ten transactions $TID_1, TID_2, \ldots, TID_{10}$, and three attributes A, B, and C. The figure 3.1 illustrates the boolean matrix view and the TID-list view of the database. The figure also gives the TID-list of all combinations of attributes. It is clear that a transaction supports an itemset (set of attributes) if and only if its TID is present in the TID-list for all the attributes in the itemset. To find the number of transactions that support a given itemset we need only find the cardinality of the intersection set of the TID-lists of these attributes.

Sections 4.1 and 4.2 present an efficient way to compute the cardinality of the intersection set without disclosing the items in that set. We now show how to generalize these protocols from a k-itemset where each party has one of the attributes to general association rules. This generalization shares no information other than through computing the size of the set intersections.

We first show how this can be generalized to the case where a single party contributes fewer or more than one attribute to the itemset. A party that has none of the attributes does not participate in the protocol. This reveals that it does not have the attribute, but which attribute is at which site is presumed to be public.

	A	B	C
TID_1	1	0	1
TID_2	0	0	1
TID_3	1	0	0
TID_4	0	1	0
TID_5	0	0	1
TID_6	1	1	1
TID_7	1	1	0
TID_8	0	1	1
TID_9	0	0	0
TID_{10}	1	1	1

A $\{ TID_1, TID_3, TID_6, TID_7, TID_{10} \}$

B $\{ TID_4, TID_6, TID_7, TID_8, TID_{10} \}$

C $\{ TID_1, TID_2, TID_5, TID_6, TID_8, TID_{10} \}$

AB $\{ TID_6, TID_7, TID_{10} \}$

AC $\{ TID_1, TID_6, TID_{10} \}$

BC $\{ TID_6, TID_8, TID_{10} \}$

ABC $\{ TID_6, TID_{10} \}$

2− and 3−Itemsets

Figure 3.1. An example database in its various representations

It is the transactions, and which attributes are in which transaction, that must be kept private. If a party contributes more than one attribute, it locally computes the intersection before participating in the protocol.

For example, suppose we want to compute if a particular 5-itemset is frequent, with

- P_1 having 2 of the attributes, A_{11}, A_{13},
- P_2 having 2 of the attributes, A_{22}, A_{23}, and
- P_3 having the remaining 1 attribute, A_{31}.

P_1, P_2 and P_3 want to know if the itemset $l = \langle A_{11}, A_{13}, A_{22}, A_{23}, A_{31} \rangle$ is frequent. P_1 locally generates the set $S_1 = S_{11} \cap S_{13}$, P_2 locally generates the set $S_2 = S_{22} \cap S_{23}$ and P_3 generates the set $S_3 = S_{31}$. $|S_1 \cap S_2 \cap S_3|$ is the frequency of the itemset.

The full procedure for computing frequent itemsets is given in Algorithm 1. In step 4, the function apriori-gen takes the set of large itemsets L_{k-1} found in the $(k-1)th$ pass as an argument and generates the set of candidate itemsets C_k. This is done by generating a superset of possible candidate itemsets and pruning this set. [5] discusses the function in detail. Given the counts and frequent itemsets, we can compute all association rules with $support \geq minsup$.

Two-itemsets pose a special problem, as the set intersection protocols only work for three or more parties. Assuming we have at least three parties overall, we can use one as an "untrusted third party" to compute the size of 2-itemsets. This is analogous to the leaf actions of Protocol 18 (Section 4.2): the two parties exchange keys, hash their items with both keys, and send the hashed sets to the third party. The third party reports the size of the set if it is greater than the threshold. The same argument as in the proof of Protocol 18 demonstrates the security of this approach.

1: $L_1 = \{\text{large 1-itemsets}\}$
2: **for** $(k = 2; L_{k-1} \neq \emptyset; k++)$ **do**
3: $L_k = \emptyset$
4: $C_k = \text{apriori-gen}(L_{k-1})$;
5: **for all** candidates $c \in C_k$ **do**
6: **if** all the attributes in c are entirely at any one party P_l **then**
7: party P_l independently calculates $c.count$
8: **else**
9: let P_1 have l_1 of the attributes, ..., P_k have l_k attributes $(\sum_{i=1}^{k} l_i = |c|)$
10: construct S_1 on P_1's side, ..., S_k on P_k's side,
11: where $S_i = S_{i1} \cap \ldots \cap S_{il_i},\ 1 \leq i \leq k$
12: compute $c.count = |\cap_{j=1..k} S_j|$ using Protocol 17 or 18
13: **end if**
14: $L_k = L_k \cup c | c.count \geq minsup$
15: **end for**
16: **end for**
17: Answer $= \cup_k L_k$

Algorithm 1: Privacy Preserving Association Rule Mining Algorithm

3.2.3 Proof of Correctness

Candidate itemsets are generated by a straightforward application of the Apriori-gen procedure. For the proof of correctness of that procedure refer to [6]. As long as the input to the procedure is correct, the C_k sets are generated correctly.

We show by induction that the L_j sets are generated correctly. For the basis step with $j = 1$, L_1 is correctly generated directly from the data. Assume that L_{k-1} has been correctly generated. Hence C_k is generated correctly from L_{k-1}. Assuming that step 12 computes the count correctly, L_k is correctly computed.

The critical step is computing *c.count*, step 12. The correctness of Protocols 17 and 18 is given in section 4.1 and section 4.2. Thus, the entire association rule mining algorithm gives correct results.

3.2.4 Computation and Communication Analysis

The communication analysis critically depends on the number of times step 12 is called. For each call to step 12, we incur the cost of the set intersection algorithm. If we let r be the maximum size of a frequent itemset (i.e., no frequent $r+1$-itemsets are found), and let $C_i, (i = 1 \ldots r)$ represent the number of candidate itemsets at each round, the total communication using Protocol 18 is:

$$\sum_{i=1}^{r} C_i * (3k - 4) \quad \text{messages}$$

$$\sum_{i=1}^{r} C_i * 2(k-1) * (m * \textit{hashed item size} + k \log_2(k) * \textit{hash key size}) \quad \text{bits}$$

Surprisingly, a much more efficient association rule mining can be constructed using Protocol 17. The goal of association rule mining is to find all frequent itemsets. If we use Protocol 17 to immediately find the size of $|\cap_{i=1..k} S_k|$, each party also has enough information to find the intersection sizes, or support, of all smaller itemsets that do not include its own items. The "flaw" in Protocol 17 becomes a benefit. Each party compute all frequent itemsets based on the hashed sets received in the initial intersection stage. If any of the frequent itemsets involve its neighbor's set, the corresponding hashed sets are forwarded to its neighbor. This gives every site

complete information, but still prevents probing. The new association rule mining algorithm can be defined as follows:

Call Protocol 17, with each attribute treated as a different site (even if both are at the same site.)

Perform local association rule mining on the TS_j to determine frequent itemsets not involving ones own attributes

if Any frequent itemsets involve ones left or right neighbor **then**

Send the support of those itemsets to the neighbor

end if

Since the only action carried out involves calling the set intersection protocol, the security analysis of this algorithm is completely reducible to the security of the set intersection protocol. The one caveat is that each site learns the support of itemsets in which it does not participate, even if they are below the support threshold. This is less secure than Algorithm 1, but still does not reveal individual itemsets or allow probing.

Since only one call to the set intersection algorithm takes place, the communication analysis is straightforward. Note that all the attributes that are frequent 1-itemsets are sent for intersection. No local intersection is carried out. The communication cost is therefore

$$\begin{array}{ll} k * (2k-2) = O(k^2) & \text{messages} \\ K * (2K-2) * m * \textit{encrypted item size} = O(K^2 m) & \text{bits} \\ k & \text{rounds} \end{array}$$

where K is the number of frequent attributes rather than the number of sites.

It is important to note that this communication cost is dependent entirely upon the number of attributes, rather, a constant times the size of the database. It is independent of the number of iterations of the Apriori [5] algorithm. This is likely to lead to huge cost savings in extremely high dimensional, high transactional databases.

The computational complexity can be easily defined in terms of the computational complexity of the underlying size of set intersection algorithm. Since, the size of set intersection algorithm is only called once, the overhead in computational complexity beyond that of Apriori is exactly the cost of a single invocation of the set intersection algorithm. This cost can be found in section 4.1.3 and section 4.2.3.

3.2.5 Security Analysis

Only steps 1, 12 and 14 require exchanging information. Since the final result $\cup_k L_k$ is known to both parties, steps 1 and 14 reveal no additional information. In Sections 4.1 and 4.2, we show how to compute step 12 revealing only limited information.

Theorem 3.2.1 *Algorithm 1 privately computes the association rules present in the database without revealing any information other than the support of all the possible itemsets.*

PROOF. We use the basic idea of proof by simulation described in Section 2.3. The key idea is to show that everything seen in the protocol can be simulated by knowing just the input and the output. The security of the protocol is based upon the security of steps 1, 12 and 14. The final result $\cup_k L_k$ is known to both parties, so steps 1 and 14 can be simulated directly from the result.

Step 12 consists of an invocation of a protocol to compute the size of set intersection (Protocol 17 or 18). The security discussion of those protocols can be found in Sections 4.1.4 and 4.2.4. By appropriately setting the parameter r to *minsupport* causes the protocol to abort if the itemset is not supported, so we only learn the support of supported itemsets. Apart from giving the final count, Protocols 17 and 18 also reveal the sizes of some (or all) subsets of the itemset being tested (further details can be found in Sections 4.1.4 and 4.2.4). However, a candidate k-itemset is only generated if all its subsets are frequent. The result $\cup_k L_k$ of Algorithm 1 includes all frequent subsets. If we also include the support of the supported itemsets in the

result, as is commonly done in association rule mining, the size of these frequent subsets is also known. These sizes can be provided as input to the simulator to prove the security of Protocols 17 and 18.

Now, we use the composition theorem 2.3.1 to prove the overall security of the algorithm. Treating Protocols 17 and 18 as f, since their security is proven, we show that the association rule computation, g, is computed privately.

This demonstrates that the association rule mining algorithm (Algorithm 1) is fully secure under the Secure Multiparty Computation definitions. □

3.3 k-means Clustering

Cluster Analysis is the problem of decomposing or partitioning a (usually multivariate) data set into groups so that the points in one group are similar to each other and are as different as possible from the points in other groups [44]. There are many situations where clustering can lead to the discovery of important knowledge but privacy/security reasons restrict the sharing of data.

Imagine the following scenario. A law enforcement agency wants to cluster individuals based on their financial transactions, and study the differences between the clusters and known money laundering operations. Knowing the differences and similarities between normal individuals and known money launderers would enable better direction of investigations. Currently, an individual's financial transactions may be divided between banks, credit card companies, tax collection agencies, etc. Each of these (presumably) has effective controls governing release of the information. These controls are not perfect, but violating them (either technologically or through insider misuse) reveals only a subset of an individual's financial records. The law enforcement agency could promise to provide effective controls, but now overcoming them gives access to an individual's entire financial history. This raises justifiable concerns among privacy advocates. What is required is a privacy preserving way of doing clustering.

We focus on k-means clustering [29,38] which is a simple technique to group items into k clusters. k-means clustering is an iterative algorithm, which starts off with random cluster centers. A single iteration assigns all objects to the closest clusters based on their distances from the cluster means and then recomputes the cluster means. Iterations are repeated until the algorithm converges to a set of stable clusters. The basic k-means clustering algorithm is given below:

Initialize the k means $\mu_1 \dots \mu_k$ to 0.
Arbitrarily select k starting points $\mu'_1 \dots \mu'_k$
repeat
  Assign $\mu'_1 \dots \mu'_k$ to $\mu_1 \dots \mu_k$ respectively
  for all points i **do**
    Assign point i to cluster j if distance $d(i, \mu_j)$ is the minimum over all j.
  end for
  Calculate new means $\mu'_1 \dots \mu'_k$.
until the difference between $\mu_1 \dots \mu_k$ and $\mu'_1 \dots \mu'_k$ is acceptably low.

The results come in two forms: Assignment of entities to clusters, and the cluster centers themselves. We assume that the cluster centers μ_i are semiprivate information, i.e., each site can learn only the components of μ that correspond to the attributes it holds. Thus, all information about a site's attributes (not just individual values) is kept private; if sharing the μ is desired, an evaluation of privacy/secrecy concerns can be performed after the values are known.

At first glance, this might appear simple – each site can simply run the k-means algorithm on its own data. This would preserve complete privacy. Figure 3.2 shows why this will not work. Assume we want to perform 2-means clustering on the data in the figure. From y's point of view (looking solely at the vertical axis), it appears that there are two clusters centered at about 2 and 5.5. However in two dimensions it is clear that the difference in the horizontal axis dominates. The clusters are actually "left" and "right", with both having a mean in the y dimension of about 3. The problem is exacerbated by higher dimensionality.

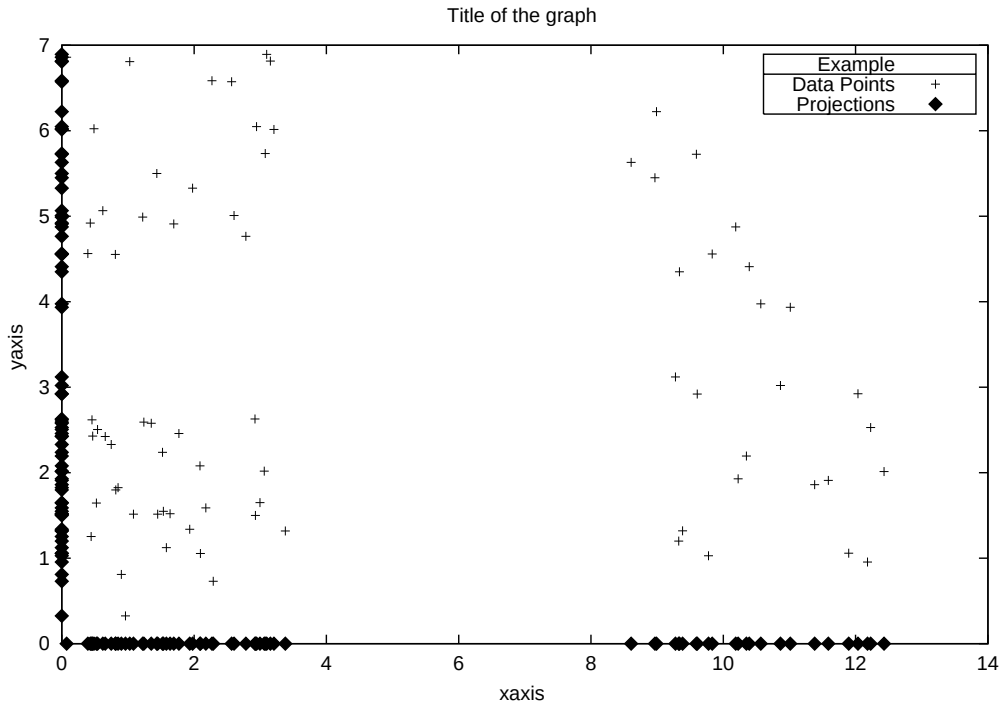


Figure 3.2. Two dimensional problem that cannot be decomposed into two one-dimensional problems

3.3.1 Basic Approach

Given a mapping of points to clusters, each site can independently compute the components of μ_i corresponding to its attributes. Assigning points to clusters, specifically computing which cluster gives the minimum $d(i, \mu_j)$, requires cooperation between the sites. We show how to privately compute this in Section 3.3.2. Briefly, the idea is that site A generates a (different) vector (of length k) for every site (including itself) such that the vector sum of all the site vectors is $\vec{0}$. Each site adds its local differences $|point - \mu_i|$ to its vector. At the same time, the vector is permuted in an order known only to A. Each site (except a single holdout) sends their permuted vector to site B. Site B sums the received vectors, then the holdout site and B perform a series of secure additions and comparisons to find the minimum

i without learning distances. B now asks A the real index corresponding to i, giving the proper cluster for the point.

The second problem is knowing when to quit, i.e., when the difference between μ and μ' is small enough; we show how to privately compute this in Algorithm 3. This makes use of secure sum and secure comparison, described on Page 36.

3.3.2 Algorithm

First, we formally define the problem. Let r be the number of parties, each having different attributes for the same set of entities. n is the number of the common entities. The parties wish to cluster their *joint* data using the k-means algorithm. Let k be the number of clusters required.

The final result of the k-means clustering algorithm is the value/position of the means of the k clusters, with each side only knowing the means corresponding to their own attributes, and the final assignment of entities to clusters. Let each cluster mean be represented as $\mu_i, i = 1, \ldots, k$. Let μ_{ij} represent the projection of the mean of cluster i on party j. Thus, the final result for party j is

- the final value/position of $\mu_{ij}, i = 1 \ldots k$
- cluster assignments: $clust_i$ for all points $(i = 1, \ldots, n)$

The k-means algorithm also requires an initial assignment (approximation) for the values/positions of the k means. This is an important issue, as the choice of initial points determines the final solution. Research has led to mechanisms producing a good initial assignment [15]. Their technique uses classic k-means clustering done over multiple subsamples of the data, followed by clustering the results to get the initial points. For simplicity, we assume that the k means are selected arbitrarily. Since the underlying operations in [15] involve k-means clustering, it is possible to extend our algorithm to search for and start off with good initial means.

Thus, for $i = 1 \ldots k$, every party selects its share μ'_{ij} of any given mean. This value is local to each party and is unknown to the other parties.

The basic algorithm directly follows the standard k-means algorithm. The approximations to the true means are iteratively refined until the improvement in one iteration is below a threshold. At each iteration, every point is assigned to the proper cluster, i.e., we securely find the cluster with the minimum distance for each point (this is described in Section 3.3.2.) Once these mappings are known, the local components of each cluster mean can be computed locally. We then use Algorithm 3 (*checkThreshold*) to test termination: was the improvement to the mean approximation in that iteration below a threshold? This is shown formally in Algorithm 2.

The *checkThreshold* algorithm (Algorithm 3) is straightforward, except that to maintain security (and practicality) all arithmetic must be mod$n$. This results in a non-obvious threshold evaluation at the end, consisting of a secure addition / comparison. *Intervals* are compared rather than the actual numbers. Since $Th < n/2$ and the domain of $-_D < n/2$, if the result of $m - Th'$ is positive, it will be less than $n/2$, and if it is negative, due to the modulo operation, it will be greater than $n/2$. Thus, $m - Th' > Th' - m(modn)$ if and only if $m < Th'$, and the correct result is returned.

Securely Finding the Closest Cluster

This algorithm is used as a subroutine in the k-means clustering algorithm to privately find the cluster which is closest to the given point, i.e., which cluster should a point be assigned to. Thus, the algorithm is invoked for every single data point in each iteration. Each party has as its input the component of the distance corresponding to each of the k clusters. This is equivalent to having a matrix of distances of dimension $k \times r$. For common distance metrics; such as Euclidean, Manhattan, or any other Minkowski; this translates to finding the cluster where the sum of the local distances is the minimum among all the clusters.

Require: r parties, k clusters, n points.

1: **for all** sites $j = 1 \ldots r$ **do**
2: **for all** clusters $i = 1 \ldots k$ **do**
3: initialize μ'_{ij} arbitrarily
4: **end for**
5: **end for**
6: **repeat**
7: **for all** $j = 1 \ldots r$ **do**
8: **for** $i = 1 \ldots k$ **do**
9: $\mu_{ij} \leftarrow \mu'_{ij}$
10: $Cluster[i] = \emptyset$
11: **end for**
12: **end for**
13: **for** $g = 1 \ldots n$ **do**
14: **for all** $j = 1 \ldots r$ **do**
15: {Compute the distance vector $\vec{X_j}$ (to each cluster) for point g.}
16: **for** $i = 1 \ldots k$ **do**
17: $x_{ij} = |data_{gj} -_D \mu_{ij}|$
18: **end for**
19: **end for**
20: Each site puts g into $Cluster[closest\_cluster]$ {$closest\_cluster$ is Algorithm 4}
21: **end for**
22: **for all** $j = 1 \ldots r$ **do**
23: **for** $i = 1 \ldots k$ **do**
24: $\mu'_{ij} \leftarrow$ mean of j's attributes for points in $Cluster[i]$
25: **end for**
26: **end for**
27: **until** *checkThreshold* {Algorithm 3}

Algorithm 2: Privacy Preserving k-means clustering

Require: Th is a threshold for termination, Random number generator *rand* produces values uniformly distributed over $0..n-1$ spanning (at least) twice the domain of the distance function $-_D$.

1: **for all** $j = 1 \ldots r$ **do**
2: $d_j \leftarrow 0$
3: **for** $i = 1 \ldots k$ **do**
4: $d_j \leftarrow d_j + |\mu'_{ij} -_D \mu_{ij}|$
5: **end for**
6: **end for**
7: {Securely compute if $\sum d_j \leq Th$.}
8: At P_1: $m = \text{rand}()$
9: **for** j=1 ... r-1 **do**
10: P_i sends $m + d_j \pmod n$ to P_{j+1}
11: **end for**
12: At P_r: $m = m + d_r$
13: At P_1: $Th' = Th + r$
14: P_1 and P_r return *secure_add_and_compare*$(m - Th' \pmod n > Th' - m \pmod n)$ {Secure comparison is described on Page 36.}

Algorithm 3: checkThreshold: Find out if the new means are sufficiently close to old means

The problem is formally defined as follows. Consider r parties $P_1, \ldots, P_r$, each with their own k-element vector $\vec{X_i}$:

$$P_1 \text{ has } \vec{X_1} = \begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{k1} \end{bmatrix}, P_2 \text{ has } \begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{k2} \end{bmatrix}, \ldots, P_r \text{ has } \begin{bmatrix} x_{1r} \\ x_{2r} \\ \vdots \\ x_{kr} \end{bmatrix}.$$

The goal is to compute the index l that represents the row with the minimum sum. Formally, find

$$\underset{i=1..k}{argmin}(\sum_{j=1..r} x_{ij})$$

For use in k-means clustering, $x_{ij} = |\mu_{ij} - point_j|$, or site P_j's component of the distance between a point and the cluster i with mean μ_i.

The security of the algorithm is based on three key ideas.

1. Disguise the site components of the distance with random values that cancel out when combined.

2. Compare distances so only the comparison result is learned; no party knows the distances being compared.

3. Permute the order of clusters so the real meaning of the comparison results is unknown.

The algorithm also requires three non-colluding sites. These parties may be among the parties holding data, but could be external as well. They need only know the number of sites r and the number of clusters k. Assuming they do not collude with each other, they learn nothing from the algorithm. For simplicity of presentation, we will assume the non-colluding sites are P_1, P_2, and P_r among the data holders. Using external sites, instead of participating sites P_1, P_2 and P_r, to be the non-colluding sites, is trivial.

The algorithm proceeds as follows. Site P_1 generates a length k random vector $\vec{V_i}$ for each site i, such that $\sum_{i=1}^{r} \vec{V_i} = \vec{0}$. P_1 also chooses a permutation π of $1..k$.

P_1 then engages each site P_i in the permutation algorithm [27] (see Section A.1.1) to generate the sum of the vector $\vec{V_i}$ and P_i's distances $\vec{X_i}$. The resulting vector is known only to P_i, and is permuted by π known only to P_1, i.e., P_i has $\pi(\vec{V_i} + \vec{X_i})$, but does not know π or $\vec{V_i}$. P_1 and $P_3 \ldots P_{r-1}$ send their vectors to P_r.

Sites P_2 and P_r now engage in a series of secure addition / comparisons to find the (permuted) index of the minimum distance. Specifically, they want to find if $\sum_{i=1}^{r} x_{li} + v_{li} < \sum_{i=1}^{r} x_{mi} + v_{mi}$. Since $\forall l, \sum_{i=1}^{r} v_{li} = 0$, the result is $\sum_{i=1}^{r} x_{li} < \sum_{i=1}^{r} x_{mi}$, showing which cluster (l or m) is closest to the point. P_r has all components of the sum except $\vec{X_2} + \vec{V_2}$. For each comparison, we use a secure circuit evaluation (see Page 36) that calculates $a_2 + a_r < b_2 + b_r$, without disclosing anything but the comparison result. After $k - 1$ such comparisons, keeping the minimum each time, the minimum cluster is known.

P_2 and P_r now know the minimum cluster in the permutation π. They do not know the real cluster it corresponds to (or the cluster that corresponds to any of the others items in the comparisons.) For this, they send the minimum i back to site P_1. P_1 broadcasts the result $\pi^{-1}(i)$, the proper cluster for the point.

The full algorithm is given in Algorithm 4. Several optimizations are possible, we discuss these when analyzing the complexity of the algorithm in Section 3.3.5.

Section A.1.1 describes the permutation algorithm. We now describe the secure addition comparison, which builds a circuit that has two inputs from each party, sums the first input of both parties and the second input of both parties, and returns the result of comparing the two sums. This (simple) circuit is evaluated securely using the generic algorithm described on Page 36. We then prove the security of the method. A graphical depiction of stages 1 and 2 is given in Figures 3.3 and 3.4.

Secure Comparison

We simply use the generic circuit based evaluation approach for the addition and comparison function. The function is first represented as a combinatorial circuit, and

Require: r parties, each with a length k vector $\vec{X}$ of distances. Three of these parties (trusted not to collude) are labeled P_1, P_2, and P_r.

1: {Stage 1: Between P_1 and all other parties}
2: P_1 generates r random vectors $\vec{V_i}$ summing to $\vec{0}$ (see Algorithm 5).
3: P_1 generates a random permutation π over k elements
4: **for all** $i = 2 \ldots r$ **do**
5: $\vec{T_i}$ (at P_i) = *add_and_permute*$(\vec{V_i}, \pi(\text{at } P_1), \vec{X_i}(\text{at } P_i))$ {This is the permutation algorithm described in Section A.1.1}
6: **end for**
7: P_1 computes $\vec{T_1} = \pi(\vec{X_1} + \vec{V_1})$
8:
9: {Stage 2: Between all but P_2 and P_r}
10: **for all** $i = 1, 3 \ldots r - 1$ **do**
11: P_i sends $\vec{T_i}$ to P_r
12: **end for**
13: P_r computes $\vec{Y} = \vec{T_1} + \sum_{i=3}^{r} \vec{T_i}$
14:
15: {Stage 3: Involves only P_2 and P_r}
16: $minimal \leftarrow 1$
17: **for** j=2..k **do**
18: **if** *secure_add_and_compare*$(Y_j + T_{2j} < Y_{minimal} + T_{2minimal})$ **then**
19: $minimal \leftarrow j$
20: **end if**
21: **end for**
22:
23: {Stage 4: Between P_r (or P_2) and P_1}
24: Party P_r sends *minimal* to P_1
25: P_1 broadcasts the result $\pi^{-1}(minimal)$

Algorithm 4: *closest_cluster*: Find minimum distance cluster

Require: Random number generator *rand* producing values uniformly distributed over $0..n-1$ spanning (at least) the domain of the distance function $-_D$.

Ensure: The sum of the resulting vectors is $\vec{0}$.

1: **for all** i = 1 ... k **do**
2: $PartSum_i \leftarrow 0$
3: **for** j = 2 ... r **do**
4: $V_{ij} \leftarrow rand()$
5: $PartSum_i \leftarrow PartSum_i + V_{ij} \pmod n$
6: **end for**
7: $V_{i1} \leftarrow -PartSum_i \pmod n$
8: **end for**

Algorithm 5: genRandom: Generates a (somewhat) random matrix $V_{k\times r}$

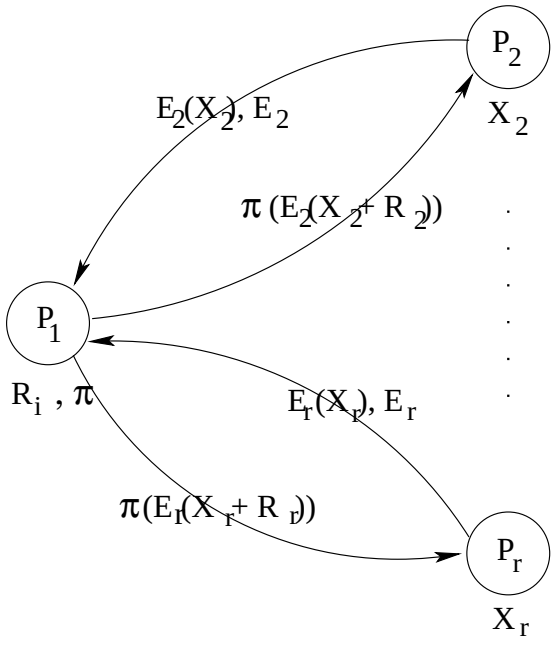


Figure 3.3. Closest cluster – stage 1

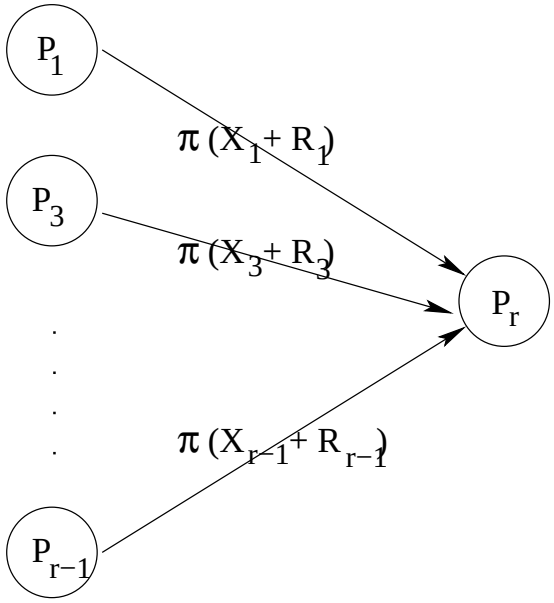


Figure 3.4. Closest cluster – stage 2

then the parties run a short protocol for every gate in the circuit. Every participant gets (randomly selected) shares of the input wires and the output wires for every gate. Since determining which share goes to which party is done randomly, a party's own share tells it nothing. Upon completion, the parties exchange their shares, enabling each to compute the final result.

While impractical for large inputs and many parties, for a limited number of simple two-party operations, such as the *secure_add_and_compare* function used in Algorithms 3 and 4, the complexity is reasonable. For two parties, the message cost is $O(circuit\_size)$, and the number of rounds is constant. We can add and compare numbers with $O(m = \log(number\_of\_entities))$ bits using an $O(m)$ size circuit.

3.3.3 Security Discussion

Closest Cluster Computation Algorithm 4 returns the index of the closest cluster (i.e., the row with the minimum row sum). To prove this algorithm is privacy preserving, we must show that each party can construct a polynomial time simulator for the view that it sees, given only its own input and this closest cluster index.

Theorem 3.3.1 *Algorithm 4 privately computes the index of the row with the minimum row sum, revealing only this result assuming parties do not collude to expose other information.*

PROOF. The simulator is constructed in stages, corresponding to the stages of the algorithm.

Stage 1: The only communication occurring in this stage occurs in the $r - 1$ calls to the Permutation Algorithm. Thus, we simply need to apply the composition theorem stated in Theorem 2.3.1, with g being the closest cluster computation algorithm and f being the permutation algorithm. What remains is to show that we can simulate the result T_i. The simulator for P_1 is exactly the algorithm used by P_1, without sending any data. For the remaining sites, since the v_i are unknown and

chosen from a uniform distribution on $(0..n-1)$, $v_i + x_i$ will also form a uniform distribution on $(0..n-1)$. Each $P_i, i = 2 \ldots r$ can simulate the vector $\vec{T_i}$ by selecting values randomly from a uniform distribution on $(0..n-1)$. This is indistinguishable from what it sees in the algorithm.

Stage 2: All the parties other than P_2 and P_r send their permuted result vectors to the receiver. Since only P_r sees new information, we need only concern ourselves with simulating what it sees. The received vectors can be simulated by P_r exactly as they were simulated by the P_i in Stage 1. The vector $\vec{Y}$ is equal to the actual distances minus $\vec{T_2}$. However, since $\vec{T_2}$ consists of values uniformly distributed over $(0..n-1)$, $\vec{Y}$ is effectively $distances - v$, and is thus also uniformly distributed over $(0..n-1)$. However, we cannot simulate it by generating random values, as we must preserve the relationship $\vec{Y} = \vec{T_1} + \sum_{j=3}^{r} T_j \pmod{n}$. Fortunately, the sum of the simulator-generated $\vec{T_i}$ will give a vector $\vec{Y}$ that both meets this constraint and is uniformly distributed over $(0..n-1)$, giving a view that is indistinguishable from the real algorithm.

Stage 3: Here P_2 and P_r engage in a series of comparisons. Again, we use the composition theorem. Each comparison is secure, so we need only show that we can simulate the sequence of comparison results.

The simulator uniformly chooses a random ordering of the k clusters from the $k!$ possible orderings. We regard this as the distance-wise ordering of the clusters relative to the point. This ordering is used to choose the appropriate result, $\leq$ or $>$, for each comparison. Effectively, the simulator runs steps 17-21, but makes the comparisons locally based on the random ordering. The probability of any given ordering is $1/k!$, the same as the probability of any given ordering achieved after the permutation π in the actual view. Therefore, the probability of any given sequence of comparison results is the same under the ordering as under the view seen in the actual algorithm.

Note that all of the possible $2^{(k-1)}$ sequences are not equally likely, e.g., the sequence of all >s corresponds to only *one* ordering, while the sequence of all $\leq$s corresponds to $(k-1)!$ orderings. However, selecting random total orderings generates sequences matching the (non-uniform) probability distribution of the actual sequences of comparisons.

Stage 4: P_r (or P_2) sends the index i to P_1. Since the true index i_t is the final result known to all the parties, and P_1 decides upon the permutation π, the simulator generates $\pi(i_t) = i$ as the message it receives.

The final result i_t is sent to all parties. Since this is the final result, obviously all the parties can simulate it.

Since this simulator is also linear in the size of the input, and we have proven the permutation algorithm to be secure, application of the composition theorem proves that Algorithm 4 preserves privacy. □

Stopping Criterion Before analyzing the security of the entire k-means algorithm, we prove the security of the threshold checking Algorithm 3.

Theorem 3.3.2 *Algorithm 3 determines if* $\sum |\mu'_{ij} -_D \mu_{ij}| < Th|$*, revealing nothing except the truth of this statement.*

PROOF. Steps 10 and 14 are the only steps of Algorithm 3 requiring communication, so the simulator runs the algorithm to this point. In step 10, party P_1 first sends $m + d_1 \pmod n$ where m is the random number known to P_1. Each of the parties $P_j, j = 2 \ldots r$ receive a message $m + \sum_{j=1}^{i} d_j$ from their left neighbor. Since m is chosen from a uniform distribution on $(0 \ldots n-1)$, and all arithmetic is $\bmod n$, this sum forms a uniform distribution on $(0 \ldots n-1)$ and can be simulated by generating a random number over that distribution:

$$Pr\left[VIEW_j^{\text{Algorithm 3 Step 10}} = x\right] \quad = \quad Pr\left[m + \sum_{i=1}^{j} d_i = x\right]$$

$$
\begin{aligned}
&= Pr\left[m = x - \sum_{i=1}^{j} d_i\right] \\
&= \frac{1}{n} \\
&= Pr\left[Simulator_j = x\right]
\end{aligned}
$$

The *secure_add_and_compare* algorithm gives only the final result: $m - Th' \pmod n > Th' - m \pmod n = \sum_{j=1}^{r} d_j \leq Th$. Step 14 is easily simulated knowing that result.

This simulator runs in the $O(k)$ time required by the Algorithm, and is thus polynomial. Applying the composition theorem with Algorithm 3 as f and taking g to be the *secure_add_and_compare* algorithm, along with the other facts given above, proves that Algorithm 3 is secure. □

Overall k-means algorithm We now analyze the security of the entire k-means algorithm. In every iteration, the following things are revealed to the parties:

- Each party's local share of the k cluster means.
- The cluster assignment for every point.

These values are the desired result of the final iteration. Since it is impossible to know in advance the number of iterations required to halt, the number of iterations needs to be accepted as part of the final output. The results from the intermediate iterations *may* be used to infer information beyond this result. For example, if the cluster centers for site j do not change between iterations, and a point moves between two clusters, site j knows that those two clusters are both relatively close to the point across the sum of the other sites. However, since the location of the point in the other dimensions is not known, this information is of little use. In any iteration the final assignment of points to clusters is the same for every party. If this intermediate assignment should not be revealed, either a genuine third party will be required or else the algorithm will be quite inefficient. Allowing the intermediate results to be accepted as part of overall results allows an efficient algorithm with provable

security properties. Forbidding knowledge of intermediate results would prevent each site from computing the next iteration locally, making the entire computation much more expensive.

We therefore state the proven overall security properties in the following theorem.

Theorem 3.3.3 *Algorithm 2 is a private algorithm computing the k clusters of the combined data set, revealing at most the point assignment to clusters at each iteration and the number of iterations required to converge.*

PROOF. All of the communication in Algorithm 2 all occurs in the calls to Algorithms 4 and 3. The results of Algorithm 4 are point assignments to clusters, and can be simulated from the known result for that iteration. The results of Algorithm 3 are easily simulated; for all but the final iteration it returns false, in the final iteration it returns true. Applying the composition theorem shows that within the defined bounds the k-means algorithm is secure. □

3.3.4 Handling Collusion

Parties P_1, and P_r have more information than the others during the execution of the above algorithm. Specifically, P_1 knows

1. the permutation π, and
2. the values of the random splits (i.e., the random matrix $V_{k\times r}$).

P_r learns

1. the permuted result vectors of the permutation algorithm ($\vec{T_i}$) for all the parties other than P_2, and
2. the comparison results.

(Note that P_2 also learns the comparison results.) While we have proven that this information is meaningless *in isolation*, collusion between P_1 and P_r provides enough

information to derive the distances between each point and each party's means. It is necessary to carefully select these two parties so that all parties are confident the two will not collude.

The assumption of non-collusion is often implicitly made in the real world. For example, take the case of lawyers for parties on opposite sides in court. While no *technical* means prevent collusion, safeguards exist in the form of severe punishments for breaking this rule as well as the business penalty of lost reputation. Similar legal and reputation safeguards could be enforced for privacy-preserving data mining. In addition, if there were not at least two parties who did not want to share information, there would be no need for a secure algorithm. Since collusion between P_1 and P_r reveals P_1's information to P_r, P_1 would be unlikely to collude simply out of self-interest.

However, technical solutions are more satisfying. Let p, $1 \leq p \leq r-1$, be a user defined anti-collusion security parameter. We present a modification of the algorithm that guarantees that at least $p+1$ parties need to collude to disclose additional information. The problem is in Algorithm 4. The key idea is that stage 1 is run p times, each time selecting a new party to act as P_1. Thus, the permutation π and the random matrix $V_{k\times r}$ is different for every run, however the row sum of each V matrix is $\vec{0}$, so the total sum is still the actual distance. In stage 4, to get the true index from the permuted index, the p parties apply their inverse permutations in order. Thus, the true index is $\pi_1^{-1}(\pi_2^{-1}(\ldots(\pi_p^{-1}(i'))\ldots))$.

3.3.5 Communication Analysis

We give a bottom-up analysis of the communication cost of one iteration of the algorithm. The total cost is dependent on the number of iterations required to converge, which is dependent on the data. Assume r parties, n data elements, and that encrypted distances can be represented in m bits.

The permutation algorithm requires only two rounds of communication. For length-n vectors, the total bit cost is $2n * m + public\_key\_size = O(n)$ bits.

The *secure_add_and_compare* algorithm is a two party protocol, implemented using secure circuit evaluation. There are several general techniques for implementing circuit evaluation that optimize different parameters such as computation cost, communication cost (number of rounds or total number of bits), etc. The basic tool used, one out of two oblivious transfer, can also be implemented in several ways. Methods exist that require a constant number of rounds of communication (by parallelizing the oblivious transfers) with bit communication cost linear in the number of gates in the circuit. An excellent survey is given in [36]. The *secure_add_and_compare* algorithm requires two addition circuits and one comparison circuit, all of $m = \log n$ bits (where n is based on the resolution of the distance). For both addition and comparison the number of gates required is linear in m. Therefore this step requires constant rounds and $O(m)$ bits of communication.

In Algorithm 4, *closest_cluster*, there are several places where communication occurs. Steps $4 - 5$ make $r - 1$ calls to the permutation algorithm with size k vectors. Steps $10 - 11$ require $r - 2$ rounds of communication and $(r - 2) * k * m$ bits. Steps $17 - 18$ use $k - 1$ calls to the *secure_add_and_compare* algorithm. Steps $24 - 25$ require two rounds and $O(r \log k)$ bit cost. Thus the total cost is $2(r - 1) + r - 2 + (k - 1) * const \approx 3r + const * k = O(r + k)$ rounds and $2k * m * (r - 1) + k * m * (r - 2) + (k - 1) * const * (\log n) \approx 3 * m * kr + kc \log n = O(kr)$ bits.

The collusion resistant variant of Section 3.3.4 multiplies the cost of steps $4 - 5$ and step 24 by a factor of p. This gives $O(pr + k)$ rounds and $O(pkr)$ bits.

We now give a communication analysis of Algorithm 3. Step 10 involves $r - 1$ rounds of communication, with bit cost $(r - 1) * m$. Step 14 makes one call to *secure_add_and_compare*, for constant rounds and $O(m)$ bits. Thus, the total cost is $O(r)$ rounds and $O(rm)$ bits.

Finally, we come to the analysis of the entire algorithm. We do not count any setup needed to decide the ordering or role of the parties. One iteration of the k-

means algorithm requires one call to the closest cluster computation for every, point and one call to the *checkThreshold* algorithm. Since all points can be processed in parallel, the total number of rounds required is $O(r + k)$. The bit communication cost is $O(nrk)$.

Optimizations The cost of secure comparisons in Stage 3 of Algorithm 4 can be eliminated with a security compromise that would often be innocuous. First, the random vector generated in step 2 is generated so the rows sum to randomly chosen r instead of 0. In Stage 2, all the parties (including P_2) send their permuted vectors to P_r. Now P_r can independently find the index of the row with the minimum row sum. Thus, the communication cost is $2(r-1) + r - 1 + 2 \approx 3r = O(r)$ rounds and $2k * m * (r-1) + k * m * (r-1) + 2(\log k) \approx 3krm$ bits.

The problem with this approach is that P_r learns the relative distance of a point to each cluster, i.e., it learns that p is 15 units farther from the second nearest cluster than from the cluster it belongs to. It does not know which cluster the second nearest is. Effectively it gets k equations (one for each cluster) in $k + 1$ unknowns. (The unknowns are the location of the point, the location of all clusters but the one it belongs in, and the distance to the closest cluster center.) Since the permutation of clusters is different for each point, as is the random R, combining information from multiples points still does not enable solving to find the exact location of a point or cluster. However, probabilistic estimates on the locations of points/clusters are possible. If the parties are willing to accept this loss of security in exchange for the communication efficiency, they can easily do so.

Let us now compare our communication cost with that of the general circuit evaluation method. For one iteration of the algorithm a circuit evaluation would be required for each point to evaluate the cluster to which the point is assigned. Even with an optimized circuit the closest cluster computation requires at least $r - 1$ addition blocks for each cluster, i.e., approx. kr addition circuits, and $k - 1$ comparison blocks. These blocks are all of width at least m bits. The best known

general method still requires at least r^2 bits of communication for every circuit. Thus, a lower bound on the amount of bits transferred is $O(kmr^3)$ bits.

A simple upper bound on *non*-secure distributed k-means is obtained by having every party send its data to one site. This gives $O(n)$ bits in one round. Privacy is adding a factor of $O(r + k)$ rounds and $O(rk)$ bit communication cost. While this tradeoff may seem expensive, if the alternative is not to perform data mining at all, it seems quite reasonable.

Clustering in the presence of differing scales, variability, correlation and/or outliers can lead to unintuitive results if an inappropriate space is used. Research has developed robust space transformations that permit good clustering in the face of such problems [55]. Such estimators need to be calculated over the entire data. An important extension to our work would be to allow privacy preserving computation of such estimators, giving higher confidence in clustering results. Similarly, extending this work to the more robust EM-clustering algorithm [25, 61] under the heterogeneous database model is a promising future direction. Another problem is to find the set of common entities without revealing the identity of entities that are not common to all parties.

3.4 Naïve Bayes Classification

Privacy-preserving classification on vertically partitioned data has many real-world applications. For example, assume a medical research study wants to compare medical outcomes based on techniques in pharmaceutical manufacturing processes (e.g., to answer the question "are generic drugs really as effective as brand-name", and more important, what manufacturing processes produce the best results?) The insurance companies can't disclose individual patient data without permission [45], and complete manufacturing processes are trade secrets (although individual techniques may be commonly known.) Similar constraints arise in many applications; European Community legal restrictions apply to disclosure of any individual data [30].

Naïve Bayes is a simple but highly effective classifier. This combination of simplicity and effectiveness has lead 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 [63]).

3.4.1 Naïve Bayes Classifier

method. The following description of a Naïve Bayes classifier is based on the discussion in Mitchell [63]. 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 V. 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, v_{MAP}, given the attribute values $< a_1, a_2, \ldots, a_n >$ that describe the instance.

$$v_{MAP} = \underset{v_j \in V}{argmax} \left(P(v_j | a_1, a_2, \ldots, a_n) \right) \tag{3.1}$$

Using Bayes theorem,

$$\begin{aligned} v_{MAP} &= \underset{v_j \in V}{argmax} \left(\frac{P(a_1, a_2, \ldots, a_n | v_j) P(v_j)}{P(a_1, a_2, \ldots, a_n)} \right) \\ &= \underset{v_j \in V}{argmax} \left(P(a_1, a_2, \ldots, a_n | v_j) P(v_j) \right) \end{aligned} \tag{3.2}$$

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

$$v_{NB} = \underset{v_j \in V}{argmax} \left(P(v_j) \prod_i P(a_i | v_j) \right) \tag{3.3}$$

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

The conditional probabilities $P(a_i|v_j)$ need to be estimated from the training set. The prior probabilities $P(v_j)$ also need to be fixed in some fashion (typically by

simply counting the frequencies from the training set). The probabilities for differing hypotheses can be computed by normalizing the values received for each hypothesis. Probabilities are computed differently for nominal and numeric attributes.

Nominal Attributes

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

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

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.4}$$

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 v_j, $P(X = x'|v_j)$, can be estimated by substituting $x = x'$ in equation 3.4.

3.4.2 Model Issues – Splitting of Model Parameters

Since different sites hold different attributes, one issue of particular interest with classification is the location and security properties of the class attribute. We can divide this into two possibilities:

- All the parties hold the (common/public) class attribute, or
- Only a subset of the parties have the (secret) class attribute.

The first case is the simplest, assuming that the class attribute of the training data is known to all parties. In some cases this is reasonable – e.g., manufacturers of

subcomponents collaborating to determine expected failure rates of fully assembled systems based on attributes of the subcomponents. In this case, it is easy to estimate all the required counts for nominal attributes and means and variances for numeric attributes locally, causing no privacy breaches. Prediction can be accomplished by independently estimating the probabilities, and securely multiplying and comparing to obtain the predicted class.

More interesting is the general case, where not all parties have the class attribute. We can simplify this to the basic case where one party has the class attribute and the other has the remaining attributes. Solving this enables us to solve any distribution of attributes. (Extension to more than two parties, or where the party with the class attribute has more information, is straightforward.)

It is also necessary that the model learned not reveal information – the model parameters (probability distribution of classes) would reveal information about the (protected) class values. Instead, we build a model where each party has random shares of the model, and collaborate to classify an instance. The only knowledge gained by either side is the class of each instance classified.

The obvious alternative, generating and sharing the classifier, reveals considerable information about both the attributes and the classes. The relative distribution of classes in the training data is likely to be sensitive, as is the mean/variance or distribution of the attribute values. With our approach, neither party learns anything new until a new instance is classified, and then the only thing learned is the predicted class of that instance. While learning the predicted class of enough instances may allow reverse-engineering the classifier, this is unavoidable given that the goal is to learn the classes of the test data. In addition, if either party feels too much is being revealed, they can simply dispose of their share of the classifier to ensure no more of their information is disclosed. Also, it is possible to use the protocols developed such that the class of each instance is learned only by the party holding the class attribute (nothing is learned by the remaining parties). In some cases, this might be preferable.

Having both parties (the data site and the class site) hold shares of all the model parameters complicates the evaluation of a new instance. Classifying a new instance is no longer a straightforward task and a joint protocol is required to classify any new instance. The method to do this is given in Section 3.4.4.

3.4.3 Building the Classifier Model

The basic idea behind our protocol is that each party ends up with shares of the conditionally independent probabilities that constitute the parameters of a Naïve Bayes classifier. By themselves, the shares appear random – only when added do they have meaning. This addition only occurs as part of evaluating the classifier on an instance – and the protocol that does this reveals only the class of the instance.

We start with computing the shares of the parameters. For nominal attributes, the parameters are $P(x_i|v_l) = n_i/n$ for each class i and attribute value l. For Numeric attributes, we need the mean and variance for the probability density function given in Equation 3.4.

Nominal Attributes

Party P_d holds the nominal attribute D, while party P_c holds the class attribute C. D has r possible values, $a_1, \ldots, a_r$. C has k possible class values $v_1, \ldots, v_k$. The goal is to compute $r \times k$ matrices S^c, S^d where the sum of corresponding entries $s^c_{li} + s^d_{li}$ gives the probability estimate for class v_i given that the attribute has value a_l.

The key idea is that to compute a given entry s_{li}, P_d constructs a binary vector corresponding to the entities in the training set with 1 for each item having the value a_l and 0 for other items. P_c constructs a similar vector with $1/n_i$ for the n_i entities in the class, and 0 for other entities. The scalar product of the vectors gives the appropriate probability for the entry.

Require: Nominal attribute D, Class attribute C

Require: n transactions, r attribute values, k class values

Require: $Const$ (field size / precision)

Ensure: $r \times k$ share matrices S^c, S^d where $S = S^c + S^d$ gives the probability values for each class/attribute

1: **for** $i = 1 \ldots k$ {For each class value} **do**
2: {P_c generates the vector $\vec{Y}$ from C:}
3: **for** $j = 1 \ldots n$ **do**
4: **if** $c_j = v_i$ **then**
5: $y_j \leftarrow \lfloor Const/n_i \rfloor$ {Class value is v_i}
6: **else**
7: $y_j \leftarrow 0$
8: **end if**
9: **end for**
10: **for** $l = 1 \ldots r$ {For each attribute value} **do**
11: {P_d generates the vector $\vec{X}$ from D:}
12: **for** $j = 1 \ldots n$ **do**
13: **if** $d_j = a_l$ **then**
14: $x_j \leftarrow 1$ {Attribute value is a_l}
15: **else**
16: $x_j \leftarrow 0$
17: **end if**
18: **end for**
19: $s^c_{li}, s^d_{li} \leftarrow \vec{X} \cdot \vec{Y}$ computed using a secure scalar product protocol (Section A.1.2)
20: **end for**
21: **end for**

Algorithm 6: Computing shares of all probabilities

Algorithm 6 defines the protocol to compute the shares of these renormalized ratios (probabilities) in detail. To accomplish the security proof, calculations must occur over a closed field; as a result values are premultiplied by a constant and truncated to integral values. To achieve full precision, this constant should be a multiple of the least common multiple of $n_1, ..., n_k$, however sharing this would reveal private information about the distribution of classes. ($n!$ would be an acceptable multiple that would not reveal class distributions, but is computationally intractable). In practice, using n on the order of word size (e.g., 2^{64}) will give reasonable precision and computational cost. To simplify presentation, we will speak of "probability" when the algorithm in fact computes $C * probability$.

Numeric Attributes

For numeric attributes, computing the probability requires knowing the mean μ and variance σ^2 for each class value.

Computing the mean is similar to the preceding algorithm – for each class, P_c builds a vector of $1/n_i$ and 0 depending on whether the training entity is in the class or not, and the mean for the class is the scalar product of this vector with the projection of the data onto the attribute. The scalar product gives each party a share of the result, such that the sum is the mean (actually a constant times the mean, to convert to an integral value.) The result is a length k vector of the shares of the means.

Computing the variances $\sigma_1^2, \ldots, \sigma_k^2$ is more difficult, as it requires summing the square of the distances between values and the mean, without revealing values to P_c or classes to P_d, or means to either. This is accomplished with homomorphic encryption: $E(a+b) = E(a) * E(b)$. Algorithm 7 describes this process in detail, we highlight some of the more confusing areas here.

Require: n data items, k class values, precision/field size $Const$
Require: P_d has data vector $\vec{D}$, P_c has class vector $\vec{C}$
1: {Compute the mean:}
2: **for** $i = 1 \ldots k$ **do**
3: **for** $j = 1 \ldots n$ **do**
4: **if** $c_j = v_i$ **then**
5: $y_j \leftarrow \lfloor Const/n_i \rfloor$ {Class value is v_i}
6: **else**
7: $y_j \leftarrow 0$
8: **end if**
9: **end for**
10: $\mu'_i, \mu''_i \leftarrow \vec{D} \cdot \vec{Y}$ {Computed with secure scalar product.}
11: {shares $\mu'_i + \mu''_i = Const * \mu_i$, where μ_i is the mean for class i}
12: **end for**
13:
14: {Compute the variance}
15: P_d: generate a homomorphic public key encryption pair E_k, D_k
16: **for** $j = 1 \ldots n$ **do**
17: $d^e_j \leftarrow E_k(Const * d_j)$
18: **end for**
19: **for** $i = 1 \ldots k$ **do**
20: $m^e_i \leftarrow E_k(\mu'_i)$
21: **end for**
22: P_d sends $\vec{D^e}, \vec{M^e}$, and E_k to P_c
23: P_c: generate the vectors $\vec{Z}$ and $\vec{X}$:
24: **for** $j = 1 \ldots n$ **do**
25: Generate random r_j
26: $z_j \leftarrow y'_j/(m_{c_j} * E_k(\mu''_{c_j} + r_j))$
27: $\{= E_k(Const * d_j - \mu'_{c_j} - \mu''_{c_j} - r_j)\}$
28: $\{= E_k(Const * (d_j - \mu_{c_j}) - r_j)\}$
29: **end for**
30: P_c sends $\vec{Z}$ to P_d
31: P_d decrypts all the transactions in $\vec{Z}$ to get $\vec{W}$ (i.e., $w_j \leftarrow D_k(E_k(Const * (d_j - \mu_l) + r_j)) = Const * (d_j - \mu_l) + r_j$)
32:
33: **for** $j = 1 \ldots n$ **do**
34: Shares $t'_j, t''_j \leftarrow (r_j + w_j)^2$ using the protocol in Section A.1.3
35: **end for**
36: **for** $i = 1 \ldots k$ **do**
37: $\{\vec{Y}$ is vector for class k as generated in steps 1-5$\}$
38: Compute shares $temp, \sigma''_j$ where $temp + \sigma''_j = \vec{T''} \cdot \vec{Y}$
39: $P_c : \sigma''_j \leftarrow \vec{T} \cdot \vec{Y} + temp$
40: {Note $\sigma'_j + \sigma''_j = Const^3 * (\frac{1}{n_j} * (\sum_j (d_j - \mu_j)^2))$}
41: **end for**

Algorithm 7: Computing Mean and Variance

In lines 15-22, P_d computes encrypted vectors of the data values and its share of the means and sends them to P_c, along with the encryption (but not decryption) key.

In the next phase (lines 23-31), P_c takes the data values and subtracts the means (both its share and the share sent by P_d) to get the distance needed to compute the variance. P_c also subtracts a random value, keeping the random value as its share of the distances. Homomorphic encryption makes this possible without decrypting. P_c sends the vector back to P_d, which decrypts to get the distance plus a random value.

Next, the parties engage in a square computation protocol (Section A.1.3) to compute shares t'_j, t''_j of the square of the sum of P_c's randoms r_j and the decrypted distance. The scalar product of P_d's share vector and the class vector $\vec{Y}$ is taken, giving two shares. To its share, P_c adds the scalar product of its vector of randoms and $\vec{Y}$. This gives each party a share of σ^2 multiplied by the probability of an item appearing in the class (again scaled to an integral value, in this case by the cube of the chosen constant.)

The scalar product and square computation subroutines are based on previous work, and are discussed in Section A.1.

3.4.4 Evaluation of an Instance

A new instance is classified according to Equation 3.3. Since both $y = x^2$ and $y = \ln x$ are monotonically increasing functions, squaring and taking the natural log still preserves the correctness of the $argmax$. Thus the equation can be rewritten as follows:

$$\begin{aligned} v_{NB} &= \underset{v_j \in V}{argmax} \left(P(v_j) \prod_i P(a_i|v_j) \right) \\ &= \underset{v_j \in V}{argmax} \left(\ln \left(P(v_j) \prod_i P(a_i|v_j) \right)^2 \right) \end{aligned}$$

$$
\begin{aligned}
&= \underset{v_j \in V}{argmax} \left((2 * \ln P(v_j)) + \sum_i \ln \left(P(a_i|v_j)^2 \right) \right) \\
&= \underset{v_j \in V}{argmax} \left(\begin{array}{l} C + (2 * \ln P(v_j)) + \\ \sum_i \ln \left(P(a_i|v_j)^2 \right) \end{array} \right)
\end{aligned} \tag{3.5}
$$

where the constant C is determined by the number and composition of the nominal attributes. If there are l nominal attributes, $C = Const_1 * \ldots * Const_l$ where each $Const_i$ is contributed by one nominal attribute due to the fact that the nominal probabilities are multiplied with a constant. By taking the logarithm, the constant multiplicative factor is converted to a constant additive factor.

For a nominal attribute,

$$\ln \left(P(a_i|v_j)^2 \right) = \ln(\frac{n_j}{n})^2 = 2 \ln(p' + p'')$$

We have already shown how to compute p' and p'' in Section 3.4.3. The parties can computes shares of the ln function securely using the secure ln method developed by Lindell and Pinkas, outlined later in Section A.1.4. Finally, they can multiply their shares by 2 to generate the necessary shares.

For a numeric attribute,

$$
\begin{aligned}
\ln \left(P(a_i|v_j)^2 \right) &= \ln \left(\frac{1}{2\pi\sigma^2} e^{-\frac{(x-\mu)^2}{\sigma^2}} \right) \\
&= -\ln(2\pi\sigma^2) - \frac{(x-\mu)^2}{\sigma^2} \\
&= -\ln(2\pi) - \ln(\sigma^2) - \frac{(x-\mu)^2}{\sigma^2}
\end{aligned} \tag{3.6}
$$

$\ln(2\pi)$ is publicly computable, but it does not even need to be computed since it is a constant that does not affect comparison. Shares of σ^2 are present with both parties. Shares of $\ln(\sigma^2)$ can again be computed using the method discussed in Section A.1.4. Shares of $(x - \mu)^2$ can be computed using the square computation method given in Section A.1.3. Finally, shares of $(x - \mu^2)/\sigma^2$ can be computed using the division protocol described in Section A.1.5. Thus, for every class value, for each attribute, the shares of the required values are present with the party owning the attribute and the party owning the class attribute. Now, evaluating equation

3.5 reduces to a simple circuit evaluation. The required circuit adds up all of the shares for each attribute for each class value and outputs the name of the class with the maximum such value. This circuit is similar to the *Secure_Add_and_Compare* circuit used in Section 4.5 except that it is extended to multiple attributes.

The Taylor series expansion is a bounded approximation to the real value. However, the result class of the algorithm can only be wrong if the true Naïve Bayes probability estimate of the correct class and the incorrect result are within some δ (increasing the number of steps in the Taylor series expansions, and thus the communication cost, allows the choice of δ to be arbitrarily small). If the correct class and the class returned are this close, then the "incorrect" result is nearly as good an answer as the best result, and likely to be adequate in practice.

3.4.5 Security Analysis

We now give a proof of security for protocols of Section 3.4.3, assuming security of the sub-blocks used, and applying the composition theorem of [39] described in Section 2.3. We start with a lemma that share splitting does in fact preserve privacy.

Lemma 3.4.1 *If a function $y = f(x_1 + x_2)$ is evaluated over a finite field $\mathcal{F}$, where the inputs x_1 and x_2 are shares known to two different parties and the output y is also split into shares, where the share y_1 is chosen randomly from an uniform distribution over the field $\mathcal{F}$ and $y_2 = y - y_1$, then both parties can independently simulate their share y_i.*

PROOF. First, we need to prove that $P(y_2 = a) = \frac{1}{|\mathcal{F}|}$.

$$\begin{aligned} P(y_2 = a) &= P(y - y_1 = a) \\ &= P(y_1 = y - a) \\ &= \frac{1}{|\mathcal{F}|} \end{aligned}$$

This is equivalent to choosing y_2 from an uniform distribution over the field $\mathcal{F}$. Note that though the *joint* distribution of y_1, y_2 is *not* necessarily uniform, independently both y_1 and y_2 can be simulated using a uniform distribution. □

Theorem 3.4.2 *Algorithm 6 privately computes the shares of all the probabilities.*

PROOF. The only communication occurs at line 19 with the invocation of the scalar product protocol. The results of the scalar product protocol are random shares, which can be simulated as shown in Lemma 3.4.1. Protocol 6 can thus be simulated, with the composition theorem 2.3.1 being applied to the scalar product protocol. □

Theorem 3.4.3 *Algorithm 7 privately computes the shares of the means and variances.*

PROOF. Communication occurs only at lines $10, 22, 30, 34, 38$. We prove the protocol secure by providing a simulator for both parties P_c and P_d. The simulator for both P_c and P_d proceeds simply by executing the actual protocol. In order to show that the view of each party can be simulated, we only need to simulate the messages received by each party.

At line 10, the results of the scalar product protocol are random shares, which can be simulated by both P_c and P_d as shown in Lemma 3.4.1.

At line 22, P_c simulates the message received by P_c generating a key pair and using the generated encryption key for E_k. It also generates a n random numbers to comprise $\vec{D}^e$ and k random numbers to form $\vec{M}^e$. Assuming security of encryption, these are computationally indistinguishable from the true vectors and encryption key.

To simulate the message received by P_d at line 30, P_d chooses n random numbers from an uniform distribution over the field $\mathcal{F}$ and encrypts these numbers with its key E_k to form the vector $\vec{Z}$. Note that each z_j simulates the encryption of

$Const * (d_j - \mu_{c_j}) - r_j$. Since the operations are over a finite field $\mathcal{F}$ and the r_j is also uniformly chosen over the finite field $\mathcal{F}$,

$$\begin{aligned} P(D_k(z_j) = x) &= P(Const * (d_j - \mu_{c_j}) - r_j = x) \\ &= P(r_j = Const * (d_j - \mu_{c_j}) - x) \\ &= \frac{1}{|\mathcal{F}|} \end{aligned}$$

Thus simulating the value is possible by choosing a random number from an uniform distribution over $\mathcal{F}$ and encrypting this random with the encryption key E_k.

At line 34, the results of the square computation are random shares, which can be simulated by both P_c and P_d as shown in Lemma 3.4.1.

At line 38, the results of the scalar product protocol are random shares, which can be simulated by both P_c and P_d as shown in Lemma 3.4.1.

Note that the scalar product in line 39 is a completely local computation by P_c and thus does not need to be simulated by P_d. Protocol 7 can thus be simulated, with the composition theorem 2.3.1 being applied to the scalar product protocol at lines 10 and 38 and to the square computation protocol at line 34. □

Theorem 3.4.4 *The evaluation protocol in Section 3.4.4 privately computes the class.*

PROOF. For nominal attributes, the shares of the probabilities are present with both the parties to begin with. The secure ln computation returns random shares to both the parties. By Lemma 3.4.1, these shares can be independently simulated by both the parties.

Similarly, for numeric attributes, the shares of the means and variances are present with both the parties. The secure ln computation returns random shares of the variance to both the parties. By Lemma 3.4.1, these shares can be independently simulated by both the parties. The shares of $(x - \mu)^2$ are computed by a call to the secure square computation protocol. Since this protocol also computes random shares, by Lemma 3.4.1, these shares can be independently simulated by both the parties. Finally, the shares of $(x - \mu)^2/\sigma^2$ are computed by an invocation

of the division protocol which also computes random shares. Therefore by Lemma 3.4.1 these shares can also be independently simulated by both the parties.

The addition and comparison circuit is a generic circuit and thus has been proven secure by [40]. The result is simply the output class, and is simulated exactly as the final result is presumed known by the simulator. Applying Theorem 2.3.1 to the secure ln computation, protocol 6, protocol 7 and square computation protocol, the evaluation protocol is also secure. □

3.4.6 Computation and Communication Analysis

For the purpose of this analysis, the number of distinct class values is assumed to be k. For a nominal attribute with r attribute values, the scalar product protocol is called a total of $r * k$ times over n-dimensional vectors. Thus depending on the cost of the scalar product (which is typically linear in n), the cost of protocol 6 is $O(rkn)$. For small values of r, k this is feasible, though for large values it may be *quite* inefficient. A mitigating factor is that if r, k are large relative to the size of the training set n, Naïve Bayes is probably not a good classifier to use anyway.

For numeric attributes, to compute the shares of the means requires k invocations of the scalar product protocol. To compute the variance, at line 22 P_d sends two n-dimensional vectors to P_c. At line 30, P_c sends one n-dimensional vector to P_d. Line 34 involves n invocations of the square computation protocol. Since the square computation protocol consists of one polynomial evaluation for a polynomial of degree 2, the communication cost of n invocations of the square computation require only linear (O(n)) communication cost where the constant is quite small. Finally, line 38 again involves k invocations of the scalar product protocol. Thus the total communication cost is clearly linear in n (O(n)), where the constant is of the degree of k. Thus the cost for numeric attributes is significantly lower than for nominal attributes.

Selecting the parameters is done off-line, while classification of a new instance can be considered "online", and is done one instance at a time. Evaluation requires one call to the secure ln protocol for every nominal attribute and one call to the secure ln protocol, one call to the square computation protocol and one call to the division protocol for every numeric attribute. Finally, it also requires one call to the generic addition and comparison circuit to find the class having the maximum. Secure ln computation requires running Yao's protocol on a circuit that is linear in the size of the inputs followed by the private evaluation of a polynomial of degree k' over the field $\mathcal{F}$. The value of this k' is user decidable depending on the accuracy / cost tradeoff. The total communication cost is dominated by the circuit evaluation and thus is $O(k' \log |\mathcal{F}| \cdot |S|)$ bits where $|S|$ is the length of the key for a pseudo-random function.

The cost of square computation protocol is insignificant (since it is a constant). Similarly, the division protocol requires only two scalar products of vectors of constant size (2 and 3). The cost for a numeric attribute is dominated by the secure ln protocol.

The single generic circuit required to find the class with the maximum value requires a total of k comparison circuits built on top of q addition circuits, where q is the total number of attributes. The cost of this is linear in $q + k$. Thus for a total of q attributes, the total cost of a single evaluation is $O(qk' \log |\mathcal{F}| \cdot |S|)$ bits.

3.5 Decision Tree Classification

Decision Tree Classifiers are used effectively in a multitude of different areas: radar signal classification, character recognition, remote sensing, medical diagnosis, expert systems, and speech recognition, to name a few. One of the most important features of decision tree classifiers is their ability to break down a complex decision making process into a collection of simpler decisions, thus providing a solution which is often easier to interpret. We look at the seminal ID3 [76] classification algorithm.

While the problem has been addressed earlier [28], the prior solution is limited to two parties and also requires that both parties have the class attribute. We present a completely general solution to the problem. The method presented here works for any number of parties, and the class attribute (or other attributes) need be known only to one party.

Privacy preservation can mean many things: Protecting specific individual values, breaking the link between values and the individual they apply to, protecting source, etc. Here, we aim for a high standard of privacy: Not only individual entities are protected, but to the extent feasible even the schema (attributes and possible attribute values) are protected from disclosure. The goal is that each site need disclose as little as possible, while still constructing a valid tree in a time suitable for practical application.

To this end, all that is revealed is the basic structure of the tree (e.g., the number of branches at each node, corresponding to the number of distinct values for an attribute; the depth of each subtree) and which site is responsible for the decision made at each node (i.e., which site possesses the attribute used to make the decision, but not what attribute is used, or even what attributes the site possesses.) This allows for efficient *use* of the tree to classify an object; otherwise using the tree would require a complex cryptographic protocol involving every party at every *possible* level to evaluate the class of an object without revealing who holds the attribute used at that level.

Each site also learns the count of classes at some interior nodes (although only the class site knows the mapping to actual classes – other sites don't even know if a class with 30% distribution at one node is the same class as one with a 60% distribution at a lower node, except to the extent that this can be deduced from the tree and it's own attributes.) At the leaf nodes, this is desirable: one often wants probability estimates, not simply a predicted class. As knowing the count of transactions at each leaf node would enable computing distributions throughout the tree anyway, this really doesn't disclose much *new* information.

Require: R, the set of attributes

Require: C, the class attribute

Require: T, the set of transactions

1: **if** R is empty **then**

2: return a leaf node, with class value assigned to most transactions in T

3: **else if** all transactions in T have the same class c **then**

4: return a leaf node with the class c

5: **else**

6: Determine the attribute A that best classifies the transactions in T

7: Let $a_1, \ldots, a_m$ be the values of attribute A. Partition T into the m partitions $T(a_1), \ldots, T(a_m)$ such that every transaction in $T(a_i)$ has the attribute value a_i.

8: Return a tree whose root is labeled A (this is the test attribute) and has m edges labeled $a_1, \ldots, a_m$ such that for every i, the edge a_i goes to the tree $ID3(R - A, C, T(a_i))$.

9: **end if**

Algorithm 8: ID3(R,C,T) tree learning algorithm

3.5.1 Privacy-Preserving ID3: Creating the Tree

The basic ID3 algorithm is given in Algorithm 8. We will introduce our distributed privacy-preserving version by running through this algorithm, describing pieces as appropriate. We then give the full algorithm in Algorithm 14. Note that for our distributed algorithm, no site knows R, instead each site i knows its own attributes R_i. Only one site knows the class attribute C. In vertical partitioning, every site knows a *projection* of the transactions $\Pi_{R_i} T$. Each projection includes a transaction identifier that serves as a join key.

We first check if R is empty. This is based on Secure Sum [50, 84], and is given in Algorithm 9. The idea is that the first party adds a random r to its count of remaining items. This is passed to all sites, each adding its count. The last site

Require: k sites P_i (the site calling the function is P_1; any other site can be P_k), each with a flag $AttRem_i = 0$ if no remaining attributes, $AttRem_i = 1$ if P_i has attributes remaining.

Require: a commutative encryption function E with domain size $m > k$.

1: P_1 chooses a random integer r uniformly from $0 \ldots m-1$.
2: P_1 sends $r + AttRem_i$ to P_2
3: **for** $i = 2..k-1$ **do**
4: Site P_i receives r' from P_{i-1}.
5: P_i sends $r' + AttRem_i \bmod m$ to P_{i+1}
6: **end for**
7: Site P_k receives r' from P_{k-1}.
8: $r' \leftarrow r' + AttRem_i \bmod m$
9: P_1 and P_k create secure keyed commutative hash keys E_1 and E_k {See Section 4.1 for discussion of commutative hash.}
10: P_1 sends $E_1(r)$ to P_k
11: P_k receives $E_1(r)$ and sends $E_k(E_1(r))$ and $E_k(r')$ to P_1
12: P_1 returns $E_1(E_k(r')) = E_k(E_1(r))$ $\{\Leftrightarrow r' = r \Leftrightarrow \sum_{j=1}^{k} AttRem_i = 0 \Leftrightarrow$ no attributes remain $\}$

Algorithm 9: IsREmpty(): Are any attributes left?

A_1	A_2	A_3	A_4	A_5	A_6
5	high	?	?	warm	?

Figure 3.5. A constraint tuple for a single site

and first then use commutative encryption to compare the final value to r (without revealing either) – if they are the same, R is empty.

Line 2 requires determining the majority class for a node, when only one site knows the class. This is accomplished with a protocol for securely determining the cardinality of set intersection, given in Section 4.1. Each site determines which of its transactions *might* reach that node of the tree. The intersection of these sets with the transactions in a particular class gives the number of transactions that reach that point in the tree, enabling the class site to determine the distribution and majority class; it returns a (leaf) node identifier that allows it to map back to this distribution.

To formalize this, we introduce the notion of a *Constraint Set*. As the tree is being built, each party i keeps track of the values of its attributes used to reach that point in the tree in a filter $Constraints_i$. Initially, this is all don't care values ('?'). However, when an attribute A_{ij} at site i is used (lines 6-7 of id3), entry j in $Constraints_i$ is set to the appropriate value before recursing to build the subtree. An example is given in Figure 3.5. The site has 6 attributes $A_1, \ldots, A_6$. The constraint tuple shows that the only transactions valid for this transaction are those with a value of 5 for A_1, *high* for A_2, and *warm* for A_5. The other attributes have a value of ? since they do not factor into the selection of an instance.

Formally, we define the following functions:

Constraints.set($attr, val$)**:** Set the value of attribute *attr* to *val* in the local constraints set. The special value '?' signifies a don't-care condition.

satisfies: *x satisfies* $Constraints_i$ if and only if the attribute values of the instance are compatible with the constraint tuple: $\forall i, (A_i(x) = v \Leftrightarrow Constraints(A_i) = v) \vee Constraints(A_i) =$ '?'.

FormTransSet:

Function $FormTransSet(Constraints)$*: Return local transactions meeting all of the constraints*

1: $Y = \emptyset$
2: **for all** transaction id $i \in T$ **do**
3: **if** t_i satisfies $Constraints$ **then**
4: $Y \leftarrow Y \cup \{i\}$
5: **end if**
6: **end for**
7: return Y

We can now determine the majority class (and distribution of classes) by computing for each class $\bigcap_{i=1..k} Y_i$, where Y_k includes a constraint on the class value. This is given in Algorithm 10.

The next issue is determining if all transactions have the same class (Algorithm 8 line 3). Note that if they are not all the same class, we don't want to disclose any more than necessary. For efficiency, we do allow the class site to learn the count of classes even if this is an interior node; since it could compute this from the counts at the leaves of the subtree below the node, this discloses no additional information. Algorithm 11 gives the details, it uses constraint sets and secure cardinality of set intersection in basically the manner describe above for computing the majority class at a leaf node. If all transactions are in the same class, we construct a leaf node. The class site maintains a mapping from the ID of that node to the resulting class distribution

Require: k sites P_i with local constraint sets $Constraints_i$

1: **for all** sites P_i except P_k **do**
2: at P_i: $Y_i \leftarrow FormTransSet(Constraints_i)$
3: **end for**
4: **for** each class $c_1, \ldots, c_p$ **do**
5: at P_k: $Constraints_k.set(C, c_i)$ {To include the class restriction}
6: at P_k: $Y_k \leftarrow FormTransSet(Constraints_k)$
7: $cnt_i \leftarrow |Y_1 \cap \ldots \cap Y_k|$ using the cardinality of set intersection protocol (Algorithm 17)
8: **end for**
9: return $(cnt_1, \ldots, cnt_p)$

Algorithm 10: DistributionCounts(): Compute class distribution given current constraints

Require: k sites P_i with local constraint sets $Constraints_i$

1: $(cnt_1, \ldots, cnt_p) \leftarrow DistributionCounts()$
2: **if** $\exists j$ s.t. $cnt_j \neq 0 \wedge \forall i \neq j$, $cnt_i = 0$ {only one of the counts is non-zero} **then**
3: Build a leaf node with distribution $(cnt_1, \ldots, cnt_p)$ {Actually, 100% class j}
4: return ID of the constructed node
5: **else**
6: return $false$
7: **end if**

Algorithm 11: IsSameClass(): Are all transactions of the same class?

The next problem is to compute the best attribute: that with the maximum information gain. If an attribute A is used to partition the data set S, the information gain can be computed as:

$$Gain(S, A) = Entropy(S) - \sum_{v \in A} \left(\frac{|S_v|}{|S|} * Entropy(S_v) \right)$$

The entropy of a dataset S is given by:

$$Entropy(S) = -\sum_{j=1}^{p} \frac{N_j}{N} \log \frac{N_j}{N}$$

where N_j is the number of transactions having class c_j in S and N is the number of transactions in S.

As we see, this again becomes a problem of counting transactions: the number of transactions that reach the node N, the number in each class N_j, and the same two after partitioning with each possible attribute value $v \in A$. Algorithm 13 details the process of computing these counts; Algorithm 12 captures the overall process.

1: **for all** sites P_i **do**
2: $bestgain_i \leftarrow -1$
3: **for** each attribute A_{ij} at site P_i **do**
4: $gain \leftarrow ComputeInfoGain(A_{ij})$
5: **if** $gain > bestgain_i$ **then**
6: $bestgain_i \leftarrow gain$
7: $BestAtt_i \leftarrow A_{ij}$
8: **end if**
9: **end for**
10: **end for**
11: return $argmax_j \, bestgain_j$ {Could implement using a set of secure comparisons}

Algorithm 12: AttribMaxInfoGain(): return the site with the attribute having maximum information gain

Once the best attribute has been determined, execution proceeds at that site. It creates an interior node for the split, then recurses.

1: $S \leftarrow DistributionCounts()$ {Total number of transactions at this node}
2: $InfoGain \leftarrow Entropy(S)$
3: **for** each attribute value a_i **do**
4: $Constraints.set(A, a_i)$ {Update local constraints tuple}
5: $S_{a_i} \leftarrow DistributionCounts()$
6: $Infogain \leftarrow Infogain - Entropy(S_{a_i}) * |S_{a_i}|/|S|$ {$|S|$ is $\sum_{i=1}^{p} cnt_i$}
7: **end for**
8: $Constraints.set(A, \text{`?'})$ {Update local constraints tuple}
9: return InfoGain

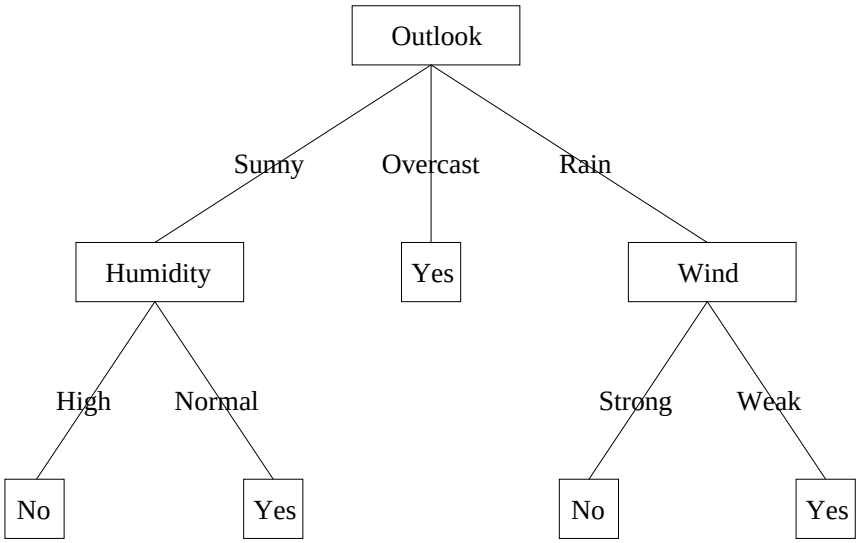
Algorithm 13: ComputeInfoGain(A): Compute the Information Gain for attribute A

Require: Transaction set T partitioned between sites $P_1, \ldots, P_k$

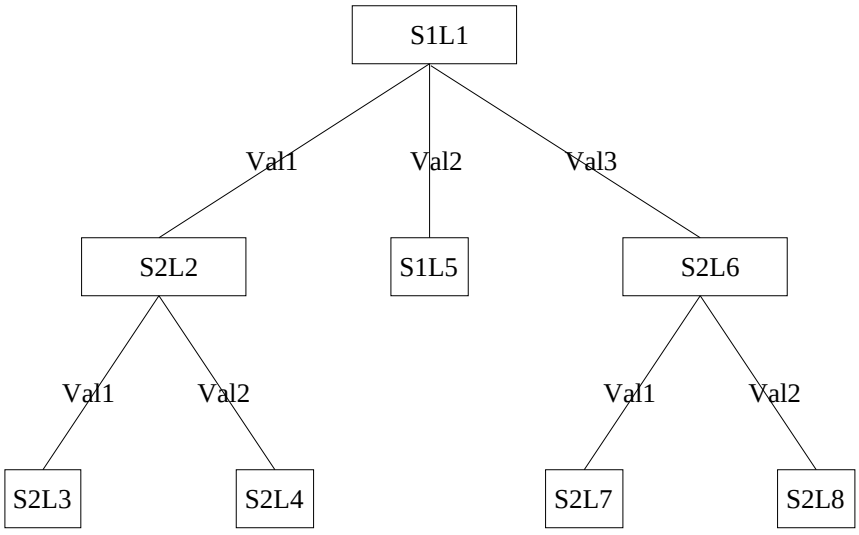
Require: p class values, $c_1, \ldots, c_p$, with P_k holding the class attribute

1: **if** $IsREmpty()$ **then**
2: Continue at site P_k up to the return:
3: $(cnt_1, \ldots, cnt_p) \leftarrow DistributionCounts()$
4: Build a leaf node with distribution $(cnt_1, \ldots, cnt_p)$
5: $\{class \leftarrow argmax_{i=1..p}\, cnt_i\}$
6: return ID of the constructed node
7: **else if** $clsNode \leftarrow$ (at P_k :) $IsSameClass()$ **then**
8: return leaf nodeId $clsNode$
9: **else**
10: $BestSite \leftarrow AttribMaxInfoGain()$
11: **Continue execution at** $BestSite$**:**
12: Create Interior Node Nd with attribute $Nd.A \leftarrow BestAtt_{BestSite}$ {This is best locally (from $AttribMaxInfoGain()$), and globally from line 8}
13: **for** each attribute value $a_i \in Nd.A$ **do**
14: $Constraints.set(Nd.A, a_i)$ {Update local constraints tuple}
15: $nodeId \leftarrow PPID3()$ {Recurse}
16: $Nd.a_i \leftarrow nodeId$ {Add appropriate branch to interior node}
17: **end for**
18: $Constraints.set(A, \text{`?'})$ {Returning to parent: should no longer filter transactions with A}
19: Store Nd locally keyed by Node ID
20: return Node ID of interior node Nd {Execution continues at site owning parent node}
21: **end if**

Algorithm 14: PPID3(): Privacy-Preserving Distributed ID3



(a) The original tree



(b) The privacy preserving tree (Mapping from identifiers to attributes and values is known only at the site holding attributes)

Figure 3.6. The $ID3$ decision tree on the weather dataset

3.5.2 Using the Tree

Instance classification proceeds as in the original ID3 algorithm, except that the nodes (and attributes of the database) are distributed. The site requesting classification (e.g., a master site) knows the root node of the classification tree. The basic idea is that control passes from site to site, based on the decision made. Each site knows the transaction's attribute values for the nodes at its site (and can thus evaluate the branch), but knows nothing of the other attribute values. The complete

1: {The start site and ID of the root node is known}
2: **if** $nodeId$ is a LeafNode **then**
3: return class/distribution saved in $nodeId$
4: **else** {$nodeId$ is an interior node}
5: $Nd \leftarrow$ local node with id $nodeId$
6: $value \leftarrow$ the value of attribute $Nd.A$ for transaction $instId$
7: $childId \leftarrow Nd.value$
8: return $childId.Site.classifyInstance(instId, childId)$ {Actually tail recursion: this site need never learn the class}
9: **end if**

Algorithm 15: classifyInstance(instId, nodeId): returns the class/distribution for the instance represented by instId

algorithm is given in Algorithm 15, and is reasonably self-explanatory if viewed in conjunction with Algorithm 14.

We now give a demonstration of how instance classification would actually happen in this instance for the tree built with the UCI "weather" dataset [13]. Assume two sites: The weather observatory collects information about relative humidity and wind, a second collects temperature and cloud cover forecast as well as the class ("Yes" or "No"). Suppose we wish to know if it is a good day to play tennis. Neither sites wants to share their forecasts, but are willing to collaborate to offer a "good tennis day" service. The classification tree is shown in Figure 3.6(b), with S1 and S2

corresponding to the site having information on that node. If today is sunny with normal humidity, high temperature, and weak wind; classification would proceed as follows: We know that Site 1 has the root node (we don't need to know anything else). Site 1 retrieves the attribute for from $S1L1$: Outlook. Since the classifying attribute is outlook, and Site 1 knows the forecast is sunny, the token $S2L2$ is retrieved. This indicates that the next step is at Site 2. Site 2 is called with the token $S2L2$, and retrieves the attribute for $S2L2$: Humidity. The humidity forecast is normal, so the token $S2L4$ is retrieved. Since this token is also present at Site 2, it retrieves the class value for nodeId $S2L4$ and returns it: we receive our answer of "Yes".

3.5.3 Security Analysis

We first analyze the security of the constituent algorithms, then the security of the complete algorithm. Although it may seem that some of the constituent algorithms leak a large quantity of information, in the context of the full algorithm the leaked information can be simulated by knowing the distribution counts at each node, so overall privacy is maintained.

Lemma 3.5.1 *Algorithm 9 reveals nothing to any site except whether the total number of attributes left is* 0.

PROOF. The algorithm has two basic phases: The sum (through P_k), and the comparison between P_k and P_1. We start with the sum: simulating the messages received at lines 2 and 7. The value received by P_i at these steps is

$$r + \sum_{j=1}^{i-1} AttRem_j \bmod m$$

We will simulate by choosing a random integer uniformly from $0 \ldots m-1$ for r'. We now show that the probability that the simulated $r' = x$ is the same as the probability that the messages received in the view $= x$.

$$Pr\{VIEW_i = x\} \quad = \quad Pr\{x = r + \sum_{j=1}^{i-1} AttRem_j \bmod m\}$$

$$
\begin{aligned}
&= Pr\{r = x - \sum_{j=1}^{i-1} AttRem_j \bmod m\} \\
&= \frac{1}{m} \\
&= Pr\{Simulator_i r' = x\}
\end{aligned}
$$

The key to the second and fourth lines is that arithmetic is mod m. r and r' are chosen uniformly from $0 \ldots m-1$, so the probability of hitting any particular value in that range is $1/m$.

Simulating the message received by P_k at line 11 is simple: Secure encryption gives messages where the distribution is independent of the key/message, so a selection from this distribution of possible encrypted messages simulates what P_k receives.

The messages received by P_1 are more difficult. The problem is that if $r = r'$, $E_k(r')$ must be such that when encrypted with E_1 it is equal to $E_k(E_1(r))$. For this, the simulator requires the ability to decrypt (as in the set intersection proof). The simulator computes $m = D_1(E_k(E_1(r)) = E_k(r)$. If $r = r'$, this is the message used to simulate $E_k(r')$. If not, a random message $\neq m$ is chosen, as in the simulator for P_k. □

Lemma 3.5.2 *Algorithm 10 reveals only the count of instances corresponding to all combinations of constraint sets for each class.*

PROOF. The only communication occurs at line 7 which consists of a call to Algorithm 17 (Cardinality of Set Intersection). This reveals only the size of the intersection set for all subsets of Y_i, which are the counts revealed. By application of the composition theorem (2.3.1), with Algorithm 10 being g and Algorithm 17 being f, Algorithm 10 is secure. □

Lemma 3.5.3 *Algorithm 11 finds if all transactions have the same class, revealing only the class distributions described in Lemma 3.5.2.*

PROOF. Line 1 is an invocation of Algorithm 10; the security of which has been discussed. Everything else is computed locally, and can be simulated from the knowledge from Lemma 3.5.2. □

Lemma 3.5.4 *Algorithm 13 reveals nothing except the counts S, S_{a_i}, and the constituent subcounts described in Lemma 3.5.2 for each attribute value a_i and class j, assuming the number of distinct class values is known.*

PROOF. The only messages received are at lines 1 and 5, invocations of the *DistributionCounts*() function. Applying the composition theorem to these, Algorithm 13 is secure. □

Lemma 3.5.5 *Algorithm 12 finds the site with the attribute having the maximum information gain while revealing only the best information gain at each site and the information discussed in Lemma 3.5.4.*

PROOF. Communication occurs at lines 4 and 11. Line 4 consists of an invocation of Algorithm 13. Line 11 is implemented by letting the site compare all the values; revealing the value of the best information gain at each site. Assuming this is revealed (part of the input to the simulator), it is trivially simulated. Repeated application of the composition theorem completes the proof. □

Further reduction of the information revealed is possible by using a secure protocol for finding the maximum among a set of numbers. This would reveal only the site having the attribute with the maximum information gain and nothing else.

Theorem 3.5.1 *Algorithm 14 computes the decision tree while revealing only:*

- *The distribution subcounts of each node, as described in Lemma 3.5.2. (The full counts, and some of the subcounts, can be computed knowing the distribution counts at the leaves.)*
- *The best information gain from each site at each interior node (as discussed above, this leak can be reduced.)*

PROOF. Knowing the final tree, the simulator at each site can uniquely determine the sequence of node computations at a site and list the function calls occurring due

to this. Given this function call list, if the messages received in each function call can be simulated, the entire algorithm can be proven to be secure.

Line 1 is an invocation of Algorithm 9. The result is simulated as either true or false depending on whether the node in question is a leaf node in the final tree or not.

Line 3 is an invocation of Algorithm 10. The actual counts are given by the counts *in* the leaf node, which are known to the site P_k that invoked the algorithm. The subcounts revealed by Algorithm 10 are presumed known.

Line 7 is an invocation of Algorithm 11. If the node in question is not a leaf node in the final tree, the result is false. Otherwise the result is the nodeId of the leaf node.

Line 10 consists of an invocation of Algorithm 12. The result is actually equal to the Site which will own the child node. This information is known from the tree structure. The subcounts and information gain values revealed during this step are presumed known.

Line 15 is a recursive invocation that returns a node identifier; a part of the tree structure.

Since all of the algorithms mentioned above have been proven secure, applying the composition theorem, Algorithm 14 is secure. The repeated invocations of the cardinality of set intersection protocol (Algorithm 17) are valid because in each invocation, a new set of keys are chosen. This ensures that messages cannot be correlated across calls. □

Theorem 3.5.2 *Algorithm 15 reveals nothing other than the leaf node classifying the instance.*

PROOF. All the computations are local. The only information passed between various sites are node identifiers. This list of node identifiers can be easily simulated from the classification tree once the final leaf is known. □

3.5.4 Computation and Communication Analysis

The communication/computation analysis depends on the number of transactions, number of parties, number of attributes, number of attribute values per attribute, number of classes and complexity of the tree. Assume that there are: n transactions, k parties, c classes, r attributes, p values per attribute (on average), and q nodes in final classification tree. We now give a rough analysis of the cost involved in terms of the number of set intersections required for building the tree (erring on the conservative side).

At each node in the tree the best classifying attribute needs to be determined. To do this, the entropy of the node needs to be computed as well as the information gain per attribute. Computing the entropy of the node requires c set intersections (1 per class). Computing the gain of one attribute requires cp set intersections (1 per attribute value and class). Thus, finding the best attribute requires cpr set intersections. Note that this analysis is rough and assumes that the number of attributes available at each node remains constant. In actuality, this number linearly decreases with the depth of the node in the tree (this has little effect on our analysis). In total, every node requires $c(1 + pr)$ set intersections. Therefore, the total tree requires $cq(1 + pr)$ set intersections.

The intersection protocol requires that the set of each party be encrypted by every other party. Since there are k parties, k^2 encryptions are required and k^2 sets are transferred. Since each set can have at most n transactions, the upper bound on computation is $O(nk^2)$ and the upper bound on communication cost is also $O(nk^2 * bitsize)$ bits.

Therefore, in total the entire classification process will require $O(cqnk^2(1 + pr)$ encryptions and $cqnk^2(1+pr)*bitsize$ bits communication. Note that the encryption process can be completely parallelized reducing the required time by an order of k.

Once the tree is built, classifying an instance requires no extra overhead, and is comparable to the original $ID3$ algorithm.

3.6 Outlier Detection

In the broadest sense, an outlier is an observation that lies an abnormal distance from other values in a random sample from a population. This definition leaves it up to the analyst (or a consensus process) to decide what will be considered abnormal. Before abnormal observations can be singled out, it is necessary to characterize normal observations. The goal of outlier detection is to find all outliers in the input data.

Outlier detection has wide application; one that has received considerable attention is the search for terrorism. Detecting previously unknown suspicious behavior is a clear outlier detection problem. The search for terrorism has also been the flash point for attacks on data mining by privacy advocates; the U.S. Terrorism Information Awareness program was killed for this reason [57].

Outlier detection has numerous other applications that also raise privacy concerns. Mining for anomalies has been used for network intrusion detection [10, 56]; privacy advocates have responded with research to enhance anonymity [41,79]. Fraud discovery in the mobile phone industry has also made use of outlier detection [34]; organizations must be careful to avoid overstepping the bounds of privacy legislation [30]. Privacy-preserving outlier detection will ensure these concerns are balanced, allowing us to get the benefits of outlier detection without being thwarted by legal or technical counter-measures.

We focus specifically on Distance Based Outliers. Knorr and Ng [53] define the notion of a Distance Based outlier as follows: *An object O in a dataset T is a DB(p,dt)-outlier if at least fraction p of the objects in T lie at distance greater than dt from O.* Other distance based outlier techniques also exist [54, 78]. The advantages of distance based outliers are that no explicit distribution needs to be defined to determine unusualness, and that it can be applied to any feature space for which we can define a distance measure. We assume Euclidean distances, although the algorithms are easily extended to general Minkowski distances.

3.6.1 Basic Approach

The problem is to find distance-based outliers without any party gaining knowledge beyond learning which items are outliers. Ensuring that data is not disclosed maintains privacy, i.e., no privacy is lost beyond that inherently revealed in knowing the outliers. Even knowing which items are outliers need not be revealed to all parties, further preventing privacy breaches.

The approach duplicates the results of the outlier detection algorithm of [53]. The idea is that an object o is an outlier if more than a percentage p of the objects in the data set are farther than distance d from o. The basic idea is that parties compute the portion of the answer they know, then engage in a secure sum to compute the total distance. The key is that this total is (randomly) split between sites, so nobody knows the actual distance. A secure protocol is used to determine if the actual distance between any two points exceeds the threshold; again the comparison results are randomly split such that summing the splits (over a closed field) results in a 1 if the distance exceeds the threshold, or a 0 otherwise.

For a given object o, each site can now sum all of its shares of comparison results (again over the closed field). When added to the sum of shares from other sites, the result is the correct count; all that remains is to compare it with the percentage threshold p. This addition/comparison is also done with a secure protocol, revealing only the result: if o is an outlier.

3.6.2 Algorithm

We now present an algorithm for Distance Based Outliers meeting the definition given of Knorr and Ng [53]. As discussed earlier, the algorithm is based on the obvious one: Compare points pairwise and count the number exceeding the distance threshold. The key is that all intermediate computations (such as distance comparisons) leave the results randomly split between the parties involved; only the final result (if the count exceeds $p\%$) is disclosed.

Note that to obtain a secure solution, all operations are carried out modulo some field. We will use the field D for distances, and F for counts of the number of entities. The field F must be over twice the number of objects. Limits on D are based on maximum distances; details on the size are given with each algorithm.

For each object i, the protocol iterates over every other object j. Each party can compute a *share* of the pairwise distance locally; the sum of these shares is the total distance. However, the distance must not be revealed, so a secure protocol is used to get shares of the pairwise comparison of distance and threshold. A second protocol allows comparing the shares with the threshold, returning 1 if the distances exceeds the threshold, or 0 if it does not. The key to this second protocol is that the 1 or 0 is actually two shares m_0 and m_{k-1}, such that $m_0 + m_{k-1} = 1$ (or 0) $\pmod F$. From one share, a party learns nothing.

These shares are added to the running total kept at parties P_0 and P_{k-1}. Once all points have been compared, the parties sum their shares. Since the shares add to 1 for distances exceeding the distance threshold, and 0 otherwise, the total sum $\pmod F$ is the number of points for which the distance exceeds the threshold. P_0 and P_{k-1} finally in a secure protocol that reveals only if the sum of the shares exceeds $p\%$. This ensures that no party learns anything except whether the point is an outlier.

An interesting side effect of this algorithm is that the parties need not reveal any information about the attributes they hold, or even the number of attributes. Each party locally determines the distance threshold for its attributes (or more precisely, the share of the overall threshold for its attributes). Instead of computing the local pairwise distance, each party computes the difference between the local pairwise distance and the local threshold. If the sum of these differences is greater than 0, the pairwise distance exceeds the threshold.

Algorithm 16 gives the full details.

In steps 5–9, the sites sum their local distances. The random x added by P_0 masks the distances from each party. In steps 11–18, Parties P_0 and P_{k-1} get shares of the pairwise comparison result. The comparison is a test if the sum is greater than

Require: k parties, $P_0, \ldots, P_{k-1}$; each holds a subset of the attributes for all objects O.
Require: dt_r : local distance threshold for P_r.
Require: Fields D larger than twice the maximum distance, F larger than $|O|$
1: **for all** objects $o_i \in O$ **do**
2: $m'_0 \leftarrow m'_{k-1} \leftarrow 0 \pmod F$
3: **for all** objects $o_j \in O, o_j \neq o_i$ **do**
4: P_0: Randomly choose a number x from a uniform distribution over the field D; $x' \leftarrow x$
5: **for** $r \leftarrow 0, \ldots, k-2$ **do**
6: At P_r: $x' \leftarrow x' + Distance_r(o_i, o_j) - dt_r \pmod D$ {$Distance_r$ is local distance at P_r}
7: P_r sends x' to P_{r+1}
8: **end for**
9: At P_{k-1}: $x' \leftarrow x' + Distance_{k-1}(o_i, o_j) - dt_{k-1} \pmod D$
10: {Using the secure comparison protocol (Section 4.5)}
11: $P_0 \leftarrow m_0$ and $P_{k-1} \leftarrow m_{k-1}$ such that:
12: **if** $0 < x' + (-x) \pmod D < |D|/2$ **then**
13: $m_0 + m_{k-1} = 1 \pmod F$
14: **else**
15: $m_0 + m_{k-1} = 0 \pmod F$
16: **end if**
17: At P_0: $m'_0 \leftarrow m'_0 + m_0 \pmod F$
18: At P_{k-1}: $m'_{k-1} \leftarrow m'_{k-1} + m_{k-1} \pmod F$
19: **end for**
20: {Using the secure comparison of Section 4.5}
21: $P_0 \leftarrow temp_0$ and $P_{k-1} \leftarrow temp_{k-1}$ such that:
22: **if** $m'_0 + m'_{k-1} \pmod F > |O| * p\%$ **then**
23: $temp_0 + temp_{k-1} \leftarrow 1$ {o_i is an outlier}
24: **else**
25: $temp_0 + temp_{k-1} \leftarrow 0$
26: **end if**
27: P_0 and P_{k-1} send $temp_0$ and $temp_{k-1}$ to the party authorized to learn the result; if $temp_0 + temp_1 = 1$ then o_i is an outlier.
28: **end for**

Algorithm 16: Finding DB(p,D)-outliers

0 (since the threshold has already been subtracted.) The random split of shares ensures that nothing is learned by either party. These two parties keep a running sum of their shares. At the end, in steps 21–26, these shares are added and compared with the percentage threshold. Both parties get a share of the result. Finally, the shares are sent to the appropriate party that is authorized to know the result. This party can sum up the shares and determine if the point is an outlier. Thus, only that party (e.g., a fraud prevention unit) learns if o_i is an outlier, the others learn nothing.

Theorem 3.6.1 *Proof of Correctness: Algorithm 16 correctly returns as output the complete set of points that are global outliers.*

PROOF. In order to prove the correctness of Algorithm 16, it is sufficient to prove that a point is reported as an outlier if and only if it is truly an outlier. Consider point q. If q is an outlier, in steps 12–16 for at least $p\% * |O| + 1$ of the other points, $m_0 + m_{k-1} = 1 \pmod F$. Since $|F| > |O|$, it follows that $m'_0 + m'_{k-1} > |O| * p\%$. Therefore, point q will be correctly reported as an outlier. If q is not an outlier, the same argument applies in reverse. Thus, in steps 12–16 at most $p\% * |O| - 1$ points, $m_0 + m_{k-1} = 1 \pmod F$. Again, since $|F| > |O|$, it follows that $m'_0 + m'_{k-1} \leq |O| * p\%$. Therefore, point q will not be reported as an outlier. □

3.6.3 Security Analysis

Theorem 3.6.2 *Algorithm 16 returns as output the set of points that are global outliers while revealing no other information to any party, provided parties do not collude.*

PROOF. All parties know the number (and identity) of objects in O. Thus they can set up the loops; the simulator just runs the algorithm to generate most of the simulation. The only communication is at lines 7, 11, 21, and 27.

Step 7: Each party P_s sees $x' = x + \sum_{r=0}^{s-1} Distance_r(o_i, o_j)$, where x is the random value chosen by P_0. $Pr(x' = y) = Pr(x + \sum_{r=0}^{s-1} Distance_r(o_i, o_j) = y) = Pr(x = y - \sum_{r=0}^{s-1} Distance_r(o_i, o_j)) = \frac{1}{|D|}$. Thus we can simulate the value received by choosing a random value from a uniform distribution over D.

Steps 11 and 21: The simulator for party P_0 (respectively P_{k-1}) again chooses a number randomly from a uniform distribution, this time over the field F. By the same argument as above, the actual values are uniformly distributed, so the probability of the simulator and the real protocol choosing any particular value are the same. Since a circuit for secure comparison is used, using the composition theorem, no additional information is leaked and steps 11 and 21 are secure.

Step 27: Since the final party knows the results (1 if o_i is an outlier, 0 otherwise), $temp_0$ is simulated by choosing a random value, $temp_1 = result$ (1 or 0) $- temp_0$ mod F. By the same argument on random shares used above, the distribution of simulated values is indistinguishable from the distribution of the shares.

The simulator clearly runs in polynomial time (the same as the algorithm). Since each party is able to simulate the view of its execution (i.e., the probability of any particular value is the same as in a real execution with the same inputs/results) in polynomial time, the algorithm is secure with respect to Definition 2.3.1. □

While the proof is formally only for the semi-honest model, it can be seen that a malicious party in isolation cannot learn private values (regardless of what it does, it is still possible to simulate what it sees without knowing the input of the other parties.) A malicious party can cause incorrect results, but it cannot learn private data values. Step 7 is particularly sensitive to collusion, but can be improved (at cost) by splitting the sum into shares and performing several such sums (see [50] for more discussion of collusion-resistant secure sum).

3.6.4 Computation and Communication Analysis

Algorithm 16 suffers the drawback of having quadratic computation complexity due to the nested iteration over all objects. While the Knorr and Ng [53] algorithm is worst-case quadratic, it stops comparing a point to others as soon as it is determined not to be an outlier. However, such early termination in a secure algorithm would reveal that the point is close to at least $(1-p)\% * |D|$ of the points to which it had been compared.

Due to the quadratic complexity, Algorithm 16 requires $O(n^2)$ secure comparisons (steps 10-16). While operation parallelism can be used to reduce the round complexity of communication, the key practical issue is the computational complexity of the encryption required for the secure comparison and scalar product protocols.

When there are three or more parties, assuming no collusion, we can develop much more efficient solutions that reveal some information. While not completely secure, the privacy versus cost tradeoff may be acceptable in some situations. We cannot simply ask one party to take the shares and do the comparisons. Since all of the parties share all of the points, partial knowledge about a point does reveal useful information to a party. Instead, one of the remaining parties is chosen to play the part of completely untrusted non-colluding party. With this assumption, a much more efficient secure comparison algorithm has been postulated by Cachin [16] that reveals nothing to the third party. The algorithm is otherwise equivalent, but the cost of the comparisons is reduced substantially.

4 PRIMITIVES DEVELOPED

This chapter describes building block primitives developed as part of this dissertation. Each following section describes a single primitive. Each section typically consists of 4 subsections –

1. Subsection 1 defines the problem to be solved.
2. Subsection 2 outlines the solution protocol.
3. Subsection 3 analyzes the communication and computation complexity of the protocol.
4. Subsection 4 provides a security analysis of the protocol.

In a few cases, different primitives solve the same problem in a different manner. In these cases, for brevity, the problem definition of the following primitives merely refers to the problem definition of the first primitive.

The final section of the chapter provides a comparative look at the protocols and discusses possible advantages/disadvantages with using any of them as constituent protocols in a global algorithm.

4.1 Securely Computing the Size of Set Intersection

The three or more party association rule mining algorithm (Section 3.2) requires computing the size of the intersection set of local sets. Apart from this, it is an interesting problem in its own right. Along with our work, there has been other concurrent work solving this problem [3, 37]. Agrawal et. al's solution [3] is similar to ours except that their solution is limited to two semi-honest parties (intersection of two sets). Freedman et al. [37] propose a completely different solution involving

the formulation of polynomials by one party and their evaluation by another. This solution is very efficient in terms of round and communication complexity. However, in terms of computation complexity, it is not scalable to the sizes required for data mining (since some of the assumptions used in their analysis no longer hold).

4.1.1 Problem Definition

Assume $k > 2$ parties, $P_0, \ldots, P_{k-1}$. Each party P_i has set $S_i \subseteq U$ chosen from a common global universe. The problem is to securely compute the size of the intersection set, $| \cap_{i=0}^{k-1} S_i |$.

4.1.2 Algorithm / Protocol

A quick overview of the algorithm idea, along with several needed definitions are now given, before presenting the entire protocol.

Algorithm Idea

The key idea behind the algorithm is simple. It is not necessary to have the actual set elements to compute the cardinality of the intersection set. Instead, the parties jointly generate a mapping from U that no party knows in its entirety. The mapping is used to transform the sets S_i, then the intersection is performed on the transformed sets. Since no party knows the mapping, they cannot reverse the mapping to find the value of any element.

A secure keyed commutative hash function can be used to perform such a mapping, and has other properties that will be useful in proving the security properties of the algorithm. We now formally define such a hash function.

Commutative One-way Hash Functions

The definitions of properties used below are collated from [62].

Definition 4.1.1 *A* commutative keyed one way hash function *(CKHF) is a function h_k, parameterized by a secret key k, with the following properties:*

1. ease of computation – *for a known function h_k, given a value k and an input x, it is easy to compute $h_k(x)$.*

2. 2nd-preimage resistance – *it is computationally infeasible to find a second input that has the same output as any specified input, i.e., given x, to find a 2nd-preimage $x' \neq x$ such that $h(x) = h(x')$.*

3. collision resistance – *it is computationally infeasible to find any two distinct inputs x, x' that hash to the same output, i.e., $h(x) = h(x')$. A stronger form of collision resistance is to require $\forall x \neq x', h(x) \neq h(x')$.*

4. commutative hashing – *given two instances of a keyed hash function h_k parameterized with 2 different keys k_1 and k_2 and an input x, $h_{k_1}(h_{k_2}(x)) = h_{k_2}(h_{k_1}(x))$.*

5. key non-recovery – *given one or more text-hash pairs $(x_i, h_k(x_i))$ for the same k, it must be computationally infeasible to recover k.*

A commutative public key encryption scheme such as Pohlig-Hellman can be used to generate a hash function satisfying all our requirements. Each party generates its own keypair (E_i, D_i). The length in bits for the keypair is commonly agreed upon and known to all parties (1024 bits is common today). The hash function h_{k_i} is simply encryption with E_i. The decryption keys are not needed.

Algorithm

There are three stages to the protocol: hashing, initial intersection, and final intersection. In the hashing stage, each party hashes (encrypts) its own items with its

own key. The parties then pass the set to their neighbor to be hashed. This continues until all sets have been encrypted by all keys. Since hashing is commutative, the resulting values from two different sets will be equal if the original values were the same (i.e., the item was present in both sets). Collision resistance ensures that this will happen only if the original values were the same.

In the initial intersection stage, each party sends the set it has to all parties except its right neighbor, the original owner of the set. All parties then compute the intersection of all the sets received. If the size of this intersection is less than a user-determined threshold r, the protocol is aborted. This is to avoid probing attacks; an attempt to probe for the existence of a particular item in the set gives at best a probability $1/r$ estimate of its existence. (A party whose items are all present in the intersection also learns of the existence of particular items, but this is an unavoidable artifact of the result. Revealing that an individual is one of a sufficiently large group is often viewed as sufficient protection of privacy [81]; this also gives a floor for r.) In the final stage, the intersections are sent to the right neighbor. The final result is the size of the intersection of the received set with the one generated in the initial intersection stage. A complete description of the algorithm is given as Protocol 17.

We now give additional clarification of the algorithm.

Hashing In this stage the sets of all the parties are hashed by all parties. Since each party hashes with a key known only to itself, and the order of items is randomly permuted, no other party can determine the mapping performed by the previous party.

Initial Intersection In this stage, every party finds the intersection of all sets except its own. The hashing prevents learning the actual values corresponding to the hashed items received. The reason a site does not get its own set is to prevent probing attacks: a site could initially generate a singleton set to probe if that item existed at another site, i.e., if the intersection of its set with that

Require: $k > 2$ sites, each having a local set S_i
Require: Maximum local set size m, and threshold r used to protect against probing
for all sites i {Parallel operations} **do**
Generate the hash key E_i
for $j = |S_i|$ to m **do**
$S_i \leftarrow S_i \cup \{\text{prefix\_not\_in\_}U.i.j\}$ {Pad S_i with items that will be unique to that site and cannot contribute to the intersection.}
end for

{Stage 1 – Hashing}
$M \leftarrow EncryptAndPermute(S_i, E_i)$
send M to site $i + 1 \bmod k$

for $p = 1 \ldots k - 2$ **do**
$M' \leftarrow$ receive from site $i - 1 \bmod k$
$M'' \leftarrow EncryptAndPermute(M', E_i)$
send M'' to site $i + 1 \bmod k$
end for

$M' \leftarrow$ receive from site $i - 1 \bmod k$
$M'' \leftarrow EncryptAndPermute(M', E_i)$
send M'' to all sites except site $i + 1 \bmod k$

{Stage 2 – Initial Intersection of sets and check for probing}
$TS_j \leftarrow$ receive from site j, $j \neq i - 1$
$TS'_i \leftarrow \cap_{p=0, p \neq i-1}^{k-1} TS_p$
if $|TS'_i| < r$ **then**
broadcast ABORT {Detect/prevent probing}
else

{Stage 3 – Final Intersection to compute Final Result}
Send TS'_i to party $i + 1 \bmod k$
Receive $TS'_{i-1 \bmod k}$ from party $i - 1 \pmod k$
$TS'_i \leftarrow TS'_i \cap TS'_{i-1 \bmod k}$
return $|TS'_i|$
end if
end for

Protocol 17: Securely computing size of intersection set

Function EncryptAndPermute(Set M, Key E_k)

Require: M is the input array to be hashed, E_k is the hash key

$C \leftarrow \emptyset$

for all $j \in M$ **do**

$C \leftarrow C \cup \{E_k(j)\}$;

end for

randomly permute C to prevent tracking values

return C

of another site is empty or of size 1. Aborting prevents probes for sets of size less than r.

This also shows the reason that we require $k > 2$ parties. With two parties, no intersection could be performed without access to the hashed values of one's own set. This prevents the probe detection/prevention.

Final Intersection Each party sends the remaining piece of the puzzle to its left neighbor. This enables all parties to compute the final intersection and find the final result, viz. the cardinality of the total intersection set.

The collision resistance property of the hash function ensures that no collisions can occur. Thus the algorithm clearly generates the correct result for the size of the intersection set. A similar technique was used by Agrawal et al. [3] to compute intersection, equijoin, intersection size and equijoin size. However, their technique is limited to two parties and to semi-honest adversaries.

4.1.3 Communication and Computation Analysis

Communication protocols of this type are generally analyzed either based on the communication cost or number of encryptions performed. The encryption cost is entirely contained in Stage 1: Each party hashes every item once, giving $k^2 * m$

encryptions. The inherent parallelism gives a factor of k speedup, for a computation time cost of $k * m$.

The communication cost for a single party is as follows. In stage 1, a set of size m is transmitted in each of the first $k-1$ rounds. In round k, a set is sent to $k-2$ of the other parties. Assuming no multicast this requires $2k-3$ messages of $m * \textit{hashed item size}$ bits. Stage 2 requires no transmission (except possibly a broadcast ABORT). In Stage 3, a message is sent containing the intersection of all but one party. This message is at most size m hashed items, but would typically be closer to the lower bound of $|S|$ items. Thus, neglecting abort, each site sends $2k-2$ message of at most size $m * \textit{hashed item size}$ bits.

The entire protocol is symmetric, so all of the parties transmit equal amounts of data. So, to calculate the total communication cost, we simply multiply the single party cost by k. Thus, the upper bound on the total communication cost of the algorithm is

$$\begin{aligned} k * (2k-2) &= O(k^2) && \text{messages} \\ k * (2k-2) * m * \textit{hashed item size} &= O(k^2 m) && \text{bits} \\ k & && \text{rounds} \end{aligned}$$

The factor of m can be reduced to $|S_i|$, at the cost of revealing the size of each site's set. This is done by skipping the padding to size m step at the beginning of Protocol 17. As this is likely to be more sensitive than the sizes of intersections, we have detailed the more secure version.

4.1.4 Security Analysis

The intersection algorithm described above clearly calculates the size of the intersection set without revealing what items are present in any set. However, it is not quite secure based on the standard of Definition 2.3.1. In addition to the size of the complete set intersection, the parties can learn the size of the intersection of subsets of the entire group. Specifically, for any set $C \subseteq \{0, \ldots, k-1\}^*$ such that $i \notin C, |C| \geq 2$, party i can compute $|\cap_{j \in C} S_j|$.

By acknowledging that these subset intersection sizes are revealed, we can prove that nothing else is disclosed by Protocol 17. We effectively augment the result with the "leaked information". We then show how to build a polynomial time simulator for the view of every party using only the augmented output for that party and their own input.

Theorem 4.1.1 *Protocol 17 privately computes the size of the intersection set* $|S| = |\cap_{p=0}^{k-1} S_p|$. *Site* i *learns at most* $|\cap_{p\in C} S_p|, \forall C \subseteq \{0,\ldots,k-1\}^*$ *such that* $i \notin C, |C| \geq 2$. *If* $|\cap_{p=0,\ldots,i-2,i,\ldots,k-1} S_p| \geq r$ *it learns that value and the final result* $|S|$.

Proof. Since the protocol is symmetric, proving that the view of one party can be simulated with its own input and output suffices to prove it for all parties. We now show how to simulate the messages received by party i. Given these, it uses its own input and hash key to simulate the rest of its view by running the appropriate parts of Protocol 17. The protocol consists of three stages. The initialization phase can be done based on i's own input, so we begin with the messages received in Stage 1.

Stage 1 At each step of this stage, party i receives a new local set from part $i-1 \bmod k$. However, each item in each of these sets has been hashed (encrypted). The preimage resistance, collision resistance, and key non-recovery properties combine to ensure that the distribution of the hashed values (as the key changes) is independent of the distribution of the data. This allows us to state that the values in these sets are computationally indistinguishable from numbers chosen from a uniform random distribution. We can simulate the received set M' by randomly choosing m values uniformly distributed over the domain of the hash function E.

This allows us to simulate a single M'. Each M' seen by i is hashed by a different set of keys, i.e., the first is hashed with $E_{i-1}(x)$, the second by $E_{i-1}(E_{i-2}(x))$, etc. (For brevity we drop the $\bmod k$, it should be assumed in all index computations.) Regardless of any relationship between items in S_{i-1} and S_{i-2}, the different hashes

and permutation ensure that i sees no relationship. Therefore, the argument that randomly choosing values allows us to simulate M' extends to the set of all $k-1$ M' sets.

Stage 2 Party i now receives $k-1$ sets TS_j, corresponding to the fully hashed sets from all but party i. TS_i is the last M'' generated in Stage 1, and is simulated with the final M'' generated in the simulation of Stage 1. While the hashing/encryption guarantees that any single item in these sets is equally likely to come from anywhere in the domain of E, we can not simply generate random values for the other TS_j. The problem is the need to simulate intersections. Since all parties have hashed all items, and because the hashing is commutative, if S_g and S_h have an item in common, then TS_{g-1} and TS_{h-1} will also have an item in common.

To simulate this, we take advantage of knowing $|\cap_{p\in C} S_p|$. The simulator first generates a directed acyclic graph from these intersection sizes, and uses this to calculate the number of items that should be common at each node. It then does a breadth-first traversal, generating the required number of items at each node and placing the items in the leaf sets reachable from that node. "Generating" an item happens in two ways: When TS_i is one of the reachable leaves, an item is chosen (without replacement) from TS_i. Otherwise, a random value from the domain of E is used.

A detailed description of this process is given as Simulator 1. A demonstration of the simulation algorithm for three parties is given in Figure 4.1.

Party i can now generate TS'_i and determine if it should send an ABORT message. It also knows if it should receive an ABORT, as this is part of the result. If the result is not an ABORT, we must simulate Stage 3.

Stage 3 Party i now receives TS'_{i-1}. Since $|TS'_{i-1}| \geq r$, i is allowed to learn the size of this set (i is not probing). This set is simulated by choosing $|S|$ items from TS'_i, and randomly choosing $|TS'_i| - |S|$ from the domain of E. The encryption arguments

Generate the hierarchical directed graph G connecting all of the intersection sets to their immediate descendents.

- $\{0, \ldots, i-1, i+1, \ldots, k-1\}$ is the root,
- All sets with $k-2$ elements are level 2,
- ...
- All 2-sets are at level $k-2$,
- $\{1\}, \ldots, \{i-1\}, \{i+1\}, \ldots, \{k-1\}$ (i.e. sets for all parties other than i) are leaves at level $k-1$,
- An edge is added from p to c if c is a subset of the set represented by p obtained by removing one number.

Each node n is assigned the size of the corresponding intersection set $|\cap_{p \in n} S_p|$, nodes at level $k-1$ are assigned m.

for $l = 1..k-1$ **do**
  for all nodes p at level l **do**
    if $i \in p$ **then**
      $items \leftarrow$ remove $p.size$ items from M''
    else
      $items \leftarrow$ remove $p.size$ items from (domain of $E - M''$)
    end if
    for all $TS_j, j \in p$ **do**
      $TS_j \leftarrow TS_j \cup items$
    end for
    for all c child of n **do**
      $c.size \leftarrow c.size - p.size$
    end for
  end for
end for

Simulator 1: GenInput: Generating Input Sets for Party i

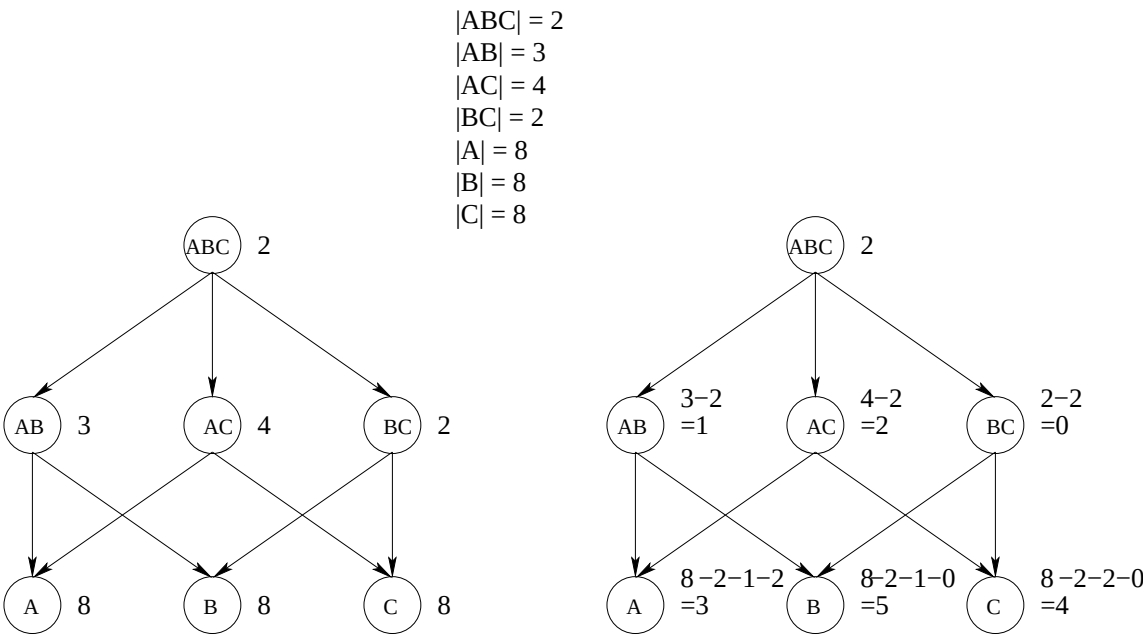


Figure 4.1. Building the simulated input set

used for Stage 1 still hold to protect the *value* of the items, and the known "leaked" information is sufficient to perform the simulation.

Definition 2.3.1 requires that this simulation be polynomial time. Stages 1 and 3 are clearly polynomial. Stage 2 requires construction and breadth-first traversal of a graph consisting of all powersets of $k-1$ nodes. The graph is exponential in k, an apparent conflict. However, the requirement is that the simulation be polynomial in the size of the input m, we can treat k as fixed. In the graph traversal of Algorithm 1, we generate $k*m$ items to fill the leaves. Since generating each item (either choosing an item from TS_i or randomly generating one) is polynomial, and we perform one such operation for each item in the input, the simulation is polynomial. □

The definitions we have given are for the semi-honest model: parties follow the protocol, but may try to learn additional details from what they see during the execution. The malicious model for Secure Multiparty Computation looks at the case where parties may not play by the rules. Protocol 17 does not quite meet malicious standards, as a malicious party can cause incorrect results without detection. From the proof of Theorem 4.1.1 we can see that the disclosure properties *do* hold in the face of a malicious party. No party sees information hashed with the same set of keys twice, so altering an outgoing message to learn how it is hashed would not enable learning anything from an incoming message. This is true as long as there is no collusion between parties. However, if two parties collude, they could jointly mount a probing attack by returning each party's fully hashed items to that party.

4.2 A More Efficient Set Intersection Protocol

The symmetric algorithm we have presented in section 4.1 is simple and proven effective at controlling the disclosure of information. We now present a more complex variant that gives asymptotically improved performance in number of rounds, number of messages, and total number of bits transmitted. It also provides a prac-

tical improvement in information disclosure; the same total information is disclosed, but each party only sees a piece of that information.

4.2.1 Problem Definition

The problem to be solved is the one defined in Section 4.1.1.

4.2.2 Algorithm / Protocol

The key insight behind this protocol is to overlap the hashing and intersection phases. Note that any arbitrary parenthesization of the intersection expression still gives the same result.

$$S_0 \cap S_1 \cap \ldots \cap S_k$$

$$\equiv$$

$$(\ldots(S_0 \cap S_1) \cap S_2) \cap \ldots \cap S_k)$$

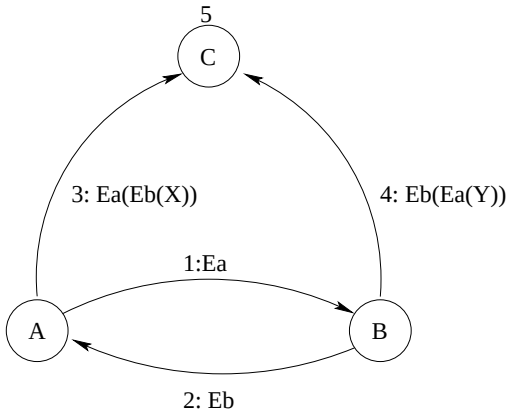
$$\equiv$$

$$(S_0 \cap S_1) \cap (S_2 \cap S_3) \cap \ldots \cap (S_{k-1} \cap S_k)$$

The second observation is that it is not necessary to hash every set with all keys before intersecting the sets. Any time two items have been hashed by the same set of keys, they can be tested for equality. With careful ordering of the hashing we can perform the innermost intersections early. Repeating this at each level, the intersections can be carried out in the form of a binary tree, reformulating the intersection as

$$(\ldots(\log k) - 1 \ldots((S_0 \cap S_1) \cap (S_2 \cap S_3)) \cap \ldots \cap (S_{k-1} \cap S_k) \ldots \log k \ldots)$$

The difficulty is with carrying out intersections of two sets. As pointed out in Section 4.1.2, a party that sees the hashed results of its own set can probe, requiring at least three parties to perform the intersection. The solution is to use a party from the opposite side of the tree as this third party. Each party hashes its set and sends it to its "intersection partner". The partner hashes it, and both send them to the



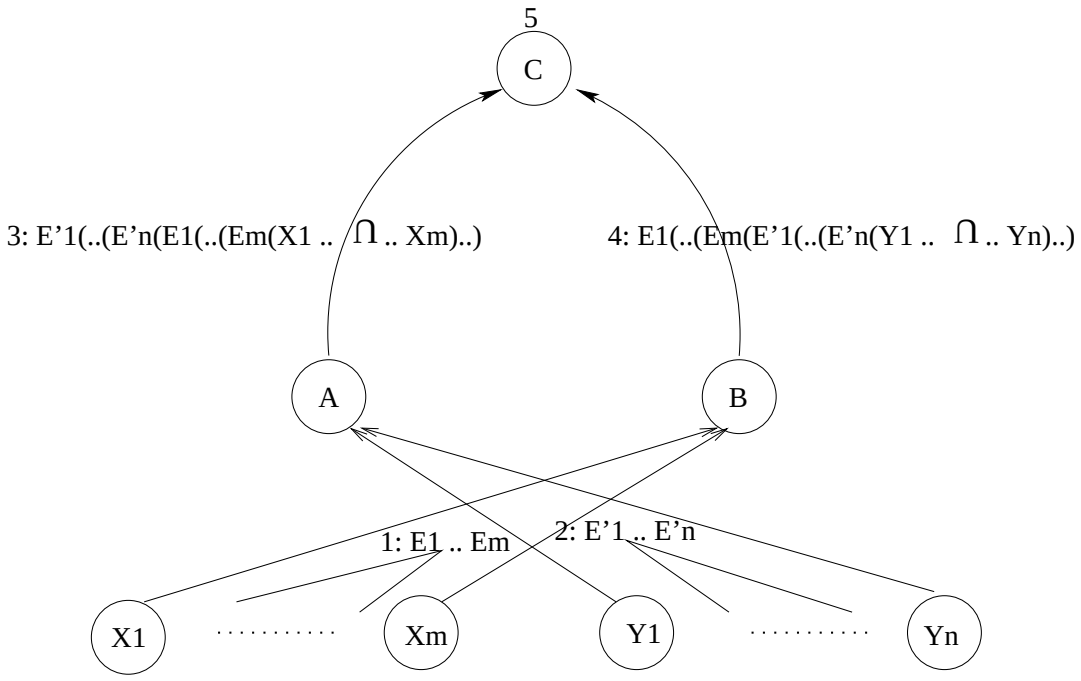
1: A sends its key, Ea, to B
2: B sends its key, Eb, to A
3: A encrypts and sends its set to C
4: B encrypts and sends its set to C
5: C intersects both sets it receives to get the result

Figure 4.2. A single binary set intersection using a third party

parent third party. The third party performs the intersection. An example of this is given in Figure 4.2.

Each parent now has the intersection of the sets of its children, hashed with the keys of its children. To compute the intersection with its sibling, it must hash these items with the keys of its sibling's children. Since it does not see any items from its sibling's children, it gains no information by having these keys. Once this is complete, the siblings can send their intersections up the tree to compute the next level of intersection. This process is repeated until the root is reached, giving the final intersection. This process is illustrated in Figure 4.3. The complete algorithm is given in Protocol 18, and depicted graphically in Figure 4.4.

Interior nodes are assigned so that no node is on the path from itself (as a leaf) to the root. This avoids any information leak based on knowing the size of the intersection of one's own set with any subset of other nodes. (The root is the only node to learn the size of a subset containing its own set, but this subset contains



1: m leaves send their encryption keys E1 .. Em to B
2: n leaves send their encryption keys E'1 .. E'n to A
3: A applies all encryption keys to its set and sends it to C
4: B applies all encryption keys to its set and sends it to C
5: C intersects what it has received from both A and B to get the result

Figure 4.3. Higher-level set intersection

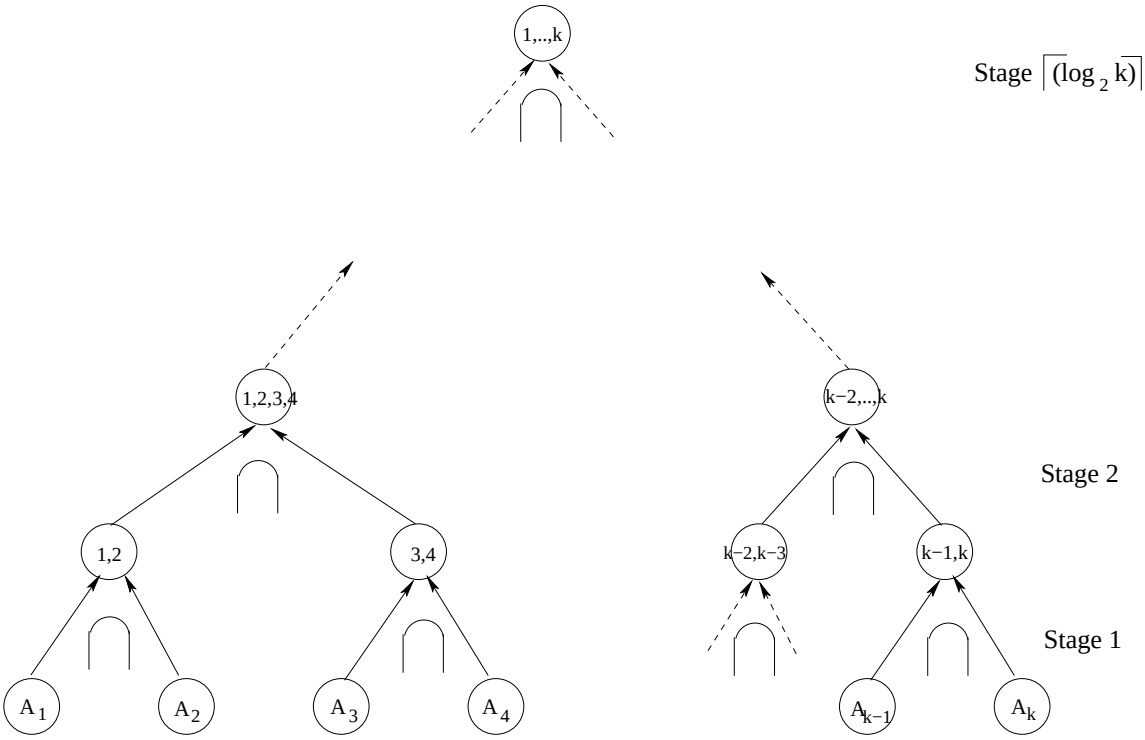


Figure 4.4. A more efficient protocol

half the nodes.) If the number of parties is a power of two, they form a complete tree with each party acting as both a leaf and at most once as an interior node. Only $k-1$ parties are needed to act as interior nodes. If the tree is not complete, we can still make such an assignment provided $k \geq 4$. Leaf nodes whose sibling is not a leaf hash their own set with their own key, and with the keys of their sibling's children. The protocol then proceeds as normal. This eliminates the need for one interior node from the leaves on the other side of the tree. Balancing the tree with respect to these singleton nodes enables an assignment such that no node is on its own path to the root.

Since the sets each parent receives are hashed with the same set of keys, the commutative hashing property guarantees that the intersection of those sets will be of the correct size. Associativity of intersection ensures that the order does not

Require: $k > 3$ sites numbered $0 \ldots k-1$, each having a local set S_i
Require: Maximum local set size m, threshold r used to protect against probing
for all sites i **do**
Generate a binary tree with k leaves at levels $\log_2(k)$ and $\log_2(k) - 1$. Even leaves are to the left of root, odd to the right. If the tree is not complete, the lowest numbered nodes form the lowest level.
Number non-leaf nodes as follows. 0 is root. The left-hand side is numbered with odd numbers using a preorder traversal, the right side with even numbers.
{Each site now has an identical view of the tree.}
Generate the hash key E_i
for $j = |S_i|$ to m **do**
$S_i \leftarrow S_i \cup \{\text{prefix\_not\_in\_}U.i.j\}$ {Pad S_i with unique items.}
end for
$M \leftarrow EncryptAndPermute(S_i, E_i)$
if the sibling of i is a leaf **then**
Send E_i to sibling
$E \leftarrow$ receive from sibling
$M \leftarrow EncryptAndPermute(M, E)$
end if
Send $\{E_i\}$ to the sibling of the parent of leaf i
if the sibling of i is a leaf **then**
Send M to parent
end if
end for
{Nodes now act based on their "interior" position in the tree. For nodes whose sibling is not a leaf, the interior and leaf positions are the same.}
for all sites $i > 0$ **do**
$KeySet_i \leftarrow$ receive key set from left child of interior node i of sibling
$KeySet_i \leftarrow KeySet_i \cup$ receive key set from right child of sibling
if the sibling of i is a leaf {site i is also an interior node} **then**
$M_l \leftarrow$ receive from left child
$M_r \leftarrow$ receive from right child
$M \leftarrow M_l \cap M_r$
end if
if $|M| < r$ **then**
Broadcast ABORT {Potential Probe}
else
for all $E \in KeySet_i$ **do**
$M \leftarrow EncryptAndPermute(M, E)$
end for
if parent of i is not 0 **then**
Send $KeySet_i$ to sibling of parent of interior node i
end if
Send M to parent of interior node i
end if
end for
if site is 0 **then**
$M_l \leftarrow$ receive from left child, $M_r \leftarrow$ receive from right child
Broadcast result $|M_l \cap M_r|$
end if

Protocol 18: Tree-based protocol for computing size of set intersection

affect the result. An inductive argument shows that the protocol generates the correct result.

Protocol 18 demonstrates another optimization. Instead of sending sets to be hashed by other sites, a site sends its key. The numbering of the tree ensures that no site sees items hashed with any key it knows (except root, which knows only its own key and sees items hashed with that plus several others.) Thus, in the absence of collusion, sending a key gives the receiver no additional information.

4.2.3 Communication and Computation Analysis

At first glance, the encryption cost appears similar to Protocol 17. Every item in the final intersection is hashed with every key. However, duplicate items are filtered out at higher levels. This reduces the number of items to be hashed. Leaf nodes perform $2m$ encryptions. Lowest level parent nodes perform 2 additional encryptions on every item in the intersection of their children; at most $2m$. The next level must hash with 4 keys. The level below the root ends up with $k/2$ keys. Multiplying by the number of nodes at each level gives $k * m$ encryptions at each level, for a worst case total of $(2k + k\log_2(k)) * m$ encryptions. Parallelism gives some benefit, but since the upper levels perform more encryptions the encryption time is still $O(k*m)$.

More important is the savings in number of rounds and messages. The number of rounds of messages is one for the leaf key exchange, and $\lceil \log_2(k) \rceil$ rounds of sending key sets and hashed sets up the tree. The key exchange requires each leaf to send one message of the number of bits in the hash key. (The one or two "extra" nodes whose sibling is not a leaf are spared sending this message.) For each edge in the tree there is one "hashed set" message and (except for the root) a corresponding key set message, for a total of $2k - 4$ messages. Each hashed set message is at most $m * \textit{hashed item size}$ bits. The key set messages grow as they grow up the tree; a total of $k * \textit{hash key size}$ bits are sent at each level.

The overall communication cost of Protocol 18 is then:

$$3k - 4 = O(k) \quad \text{messages}$$

$$k \log_2(k) * \textit{hash key size} + 2(k-1)m * \textit{hashed item size} \approx O(k * m) \quad \text{bits}$$

$$\lceil \log_2(k) \rceil = O(\log k) \quad \text{rounds}$$

This is a substantial improvement over the $O(k^2)$ messages, $O(k^2 m)$ bits, and $O(k)$ rounds of Protocol 17.

The number of bits and number of encryptions is in practice a pessimistic estimate. It is likely that the size of intersections will be significantly smaller than m, and will rapidly approach $|S|$. While the asymptotic results do not change, the effect of parallelism on the encryption cost is likely to improve significantly, as the majority of encryptions occur only at the low levels. The set message sizes at higher levels will also shrink, although each key transmission message will grow. Thus, the effective total time to run the algorithm should be closer to $O(m + \log k)$ than $O(m \log k)$.

4.2.4 Security Analysis

Protocol 17 is symmetric, and reveals to all parties the intersections of any subset of items except its own. Protocol 18 reveals the same type of information, however, each site learns the intersection size of at most three subsets:

- $|\cap_{p \in descendants} S_p|$
- $|\cap_{p \in left\ descendants} S_p|$
- $|\cap_{p \in right\ descendants} S_p|$

The total information revealed is considerably less, $O(k)$ intersections as opposed to $O(2^k)$. In addition, the limited amount revealed to any party enables an assignment of parties to nodes based on trust and which specific intersections can be disclosed. This gives considerable flexibility in meeting specific security policy goals.

Theorem 4.2.1 *Protocol 18 privately computes the size of the intersection set* $|S| = |\cap_{p=0}^{k-1} S_p|$. *Each site learns the final result, and if it serves as an interior node in the tree it learns:*

- $|\cap_{p \in descendants} S_p|$
- $|\cap_{p \in left\ descendants} S_p|$
- $|\cap_{p \in right\ descendants} S_p|$

PROOF. The proof proceeds by simulating the view of a party i as it proceeds through Protocol 18. The simulator effectively runs Protocol 18, all we need to show is that the received messages can be simulated.

First, let us look at the case of an "extra" node: a leaf whose sibling is not a leaf. These nodes (at most two, one odd and one even) serve in the same spot as a leaf and interior node. If i is one of these nodes, it will receive the keys E_{i-2} and E_{i-4}. This is the only message it receives. Since these were not used to generate any information i will receive, they can be simulated by randomly generating keys for the hash function E. Protocol 18 generates the rest of the view for these nodes, except for receiving the final result. As the result is known to the simulator, generating it is trivial.

The remaining non-root nodes are slightly more complex, as they receive three sets of messages. The first message received is the key of their sibling in their position as a leaf. This is simulated by randomly generating a key for the hash function E. The next are the two sets of keys of their sibling's descendants based on their position as an interior node. These are simulated by randomly generating keys for the hash function E, the number of keys to generate is known from i's position in the tree. The final messages received by i are the intersection sets of i's left and right descendants. To simulate these, i takes advantage of three facts:

1. i knows the sizes $|\cap_{p \in left\ descendants} S_p|$ and $|\cap_{p \in right\ descendants} S_p|$ of the received sets,
2. i knows the number of items the two sets have in common $|\cap_{p \in descendants} S_p|$, and
3. i has *no* knowledge of the keys used to hash the items in the sets.

Using fact 2, the simulator for i generates $|\cap_{p \in descendants} S_p|$ items, by randomly selecting items from a uniform distribution over the domain of E, and places them in both the simulated M_l and M_r. Fact 1 is then used to complete M_l and M_r, by generating $|\cap_{p \in left\ descendants} S_p| - |\cap_{p \in descendants} S_p|$ to insert into M_l and $|\cap_{p \in right\ descendants} S_p| - |\cap_{p \in descendants} S_p|$ to insert into M_r.

The simulated sets M_l and M_r are now the same size as those seen in the actual protocol, and their intersection contains the same number of items as those in the actual protocol. Since i has no knowledge of the keys used to hash the items (fact 3), security of hashing/encryption guarantees values in hashed sets are computationally indistinguishable from values chosen from a uniform random distribution over the domain of the hash function E. Therefore, the simulated view is computationally indistinguishable from that seen by i during the actual execution of the protocol.

The argument for site 0, the root, is slightly different. The simulator is the same as other interior nodes. This site receives only E_2, the key of its sibling. It does see items hashed with E_2, as well as its own key E_0, so fact 3 does not hold. However, by the time it sees items hashed with E_0 and E_2, they have also been hashed with (at least) E_1. Since 0 does not know E_1, the computational indistinguishability argument still holds.

The simulator requires one key generation or selection of a random value from the domain of E for each item in the received messages. The number of items is bounded by the maximum set size m. Assuming key generation or random value selection is polynomial in the size of the input, and that Protocol 18 runs in polynomial time (see Section 4.2.3), the view seen by any site i can be simulated in time polynomial in the size of the input. □

Absent collusion, Protocol 18 is as effective as Protocol 17 with malicious parties. A malicious party can alter what it sends, however, since it never sees anything based on messages it has sent except the final output it can only gain information from the final result. This constitutes a probing attack, and the information gain possible is restricted by the minimum size threshold r. Site 0 does see information based on its

first set of messages, however the intermediate hashing and threshold test prevents it from gaining additional information from what it sent as a leaf.

Collusion poses a significant problem. Collusion between the parent of a leaf node and its right child can give it both the hash key E_l and the hashed set M_l of the left child. It can now probe for the existence of any item I in that set, by testing if $E_l(I) \in M_l$. Collusion with its own sibling or the sibling of its ancestors also gives it this key. Even if some sites were not trusted (i.e., they may collude with some other sites), it would often be possible to assign sites to tree nodes in such a way that the untrusted sites would not gain by colluding.

4.3 Algebraic Method for Computing Dot Product

4.3.1 Problem Definition

Consider two real-valued vectors $\vec{X}$ and $\vec{Y}$ of cardinality n, $\vec{X} = (x_1, \cdots, x_n), \vec{Y} = (y_1, \cdots, y_n)$. The scalar product (or dot product) of $\vec{X}$ and $\vec{Y}$ is defined as $\vec{X} \cdot \vec{Y} = \sum_{i=1}^{n} x_i * y_i$. If party A has the vector $\vec{X}$ and party B has the vector $\vec{Y}$, securely compute the scalar product $\vec{X} \cdot \vec{Y}$.

4.3.2 Protocol

Scalar product protocols have been proposed in the Secure Multiparty Computation literature [9], however these cryptographic solutions do not scale well to data mining problems. We give an algebraic solution that hides true values by placing them in equations masked with random values. The knowledge disclosed by these equations only allows computation of private values if one side learns a substantial number of the private values from an outside source. (A different algebraic technique has recently been proposed [46], however it requires at least twice the bitwise communication cost of the method presented here.)

We assume without loss of generality that n is even.

Step 1: A generates randoms $R_1 \ldots R_n$. From these, $\vec{X}$, and a matrix C forming coefficients for a set of linear independent equations, A sends the following vector $\vec{X}'$ to B:

$\langle x_1 + c_{1,1} * R_1 + c_{1,2} * R_2 + \cdots + c_{1,n} * R_n \rangle$

$\langle x_2 + c_{2,1} * R_1 + c_{2,2} * R_2 + \cdots + c_{2,n} * R_n \rangle$

$\vdots$

$\langle x_n + c_{n,1} * R_1 + c_{n,2} * R_2 + \cdots + c_{n,n} * R_n \rangle$

In step 2, B computes $S = \vec{X}' \cdot \vec{Y}$. B also calculates the following n values:

$\langle c_{1,1} * y_1 + c_{2,1} * y_2 + \cdots + c_{n,1} * y_n \rangle$

$\langle c_{1,2} * y_1 + c_{2,2} * y_2 + \cdots + c_{n,2} * y_n \rangle$

$\vdots$

$\langle c_{1,n} * y_1 + c_{2,n} * y_2 + \cdots + c_{n,n} * y_n \rangle$

But B can't send these values, since A would then have n independent equations in n unknowns $(y_1 \ldots y_n)$, revealing the y values. Instead, B generates r random values, $R'_1 \ldots R'_r$. The number of values A would need to know to obtain full disclosure of B's values is governed by r.

B partitions the n values created earlier into r sets, and the R' values are used to hide the equations as follows:

$\langle c_{1,1} * y_1 + c_{2,1} * y_2 + \cdots + c_{n,1} * y_n + R'_1 \rangle$

$\vdots$

$\langle c_{1,n/r} * y_1 + c_{2,n/r} * y_2 + \cdots + c_{n,n/r} * y_n + R'_1 \rangle$

$\langle c_{1,(n/r+1)} * y_1 + c_{2,(n/r+1)} * y_2 +$

$\quad \cdots + c_{n,(n/r+1)} * y_n + R'_2 \rangle$

$\vdots$

$\langle c_{1,2n/r} * y_1 + c_{2,2n/r} * y_2 + \cdots + c_{n,2n/r} * y_n + R'_2 \rangle$

$\vdots$

$\langle c_{1,((r-1)n/r+1)} * y_1 + c_{2,((r-1)n/r+1)} * y_2 +$

$\quad \cdots + c_{n,((r-1)n/r+1)} * y_n + R'_r \rangle$

$\vdots$

$\langle c_{1,n} * y_1 + c_{2,n} * y_2 + \cdots + c_{n,n} * y_n + R'_r \rangle$

Then B sends S and the n above values to A, who now has:

$$\begin{aligned} S =&(x_1 + c_{1,1} * R_1 + c_{1,2} * R_2 + \cdots + c_{1,n} * R_n) * y_1 \\ &+(x_2 + c_{2,1} * R_1 + c_{2,2} * R_2 + \cdots + c_{2,n} * R_n) * y_2 \\ &\vdots \\ &+(x_n + c_{n,1} * R_1 + c_{n,2} * R_2 + \cdots + c_{n,n} * R_n) * y_n \end{aligned}$$

Simplifying further and grouping the $x_i * y_i$ terms gives:

$$\begin{aligned} S =&(x_1 * y_1 + x_2 * y_2 + \cdots + x_n * y_n) \\ &+(y_1 * c_{1,1} * R_1 + y_1 * c_{1,2} * R_2 + \cdots + y_1 * c_{1,n} * R_n) \\ &+(y_2 * c_{2,1} * R_1 + y_2 * c_{2,2} * R_2 + \cdots + y_2 * c_{2,n} * R_n) \\ &\vdots \\ &+(y_n * c_{n,1} * R_1 + y_n * c_{n,2} * R_2 + \cdots + y_n * c_{n,n} * R_n) \end{aligned}$$

The first line of the right hand side can be succinctly written as $\sum_{i=1}^{n} x_i * y_i$, the desired final result. In the remaining portion, we group all multiplicative components vertically, and rearrange the equation to factor out all the R_i values, giving:

$$\begin{aligned} S =&\sum_{i=1}^{n} x_i * y_i \\ &+R_1 * (c_{1,1} * y_1 + c_{2,1} * y_2 + \cdots + c_{n,1} * y_n) \\ &+R_2 * (c_{1,2} * y_1 + c_{2,2} * y_2 + \cdots + c_{n,2} * y_n) \\ &\vdots \\ &+R_n * (c_{1,n} * y_1 + c_{2,n} * y_2 + \cdots + c_{n,n} * y_n) \end{aligned}$$

Adding and subtracting the same quantity from one side of the equation does not change the equation in any way. Hence, the above equation can be rewritten as follows:

$$
\begin{aligned}
S =& \sum_{i=1}^{n} x_i * y_i \\
&+\{R_1 * (c_{1,1} * y_1 + c_{2,1} * y_2 + \cdots + c_{n,1} * y_n) \\
&\quad + R_1 * R'_1 - R_1 * R'_1\} \\
&\vdots \\
&+\{R_{n/r} * (c_{1,n/r} * y_{n/r} + c_{2,n/r} * y_2 + \cdots + c_{n,n/r} * y_n) \\
&\quad + R_{n/r} * R'_1 - R_{n/r} * R'_1\} \\
&+\{R_{n/r+1} * (c_{1,n/r+1} * y_{n/r+1} + c_{2,n/r+1} * y_2 + \\
&\qquad \cdots + c_{n,n/r+1} * y_n) \\
&\quad + R_{n/r+1} * R'_2 - R_{n/r+1} * R'_2\} \\
&\vdots \\
&+\{R_{2n/r} * (c_{1,2n/r} * y_{2n/r} + c_{2,2n/r} * y_2 + \\
&\qquad \cdots + c_{n,2n/r} * y_n) \\
&\quad + R_{2n/r} * R'_2 - R_{2n/r} * R'_2\} \\
&\vdots \\
&\vdots \\
&+\{R_{(r-1)n/r+1} * (c_{1,(r-1)n/r+1} * y_{(r-1)n/r+1} + \\
&\qquad c_{2,(r-1)n/r+1} * y_2 + \cdots + c_{n,(r-1)n/r+1} * y_n) \\
&\quad + R_{(r-1)n/r+1} * R'_r - R_{(r-1)n/r+1} * R'_r\} \\
&\vdots \\
&+\{R_n * (c_{1,n} * y_1 + c_{2,n} * y_2 + \cdots + c_{n,n} * y_n) \\
&\quad + R_n * R'_r - R_n * R'_r\}
\end{aligned}
$$

Now A factors out the R_i from the first two components and groups the rest vertically, giving:

$$
\begin{aligned}
S \;=\; & \sum_{i=1}^{n} x_i * y_i \\
& + R_1 * (c_{1,1} * y_1 + c_{2,1} * y_2 + \cdots + c_{n,1} * y_n + R'_1) \\
& \vdots \\
& + R_{n/r} * (c_{1,n/r} * y_{n/r} + c_{2,n/r} * y_2 + \\
& \qquad \cdots + c_{n,n/r} * y_n + R'_1) \\
& + R_{n/r+1} * (c_{1,n/r+1} * y_{n/r+1} + c_{2,n/r+1} * y_2 + \\
& \qquad \cdots + c_{n,n/r+1} * y_n + R'_2) \\
& \vdots \\
& + R_{2n/r} * (c_{1,2n/r} * y_{2n/r} + c_{2,2n/r} * y_2 + \\
& \qquad \cdots + c_{n,2n/r} * y_n + R'_2) \\
& \vdots \\
& + R_{(r-1)n/r+1} * (c_{1,(r-1)n/r+1} * y_{(r-1)n/r+1} + \\
& \qquad c_{2,(r-1)n/r+1} * y_2 + \cdots + c_{n,(r-1)n/r+1} * y_n + R'_r) \\
& \vdots \\
& + R_n * (c_{1,n} * y_1 + c_{2,n} * y_2 + \cdots + c_{n,n} * y_n + R'_r) \\
& - R_1 * R'_1 - \cdots - R_{n/r} * R'_1 \\
& - R_{n/r+1} * R'_2 - \cdots - R_{2n/r} * R'_2 \\
& \vdots \\
& - R_{(r-1)n/r+1} * R'_r - \cdots - R_n * R'_r
\end{aligned}
$$

A already knows the n R_i values. B also sent n other values, these are the coefficients of the n R_i values above.

A multiplies the n values received from B with the corresponding R_i and subtracts the sum from S to get:

$$\begin{aligned} Temp &= \sum_{i=1}^{n} x_i * y_i \\ &\quad -R_1 * R'_1 - \cdots - R_{n/r} * R'_1 \\ &\quad -R_{n/r+1} * R'_2 - \cdots - R_{2n/r} * R'_2 \\ &\quad \vdots \\ &\quad -R_{(r-1)n/r+1} * R'_r - \cdots - R_n * R'_r \end{aligned}$$

Factoring out the R'_i gives:

$$\begin{aligned} &Temp = \\ &\quad \sum_{i=1}^{n} x_i * y_i \\ &\quad -(R_1 + R_2 + \cdots + R_{n/r}) * R'_1 \\ &\quad -(R_{n/r+1} + R_{n/r+2} + \cdots + R_{2n/r}) * R'_2 \\ &\quad \vdots \\ &\quad -(R_{((r-1)n/r)+1} + R_{((r-1)n/r)+2} + \cdots + R_n) * R'_r \end{aligned}$$

To get the desired final result (viz. $\sum_{i=1}^{n} x_i * y_i$), A needs to add the sum of the r multiplicative terms to Temp.

In step 3, A sends the r values to B, and B (knowing R') computes the final result. Finally B replies with the result.

Selection of $c_{i,j}$

The above protocol requires a matrix C of values that form coefficients of linear independent equations. The necessity of this is obvious from the fact that the equations are used to hide the data values. If any equation can be eliminated using less than half of the other equations, a linkage between less than $n/2$ of the unknowns is created.

Table 4.1
Communication cost

Rounds	*Bitwise cost*	
4	$2 * n * MaxValSz$	$O(n)$

MaxValSz = Maximum bits to represent any input value

With high probability, a coefficient matrix generated by a pseudo-random function will form linearly independent equations. This enables construction of the $c_{i,j}$ matrix by sharing only a seed and a generating function.

4.3.3 Communication and Computation Analysis

We first look at the computation cost. In step 1, A has to generate n random numbers, and perform n^2 multiplications and additions. In step 2, B performs n^2+n additions and multiplications. In step 3, A performs $n + r$ multiplications and additions (where $r << n$). In step 4, B performs r computations. Overall, it is quite clear that the protocol requires $O(n^2)$ additions and multiplications, which is quite low since these are simple arithmetic operations.

Computing support for each candidate itemset requires one run of the component scalar product protocol. The cost of each run (based on the number of items n is as follows: A sends one message with n values. B replies with a message consisting of $n + 1$ values. A then sends a message consisting of r values. Finally B sends the result, for a total of four communication rounds. The bitwise communication cost is O(n) with constant approximately 2 (assuming r is constant). This is summarized in Table 4.1.

There is also the quadratic cost of communicating the $c_{i,j}$ values. However, this cost can made constant by agreeing on a function and a seed value to generate the values.

Table 4.2
Security analysis of protocol

	Protected values	*Number of randoms generated*	*Total number of unknowns*	*Number of equations revealed*
A	$x_1 \cdots x_n$	n	$2n$	$n + r$
B	$y_1 \cdots y_n$	r	$n + r$	n

4.3.4 Security Analysis

The security of the scalar product protocol is based on the inability of either side to solve k equations in more than k unknowns. Some of the unknowns are randomly chosen, and can safely be assumed as private. However, if enough data values are known to the other party, the equations can be solved to reveal all values. Therefore, the disclosure risk in this method is based on the number of data values that the other party might know from some external source. Table 4.2 presents the number of unknowns and equations generated. This shows the number of data values the other party must have knowledge of to obtain full disclosure.

4.4 Cryptographic Method for Computing Boolean Dot Product

This section presents a purely cryptographic primitive for computing the dot product for boolean vectors. To be precise, we show how to compute the number of 1s in the logical AND vector of several boolean vectors.

4.4.1 Problem Definition

Let k be the total number of parties with the parties being $P_1, P_2 \cdots, P_k$. Each party has a corresponding n dimensional vector X_i. I.e.,

P_1 has the vector $\vec{X}_1 = (x_{11}, \ldots, x_{1n})$

P_2 has the vector $\vec{X}_2 = (x_{21}, \ldots, x_{2n})$

$\vdots$

P_k has the vector $\vec{X}_k = (x_{k1}, \ldots, x_{kn})$

where all the x_{ij} are boolean (either 0 or 1).

Thus, the problem can be defined as follows:

Component Multiplication of k n-dimensional vectors $\vec{X}_1 = (x_{11}, \ldots, x_{1n})$, $\vec{X}_2 = (x_{21}, \ldots, x_{2n})$, $\cdots$, $\vec{X}_k = (x_{k1}, \ldots, x_{kn})$ is defined as $\vec{X} = \prod_{i=1}^{k} \vec{X}_i$ ie. $\forall_{i=1}^{n} X_i = \prod_{j=1}^{k} X_{ji}$

Now, we wish to calculate the sum of the elements of the resulting vector, $Sum = \sum_{i=1}^{n} X_i$ The final step is to check if this sum is greater than the threshold t, ie. $Sum > t$? We now show how to compute Sum.

4.4.2 Generic Encryption System

The protocol described below requires a homomorphic probabilistic encryption system. The generic system used can be described as below (the presentation is patterned from [86]):

- A security parameter s. This is used to derive several finite domains ($R(s)$, $X(s)$, $Y(s)$) which are identified with initial subset of integers. Thus we use $R(s)$ for $\{x : 0 < x < r(s)\}$, $X(s)$ for $\{x : 0 \leq x < x(s)\}$ and similar notation for $Y(s)$.

- A public probabilistic encryption function, $f : R(s) \times X(s) \rightarrow Y(s)$, and a private decryption algorithm $g : Y(s) \rightarrow X(s)$ such that

$$(\forall (r, x) \in R(s) \times X(s)) \quad g(f(r, x)) = x \tag{4.1}$$

Note that the existence of a decryption algorithm implies that the function is injective with respect to its second parameter, that is, for $(r_1, x_1), (r_2, x_2) \in R(s) \times X(s)$, if $f(r_1, x_1) = f(r_2, x_2)$, then $x_1 = x_2$

We also require several additional important properties, which are described as follows:

1. The encryption function should be homomorphic, that is:

   $\forall (r_1, x_1), (r_2, x_2) \in R(s) \times X(s)$,

   $f(r_1, x_1) f(r_2, x_2) = f(r_3, x_1 + x_2 \bmod x(s))$

   where r_3 can be computed in polynomial time from r_1, r_2, x_1 and x_2.

2. We ask that the encryption function have *semantic security*. Informally, this means that for a polynomially bounded adversary, the analysis of a set of ciphertexts does not give more information about the cleartexts than what would be available without knowledge of the ciphertexts. [42] provides a formal definition.

3. As a result of the prior properties, one more can be deduced. There exists a "hiding" function $hide : R(s) \times Y(s) \rightarrow Y(s)$, depending only on the public parameters of the system and such that:

   $\forall (r, x) \in R(s) \times X(s), \forall s \in R(s)$,

   $hide(s, f(r, x)) = f(sr' \bmod r(s), x)$

   where r' can be computed in polynomial time from r, x. Indeed, *hide* can be defined by $hide(s, x) = f(s, 0) * x$.

4. Finally, we ask that the domain and range of the system are suitably high (to compute the required sum)

Several real encryption systems satisfy all of the properties required above. Examples are the Goldwasser-Micali cryptosystem [14], the Benaloh cryptosystem [12], the Naccache-Stern cryptosystem [65] and the Okamoto-Uchiyama cryptosystem [68].

Any of these cryptosystems, excepting the Goldwasser-Micali cryptosystem (since it is limited to arithmetic mod 2) can be used for our purpose.

4.4.3 Algorithm

We assume that the parties jointly decide on one of the suitable cryptosystems. The parties also randomly order themselves into a ring. To simplify the presentation, we assume that this order is the canonical order $P_1, \ldots, P_k$. In general, any order is acceptable. The first party, P_1, generates a public key E and private key D for the cryptosystem decided on above. The public key, E, is broadcast to all the other parties. The private key, D, is secret and known only to P_1. The basic idea of the algorithm is as follows: for each bit, $x_{1,i}$, in its vector, P_1 generates a random encryption of that bit ($M_i = E(r, x_{1,i})$, where r is randomly chosen from $R(s)$). P_1 sends M_i to P_2. Parties $P_2, \ldots, P_{k-1}$ act as follows: When Party P_j receives a message M_i from P_{j-1}, if its own bit is one (i.e., $x_{j,i} = 1$), P_j simply hides the message it receives (computes f(s,0)*x) and sends the hidden message on to the next party. However, if its bit is 0, P_j then sends a random encryption of 0 to the next party. Party P_k follows the same computation, so that it has a random encryption of either 0 or 1 (it does not know which). Now, P_k multiplies all of the encryptions it has together. Due to the homomorphic property of the encryption, this results in the sum of all of the component bits. P_k sends this final encrypted result back to P_1. P_1 decrypts this message to get the sum, Sum. It can now check whether $Sum > t$, which gives the required result. Protocol 19 gives the complete details of the algorithm.

Proof of Correctness

Theorem 4.4.1 *Algorithm 19 correctly computes Sum.*

Require: k parties $P_1, \ldots, P_k$, n-dimensional vectors

1: P_1 generates a public and private key pair E, D for the homomorphic encryption system agreed upon.
2: P_1 broadcasts the public key E to all other parties.
3:
4: **for** $i = 0; i < n; i++$ {For each bit} **do**
5: P_1:
6: Compute $M_{1,i} = E(r, x_{1,i})$, (r randomly chosen from R(s))
7: Send $M_{1,i}$ to P_2
8:
9: $P_j, j = 2, \ldots, k-1$:
10: **if** $x_{j,i} = 0$ **then**
11: Compute $M_{j,i} = hide(r, M_{j-1,i})$, ($r$ randomly chosen from R(s))
12: **else**
13: Compute $M_{j,i} = E(r, x_{ji})(= E(r, 1))$, ($r$ randomly chosen from R(s))
14: **end if**
15: Send $M_{j,i}$ to P_{j+1}
16:
17: P_k:
18: **if** $x_{k,i} = 0$ **then**
19: Compute $M_{k,i} = hide(r, M_{k-1,i})$, ($r$ randomly chosen from R(s))
20: **else**
21: Compute $M_{k,i} = E(r, x_{k,i})(= E(r, 1))$, ($r$ randomly chosen from R(s))
22: **end if**
23: **end for**
24: $P_k : EncSum \leftarrow \prod_{j=1}^{n} M_{k,j}$
25: P_k : Send $EncSum$ to P_1
26: P_1: Compute $Sum = D(EncSum)$

Algorithm 19: Computing the boolean dot product, Sum

PROOF. All of the bits in the vector undergo the same operations. Thus to prove the correctness of the entire algorithm, it is sufficient to prove that a single bit of the result is the correct componentwise multiplication of the corresponding bits of the vectors $\vec{X_1}, \ldots, \vec{X_k}$ (i.e., we just need to prove that $D(m_{k,i}) = x_{1,i} * \ldots * x_{k,i}$).

Observe that if $x_{j,i} = 0$, then party P_j sends forward a random encryption of 0. When P_j has 1, it simply sends forward an obfuscated form of the message it receives (with P_1 sending an encryption of 1 to begin with). Thus $M_{k,i} = E(r, 1)$ if and only if $\forall j, x_{j,i} = 1$. Other wise $M_{k,i} = E(r, 0)$, for some $r \in R(s)$. Now, due to homomorphic property of the encryption, multiplying all of the $M_{k,i}$ together gives the encryption of the sum. Thus the decryption of $EncSum$ correctly gives the required sum, Sum. □

4.4.4 Communication and Computation Analysis

The entire protocol is quite efficient. P_1 broadcasts the key E to all other parties. Each party also sends the entire (encrypted) vector to the next party once. P_k finally sends the encrypted sum back to P_1. Thus the total communication cost is $(k-1)*keysize+(k*n+1)*encrypted\_msg\_size = O(kn)$ bits, and $k-1+k = 2k-1$ messages (assuming the entire vector can be sent off as a single message.

In terms of computation, every party has to perform n encryptions (one for each bit in its vector), p_k has to perform n multiplications and finally P_1 has to perform 1 decryption to get the final result. Thus, there is a total of kn encryptions and 1 decryption.

We ran tests on a SUN Blade 1000 workstation with a 900 Mhz processor and 1 gig of RAM. A C implementation of the Okamoto-Uchiyama [68] encryption system was tested. The key size was fixed at 1152 bits. The computation time required for different values of n are summarized in Table 4.3. The encryption/decryption cost approximately linearly increases with the number of items. The cost of multiplication is much lower than the cost of decryption. Using this table, it is very easy to estimate

	100	1000	10000	100000
encrypt	1.33s	13.06s	2.2min	21.5min
decrypt	2.11s	20.80s	3.5min	35.3min

Table 4.3
Computation time required for encryption/decryption

the actual time required for different number of parties and different vector sizes. For example, 5 parties with vectors of size $100,000$ would require approximately 150 minutes. The time required would be significantly lower with smaller key sizes and with use of special purpose encryption hardware.

4.4.5 Security Analysis

We now give a proof of security for the entire protocol.

Theorem 4.4.2 *Protocol 19 computes the required sum, Sum while revealing nothing to any site other than its input and the final output.*

PROOF. A simulator is presented for the view of each party. We only show how to simulate the messages received. The rest of the proof trivially follows from this.

P_1: The only message received is on line 25. Since the final result (Sum) is known to P_1, it simply generates a random encryption of this to simulate the message it receives (choose a random r from $R(s)$ and compute $E(r, Sum)$. This is computationally indistinguishable from the message it receives since the only thing different is the choice of random r.

P_2: The only messages received are on line 2 and 7. The public key E is simulated simply by randomly choosing a key E over the space of possible keys. The message $M_{1,i}$ can be simulated by randomly choosing a bit b (0 or 1), uniformly choosing

a random r from $R(s)$, and computing $E(r, b)$. The semantic security property of the encryption system guarantees that no advantage or information can be gained from the ciphertext resulting from the encryption algorithm (even while knowing the public key, as long as the private key is secret). In other words, it is not computationally possible to distinguish between the encryption of a 0 or a 1 when r is randomly chosen with uniform probability over $R(s)$. Thus, by selecting a random r and a random bit b (0 or 1), the encrypted message generated is computationally indistinguishable from the message received.

$P_3, \ldots, P_k$: The only messages received are on lines 2 and 15. The same argument as for P_2 applies. The public key E is simulated simply by randomly choosing a key E over the space of possible keys. The message $M_{j,i}$ $(j = 2, \ldots, k-1)$ can be simulated by randomly choosing a bit b (0 or 1), uniformly choosing a random r from $R(s)$, and computing $E(r, b)$. This message is computationally indistinguishable from the message received since the semantic security of the encryption system guarantees that no extra information is revealed (read prior paragraph for detailed discussion). □

4.5 Modified Secure Comparison Protocol

In many protocols, at some stage we need to securely compare the sum of two numbers with some threshold, with the output split between the parties holding those numbers. This can be accomplished using the generic circuit evaluation technique first proposed by Yao [92]. Formally, we need a modified secure comparison protocol for two parties, A and B. The local inputs are x_a and x_b and the local outputs are y_a and y_b. All operations on input are in a field F_1 and output are in a field F_2. $y_a + y_b = 1 \pmod{F_2}$ if $x_a + x_b \pmod{F_1} > 0$, otherwise $y_a + y_b = 0 \pmod{F_2}$. A final requirement is that y_a and y_b should be independently uniformly distributed over F (clearly the joint distribution is not uniform).

This builds on the standard secure multiparty computation circuit-based approach for solving this problem [39]. Effectively, A chooses y_a with a uniform distri-

bution over F, and provides it as an additional input to the circuit that appropriately computes y_b. The circuit is then securely evaluated, with B receiving the output y_b. The complexity of this is equivalent to the complexity of Yao's Millionaire's problem (simple secure comparison). The security of the protocol is also obvious, since the generic circuit evaluation technique is used.

5 EXPERIMENTAL VALIDATION

Apart from presenting algorithms to solve the problem, we must pay attention to the actual realization of those algorithms in practice. Thus, instead of simply presenting prototype versions of some of the algorithms developed, we would like to build a framework in which privacy preserving data mining can be *demonstrated*. As a part of the experimental validation, we give experimental results on two problems – decision tree classification and association rule mining. Other solutions are also being implemented, but the first two serve as a starting point to validate the techniques developed.

5.1 Weka

To demonstrate real practicality, we implemented the methods as part of an existing and widely used Data Mining toolkit. Weka [91], developed at the University of Waikato in New Zealand, is a collection of machine learning algorithms for data mining tasks implemented in Java. Apart from providing algorithms, it is a general implementation framework, along with support classes and documentation. It is extensible and convenient for prototyping purposes. However, the Weka system is a centralized system meant to be used at a single site. We extended the Weka core classes "Instance and Instances" to provide support for distributed instances. A distributed instance consists of only the key identifier and the site identifiers for the sites that together contain the whole instance.

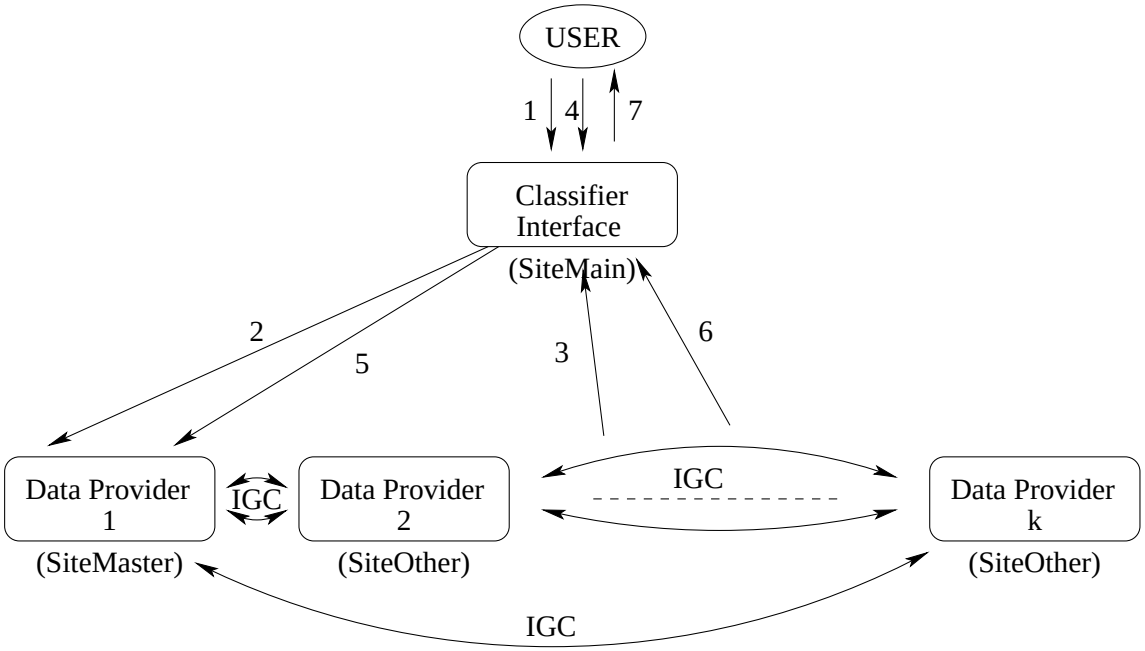
5.2 Decision Tree Classification

We first developed a general model of operation to extend Weka for privacy preserving distributed classification. The general model of privacy preserving distributed classification is as follows. The user initiates a request to build a classifier and then request(s) classification of an instance whenever required. The process of building the classifier needs to be co-ordinated so that the data sites locally construct enough state to enable them to jointly satisfy a classification request. To this end, every centralized classification class must be extended with a distributed class that provides the same functionality, however the implementation of these functions/messages is in a distributed manner.

In the current case, we extend the ID3 class with the new class distId3 that fulfills the same contracts as promised by ID3. When called with normal instance(s) the behavior is identical to ID3, when called with distributed instance(s) the algorithm performs in a distributed fashion. There is one global co-ordinating class/interface that provides access to the classification functionality. Figure 5.1 demonstrates the basic usage model.

We ran experiments with two and three sites on two data sets from the UCI repository [13]. Each of the processors used in the experimentation was a SUN Blade 1000 with a $900Mhz$ processor and $1gig$ of RAM. The trees, as expected, are identical to the original ID3 trees. The weather dataset consists of four attributes plus the class, and fourteen transactions. The $car - large$ is the UCI *car* dataset; the $car - small$ is a random subset of 1/2 the transactions, used to demonstrate scaling in number of transactions. These datasets have six attributes with about four distinct values each, and a four-class class attribute. There are 885 transactions in $car - small$, 1728 in $car - large$. The sample experimental results are given in Tables 5.1 and 5.2.

The ID3 trees for the car datasets are over 300 nodes. This is quite complex. We can see that this scales linearly in the number of transactions, as expected.



1: User Requests a classifier
2: Main Site sends request to Master to initiate model construction
IGC: Inter Group Communication (Protocol)
3: Classifier is built
4: User requests classification of an instance
5: Main site asks Master to initiate classification
6: Class of instance returned to Master
7: Class of instance returned to User

Figure 5.1. Basic classifier model

Table 5.1
Building the classifier on real data sets

Sites:	2	3
Weather	10s	86s
Car-Small	3.5 hrs	27 hrs
Car-large	7.1 hrs	62hrs

Table 5.2
Classifying an instance on real data sets

Sites:	2	3
Weather	$< 0.01s$	0.02s
Car-Small	0.125s	0.2s
Car-large	0.14s	0.46s

Increase in the number of parties causes a quadratic expansion in the amount of time required. One of the most important factors affecting the computation time of the protocol is the size of the tree built. Simpler trees are much faster to build. A good thing to note is that once the classification tree is built, classifying an instance takes very little time. Thus, if the (much more expensive) protocol to build the tree has already been executed, it is an easy (and much less computationally intensive) task to classify any given instance.

The current implementation is multi-threaded and does exploit parallelism to the extent possible. Readily available hardware for encryption or implementation in more highly optimized languages than Java would result in significant improvement. This prototype is meant as a demonstration of the viability and correctness of the protocol.

Table 5.3
Computation and communication cost of encryption

Number of items encrypted	Key Size			Transfer Time
	256	512	1024	
1k	< 0.0001s	5s	29s	0.0027s
10k	10s	47s	286s	0.007s
100k	90s	467s	2827s	0.04s
1M	900s	4660s	28762s	0.41s

5.3 Association Rule Mining

We have run experiments to evaluate what the actual cost would be for a number of different cases. The experiments were run on a SUN Blade 1000 workstation with a 900Mhz processor and 1GB of RAM. First, we tabulate the pure encryption cost for different key sizes. An encryption key size of 512 bits is sufficient for typical applications. It can be seen that the computation cost rises linearly with the number of item to be encrypted (as expected). Note that encryption can proceed at different sites in parallel. Thus Table 5.3 gives the encryption time per round.

We also measured the transfer time required to send the encrypted data from one site to another over a 100Mb network. The encrypted data required comparable size in the GNU GMP raw bit format regardless of key size.

Using the data generated in the prior table we can easily estimate the extra cost incurred by privacy while doing association rule mining in a particular situation (characterized by the number of transactions, attributes and parties). Table 5.4 estimates the computation cost assuming that the encryption key size is 512 bits. Table 5.5 estimates the communication cost assuming communication is over a 100Mb network. Both assume that attributes can have at most $100k$ transactions. We give a worst case scenario estimate assuming that all the attributes are frequent 1-itemsets

and also encrypting and communicating the entire attribute. In practice, the cost would be much lower (at least an order of magnitude), since all attributes may not be frequent and even the frequent attributes are present in only a fraction of the total number of transactions. The cost for other values of key size and communication bandwidth can be easily extrapolated using the data provided above. It is clear from this data that the computation cost greatly exceeds the communication cost. Computation cost can be drastically reduced by optimizing the code (we used the generic variant of GNU gmp), or through widely-available special-purpose encryption hardware. Note that the cost described here is the additional cost of assuring privacy. We still need to compute the association rules at each site. Overall, though expensive, the process is much faster than obtaining necessary approval to release data, assuming such approval could be obtained.

5.4 Summary

A first look at the experimental results may suggest that in comparison to centralized data mining algorithms, our performance is exceedingly slow. However, there are two caveats to this. First, our performance is *much* better than the general secure solutions possible. Second, and even more importantly, privacy is not free. If one has no privacy/security concerns, there is no reason why any of these algorithms should be used. It is simple enough to simply send all the data to a central site and let it do the mining, or use other distributed data mining techniques. However, when privacy/security concerns *do* exist, one can clearly see that the true comparison is between the time taken by our algorithms versus the time required to get approval, e.g., through an Institutional Review Board for Human Subjects Research, if it is even possible. In this sense, our algorithms are clearly practical and *enable* functionalities which would otherwise be prohibited.

Table 5.4
Worst-case added computation to achieve privacy

Number of attributes	Number of Sites				
	2	3	5	10	20
10	9340s	14010s	23350s	46700s	-
50	13hr	19.5hr	32.5hr	65hr	130hr
100	26hr	39hr	65hr	130hr	260hr
200	52hr	78hr	130hr	260hr	520hr

Table 5.5
Worst-case communication cost increase to achieve privacy

Number of attributes	Number of Sites				
	2	3	5	10	20
10	1.6s	3.6s	10s	40s	-
50	8s	18s	50s	200s	800s
100	16s	36s	100s	400s	1600s
200	32s	72s	200s	800s	3200s

6 SUMMARY

To summarize, the thesis is that privacy-preserving data mining over vertically partitioned data is both *feasible* and *practical*. The dissertation has presented a set of underlying techniques, which are used to construct several privacy-preserving data mining algorithms operating over vertically partitioned data, which enable the "mining" of knowledge.

Privacy/Security concerns have become an enduring part of society and commerce. It is increasingly necessary to ensure that useful computation does not violate legal/commercial norms for the safety of personal data. The thesis demonstrates that Privacy and Data Mining are not inherently in conflict. The major contribution has been to develop solutions for representatives of all of the major data mining tasks: classification, clustering, association rule mining and outlier detection.

Some of the tools developed are interesting in and of themselves. They are definitely applicable even beyond the scope of data mining. For example, we have developed privacy preserving solutions for optimization problems (such as linear programming) by utilizing some of the underlying techniques developed. In the future, we intend to look at other interesting practical problems.

One of the big drawbacks of Secure Multiparty Computation is that it is restricted to securing the process. There is no analysis of what the results themselves might reveal. This is an important problem which needs to be solved for any practical application of the techniques developed. Also, most of the solutions developed are valid within the semi-honest model of computation. Some go beyond that, but none are suitable for completely malicious behavior. It would be interesting to see how to extend our techniques to the malicious model *without* giving up on efficiency.

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, CA, May 21-23 2001. ACM.

[3] Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. Information sharing across private databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, San Diego, CA, June 9-12 2003.

[4] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., May 26–28 1993.

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

[6] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. Technical Report RJ9839, IBM Research, June 1994.

[7] 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.

[8] 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, IL, November 8 1999.

[9] Mikhail J. Atallah and Wenliang Du. Secure multi-party computational geometry. In *Seventh International Workshop on Algorithms and Data Structures (WADS 2001)*, Providence, RI, August 8-10 2001.

[10] Daniel Barbará, Ningning Wu, and Sushil Jajodia. Detecting novel network intrusions using bayes estimators. In *First SIAM International Conference on Data Mining*, Chicago, IL, April 5-7 2001.

[11] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, IL, May2-4 1988.

[12] 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.

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

[14] 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.

[15] Paul S. Bradley and Usama M. Fayyad. Refining initial points for K-Means clustering. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 91–99. Morgan Kaufmann, San Francisco, CA, 1998.

[16] Christian Cachin. Efficient private bidding and auctions with an oblivious third party. In *Proceedings of the Sixth ACM conference on Computer and communications security*, pages 120–127, Kent Ridge Digital Labs, Singapore, 1999. ACM Press.

[17] Philip Chan. *An Extensible Meta-Learning Approach for Scalable and Accurate Inductive Learning*. PhD thesis, Department of Computer Science, Columbia University, New York, NY, 1996.

[18] Philip Chan. On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information Systems*, 8:5–28, 1997.

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

[20] Rong Chen, Krishnamoorthy Sivakumar, and Hillol Kargupta. Distributed web mining using bayesian networks from multiple data streams. In *The 2001 IEEE International Conference on Data Mining*, San Jose, CA, November 29 – December 2 2001. IEEE.

[21] 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.

[22] Chris Clifton, Murat Kantarcioglu, Xiaodong Lin, Jaideep Vaidya, and Michael Zhu. Tools for privacy preserving distributed data mining. *SIGKDD Explorations*, 4(2):28–34, January 2003.

[23] 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 2 1996. ACM SIGMOD.

[24] 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.

[25] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society*, B 39:1–38, 1977.

[26] Wenliang Du. *A Study of Several Specific Secure Two-party Computation Problems*. PhD thesis, Purdue University, West Lafayette, Indiana, 2001.

[27] Wenliang Du and Mikhail J. Atallah. Privacy-preserving statistical analysis. In *Proceeding of the Seventeenth Annual Computer Security Applications Conference*, New Orleans, LA, 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] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, June 1973.

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

[32] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the Twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 211–222, San Diego, CA, June 9-12 2003. ACM Press.

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

[34] Kazuo J. Ezawa and Steven W. Norton. Constructing bayesian networks to predict uncollectible telecommunications accounts. *IEEE Expert*, 11(5):45–51, October 1996.

[35] Mr. Feingold, Mr. Corzine, Mr. Wyden, and Mr. Nelson. Data-mining moratorium act of 2003. U.S. Senate Bill (proposed), January 16 2003.

[36] M. Franklin and M. Yung. Varieties of secure distributed computing. In *Proceedings of Sequences II, Methods in Communications, Security and Computer Science*, pages 392–417, Positano, Italy, June 1991.

[37] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology - Eurocrypt 2004, International Conference on the Theory and Applications of Cryptographic Techniques*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19, Interlaken, Switzerland, May 2-6 2004. International Association for Cryptologic Research (IACR), Springer-Verlag.

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

[39] Oded Goldreich. *The Foundations of Cryptography*, volume 2, chapter 7: General Cryptographic Protocols. Cambridge University Press, may 2004.

[40] Oded Goldreich, Silvio Micali, and Avi Widgerson. How to play any mental game - a completeness theorem for protocols with honest majority. In *Proceedings of the Nineteenth Annual ACM Symposium on the Theory of Computing*, pages 218–229, New York City, NY, May 25-27 1987.

[41] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, February 1999.

[42] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth annual ACM symposium on Theory of computing*, pages 291–304, Providence, RI, 1985. ACM Press.

[43] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, April 2000.

[44] David Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. MIT Press, August 2001.

[45] Standard for privacy of individually identifiable health information. *Federal Register*, 66(40), February 28 2001.

[46] Ioannis Ioannidis, Ananth Grama, and Mikhail Atallah. A secure protocol for computing dot-products in clustered and distributed environments. In *The 2002 International Conference on Parallel Processing*, Vancouver, British Columbia, August 18-21 2002.

[47] Murat Kantarcioglu 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.

[48] 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, WI, June 2 2002.

[49] Murat Kantarcıoğlu and Chris Clifton. Privately computing a distributed k-nn classifier. In *Proceedings of the Eighth European Conference on Principles and Practice of Knowledge Discovery in Databases*, Pisa, Italy, September 20-24 2004.

[50] Murat Kantarcıoğlu and Chris Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering*, to appear.

[51] Hillol Kargupta and Philip Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery*. AAAI/MIT Press, September 2000.

[52] 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, FL, November 19-22 2003.

[53] Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proceedings of Twenty-fourth International Conference on Very Large Data Bases (VLDB 1998)*, pages 392–403, New York City, NY, August 24-27 1998.

[54] Edwin M. Knorr, Raymond T. Ng, and Vladimir Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 8(3-4):237–253, 2000.

[55] Edwin M. Knorr, Raymond T. Ng, and Ruben H. Zamar. Robust space transformations for distance-based operations. In *Proceedings of the Seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 126–135, San Francisco, CA, 2001. ACM Press.

[56] Aleksandar Lazarevic, Aysel Ozgur, Levent Ertoz, Jaideep Srivastava, and Vipin Kumar. A comparative study of anomaly detection schemes in network intrusion detection. In *SIAM International Conference on Data Mining (2003)*, San Francisco, CA, May 1-3 2003.

[57] Mr. Lewis. Department of defense appropriations act, 2004, July 17 2003. Title VIII section 8120. Enacted as Public Law 108-87.

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

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

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

[61] Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. John Wiley & Sons, 1997.

[62] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, October 1996.

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

[64] 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.

[65] David Naccache and Jacques Stern. A new public key cryptosystem based on higher residues. In *Proceedings of the Fifth ACM conference on Computer and communications security*, pages 59–66, San Francisco, CA, 1998. ACM Press.

[66] 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, GA, 1999. ACM Press.

[67] 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.

[68] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology – Eurocrypt '98, LNCS 1403*, pages 308–318. Springer-Verlag, 1998.

[69] Stanley R. M. Oliveira and Osmar R. Zaiane. Foundations for an access control model for privacy preservation in multi-relational association rule mining. In Chris Clifton and Vladimir Estivill-Castro, editors, *IEEE ICDM Workshop on Privacy, Security and Data Mining*, volume 14 of *Conferences in Research and Practice in Information Technology*, pages 19–26, Maebashi City, Japan, 2002. ACS.

[70] Stanley R. M. Oliveira and Osmar R. Zaiane. Privacy preserving frequent itemset mining. In Chris Clifton and Vladimir Estivill-Castro, editors, *IEEE ICDM Workshop on Privacy, Security and Data Mining*, volume 14 of *Conferences in Research and Practice in Information Technology*, pages 43–54, Maebashi City, Japan, 2002. ACS.

[71] Stanley R. M. Oliveira and Osmar R. Zaïane. Privacy preserving clustering by data transformation. In *Proceedings of the Eighteenth Brazilian Symposium on Databases*, pages 304–318, Manaus, Amazonas, Brazil, October 6-10 2003.

[72] Stanley R. M. Oliveira and Osmar R. Zaïane. Protecting sensitive knowledge by data sanitization. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*, Melbourne, FL, November 19-22 2003.

[73] 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.

[74] 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, FL, November 19-22 2003.

[75] Andreas Prodromidis, Philip Chan, and Salvatore Stolfo. *Advances in Distributed and Parallel Knowledge Discovery*, chapter 3: Meta-learning in distributed data mining systems: Issues and approaches. AAAI/MIT Press, September 2000.

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

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

[78] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 427–438, Dallas, TX, 2000. ACM Press.

[79] 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.

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

[81] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1998.

[82] Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of Twenty-first International Conference on Very Large Data Bases*, pages 432–444, Zurich, Switzerland, September 11-15 1995. VLDB.

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

[84] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, 2nd edition, 1995.

[85] Pradeep Shenoy, Jayant R. Haritsa, S. Sundarshan, Gaurav Bhalotia, Mayank Bawa, and Devavrat Shah. Turbo-charging vertical mining of large databases. In *Proceedings of the Nineteenth ACM SIGMOD International Conference on Management of Data*, pages 22–33, Dallas, TX, 2000.

[86] Julien P. Stern. A new and efficient all or nothing disclosure of secrets protocol. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology–ASIACRYPT'98*, number 1514 in Lecture Notes in Computer Science, pages 357–371. Springer-Verlag, 1998.

[87] Doug Struck. Don't store my data, Japanese tell government. *International Herald Tribune*, page 1, August 24-25 2002.

[88] Total information awareness (TIA) system.

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

[90] Rüdiger Wirth, Michael Borth, and Jochen Hipp. When distribution is part of the semantics: A new problem class for distributed knowledge discovery. In *Ubiquitous Data Mining for Mobile and Distributed Environments workshop associated with ECML'01 and PKDD'01, Freiburg, Germany, September 3-7 2001.*

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

[92] Andrew C. Yao. How to generate and exchange secrets. In Proceedings of the Twenty-seventh IEEE Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27-29 1986. IEEE.*

[93] *Mohammed J. Zaki. Parallel and distributed association mining: A survey.* IEEE Concurrency, special issue on Parallel Mechanisms for Data Mining, *7(4):14–25, December 1999.*

A DISCUSSION OF SECURE MULTIPARTY COMPUTATION TECHNIQUES

We first discuss some of the other primitives used from the SMC literature. We then present a complete solution and analysis using the General Secure Multiparty approach to a single problem to serve as a reference point for comparison.

A.1 Primitive Used from the Literature

The methods in Chapter 3 made use of several previously developed primitives. For completeness, they are described here.

A.1.1 Permutation Algorithm

The secure permutation algorithm developed by Du and Atallah simultaneously computes a vector sum and permutes the order of the elements in the vector. We repeat the idea here for completeness, for more details see [27]. We do present a more formal proof of the security of the algorithm than that in [27].

The permutation problem is an asymmetric two party algorithm, formally defined as follows. There exist 2 parties, A and B. B has an n-dimensional vector $\vec{X} = (x_1, \ldots, x_n)$, and A has an n-dimensional vector $\vec{V} = (v_1, \ldots, v_n)$. A also has a permutation π of the n numbers. The goal is to give B the result $\pi(\vec{X} + \vec{V})$, without disclosing anything else. In particular, neither A nor B can learn the other's vector, and B does not learn π. For our purposes, $\vec{V}$ is a vector of random numbers from a uniform random distribution, used to hide the permutation of the other vector.

The solution makes use of a tool known as *Homomorphic Encryption*. For a detailed discussion, see Section 4.4.2. The key is that homomorphic encryption allows us to perform addition of encrypted data without decrypting it.

The permutation algorithm consists of the following steps:

1. B generates a public-private keypair (E_k, D_k) for a homomorphic encryption scheme.
2. B encrypts its vector $\vec{X}$ to generate the encrypted vector $\vec{X}' = (x'_1, \ldots, x'_n), x'_i = E_k(x_i)$.
3. B sends $\vec{X}'$ and the public key E_k to A.
4. A encrypts its vector $\vec{V}$ generating the encrypted vector $\vec{V}' = (v'_1, \ldots, v'_n), v'_i = E_k(v_i)$.
5. A multiplies the components of the vectors $\vec{X}'$ and $\vec{V}'$ to get $\vec{T}' = (t'_1, \ldots, t'_n)$, $t'_i = x'_i * v'_i$.

   Due to the homomorphic property of the encryption,

   $$x'_i * v'_i = E_k(x_i) * E_k(v_i) = E_k(x_i + v_i)$$

   so $\vec{T}' = (t'_1, \ldots, t'_n), t'_i = E_k(x_i + v_i)$.
6. A applies the permutation π to the vector $\vec{T}'$ to get $\vec{T}'_p = \pi(\vec{T}')$, and sends $\vec{T}'_p$ to B.
7. B decrypts the components of $\vec{T}'_p$ giving the final result $\vec{T}_p = (t_{p_1}, \ldots, t_{p_n}), t_{p_i} = x_{p_i} + v_{p_i}$.

Security Analysis

The permutation algorithm reveals nothing to A, so A's view must be simulated using only it's own input. B gets the result vector.

Theorem A.1.1 *The Permutation Algorithm (Section A.1.1) privately computes a permuted vector sum of two vectors, where one party knows the permutation π and the other gets permuted sum $\pi(\vec{X} + \vec{V})$.*

PROOF.

A's view:

A receives an encryption key E_k and a encrypted vector $\vec{X}'$ of size n. It can simulate the encryption key by generating a single random number from a uniform random distribution. Assuming security of encryption and since A knows the n, the vector $\vec{X}'$ can also be simulated simply by generating n randoms from an uniform distribution. Using its own vector $\vec{V}$ and the simulated input, the simulator for A can perform steps 4–6 to complete the simulation of A's view.

B's view:

The simulator for B performs steps 1 and 2 to generate E_k and $\vec{X}'$. In step 6 B receives a size n vector $\vec{T}'_p$. To simulate $\vec{T}'_p$, B encrypts the components of the result $T_p = \pi(\vec{X} + \vec{V})$: $t'_{pi} = E_k(t_{pi})$.

The simulator for both runs in time linear in the size of the input vectors, meeting the requirement for a polynomial-time simulation. □

A.1.2 Scalar Product Protocol

One of the key sub-protocols required is a protocol for computing the scalar product of two vectors. Many scalar product protocols have been proposed in the past [27, 46, 89]. We now briefly describe one of the scalar product protocols given in [27]. The problem is defined as follows: Alice has a n-dimensional vector $\vec{X}$ while Bob has a n-dimensional vector $\vec{Y}$. At the end of the protocol, Alice should get $r_a = \vec{X} \cdot \vec{Y} + r_b$ where r_b is a random number chosen from an uniform distribution and is known only to Bob. The key idea behind the protocol is as follows: Alice splits up its vector into multiple parts. She then hides each part with some other random vectors and sends them to Bob. Bob computes the scalar product of his vector with all the vectors he receives while adding a random he generates to the result. Alice then uses Oblivious Transfer to get back the correct part results from all the numbers that Bob has generated. The Oblivious Transfer primitive

is described in the following subsection. First, here is the actual scalar product protocol:

Alice and Bob agree on two numbers p and m, such that p^m is considered large enough (for security).

Alice generates m random vectors $\vec{V_1}, \ldots, \vec{V_m}$ such that $\vec{X} = \sum_{i=1}^{m} \vec{V_i}$.

Bob generates m random numbers $r_1, \ldots, r_m$ such that $r_b = \sum_{i=1}^{m} r_j$.

for $j = 1 \ldots m$ **do**

Alice generates a secret random number $k, 1 \leq k \leq p$.

Alice sends $(\vec{H_1}, \ldots, \vec{H_p})$ to Bob, where $\vec{H_k} = \vec{V_j}$, and the rest of $\vec{H_i}'s$ are random vectors. Since k is secret, Bob does not know the position of $\vec{V_j}$.

for $i = 1 \ldots p$ **do**

Bob computes $Z_{j,i} = \vec{H_i} \cdot \vec{Y} + r_j$.

end for

Using the 1-out-of-p Oblivious Transfer protocol, Alice get $Z_j = Z_{j,k} = \vec{V_j} \cdot \vec{Y} + r_j$, while Bob learns nothing about k

end for

Alice computes $r_a = \sum_{j=1}^{m} Z_j = \vec{X} \cdot \vec{Y} + r_b$

The key primitive used in this protocol is the 1-out-of-p Oblivious Transfer.

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 [31] as a generalization of Rabin's "oblivious transfer" [77]. Naor and Pinkas [67] 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.

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. 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.

A.1.3 Square Computation

The problem is defined as follows: There exist two sites, A and B. A holds x_a, while B holds x_b. Together they wish to compute shares of the function $f = (x_a + x_b)^2$. Thus, at the end of the protocol, A should have y_a and B should have y_b such that $y_a + y_b = (x_a + x_b)^2$. An obvious way to do this is using oblivious evaluation of polynomials. A first generates a random value y_a. A then forms the polynomial $P(z) = (1)z^2 + (2x_a)z + (x_a^2 - y_a)$. An oblivious evaluation of $P(x_b)$ by B gives B, $y_b = P(x_b)$. Note that $y_b + y_a = x_b^2 + 2x_a x_b + x_a^2 - y_a + y_a = (x_a + x_b)^2$ as required.

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 [66]. Subsequently, there have been more protocols improving the communication/computation efficiency [24] as well as extending the problem to floating point numbers [19]. For our protocols, we use the protocol given in [24] since it requires only $O(k)$ exponentiations 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 [59]: 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 (DDH) assumption holds. The protocol can be converted to one computing $P(x)$ using the methods of Paillier [73], who presented a trapdoor for computing discrete logs. The protocol is quite simple when 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'.

A.1.4 Privately Computing $\ln x$

In Section 3.4.4, 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 [59] was a cryptographic protocol for this computation. We now describe the protocol in brief: Note that $\ln x$ is *Real* while cryptography works over finite fields. Thus there needs to be some way of doing numerical analysis. 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 that 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 $Q(\dot{)}$ is defined as

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

P_1 and P_2 then execute an oblivious polynomial evaluation with P_1 inputting $Q(\cdot)$ and P_2 inputting α_2, where 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 [60].

Theorem A.1.1 *Protocol A.1.4 privately computes* $\ln x$*, for* $x = x_1 + x_2$ *split between two parties.*

PROOF. Refer to [60]. □

A.1.5 Division Protocol

The problem of division is described as follows: Alice has inputs a_1, a_2. Bob has inputs b_2, b_2. At the end of the protocol, Alice and Bob get shares of $(a_1+b_1)/(a_2+b_2)$. Thus Alice should get c_a while Bob gets c_b such that:

$$c_a + c_b = \frac{a_1 + b_1}{a_2 + b_2}$$

We present an efficient protocol to do this based on the Division Protocol developed in [27].

Alice chooses 3 randoms c_a, r_1, r_2.

Alice forms $\vec{U} = \begin{bmatrix} r_1(a_1 - c_a * a_2) \\ r_1 \\ -r_1 * c_a \end{bmatrix}$ Bob forms $\vec{V} = \begin{bmatrix} 1 \\ b_1 \\ b_2 \end{bmatrix}$

Alice and Bob engage in a secure scalar product so that (only) Bob gets

$$\begin{aligned} x_1 &= \vec{U} \cdot \vec{V} \\ &= r_1(a_1 - c_a * a_2) + r_1 * b_1 - r_1 * c_a * b_2 \\ &= r_1(a_1 + b_1 - c_a(a_2 + b_2)) \end{aligned}$$

Alice forms $\vec{W} = \begin{bmatrix} r_2 * a_2 \\ r_2 \end{bmatrix}$ Bob forms $\vec{X} = \begin{bmatrix} 1 \\ b_2 \end{bmatrix}$

Alice and Bob engage in a secure scalar product so that (only) Bob gets

$$\begin{aligned} x_2 &= \vec{W} \cdot \vec{X} \\ &= r_2 a_2 + r_2 b_2 \\ &= r_2(a_2 + b_2) \end{aligned}$$

Alice sends $x_3 = \frac{r_2}{r_1}$ to Bob

Bob computes

$$\begin{aligned} c_b &= \frac{x_1}{x_2} * x_3 \\ &= \frac{a_1 + b_1 - c_a(a_2 + b_2)}{a_2 + b_2} \\ &= \frac{a_1 + b_1}{a_2 + b_2} - c_a \end{aligned}$$

Though neither this protocol nor the underlying division protocol have been formally proven secure, it is also possible to do division in a provably secure fashion using the generic circuit evaluation method.

A.2 A Complete Solution and Analysis using the General Secure Multiparty Approach

This dissertation is predicated on the impracticality of generic circuit evaluation for solving problems involving large datasets. To demonstrate this, we present a construction of the size of set intersection as a comparison with the efficient solutions in Chapter 4.

For demonstration, we limit the problem to 2 parties. We also assume that the input set sizes can be at most n and that all numbers are m bit. A simple circuit to do this would compare each value of one set in succession with all of the values of the other set. The comparator returns 1 if the values are equal, 0 otherwise. Assuming that we are looking at sets (i.e. objects cannot be repeated), at most one of the comparators can return 1. Thus, we need n^2 comparator circuits. We also need a layer of addition circuits to add up all the outputs of the comparators. The result of the final addition circuit gives the total number of items that are common to both of the input sets. Thus we need a total of n^2 comparators and n^2 adders. The comparators operate over m bit numbers while the adders have to operate over $\log_2 n$ bit numbers. In terms of depth, the complete circuit has $log_2 n + 1$ layers (one for each of the addition layers and one for the comparators). It is possible to actually determine the number of AND gates required for a single adder and a single comparator. That number gives the number of Oblivious Transfers required to compute the circuit. In general, while polynomial in the size of the circuit, this is extremely inefficient and thus infeasible.

VITA

Jaideep Vaidya was born in Mumbai, India. He received his Bachelor of Engineering degree in computer engineering in August 1999 from University of Mumbai. He got his master's degree in computer science in May 2001 from Purdue University, and a Ph.D. in computer science from Purdue University in 2004. While at Purdue, Jaideep was a member of the Center of Education and Research in Information Assurance and the Indiana Center of Database Systems. He will shortly be joining the faculty of Rutgers, the State University of New Jersey, as an Assistant Professor of Management Science and Information Systems.

During his Ph.D. studies, Jaideep interned with Microsoft in 2000 and NEC Research in 2002. For his doctoral work, he developed new methods to enable mining of knowledge from data distributed across multiple entities, while respecting privacy concerns. His current research interests lie at the confluence of security/privacy, data mining and databases. He is also interested in the general problem of secure computation, especially in relation to real problems/applications.

Jaideep has been an invited speaker and guest lecturer at several institutions. His paper on Privacy-Preserving K-means Clustering won the runner up award for best research paper at ACM SIGKDD 2003. He has reviewed papers for several journals and conferences including IEEE TKDE, ACM KDD and PKDD.