

© 2010

GEETHA JAGANNATHAN

ALL RIGHTS RESERVED

DATA PRIVACY IN KNOWLEDGE DISCOVERY

By

GEETHA JAGANNATHAN

A Dissertation submitted to the
Graduate School—New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Computer Science
written under the direction of
Rebecca N. Wright
and approved by

New Brunswick, New Jersey

May, 2010

ABSTRACT OF THE DISSERTATION

Data Privacy In Knowledge Discovery

By GEETHA JAGANNATHAN

Dissertation Director:

Rebecca N. Wright

This thesis addresses data privacy in various stages of extracting knowledge embedded in databases. Advances in computer networking and database technologies have enabled the collection and storage of vast quantities of data. Legal and ethical considerations might require measures to protect an individual's privacy in any use or release of the data.

In this thesis, we address the problem of preserving privacy in the two following cases:

- in distributed knowledge discovery;
- in situations where the output of a data mining algorithm could itself breach privacy.

We present results in two different models, namely *secure multiparty computation* (SMC) and *differential privacy*. The first part of the thesis presents privacy preserving protocols in the SMC model. Secure multiparty computation involves the collaborative computation of functions based on inputs from multiple parties. The privacy goal is to ensure that all parties receive only the final output without any party learning anything beyond what can be inferred from the output. Within this framework we address the problem of preserving privacy in the preprocessing and the data mining stages of knowledge discovery in

databases. For the preprocessing stage, we present private protocols for the imputation of missing data in a dataset that is shared between two parties. For the data mining stage, we introduce the notion of arbitrarily partitioned data that generalizes both horizontally and vertically partitioned data. We present a privacy-preserving protocol for k -means clustering of arbitrarily partitioned data. We also develop a new simple k -clustering algorithm that was designed to be converted into a communication-efficient protocol for private clustering.

The second part of the thesis deals with privacy in situations where the output of a data mining algorithm could itself breach privacy. In this setting, we present private inference control protocols in the SMC model for On-line Analytical Processing systems. In the *differential privacy* model, the goal is to provide access to a statistical database while preserving the privacy of every individual in the database, irrespective of any auxiliary information that may be available to the database client. Under this privacy model, we present a practical privacy preserving decision tree classifier using random decision trees.

Table of Contents

Abstract	ii
List of Algorithms	xi
List of Protocols	xii
List of Figures	xiii
List of Tables	xviii
Acknowledgements	xix
1. Introduction	1
1.1. Motivation	1
1.2. Related Work	4
1.2.1. Privacy Models	5
1.2.2. Privacy-Preserving Distributed Data Mining	8
1.2.3. Output Privacy in Data Mining	10
1.3. Part-I : Privacy-Preserving Knowledge Discovery in Distributed Databases .	13
1.3.1. Privacy-Preserving Data Imputation	13
1.3.2. Privacy-Preserving Clustering Algorithms	14
1.4. Part-II : Output Privacy in Data Mining	16
1.4.1. Private Inference Control in OLAP Systems	16

1.4.2.	A Practical Differentially Private Random Decision Tree Classifier	17
2.	Preliminaries	19
2.1.	Privacy Models	19
2.1.1.	Secure Multiparty Computation	19
2.1.2.	Differential Privacy	21
2.2.	Primitives	22
2.2.1.	Random Shares	22
2.2.2.	Oblivious Transfer	23
2.2.3.	Yao’s Circuit Evaluation Protocol	24
2.2.4.	Homomorphic Encryption	24
2.2.5.	Secure Scalar Product Protocol	25
I	Privacy-Preserving Distributed Data Mining	26
3.	Privacy-Preserving Imputation of Missing Data	27
3.1.	Introduction	27
3.1.1.	Related Work	28
3.1.2.	Our Contributions	31
3.2.	Preliminaries	32
3.2.1.	Lazy Decision Tree Algorithm	33
3.2.2.	Privacy Definition	35
3.2.3.	Cryptographic Primitives	36
3.3.	Privacy-Preserving Imputation Based on Lazy Decision Trees	37
3.3.1.	Our Basic Protocol	38

3.3.2. Secure Computation of the Number of Instances	42
Secure Hamming Distance	42
Secure Computation of the Total Number of Instances in a Subset with a Given Class Label	43
3.3.3. Secure Protocol to Compute Split Entropy	44
3.3.4. Secure Split Protocol	47
3.3.5. Secure Protocol to Check if D is Pure	48
3.3.6. Secure Protocol for Majority Label	49
3.3.7. Performance and Privacy Analysis	49
3.3.8. Enhancing Privacy	51
3.4. Other Imputation Protocols	52
3.5. Summary	54

4. Privacy-Preserving Distributed k -means Clustering over Arbitrarily Partitioned Data

4.1. Introduction	55
4.1.1. Related Work	56
4.1.2. Our Contributions	57
4.2. Preliminaries	59
4.2.1. Arbitrarily Partitioned Data	59
4.2.2. Privacy Properties	62
4.2.3. Clustering and the k -Means Algorithm	63
4.3. Privacy-Preserving k -Means Clustering Protocol	65
4.3.1. Secure Protocol to Compute Closest Cluster	68

4.3.2.	Secure Protocol for Recomputing the Mean	71
4.3.3.	Secure Protocol for Termination of Iterations	72
4.4.	Performance Analysis	74
4.5.	Enhancing Privacy	76
4.6.	Summary	79
5.	Communication-Efficient Privacy-Preserving Clustering Algorithms . .	80
5.1.	Introduction	80
5.1.1.	Our Contributions	81
5.2.	Preliminaries	82
5.3.	The k -Clustering Algorithm	83
5.3.1.	Overview	83
	Error Measure.	84
5.3.2.	Extension to Data Streams	86
5.3.3.	Distributed k -Clustering Protocol for Two Parties	87
5.4.	Experimental Results	88
5.4.1.	Synthetic Data Generator	88
5.4.2.	Realistic Data	90
5.4.3.	Results for ReCluster	90
5.4.4.	Results for Distributed ReCluster	95
5.5.	Privacy-Preserving k -Clustering Protocol	96
5.5.1.	Overview of the Private Clustering Protocol	96
5.5.2.	Secure Protocol to Share Data	100
5.5.3.	Secure Protocol to Merge Clusters.	101

5.5.4.	Overall Privacy and Performance of the Private ReCluster Protocol	104
5.6.	Summary	107
II	Output Privacy in Data Mining	108
6.	Private Inference Control For Aggregate Database Queries	109
6.1.	Introduction	109
6.1.1.	Related Work	111
6.1.2.	Our Contributions	113
6.2.	Preliminaries	114
6.2.1.	Selective Private Function Evaluation	114
6.2.2.	Private Inference Control	115
6.2.3.	Our Model	117
6.3.	Private Inference Control for Aggregate Queries	119
6.3.1.	The First Protocol	121
	Correctness	122
	Client Privacy	122
	Inference Control	122
6.3.2.	The Second Protocol	125
6.3.3.	The Third Protocol	127
6.3.4.	A Comparison	131
6.4.	Extensions and Special Cases	132
6.4.1.	Relaxing the Inference Control Rule	132
6.4.2.	Special Cases: SUM and COUNT	133

The Sum Function	133
Counting Frequencies	134
6.4.3. Horizontally Partitioned Database	135
6.5. Summary	136
7. A Practical Differentially Private Random Decision Tree Classifier . .	138
7.1. Introduction	138
7.1.1. Related Work	139
7.1.2. Our Contributions	141
7.2. Preliminaries	142
7.3. ID3 Trees from Private Sum Queries	143
7.4. Random Decision Trees: An Overview	147
7.5. Differentially Private Random Decision Trees	150
7.5.1. Private Random Decision Tree Algorithm	151
7.5.2. Updating Random Decision Trees	154
7.5.3. Private Random Decision Trees for Distributed Databases	156
7.6. Experiments	158
7.6.1. Accuracy of Private Random Decision Tree Ensembles	158
7.6.2. Updating Random Decision Trees	165
7.6.3. Private Random Decision Trees on Vertically Partitioned Data	166
7.7. Summary	168
8. Conclusions	169
References	172

Curriculum Vita	183
---------------------------	-----

List of Algorithms

1.	Lazy Decision Tree	34
2.	k -means clustering algorithm	64
3.	ReCluster	82
4.	RecursiveCluster	83
5.	MergeCenters	85
6.	StreamCluster	87
7.	The <i>ID3</i> decision tree learning algorithm.	146
8.	Random Decision Tree (RDT)	149
9.	Classify	150
10.	Private-RDT	153

List of Protocols

1.	Private Lazy Decision Tree Imputation	40
2.	Secure Hamming Distance	43
3.	Total Instances with Class Label	44
4.	Secure Split Entropy	45
5.	Privacy-preserving k -means clustering over arbitrarily partitioned data . . .	66
6.	Privacy-preserving k -clustering protocol	99
7.	Secure protocol to share data	101
8.	Private Inference Control for Aggregate Queries	123
9.	Aggregate Private Inference Control Protocol: A Second Approach	128

List of Figures

4.1. Arbitrarily partitioned data	61
5.1. The effect of choosing error_r over error_w . ReCluster prefers to merge C_s and C_t , Ward's algorithm would prefer to merge C_p and C_q	86
5.2. This synthetic data set consists of about 10,000 points concentrated around each randomly chosen cluster center plus 5% "noise" points occurring elsewhere. The points concentrated at each chosen cluster center are illustrated by a large open ellipse around each center. The noise points are shown as points. Cluster centers shown are those identified by (a) ReCluster ($\text{ESS} = 5.71 \times 10^7$) (b) the worst run of k -means algorithm ($\text{ESS} = 8.49 \times 10^7$) and (c) the best run of k -means algorithm ($\text{ESS} = 5.48 \times 10^7$) are denoted by x.	92
5.3. An example data set in which ReCluster did not accurately identify all cluster centers. The data set consists of about 10,000 points concentrated around each randomly chosen cluster center plus 5% "noise" points occurring elsewhere. The points concentrated at each cluster center are illustrated by a large open ellipse around each center. The noise points are shown as points. Also shown are the centers identified by the best run of the k -means algorithm denoted as.	93

- 5.4. Cluster centers identified in a grid data set. The data set consists of about 1000 points concentrated around each grid point plus 5% “noise” points occurring elsewhere. The points concentrated at each grid point are illustrated by a large open circle around each grid point. The noise points are shown as points. On the left are cluster centers identified by ReCluster. On the right are the best (identified by small filled circles) and worst (identified by small open circles) runs of the k -means algorithm. 94
- 5.5. Cluster centers identified in an offset grid data set. The data set consists of about 1000 points concentrated around each grid point plus 5% “noise” points occurring elsewhere. The points concentrated at each grid point are illustrated by a large open circle around each grid point. The noise points are shown as points. On the left are cluster centers identified by ReCluster. On the right are the best (identified by small filled circles) and worst (identified by small open circles) runs of the k -means algorithm. 94

5.6. Some typical runs of the distributed clustering algorithm. The data sets consists of about 20,000 points concentrated around each randomly chosen cluster center owned by the first party (denoted by 1), 10,000 points concentrated around each randomly chosen cluster center owned by the second party (denoted by 2), plus 5% “noise” points occurring elsewhere. The points concentrated at each cluster center are illustrated by a large open ellipse around each center. The noise points are shown as points. Figures (a) and (b) are example data sets in which the first party “owns” the data for three of the clusters and the second party has data for the two remaining clusters. Figures (c) and (d) are example data sets in which the first party has data for four of the clusters and the second party has data for only one cluster. In Figures (a) and (c) each party has 25% of the data from the other party’s clusters. In Figures (c) and (d) each party has no information regarding the clusters of the other party. In all cases the letter X marks the centers detected by the distributed clustering protocol. Also shown are the error sum of squares value for distributed ReCluster and the average error sum of squares for k -means. 97

5.7. Typical runs of the distributed clustering algorithm on more “skewed” data.	
Clusters owned by the first party have five times as many points as clusters owned by the second party (5000 points on average versus 1000 points on average). The first party’s cluster centers are denoted as 1 and the second party’s cluster centers are denoted as 2. Each party has no data about the clusters owned by the other party. In all cases the letter X marks the centers detected by the distributed ReCluster protocol. Also shown are the error sum of squares value for distributed ReCluster and the average error sum of squares for k -means.	98
7.1. RDT Algorithm is not private: Example databases that differ in at most one element and randomly generated tree structure	152
7.2. RDT Algorithm is not private: Final trees for D_1 and D_2	152
7.3. The decision tree used to generate synthetic data set 2.	161
7.4. Accuracy on the Nursery data set from the UCI Repository. Displayed are the average and maximum accuracy for $\epsilon \in \{0.25, 0.5, 0.75, 1, \infty\}$	163
7.5. Accuracy on the Mushroom data set from the UCI Repository.	163
7.6. Accuracy on the Congressional Voting Records data set from the UCI Repository.	164
7.7. Accuracy on synthetic data set 1. The decision tree for this data set (not shown) has 10 attributes.	164
7.8. Accuracy on synthetic data set 2. See Figure 7.3 for underlying decision tree.	164
7.9. Accuracy on synthetic data set 3. The decision tree for this data set (not shown) has 7 attributes.	165
7.10. Performance of the update algorithm on the Nursery data set.	166

7.11. Performance of the update algorithm on the Mushroom data set.	167
7.12. Performance of the update algorithm on synthetic data set 1.	167
7.13. Performance of the vertically partitioned database algorithm on randomly partitioned Congressional Voting Records dataset.	168

List of Tables

4.1. Comparison of privacy-preserving clustering protocols; in the Privacy column, an entry of L indicates that the protocol leaks candidate cluster centers and an entry of F indicates that the protocol is fully private.	58
5.1. Sample parameters for our synthetic data set generator while generating a Random Centers data set	89
5.2. The error sum of squares for a subset of the Random Centers data sets. In each case the number of clusters was 5, with approximately 10,000 points in each cluster. Noise was added at the rate of 5%.	91
5.3. Comparison of the error sum of squares for the network intrusion 10% data set. An asterisk indicates that BIRCH failed to find five clusters due to insufficient memory.	95
7.1. Implementing a privacy-preserving version of <i>ID3</i> using low-level differentially private queries produces very poor accuracy on many widely used data sets.	147
7.2. Experimental Data Characteristics	163

Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Rebecca N. Wright, for her exceptional guidance and for providing me with an excellent research atmosphere. I am especially grateful for her patience in correcting my writing and for financially supporting my research. Without her persistent help and support I would never have been able to complete my dissertation.

I would like to thank my committee members Prof. Amelie Marian, Prof. S. Muthukrishnan and Prof. Vitaly Shmatikov for their comments and questions. Their insights have helped me find new directions into which I can expand my future work. Special thanks go to Prof. Danfeng Yao for her encouragement and support throughout my stay at Rutgers.

I offer my sincere thanks to the administrative staff of the Department of Computer Science and of DIMACS. In particular, I thank Carol DiFrancesco for eliminating many administrative bottlenecks.

I am grateful for the support of the National Science Foundation through its grants, CCR-0331584 and CNS-0822269.

Finally, my thanks go to my friends and family, especially my children Keshav and Diya, who constantly challenged me in their own ways.

Chapter 1

Introduction

1.1 Motivation

The cost per byte of secondary storage has fallen steeply over the years. Many organizations have taken advantage of this reduction in cost by creating large databases of transactional and other data. These databases could contain information about retail transactions, customer and client information, geospatial information, web server logs, etc. These large data warehouses have been “mined” for knowledge that could improve the efficiency and effective functioning of an organization. However, when these data warehouses contain information regarding individuals, the use or release of this knowledge could threaten an individual’s privacy. In this thesis, we address the problem of achieving data privacy in various stages of extracting knowledge embedded in databases.

A data owner may either extract the knowledge by himself or might provide to external researchers the individual data records that make up the databases so that new and innovative knowledge mining techniques could be applied to them. For example, a hospital may create a database containing the medical history of a subset of its patients. The hospital may use the data to create a system that predicts the potential for cancer recurrence in a patient. Or, the hospital might want to help advance medical research by distributing the database it owns. Whether the owner of the data releases the data itself, or the computational results on the data, he should be aware that they may contain sensitive information

about individuals and businesses. Legal and ethical considerations might require measures to protect an individual’s privacy in any use or release of the data. Beyond legal and ethical considerations, however, is also the issue of trust—the lack of trust can lead to poor data quality and low response rates [101].

Knowledge discovery in databases, or KDD, is the process of identifying valid, novel, potentially useful, and ultimately understandable structure in data. KDD refers to the overall process including the integration and cleaning of the data, applying specific algorithms to extract patterns from the data and finally interpreting the results to obtain knowledge from it. Within KDD, preparing the raw collected data for knowledge extraction is called *preprocessing*. The process of applying algorithms to extract patterns from data is known as *data mining*. This thesis addresses privacy in two important steps of knowledge discovery namely, preprocessing and data mining.

Preprocessing

Due to human error and systemic reasons, real world data sets tend to be noisy, inconsistent and incomplete. Data preprocessing includes various tasks such as data cleaning, data transformation, data reduction, data integration and linkage. Data cleaning, an important preprocessing step, deals with detecting and removing errors and inconsistencies from the collected data. An uncleaned dataset may lead to wrong predictions that may cost an organization in many ways. For example, consider a manufacturer that surveys its customers to gauge interest in a new product. If the data collected is not cleaned, the organization could potentially manufacture a product that will not be favored by its clients, which could lead to large financial losses. This problem of preprocessing becomes more complicated when the data is distributed among multiple parties. Running a preprocessing algorithm

on the combined data may lead to better results but privacy laws may prevent such pooling of data. In order to make use of data cleaning algorithms while maintaining privacy, privacy-preserving methods of data cleaning are required. This thesis considers privacy-preserving data cleaning in the two-party setting.

Data Mining

Data mining refers to the process of applying algorithms for extracting patterns from data. It has been successfully applied in a wide range of domains extending from DNA data analysis [67] to financial data analysis [9] to fraud detection [100]. Although recognizing patterns in a given database may yield valuable insights about a disease or an organization, there also exists great potential for privacy breaches when the input data has sensitive information about individuals [108]. The hidden patterns learned as a result of running a data mining algorithm can reveal identifying personal information. For example, consider a cancer center that produces a classifier to predict the probability that an individual might suffer from some form of cancer in the future. Such a classifier could be valuable to a health care provider. But if the classifier itself reveals that a certain individual's data was included in its creation, a health insurance company could refuse to write a policy for that person. It is important to emphasize that in this situation we are not considering the loss of privacy resulting from the use of the classifier—rather, the classifier itself, as summary data, could breach privacy.

Yet another privacy issue that arises is due to the pooling of data. While much of knowledge discovery happens within an organization, it is quite common to use data from multiple sources in order to yield more precise or useful knowledge. Data may be distributed

among multiple databases owned by different entities. Aggregating all data into a centralized location could lead to a serious privacy concern since the data miner can easily “stitch together” information from multiple sources to find sensitive information about many individuals. This kind of collation could be less feasible if the data remained distributed among the original entities. Distributed data mining algorithms [115] have been designed to mine for patterns in distributed data, when pooling to one centralized location may be infeasible or undesirable. However, these algorithms do not address privacy breaches.

In this thesis, we address the problem of achieving privacy in the following two cases:

- privacy in a distributed setting, which enables knowledge discovery to be performed when the data is distributed among multiple sources,
- privacy in situations where the output of a data mining algorithm could itself breach privacy.

The rest of this chapter is organized as follows. In Section 1.2 we broadly survey related work. We outline our results in privacy-preserving knowledge discovery in distributed databases in Section 1.3. In Section 1.4 we outline our results in output privacy in data mining.

1.2 Related Work

There has been a long-standing tension between the desire to use data for various purposes and the desire to protect the security and privacy associated with data. The problem of maximizing the usage of the data collected and preventing the disclosure of sensitive or confidential information at the same time has been the subject of research in the statistical sciences (where it is called “statistical disclosure control”) and in computer science for a

very long time [1]. In this section, we describe some work related to the thesis. We discuss additional appropriate related work in each chapter.

1.2.1 Privacy Models

Preserving data privacy and extracting useful information from data are inherently conflicting tasks. Perfect privacy can only be achieved with the absolute loss of the utility of a data set. Under this circumstance, the revised objective has been to maximize the utility of a data set while minimizing the risk of privacy loss due to the use of the data set. This dual goal of maximizing confidentiality and data utility together initiated the study of various privacy models.

PERTURBATION MODELS.

This approach attempts to achieve privacy through masking—the addition of random noise drawn from certain carefully chosen distributions. The privacy goal in this setting is to mask sensitive data in the published data sets. As an example, perturbation was used by Agrawal and Srikant [2] in the construction of privacy-preserving decision tree classifiers. Perturbation techniques fall into two categories: (a) Input Perturbation—In this model, noise is added to the entries of the database prior to the evaluation of queries. (b) Output Perturbation—In this model, the queries are evaluated on the original data and noisy versions of the “correct” answers are released. These techniques have been used in the security of some statistical databases [1]. However, it has been shown in [83] that if perturbation is not performed with care, in many cases it is possible to estimate the original values accurately, thereby leading to a loss of privacy. A formal study of privacy through perturbation eventually led to the much stronger model of differential privacy, although

differential privacy does not exclusively depend on perturbation.

k -ANONYMITY AND ITS VARIANTS.

The k -anonymity model proposed by Samarati and Sweeney [124, 131] achieves privacy by enforcing the constraint that every row of the released database should be indistinguishable from at least k other rows with respect to a selected set of attributes called quasi-identifiers. This is usually achieved by suppressing and generalizing some of the entries in the database. A variant of k -anonymity known as ℓ -diversity was introduced by Machanavajjhala et al. [98]. This guarantees privacy in certain situations where k -anonymity does not, such as when there is little diversity in the sensitive attributes or when the adversary has some background information. Yet another variant called t -closeness was introduced by Li et al. [92]. This privacy model enforces the condition that the distances between the distribution of the sensitive attributes in any equivalence class and the distribution of the attribute in the overall table should be less than a threshold t . However, recently it has been shown by Ganta et al. [57] that these models do not offer privacy in the presence of auxiliary information. Brickell and Shmatikov [15] studied the trade-off between privacy and utility of these anonymization algorithms and showed that in many cases trivial sanitization provides better utility and privacy than k -anonymity and its variants.

SECURE MULTIPARTY COMPUTATION (SMC).

This setting involves the collaborative computation of functions based on inputs from multiple parties. Pioneered by Yao [145], the privacy goal in this setting is to ensure that all parties receive only the final output, without any party learning anything beyond what can be inferred from the output. Subsequent to [145], a large number of extensions have been

developed [62, 8, 19]. In this thesis, we present four results under this model:

1. A protocol for the privacy-preserving imputation of missing data, presented in Chapter 3.
2. A protocol for privacy-preserving distributed k -means clustering over arbitrarily partitioned data, in Chapter 4.
3. Protocols for communication-efficient privacy-preserving clustering, in Chapter 5.
4. Protocols for private inference control for aggregate database queries, in Chapter 6.

We describe this model in greater detail in Chapter 2.

DIFFERENTIAL PRIVACY.

This privacy model was introduced by Dwork et al. [43]. In this framework the data owner makes the data available through a statistical database on which only aggregate queries are permitted. The goal is to answer queries while preserving the privacy of every individual in the database, irrespective of any auxiliary information that may be available to the database client. The differential privacy model assures that the data from no single individual has a substantial impact on the output produced by a differentially private mechanism. Effectively, no individual data item stands out when accessed through a differential-privacy-preserving mechanism. Under this model we present, in Chapter 7, a practical algorithm for private random decision tree classifiers. We describe this model in greater detail in Chapter 2.

1.2.2 Privacy-Preserving Distributed Data Mining

Although much of knowledge discovery is based on data within an organization, it is quite common to use data from multiple sources, owned by different entities, in order to obtain more accurate knowledge. The data sources may be distributed across the network and hence pooling them at a centralized location may not be possible due to limitations in computational and communication resources. Distributed data mining provides algorithms to perform data mining in a distributed setting without pooling the data into one location [115].

A (virtual) database table is said to be *horizontally partitioned* among a number of parties if each party has a database table with the same set of attributes. The virtual table is the union of all these tables. As an example of horizontally partitioned data, consider a situation where a number of hospitals wish to collaboratively compute some quantity regarding patients with cancer. All collaborators would agree ahead of time about the attributes they have in common for the patients in their care. Note that it is not necessary that each hospital maintain their records in this agreed upon common format. All they need to be able to do is compute as necessary the attributes needed for collaborative computation. The union of all such patient data from these hospitals is the virtual database table that has been horizontally partitioned among the various hospitals.

A virtual database table is said to be *vertically partitioned* among a number of parties if each party has data for some fixed set of attributes about individuals that are known to all the parties. The virtual database table is the “join” of all the distributed tables. As an example of vertically partitioned data between two parties, consider a credit-card issuer collaborating with an airline company to find correlations between spending patterns and flight travel. Each party holds a different set of attributes, except for a common identifier

attribute that relates a row in the first database table with a row in the second.

Privacy considerations can prohibit organizations from sharing their data with each other. Privacy-preserving distributed algorithms arose as a solution to this problem by allowing parties to cooperate in the extraction of knowledge without any of the cooperating parties having to reveal their individual data items to each other or any other party.

Techniques from secure multiparty computation [61] form one approach to privacy-preserving distributed data mining. Yao’s general protocol [145] for secure circuit evaluation can be used, in theory, to compute any function over data partitioned between two parties, without revealing anything to either party beyond the computed output. In the multiparty setting one could use the result by Goldreich et al. [62]. However, because data mining usually involves millions or billions of data items, the communication costs of these protocols render them impractical for these purposes. This has led to the search for problem-specific protocols that have efficient communication complexity. In many cases, the more efficient solutions still make use of a general solution such as Yao’s, but only to carry out a much smaller portion of the computation. The rest of the computation uses other methods to ensure the privacy of the data. This field has attracted the attention of the research community since it was introduced by Lindell and Pinkas [94].

Lindell and Pinkas use cryptographic techniques in their protocol. Their work differs from general secure multi-party computation in the sense that most computation is done locally by the individual parties. The protocol involves a small number of secure evaluations of small-sized circuits (thus resulting in a protocol of low communication complexity). Following the work of Lindell and Pinkas, privacy-preserving distributed protocols have been developed for many data mining problems.

Du and Zhan [40] presented a privacy-preserving protocol for constructing decision trees

for a two-party vertically partitioned database. This was extended to the multiparty case (with three or more parties) by Vaidya and Clifton [134]. They also presented a privacy-preserving association rule mining protocol for vertically partitioned data and a privacy-preserving naive-Bayes classifier protocol for vertically partitioned data [135]. Kantarcioglu and Vaidya presented a naive Bayes classifier protocol for horizontally partitioned data [82]. Kantarcioglu and Clifton [81] gave a privacy-preserving association rule mining algorithm for horizontally partitioned data and Vaidya and Clifton [132] presented the same for vertically partitioned data. A solution for privacy-preserving Bayesian network computation on vertically partitioned data was presented by Yang and Wright [142].

1.2.3 Output Privacy in Data Mining

The protocols obtained in the secure multiparty computation model give a strong privacy guarantee, namely, that nothing is revealed by these protocols beyond what can be inferred from the final output. But sometimes the output itself, possibly combined with external information, or outputs from multiple runs put together, can be used to infer an individual's private information.

Inference Control

Inference control refers to techniques that prevent a user of a database from inferring sensitive data from a sequence of non-sensitive queries. The problem of inference control has been widely studied in regular databases, statistical databases, and even in database systems supporting multilevel security [1, 49, 22, 32, 117]. Adam and Worthmann provide a summary of inference control for statistical databases [1]. The inference control techniques typically fall into three main categories, namely, Query Restriction, Input Perturbation and

Output Perturbation [13]. In the case of query restriction, queries are answered accurately but not all queries are answered. The queries whose answers are seen as potential threats to the privacy of sensitive information are rejected. The most common query restriction techniques include (a) Query-Set-Size, which rejects any query whose output involves a small number of database items, (b) Query-Set-Overlap [35], which bounds the amount of overlap between pairs of queries, (c) Auditing [24], which looks at the history of queries to check if the database is compromised, (d) Partitioning [23], which clusters the database into similar groups and ensures privacy in each cluster, and (e) Cell Suppression [71], which identifies a collection of database entries to be suppressed. In the case of input perturbation, noise is added to the entries of the database prior to the evaluation of queries [2]. In the case of output perturbation, the queries are evaluated on the original data and noisy versions of the “correct” answers are released [1].

On the other hand, the issue of privacy could also be important from the database client’s point of view. A client who queries the database may not want the database server to know what information the client is querying. This could be perhaps because the client wishes to avoid being subjected to targeted marketing, or because the queries themselves could reveal crucial business information. The problem is thus the assurance of the client’s privacy along with an assurance to the server who owns the database that the client does not learn information beyond the query results. Until the work of Woodruff and Staddon [141], previous work on inference control assumed the database server knew what queries were being made and retained this information in deciding whether to allow or deny queries. Some of the work in this thesis generalizes Woodruff and Staddon’s notion of private inference control (see Chapter 6) by applying it to aggregate queries in statistical databases.

Differential Privacy

A new line of study has emerged in the last five years that seeks to guarantee the privacy of every individual in a database. Dinur and Nissim [34] first initiated a formal study of privacy in the context of output perturbation in statistical databases. They defined the notion of privacy with respect to a bounded adversary with no prior knowledge. Their privacy model known as the Sub-Linear Queries Output Perturbation framework, also known as SuLQ, supports queries that return the fraction of database rows that satisfy a given predicate. The database returns a noisy version of the true answer to such a query. Blum et al. [10] used a more general form of these basic SuLQ operations to construct privacy-preserving versions of more complex structures such as k -means clusterings, *ID3* classifiers, perceptrons and others.

The SuLQ frameworks of [34] and [10] led to the development of differential privacy. This model, proposed by Dwork et al. [43], assures that a change in any single record of a database has low impact on the output produced by any mechanism that conforms to the model. The implication is that the data of any single individual in the database is likely to have a very small influence on the output of the database access mechanism. Effectively, no individual “stands out” from the database. Dwork [41] observed that the counting queries used in the SuLQ framework permit differentially private computations of many standard data mining algorithms. Following the work in [43], many differentially private data analysis algorithms and differentially private database query mechanisms have been developed [110, 97, 84, 102, 42, 18]. Most of the work in this framework addresses the problem of issuing noisy answers to client queries (the so-called “interactive mechanisms”). More recently, researchers have studied non-interactive mechanisms [11, 103, 50] where the collector of the data publishes a noisy version of the database.

This thesis addresses data privacy in various stages of extracting knowledge embedded in databases using two different models namely secure multiparty computation and differential privacy that are described in Section 1.2.1. The first part of the thesis addresses privacy in various stages of knowledge discovery when the data is distributed among multiple parties using the secure multiparty computation model. The second part of the thesis deals with privacy in situations where the output of a data mining algorithm itself breaches privacy. We address this problem using both the secure multiparty computation and differential privacy approaches. In the next section, we describe various problems addressed in this thesis and give a brief overview of our results.

1.3 Part-I : Privacy-Preserving Knowledge Discovery in Distributed Databases

The first set of results in this thesis are protocols for privacy-preserving knowledge discovery in databases that are distributed among two parties. All of the results in this part of the thesis fall within the secure multiparty computation model.

1.3.1 Privacy-Preserving Data Imputation

Due to human error and systemic reasons, real-world data sets, particularly those collected from multiple sources, tend to be incomplete, noisy and inconsistent. If unprocessed raw data is used as an input for data mining process, the extracted knowledge is likely to be of poor quality as well. Therefore, data cleaning techniques are applied in order to improve the quality of the data and make it more reliable for mining purposes. Data cleaning algorithms attempt to smooth noise in the data, identify and eliminate inconsistencies, and remove missing values or replace them with values imputed from the rest of the data.

In the case that data is distributed, imputing missing values from “local” data alone

may lead to biases. Performing imputation on all of the distributed data should result in more accurate predictions. Thus, in order to maintain privacy in the data mining process, privacy-preserving methods of data cleaning are required. Such methods are a critical component of enabling privacy-preserving data mining algorithms to be used in practice, because if the data needs to be revealed in order to perform data cleaning, then privacy is lost, while if data cleaning is not performed, then data mining results could be inaccurate.

We provide a privacy-preserving solution to the distributed data imputation problem. In Chapter 3 we present a privacy-preserving protocol for filling in missing values using a lazy decision tree imputation algorithm. The protocol applies to data that is horizontally partitioned between two parties. The participants of the protocol learn only the imputed values; the computed decision tree is not learned by either party. We also show privacy-preserving protocols for several other methods of data imputation. Our results (joint work with Rebecca N. Wright) were originally published in [78].

1.3.2 Privacy-Preserving Clustering Algorithms

Clustering is a well-studied combinatorial problem [70, 79, 69], and there have been a large number of algorithms developed to solve the various formulations of the problem. The task is to group similar items in a given data set into *clusters*. The goal is to obtain a clustering that minimizes an *objective function* mapping a given clustering into a numerical value. We present results for privacy-preserving distributed clustering.

PRIVACY-PRESERVING DISTRIBUTED k -MEANS CLUSTERING OVER ARBITRARILY PARTITIONED DATA.

We introduce the concept of *arbitrarily partitioned data* which generalizes both horizontally and vertically partitioned data. In arbitrarily partitioned data, different attributes for different items can be owned by either party. Although extremely “patchworked” data is unlikely in practice, one advantage of considering arbitrarily partitioned data is that protocols in this model apply both to horizontally and vertically partitioned data, as well as to hybrids that are mostly, but not completely, vertically or horizontally partitioned. We present a privacy-preserving protocol for k -means clustering in this setting of data that is arbitrarily partitioned between two parties. We describe this work in detail in Chapter 4. After the initial publication of this work [76] (joint work with Rebecca N. Wright), Bunn and Ostrovsky [16] presented a secure two-party protocol for k -means clustering of arbitrarily partitioned databases.

COMMUNICATION-EFFICIENT PRIVACY-PRESERVING CLUSTERING ALGORITHMS

Most of the privacy-preserving protocols available in the literature convert existing data mining algorithms or distributed data mining algorithms into privacy-preserving protocols. The resulting protocols either leak intermediate information [134, 80], thereby breaching privacy, or have high communication complexity [16].

In Chapter 5 we describe a k -clustering algorithm that we specifically designed to be converted into a communication-efficient protocol for private clustering. The resulting protocol does not leak any information about the original data. Our algorithm is also I/O-efficient in that each data item is examined only once, and it only uses sequential access to the data. We also present a modified form of our k -clustering algorithm that works for data streams. Our

results (joint work with Krishnan Pillaipakkamnatt, Daryl Umano and Rebecca N. Wright) initially appeared in [75].

1.4 Part-II : Output Privacy in Data Mining

In the second part of the thesis we present privacy-preserving alternatives for data mining algorithms whose outputs themselves breach privacy.

1.4.1 Private Inference Control in OLAP Systems

On-line analytical processing systems, commonly known as OLAP systems, store historical data or data related to multiple organizations and are mainly used in knowledge discovery. Aggregate queries play a major role in OLAP systems. Whenever a database is used to discover knowledge, sensitive information about individuals must be protected. In a system that permits aggregate queries to authorized users, no single query may reveal sensitive data. However, an authorized user might invoke a sequence of queries, each of which is under his privileges, but whose results can be combined to infer some additional sensitive information about the data. Various “inference control” methods have been developed in the past to prevent users from inferring sensitive information through a sequence of queries [1].

Another privacy concern is about the server’s knowledge of the client’s queries. This is because the queries themselves could reveal the intent, knowledge and processes of the client. In Chapter 6 of this thesis we provide solutions to the problem of applying inference control policies to aggregate queries in a privacy-preserving manner. These protocols were originally published in [77] and are joint work with Rebecca N. Wright.

1.4.2 A Practical Differentially Private Random Decision Tree Classifier

The framework of differential privacy for private data analysis assures that any change in a single record of a database has a low impact on the output produced by an appropriate database mechanism. Much of the work in the differential privacy framework addresses the problem of issuing noisy answers to queries whose results can be computed using simple algorithms. These “low-level” queries include simple aggregations such as the SUM and COUNT queries. These queries can be used in the construction of differentially private data mining algorithms such as for decision trees [10]. However, a substantial practical problem arises when complex structures are created using low-level queries via the method described in [10]. If an algorithm makes q such queries, the overall privacy guarantee of the structure is reduced by a factor of q . This increases the amount of noise added to each low-level query, and can have a significant negative impact on the utility of the high-level structure the user wants to compute.

We study the problem of constructing private classifiers using decision trees, within the framework of differential privacy. We first construct privacy-preserving *ID3* decision trees using differentially private sum queries. Our experiments show that for many data sets a reasonable privacy guarantee can only be obtained via this method at a steep cost of accuracy in predictions.

We then present a differentially private decision tree ensemble algorithm using the random decision tree approach. We demonstrate experimentally that our approach yields good prediction accuracy even when the size of the datasets is small. We also present a differentially private algorithm for the situation in which new data is periodically appended to an existing database. Our experiments show that our differentially private random decision tree classifier handles data updates in a way that maintains the same level of privacy

guarantee. We describe this work in Chapter 7. Our algorithms (joint work with Krishnan Pillaipakkamnatt and Rebecca N. Wright), originally appeared in [74].

The research described in this thesis was supported by the National Science Foundation through its grants, CCR-0331584 and CNS-0822269.

Chapter 2

Preliminaries

In this chapter, we define the privacy models and describe the various cryptographic primitives we use in this thesis.

2.1 Privacy Models

Various models have been proposed in the literature for data privacy, including perturbation models, k -anonymity and its variants, secure multiparty computation, and differential privacy. The results in this thesis fall within two of these models, namely secure multiparty computation and differential privacy. We now describe these models in greater detail.

2.1.1 Secure Multiparty Computation

Secure multiparty computation was first introduced by Yao [144] as the following problem for two or more network participants: How can they jointly compute a function of their inputs while satisfying constraints of privacy and security? Privacy is said to be satisfied when no one is able to learn any of the inputs of any of the participants. Security is said to be obtained when no external entity can learn the private inputs of any of the participants, and when no external entity can cause the joint computation to fail.

Secure multiparty computation (SMC) [145, 17] is a model of distributed computation in which n parties P_1, P_2, \dots, P_n who hold data values x_1, \dots, x_n , respectively, wish to evaluate

a function $f(x_1, \dots, x_n)$. If these n parties completely trust each other, then they could send their inputs to each other and they could all evaluate the function f . However, it is often the case that no party completely trusts any other and, thus, party P_i wishes to keep its data value x_i secret. This makes the joint evaluation of f a more difficult problem. If all the n parties trust a third party, they could securely send their inputs to this third party, who can then compute the value of the function f and send the output to all the n parties. This is known as the *ideal model*. The idea behind secure multiparty computation protocols is to have the parties perform the computation themselves without the use of a third party. This is known as the *real model*. Informally, a SMC protocol is said to be secure if any data an adversary can obtain in the real model can also be obtained in the ideal model. The common type of adversaries considered are semi-honest (passive adversaries) or malicious. We describe below more formally the privacy models with respect to these adversaries with respect to the case where $n = 2$ (the two-party case).

Semi-honest Model

A semi-honest adversary faithfully follows the specified protocol, but may retain intermediate messages in an attempt to infer as much information about the other players' secret as possible. A protocol securely computes a functionality f under the semi-honest model if whatever the party can obtain during the run of the entire protocol can be learned from the input and the output available to the party.

Most of the privacy-preserving protocols in this thesis involves sequential invocation of smaller privacy-preserving protocols. The privacy of the composition of the private protocols is provided by the following theorem. Whenever there is a sequential execution of two protocols the output of the first protocol is randomly shared between the parties and

these random shares are presented as inputs to the second protocol.

An oracle-aided protocol is said to be using the oracle-functionality f if the oracle answers according to f . An oracle-aided protocol is said to privately reduce g to f , if it privately computes g when using the oracle functionality f .

Theorem 1 [61] (Composition Theorem for Semi-honest Model) Suppose that g is privately reducible to f and that there exists a protocol for privately computing f . Then, the protocol defined by replacing each oracle-call to f by a protocol that privately computes f , is a protocol for privately computing g .

2.1.2 Differential Privacy

The *differential privacy* model introduced by Dwork et al. [43] assures that the removal or addition of a single item in a database does not have a substantial impact on the output produced by a private database access mechanism.

Let $\mathcal{D}_1, \dots, \mathcal{D}_k$ denote domains, each of which could be categorical or numeric. Our database D consists of n rows, $\{x_1, x_2, \dots, x_n\}$, where each $x_i \in \mathcal{D}_1 \times \dots \times \mathcal{D}_k$. Two databases D_1 and D_2 *differ in at most one element* if one is a proper subset of the other and the larger database just contains one additional row [41].

Definition 1 ([41]) *A randomized database access mechanism \mathcal{M} satisfies ϵ -differential privacy if for all databases D_1 and D_2 differing on at most one element, and all $S \subseteq \text{Range}(\mathcal{M})$,*

$$\Pr[\mathcal{M}(D_1) \in S] \leq \exp(\epsilon) * \Pr[\mathcal{M}(D_2) \in S] \quad (2.1)$$

The probability is taken over the coin tosses in \mathcal{M} .

Note that smaller values of ϵ correspond to higher levels of privacy. Let f be a function on

databases with range \mathbb{R}^m . A standard technique by which a mechanism \mathcal{M} that computes a noisy version of f over a database D can achieve ϵ -*differential privacy* is to add noise from a suitably chosen distribution to the output $f(D)$.

2.2 Primitives

In this section we enumerate and describe the cryptographic primitives we use throughout this thesis. Although we use these primitives as black boxes, the efficiency of our protocols depends on the efficient implementation of these primitives.

2.2.1 Random Shares

A secret sharing method, introduced by Shamir [128], is a way to distribute a secret value among multiple participants such that the secret value can be reconstituted only if a certain minimum number of participants are willing to combine their values. Any combination of shares fewer in number than this threshold reveals nothing about the secret. In this thesis we use two-party secret sharing methods as primitive operations.

We say that two parties Alice and Bob have random shares (or secret shares) of a value x to mean that x is divided into two pieces (shares) a and b such that Alice knows a , Bob knows b , x can be recovered from a and b , and x cannot be recovered without both a and b . In this thesis, we use both additive sharings and XOR sharings. In additive sharing, we choose N to be a large prime and a field \mathbb{F} isomorphic to \mathbb{Z}_N . Alice and Bob have additive random shares of a value $x \in \mathbb{F}$ if Alice knows a random value $a \in \mathbb{F}$ and Bob knows a random value $b \in \mathbb{F}$ such that $(a + b) \bmod N = x$. By XOR sharing, a bit x is shared as $x = a \oplus b$ for random bits a and b . An important property of additive shares is that they allow local computation of additions and subtractions in \mathbb{F} . That is, if Alice and Bob share

values x and y via random shares $a_x, a_y \in \text{Alice}$ and $b_x, b_y \in \text{Bob}$, then $(a_x + a_y) \bmod N$ and $(b_x + b_y) \bmod N$ are random shares of $(x + y) \bmod N$. If $x + y \leq N$, then these are random shares of $x + y$.

2.2.2 Oblivious Transfer

Oblivious transfer, also known as Symmetric Private Information Retrieval (SPIR), is a primitive that allows a receiver to learn one of a number of values held by a sender without revealing which value was learned. 1-out-of-2 oblivious transfer (denoted as OT_1^2), introduced by Rabin [121], is a two-party protocol in which the sender has two messages m_0, m_1 and the receiver has a bit b . At the end of the protocol, the receiver receives m_b without the sender knowing what b is while the sender ensures that the receiver receives only m_b and not the other message. 1-out-of- n oblivious transfer (denoted as OT_1^n) is an extension of OT_1^2 where the sender has n messages m_1, \dots, m_n and the receiver has an index i and the receiver wishes to receive the i th message of the sender without the sender knowing what message that has been sent. Aiello et al. [3] gave an OT_1^n protocol using a homomorphic encryption scheme. This scheme involves the communication of n encryptions and the computational overhead of $O(n)$ public key encryptions. Naor and Pinkas [107] gave an efficient construction for OT_1^n that requires only $\log n$ executions of OT_1^2 .

In this thesis, we use a generalized version of oblivious transfer in which k items are retrieved from a database of n items, where each item is of length ℓ bits. This can be naively achieved by $k\ell$ invocations of OT_1^n . Naor and Pinkas [107] gave an efficient k -out-of- n OT (OT_k^n) protocol which involves $O(k \log n)$ invocations of OT_1^2 .

2.2.3 Yao’s Circuit Evaluation Protocol

Secure multiparty computation refers to methods for a number of distributed participants, each with a secret input, to jointly compute a known function based on their inputs. Yao’s two-party secure circuit-evaluation protocol [145] allows two parties namely Alice and Bob holding inputs a and b respectively to privately compute any function $f(a, b)$ without revealing a and b to each other. Roughly, Alice computes an “encrypted” circuit of the function $f(a, \cdot)$ and sends it to Bob. Bob does not learn anything since the circuit is in an encrypted form. Bob has to decrypt the circuit to learn only the output $f(a, b)$ without learning any intermediate information. Bob accomplishes this by recovering keys from Alice corresponding to its input b using 1-out-of-2 oblivious transfer (OT_1^2) protocols. The performance of Yao’s protocol depends heavily on the size of a Boolean circuit for f and on the performance of OT_1^2 . In theory, Yao’s protocol could be applied to any distributed two-party privacy-preserving data mining problem. However, as a practical matter, the circuits for even simple computations on megabyte-sized databases would be intractably large. We make use of Yao’s protocol for private computation in several cases, but only on functions involving a small number of small inputs.

2.2.4 Homomorphic Encryption

Homomorphic encryption schemes allow certain computations on encrypted values. In particular, an encryption scheme is *additively homomorphic* if there is some operation \otimes on encryptions such that for all plaintext values a and b , $E(a) \otimes E(b) = E(a + b)$. Our solutions make use of a semantically secure additively homomorphic encryption scheme. Examples, under suitable cryptographic hardness assumptions, include the Paillier encryption scheme [114] and the D amgaard-Jurik generalizations of it [28]. In Paillier’s encryption

scheme, the encryption of a message $m \in [1, N]$, where N is an RSA modulus, requires two exponentiations modulo N^2 , and decryption requires one exponentiation. In the Damgård-Jurik encryption scheme messages belong to $[1, N^s]$, where N is an RSA modulus and s is a natural number. In this scheme, the expansion factor is less than that of Paillier's scheme. This allows for efficient encryption of larger messages than Paillier's scheme. Other homomorphic encryption schemes include the Boneh-Goh-Nissim cryptosystem [12], which supports arbitrary addition and one multiplication, and Gentry's fully homomorphic encryption scheme [58], which supports evaluation of arbitrary circuits over encrypted data without having to decrypt. In the Boneh-Goh-Nissim encryption scheme, decryption takes polynomial time in the size of the message space. This is efficient for encrypting short messages. The fully homomorphic encryption scheme can be potentially used to solve most of the problems addressed in this thesis. However, the computational time and the size of the ciphertext are high-degree polynomials in the security parameter. This makes the scheme impractical for many applications.

2.2.5 Secure Scalar Product Protocol

Alice has a vector $X = (x_1, \dots, x_n)$ and Bob has $Y = (y_1, \dots, y_n)$. They need to securely compute the scalar product as $s_A + s_B = X \cdot Y$, where s_A and s_B are the random shares of Alice and Bob. These computations are carried out modulo N . A secure scalar product protocol is given by Goethals et al. [60]. This involves the communication of $O(n)$ encryptions and a computational overhead of $O(n)$ encryptions, $O(1)$ decryptions and $O(n)$ exponentiations.

Part I

Privacy-Preserving Distributed Data Mining

Chapter 3

Privacy-Preserving Imputation of Missing Data

3.1 Introduction

Due to human error and systemic reasons, large real-world data sets, particularly those from multiple sources, tend to be “dirty”—the data is incomplete, noisy, and inconsistent. If unprocessed raw data is used as input for data mining processes, the extracted knowledge is likely to be of poor quality as well. Therefore, *data cleaning*, which seeks to improve the quality of the data and make the data more reliable for mining purposes, is an important preliminary step in KDD. Data cleaning algorithms attempt to smooth noise in the data, identify and eliminate inconsistencies, and remove missing values and replace them with values imputed from the rest of the data.

Since some data can be potentially sensitive, privacy concerns and regulations often prevent the sharing of data between multiple parties. Privacy-preserving distributed algorithms (e.g., [2, 94]) allow cooperative computation of the required function without requiring the participating organizations to reveal their individual data items to each other. Existing privacy-preserving algorithms assume data is complete. In order to make use of those algorithms while maintaining privacy in the data mining process, privacy-preserving methods of data cleaning are also required.

In this chapter, we consider privacy-preserving imputation for data that is horizontally

partitioned between two parties. Our main result is a privacy-preserving imputation algorithm based on ID3 decision trees. In the rest of this introduction, we place our results in the context of related work and describe our results in more detail.

3.1.1 Related Work

Like data mining itself, pre-processing algorithms perform best when they have access to sufficient data. Pre-processing of distributed data has been previously investigated in the context of sensor networks [29, 68]. However, these existing methods are driven by efficiency concerns and do not protect the privacy of the data. In our context, we view privacy as a primary concern (while also seeking as much efficiency as possible, as well as good imputation results).

Several methods for dealing with missing values have been proposed. One general approach to handling missing values is to create data mining algorithms that “internally” handle missing values and still produce good results. For example, the CART decision tree learning algorithm [14] internally handles missing values essentially using an implicit form of imputation based on regression. However, in this work, we follow the more common “modular” approach, where pre-processing is performed first and the resulting data is suitable for use with a variety of data mining algorithms. This is particularly needed in the setting of privacy-preserving data mining because, to date, the existing privacy-preserving data mining algorithms do not make any special internal handling of missing data. A stand-alone approach to privacy-preserving imputation can therefore be used in combination with any existing privacy-preserving data mining algorithm for the same distributed setting. In particular, our results in this chapter are suitable for use with any privacy-preserving data mining algorithm for data that is horizontally partitioned between two parties (e.g., [94, 82]).

Some of the simpler pre-processing techniques for handling missing data have limited applicability or introduce bias into the data [95]. One of the easiest approaches to dealing with missing values is simply to delete those rows that have missing values. However, unless the missing values are distributed identically to the non-missing values, this produces poor quality data [125]. Another common technique for dealing with missing values is to create a new data value (such as “missing”) and use it to represent missing values. However, this has the unfortunate side effect that data mining algorithms may try to use `missing` as a legal value, which is likely to be inappropriate. It also sometimes has the effect of artificially inflating the accuracy of some data mining algorithms on some data sets [55].

Data imputation replaces missing values with estimated values, typically producing better results in subsequent data mining than either of the above methods. Imputation techniques range from fairly simple ideas (such as using the mean or mode of the attribute as the replacement for a missing value [26, 72]) to more sophisticated ones that use regression [6], Bayesian networks [27], and decision tree induction [90]. Using the mean or mode is generally considered a poor choice [95], as it distorts other statistical properties of the data (such as the variance) and does not take dependencies between attributes into account. Hot-deck imputation [51] fills in a missing value using values from other rows of the database that are similar to the row with the missing value. Hot-deck imputation has been performed with k -nearest neighbors algorithms [20, 5] and clustering algorithms [91].

Classification is generally considered the best method for imputing missing data [48]. For each attribute with missing values, the attribute with missing data is used as the dependent attribute (the class attribute), and the remaining attributes are used as the independent attributes for the data mining algorithm. The row with the missing attribute is used as an instance that requires prediction and then the predicted value is used for the

missing value. While any classification or prediction algorithm can be used for imputation, the most commonly used methods are regression-based imputation and decision-tree-based imputation.

Regression imputation [6] imputes missing values with predicted values derived from a regression equation based on variables in the data set that contain no missing values. Regression assumes a specific relationship between attributes that may not hold for all data sets. A privacy-preserving linear regression protocol is presented in [39]; this would be useful for privacy-preserving imputation in cases where the missing values are linearly related with existing data values.

Decision-tree imputation uses a decision-tree based learning algorithm such as ID3 [118] or C4.5 [119] to build a decision-tree classifier using the rows with no missing values, with the attribute that has the missing value as the class attribute. The tree is evaluated on the row with the missing value to predict the missing value [90, 48]. It has been observed [90, 48] that single imputation using decision trees is more accurate than imputation based on clustering. In some cases, the decision tree construction can be *lazy* [55], in that only the needed path or paths of the tree is constructed. This has an efficiency advantage because it avoids time spent on constructing the parts of the tree that will not be needed.

Lindell and Pinkas [94] provide a privacy-preserving algorithm for computing the ID3 tree for databases that are horizontally partitioned between two parties. Although this is the same distributed setting we consider, we are unable to use their solution directly because it allows the parties to learn the computed decision tree. (Indeed, that is its goal.) In our case, we want one or both of our participants to learn the imputed values determined by using the computed decision tree for classification, but we do not want the participants to learn the decision tree itself. Specifically, while our privacy-preserving data imputation

solution uses ID3 trees, it differs from their algorithm in that (for efficiency reasons) we only construct the path that is needed and (for privacy reasons) we use the path for classification without either party learning the constructed path. We do, however, make use of some of Lindell and Pinkas’s subprotocols, which we describe in Section 3.2.3.

3.1.2 Our Contributions

In this chapter, we focus on decision-tree imputation because it generally produces superior results. Our main result is a privacy-preserving data imputation protocol for databases that are horizontally partitioned between two parties. Our solution uses a lazy decision tree algorithm based on ID3 trees. As previously mentioned, the use of a lazy decision tree algorithm provides an efficiency improvement when only a small number of paths of the tree are needed. Our algorithm allows either party to compute missing values without requiring the parties to share any information about their data and without revealing the decision tree or the traversed path to either party. We present two versions of the protocol that represent a privacy/efficiency trade-off. The first version is more efficient, but reveals the number of nodes traversed by the protocol in the undisclosed decision tree. The second version does not leak any information beyond the computed imputed value, but incurs slightly increased communication and computation costs.

We also briefly describe private protocols for data imputation based on other well-known imputation methods—namely, mean, mode, linear regression and clustering, noting that the simpler methods generally produce inferior results unless it is known that the data itself or the data mining methods to be applied on the processed data are suited to those methods.

We begin in Section 3.2 by introducing some definitions and existing subprotocols. We describe our privacy-preserving lazy decision tree imputation protocol in Section 3.3. In

Section 3.4, we outline private protocols for data imputation using the mean, the mode, linear regression, and clustering-based prediction.

3.2 Preliminaries

We describe our solution in the simplified scenario in which there is exactly one missing value to be learned. In practice, the solutions for multiple missing values could be combined to make use of common subproblems for efficiency. In our distributed setting, there are two parties, who we call Alice and Bob. We use the notation $\alpha \in \text{Alice}$ to indicate that Alice holds the value or object α , and analogously, we write $\beta \in \text{Bob}$ if Bob holds β .

Alice and Bob have a horizontally partitioned database. That is, Alice holds a database $D_A = (d_1, \dots, d_m)$ and Bob holds a database $D_B = (d_{m+1}, \dots, d_n)$, which are defined over a common set $\{A_1, \dots, A_\ell\} \cup M$ of attributes. Because the attribute M will be the one in which there is a missing value, we also refer to it as the *class attribute*. Together, D_A and D_B form a single complete joint database $D = D_A \cup D_B$ with no missing values¹. Additionally, there is another instance $I \in \text{Bob}$ (not included in (d_1, \dots, d_n)) that has a missing value for the attribute M . Bob wishes to compute the missing value $I(M)$ using $D = D_A \cup D_B$ via a data imputation algorithm agreed to by both Alice and Bob. Ideally, nothing else about each other's data should be revealed to either party.

Throughout this chapter, n denotes the number of instances in the joint database, ℓ denotes the number of attributes, k denotes the maximum number of values any attribute can take, and g denotes the number of values the class attribute can take. Our solutions also make use of encryption in various places (usually indirectly through their subprotocols). We

¹Technically, these databases are sequences, not sets, but we occasionally abuse notation slightly and treat them as if they were sets.

use w to denote the maximum number of bits required to represent any public key encryption and s to denote the maximum number of bits required to represent any symmetric key encryption. All computations are performed in a field \mathbb{F} of size N .

In the rest of this section, we describe the lazy decision tree algorithm on which our main distributed protocol is based, our privacy model, and some cryptographic primitives we use in our solutions.

3.2.1 Lazy Decision Tree Algorithm

In using a decision tree to predict a single value in a single instance, the entire decision tree is not needed. Rather, only a single path is traversed, whose values are determined by the instance containing the missing value. For efficiency reasons, we therefore use a lazy decision tree. We base our distributed solution on a lazy decision tree algorithm that is a straightforward simplification of ID3.

In this section, we describe this solution as it would proceed in a centralized setting with access to all the joint data. This algorithm lends itself to an efficient privacy-preserving distributed solution. Comparing our algorithm to the lazy decision tree algorithm **LazyDT** of Friedman et al. [55], their algorithm is more complex, less efficient, and less easily amenable for conversion to a privacy-preserving protocol, but also slightly more accurate. (Experiments in [55] indicate that **LazyDT**, on average, has a small improvement in accuracy over ID3 (84% for **LazyDT** vs. 80% for ID3).) As in any lazy learning algorithm, our algorithm does not create an explicit decision tree model from the training data. Instead, the test instance to be classified is used to directly trace the path that would have been taken if an ID3 tree had been built from the training data.

Our algorithm, shown in Figure 1, starts by using ID3’s information gain criterion to

Algorithm 1 Lazy Decision Tree

Input: A database D of labeled instances (with set $R = \{A_1, \dots, A_\ell\}$ of attributes) and an unlabeled instance I to be classified.

Output: A label for instance I .

1. If R is empty, return the majority label of instances in D .
2. If D is pure, return the single occurring label c .
3. Otherwise,
 - (a) for $i = 1$ to ℓ

$$\text{Entropy}(A_i) = \text{Entropy}(D, A_i)$$
 - (b) $A_{\min} = \text{Attribute with least entropy.}$
 - (c) $D = \{X \in D \mid X(A_{\min}) = I(A_{\min})\}$
 - (d) $R = R - \{A_{\min}\}$
4. Go to Step 1

compute the attribute to be tested at the root of the tree. Those rows of the training data which match the test instance on the root attribute are filtered into the next iteration of the algorithm. A database is said to be *pure* if all the instances in it have the same class label. The algorithm repeats the process of choosing an attribute of high information gain, and then winnowing the training data to those instances that match the test data on the chosen attribute. This process is repeated until the set of remaining instances is pure or all attributes have been exhausted. The algorithm then predicts the class label of the test instance as the most frequently occurring class in the remaining set of instances.

The algorithm does not directly calculate the attribute with the highest information gain. Instead, it calculates the attribute that results in the current set of instance having the least amount of entropy. Denote the current set (or subset) of instances by D , the set of values taken by the class attribute by $\{c_1, \dots, c_g\}$, the set of values taken by an attribute A as $\{a_1, \dots, a_k\}$, the set of instances of D in which A has value a_j by $D(a_j)$, and the number of instances of $D(a_j)$ in which the class label is c_i for $1 \leq i \leq g$ by p_{ji} . Then the conditional

entropy after splitting on attribute A is defined as

$$\text{Entropy}(D, A) = - \sum_{j=1}^k \frac{|D(a_j)|}{|D|} \left(\sum_{i=1}^g \frac{p_{ji}}{|D(a_j)|} \log \frac{p_{ji}}{|D(a_j)|} \right)$$

The computation of this algorithm in a distributed privacy-preserving manner is the main result of this chapter and is described in Section 3.3.

3.2.2 Privacy Definition

The paradigm of secure distributed computation provides cryptographic solutions for protecting privacy in any distributed computation [145]. In that setting, the notion of privacy is defined by comparing the information that parties learn in carrying out a distributed protocol to the information that parties would learn if a trusted third party (TTP) were available to perform the computation for them. We use the same privacy definitions, but rather than using the general solutions provided by Yao for secure two-party computation [145], we provide a less general but more efficient solution for our specific two-party computation. We do, however, use general two-party computation as a building block for some smaller parts of our computation to design a tailored, more efficient, solution to privacy-preserving imputation.

As mentioned, our notion of privacy is in relation to the TTP setting in which there is a trusted third party to whom Alice and Bob send their data. The TTP uses the imputation algorithm chosen by Alice and Bob to compute a missing value and sends the computed value to Bob. In a private protocol, Alice and Bob compute the missing value by solely communicating with each other instead of using the trusted third party; in doing so, they should not learn anything that they would not learn in the TTP setting. We assume that both Alice and Bob are *semi-honest*. That is, both parties faithfully follow their specified protocols, but they may record intermediate messages in an attempt to infer information

about the other party’s data. The desired privacy condition in this model is that anything Alice or Bob learns from participating in the protocol could have been learned by simply giving them each their initial input and final output.

The privacy of our solution relies on composition of privacy-preserving protocols. In our solution, we use composition of a number of privacy-preserving subprotocols in which all intermediate outputs from one subprotocol that are inputs to the next subprotocol are computed as secret shares (see Section 3.2.3.) Using composition, it follows that if each subprotocol produces only secret shares and is privacy-preserving, then the resulting composition is also privacy-preserving [61].

3.2.3 Cryptographic Primitives

We use several existing cryptographic primitives, some are described in Chapter 2 and we describe the rest in this section. We assume suitable hardness assumptions under which the various cryptographic primitives described in this section are secure.

PURITY CHECKING PROTOCOL. In a purity checking protocol, Alice has a vector of values $X = (x_1, \dots, x_n)$ and Bob has a vector of values $Y = (y_1, \dots, y_n)$. The protocol outputs c if $x_1 = \dots = x_n = y_1 = \dots = y_n = c$ (i.e., if the set of values in $X \cup Y$ is pure and includes only the value c), or \perp otherwise. The parties learn nothing else. A simple purity checking protocol based on secure equality testing is described in [94].

SECURE $x \ln(x)$ PROTOCOL. When Alice and Bob have random shares of x , denoted as v_1 and v_2 , respectively, a secure $x \ln(x)$ protocol (such as the one in [94]), computes the shares of $x \ln(x)$ as w_1 and w_2 for Alice and Bob, respectively. The parties learn nothing else.

PRIVATE INDIRECT INDEX PROTOCOL. In a private indirect index protocol (PIX), Bob has

a vector of values $X = (x_1, \dots, x_n)$ and Alice and Bob have random XOR shares of an index i . That is, $i_1 \in \text{Alice}$ and $i_2 \in \text{Bob}$ and $i = i_1 \oplus i_2$. As the output of the protocol, Alice and Bob learn random shares of x_i . The parties learn nothing else. A PIX protocol is given in [106]. However, it outputs an XOR-sharing of x_i , while for our purposes we want an additive sharing instead. Fortunately, it can easily be modified to give an additive sharing instead. This protocol requires one invocation of OT_1^n .

3.3 Privacy-Preserving Imputation Based on Lazy Decision Trees

We now describe our main privacy-preserving data imputation protocol. It is based on the lazy decision tree algorithm described in Section 3.2.1. Recall the definitions of D_A , D_B and I given in Section 3.2 and recall that the class attribute M takes the values $\{c_1, \dots, c_g\}$. Bob wishes to compute the missing value $I(M)$ by applying the lazy decision tree imputation on $D = D_A \cup D_B$. At the end of the protocol, Bob learns only the missing value $I(M)$ and Alice learns nothing. Both parties learn nothing else. (For missing values in Alice's database, they would reverse roles from what we describe here.) In particular, Alice and Bob should learn nothing about the path of the decision tree that leads to the missing value including the remaining instances at each node and the value taken by the attributes along the path.

In Section 3.3.1, we first describe a basic version of the protocol. Besides the subprotocols already described in Section 3.2.3, the protocol requires four additional private subprotocols: a protocol that computes split entropy (shown in Section 3.3.3), a split protocol (shown in Section 3.3.4), a protocol that checks if the split database is pure (shown in Section 3.3.5), and a majority computation protocol (shown in Section 3.3.6). These subprotocols themselves use two additional subprotocols presented first in Section 3.3.2.

We discuss the efficiency and privacy of the protocol in Section 3.3.7. As described, the basic version of our protocol has a small privacy leak. Specifically, it reveals to the parties the number of nodes traversed by the protocol in the undisclosed decision tree used for imputation. In Section 3.3.8, we show how this leak can be removed at a slightly increased cost of communication and computation.

3.3.1 Our Basic Protocol

We rephrase the basic steps of the lazy decision tree algorithm from Section 3.2.1 so that it functions more clearly as a data imputation algorithm. The lazy decision tree algorithm computes as the root of the tree the attribute with the highest information gain. This is followed by the repeated execution of two steps until the remaining instances are pure or all attributes have been exhausted: First, extract the subset of the instances that match I on the chosen attribute. Second, choose an attribute of high information gain for the next iteration. Once the repetition of those two steps has completed, the algorithm outputs to Bob the majority label on the remaining instances as the missing value. (Again, if Alice started with the missing value, their roles would be reversed.)

Our privacy-preserving protocol follows the same basic outline. However, the privacy requirements prohibit the protocol from revealing any information other than the final output. In particular, this implies that none of the attributes that have been chosen along the way may be revealed to either party. Further, the subset of the instances that match I on the chosen attributes cannot be revealed either. To meet these requirements, we make extensive use of random sharings. The protocol handles the first condition by storing the index s of the chosen attribute A_s in any iteration as a random sharing between the two parties. That is, Alice would have s_A and Bob would have s_B such that $s_A \oplus s_B = s$. To

satisfy the second condition, the protocol uses a randomly shared bit-vector representation of any $D' \subseteq D$. That is, Alice and Bob have, respectively, bit vectors (p_1, \dots, p_n) and (q_1, \dots, q_n) such that $p_i \oplus q_i = 1$ if $d_i \in D'$ and $p_i \oplus q_i = 0$ otherwise, for $1 \leq i \leq n$. The entropy computed for each attribute in each iteration is also held as random shares by the two parties.

We now describe our protocol in more detail. The complete protocol is shown in Figure 1. At the beginning of the protocol, Alice and Bob jointly check if the database D is pure. If D is pure, Bob learns the unique label. This is done without revealing either their data or one-sided information about purity to each other using a purity checking protocol (see Section 3.2.3).

To compute the root of the lazy decision tree, Alice and Bob compute random shares of the entropy for every attribute $\{A_1, \dots, A_\ell\}$. At the root level, computing the shares of the entropy is straightforward. The conditional entropy of a database D (as explained in Section 3.2.1) with respect to the attribute A can be rewritten as

$$\text{Entropy}(D, A) = -\frac{1}{|D|} \left(\sum_{j=1}^k \sum_{t=1}^g p_{jt} \log(p_{jt}) - \sum_j |D(a_j)| \log(|D(a_j)|) \right) \quad (3.1)$$

where p_{jt} and $D(a_j)$ are as explained in Section 3.2.1. Because the database is horizontally partitioned between Alice and Bob, they can compute shares of p_{jt} and $D(a_j)$ independently. Using a secure $x \log x$ protocol (see Section 3.2.3) and local additions and subtractions, Alice and Bob can jointly compute a random sharing of $\text{Entropy}(D, A)$.

After this computation, Alice has a vector $(\text{Ent}_1^A, \dots, \text{Ent}_\ell^A)$ of entropy shares and Bob, correspondingly, has $(\text{Ent}_1^B, \dots, \text{Ent}_\ell^B)$ such that $\text{Ent}_i^A + \text{Ent}_i^B \equiv \text{Entropy}(D, A_i) \pmod{N}$ for $1 \leq i \leq \ell$. The index of the attributes that yields the least entropy is computed as random shares ($\min_A \in \text{Alice}$ and $\min_B \in \text{Bob}$ such that $\min = \min_A \oplus \min_B$) between Alice and Bob using Yao's protocol. Note that either $\text{Ent}_i^A + \text{Ent}_i^B = \text{Entropy}(D, A_i)$ or

Protocol 1 Private Lazy Decision Tree Imputation

Input: A database $D = D_A \cup D_B$ of labeled instances (with attributes $\{A_1, \dots, A_\ell\} \cup M$), where Alice owns $D_A = (d_1, \dots, d_m)$ and Bob owns $D_B = (d_{m+1}, \dots, d_n)$ and an instance I with a missing value $I(M)$.

Output: Bob learns $I(M)$.

1. Using a purity checking protocol (Section 3.2.3), if D is pure, Bob learns unique label and exits.
 2. Alice and Bob communicate with each other to compute the root attribute using the following steps:
 - (a) For $i = 1$ to ℓ , using a secure $x \log x$ protocol (Section 3.2.3) and local computation, Alice and Bob compute random shares of $\text{Entropy}(D, A_i)$ as $\text{Ent}_i^A + \text{Ent}_i^B \equiv \text{Entropy}(D, A_i) \pmod{N}$.
 - (b) Using Yao's protocol (Section 3.2.3), Alice and Bob compute \min_A and \min_B such that $\min_A \oplus \min_B = \min$ where $\text{Ent}_{\min}^A + \text{Ent}_{\min}^B \equiv \text{minimum}\{\text{Ent}_i^A + \text{Ent}_i^B\}, 1 \leq i \leq m$.
 3. for $j = 1$ to $\ell - 1$
 - (a) Alice and Bob jointly compute the set $D_j = \{d \in D_{j-1} \mid d(A_{\min}) = I(A_{\min})\}$ (represented by bit vectors P and Q) as follows:
 - Alice and Bob compute random shares of $I(A_{\min})$ as $\alpha \in \text{Alice}$ and $\beta \in \text{Bob}$ using PIX (Section 3.2.3) with inputs $\min_A \in \text{Alice}$ and $I, \min_B \in \text{Bob}$.
 - Alice and Bob run the Secure Split protocol of Section 3.3.4 on each of their instances and α, β to obtain two bit vectors $P = (p_1, \dots, p_n) \in \text{Alice}$ and $Q = (q_1, \dots, q_n) \in \text{Bob}$ such that $p_i \oplus q_i = 1$ if $d_i(A_{\min}) = I(A_{\min})$ and $p_i \oplus q_i = 0$ otherwise.
 - (b) Using the Subset Purity Checking protocol of Section 3.3.5, if D_j is pure, Bob learns the unique label and exits. Otherwise:
 - for $i = 1$ to ℓ , Alice and Bob run the Secure Split Entropy protocol of Section 3.3.3 on D_j, P, Q , and A_i to learn shares Ent_A^i and Ent_B^i .
 - Using Yao's protocol (Section 3.2.3), Alice and Bob compute \min_A and \min_B such that $\min_A \oplus \min_B = \min$ where $\text{Ent}_{\min}^A + \text{Ent}_{\min}^B \equiv \text{minimum}\{\text{Ent}_i^A + \text{Ent}_i^B\}, 1 \leq i \leq \ell$.
 4. Using the Majority Label protocol of Section 3.3.6, Bob learns the majority label to use as the missing value $I(M)$.
-

$\text{Ent}_i^A + \text{Ent}_i^B = \text{Entropy}(D, A_i) + N$. The circuit first computes $\text{Ent}_i^A + \text{Ent}_i^B$ and if it is greater than $N - 1$ it subtracts N . It then selects the attribute with the minimum entropy. We emphasize that our privacy criterion requires that the attributes that have been chosen at various stages in the path of the lazy decision tree computation not be revealed to either party. To achieve this, we maintain them as random shares between Alice and Bob.

We write D_j to denote the subset of D that has “filtered” through to level j of the lazy decision tree. To compute the attribute at level j of the lazy decision tree, Alice and Bob should split the database D_{j-1} on the attribute A_s chosen at level $j - 1$. This involves finding $I(A_s)$. Here the instance I is known to Bob, but the attribute index s is randomly shared between Alice and Bob. Alice and Bob compute a random sharing of $I(A_s)$ using PIX. Since D_j should be unknown to either party, we store the set in the form of two bits $p_i \in \text{Alice}$ and $q_i \in \text{Bob}$ per instance ($1 \leq i \leq n$) such that

$$p_i \oplus q_i = \begin{cases} 1 & \text{if } d_i(A_s) = I(A_s) \text{ and } d_i \in D_{j-1} \\ 0 & \text{otherwise} \end{cases}$$

The protocol for performing this computation privately is described in Section 3.3.4. Note that the information about the inclusion of d_i in D_{j-1} is also shared between Alice and Bob. For the root level (which contains all of D), we set $p_i = 1$ and $q_i = 0$ for all i .

If D_j is pure, then Bob learns the unique label using the secure protocol that checks if D_j is pure and returns the unique label. It is important to observe that since neither party knows which of their instances are in D_j , we cannot use here the purity checking protocol described in Section 3.2.3. Instead, we provide an alternate variation of a purity checking protocol, described in Section 3.3.5, that works on a secret-shared representation of the database as required in our case. If D_j is not pure, Alice and Bob engage in the secure

entropy computation protocol of Section 3.3.3 for the split D_j for each of the attributes $\{A_1, \dots, A_\ell\}$, and compute the random shares of the split entropy $\text{Entropy}(D, A_i)$. (i.e., $\text{Ent}_i^A + \text{Ent}_i^B \equiv \text{Entropy}(D, A_i) \bmod N$) The attribute with the least entropy is computed using Yao's protocol. To simplify the protocol, the entropy is evaluated for all attributes at all levels of the decision tree. This does not impact the correctness of the protocol, as the information gain is zero for any attribute that has already been chosen at a previous level.

3.3.2 Secure Computation of the Number of Instances

In this section, we present two protocols for computing shares of the total number of instances in a subset of the database that is only identifiable by random shares held by Alice and Bob, respectively.

Secure Hamming Distance

A secure Hamming distance protocol, shown in Figure 2, is useful for us when applied to a subset $D' \subseteq D$ represented by bit vectors $X = (x_1, \dots, x_n)$ and $Y = (y_1, \dots, y_n)$ known to Alice and Bob, respectively, where $x_i \oplus y_i = 1$ if instance $d_i \in D'$ and $x_i \oplus y_i = 0$ otherwise. At the end of the protocol, Alice and Bob learn random shares of this Hamming distance, which is also equal to $|D'|$. The protocol uses one invocation of SPP. Because of the linear relationship for binary data between the scalar product and the Hamming distance, the parties are then able to compute their needed results locally.

The only communication that happens between the two parties is during the invocation of SPP using two vectors of size n . The communication complexity of the SPP protocol of [60] is $O(wn)$. (Recall that w denotes the number of bits needed to represent a public key encryption.) Its computation complexity involves n encryptions and one decryption

for Alice and n modular exponentiations and one encryption for Bob. The SPP protocol is secure and it provides no useful information other than the random shares to the two parties. Hence, this protocol is secure.

Protocol 2 Secure Hamming Distance

Input: Alice has a bit vector $X = (x_1, \dots, x_n)$ and Bob has a bit vector $Y = (y_1, \dots, y_n)$,

Output: Alice and Bob learn α and β , respectively, such that $(\alpha + \beta) \bmod N = v$, where v is the Hamming distance of X and Y —i.e., total number of entries for which $x_i \oplus y_i = 1$.

1. Alice and Bob run SPP on X and Y to securely compute shares $\mu \in$ Alice and $\lambda \in$ Bob of their scalar product.
 2. Alice computes $\sum_{i=1}^n x_i = \gamma$.
 3. Bob computes $\sum_{i=1}^n y_i = \delta$.
 4. Alice computes $\alpha = \gamma - 2\mu$ and Bob computes $\beta = \delta - 2\lambda$.
-

Secure Computation of the Total Number of Instances in a Subset with a Given Class Label

Let D'_c denote the set of instances in D' in which the attribute M takes the value c . This protocol takes as input a subset $D' \subseteq D$ and a value $c \in \{c_1, \dots, c_g\}$ and computes shares of total number of instances in the set D'_c . The input D' is represented by the bit vectors $X = (x_1, \dots, x_n)$ and $Y = (y_1, \dots, y_n)$ known to Alice and Bob, respectively, where $x_i \oplus y_i = 1$ if instance $d_i \in D'$ and $x_i \oplus y_i = 0$ otherwise.

Alice and Bob first locally determine which of their instances have the right class value c . Specifically, for $1 \leq i \leq m$, Alice computes $w_i = 1$ if $d_i(M) = c$ and $w_i = 0$ otherwise; Bob does the same for $m + 1 \leq i \leq n$. Then the total number of instances in D' with class label c is the number of the instances that satisfy the relation $(x_i \oplus y_i) \wedge w_i = 1$ for $1 \leq i \leq n$. This relation can be rewritten as $(\overline{x_i w_i})y_i \vee (x_i w_i)\overline{y_i} = 1$. Since both $(\overline{x_i w_i})y_i$ and $(x_i w_i)\overline{y_i}$ cannot hold at the same time, this number is equal to the sum of the number of

Protocol 3 Total Instances with Class Label

Input: A value c for attribute M known to both Alice and Bob and a database D of labeled instances where Alice owns $D_A = (d_1, \dots, d_m)$ and Bob owns $D_B = (d_{m+1}, \dots, d_n)$, $D' \subseteq D$ represented by bit vectors $X = (x_1, \dots, x_n) \in$ Alice and $Y = (y_1, \dots, y_n) \in$ Bob such that $x_i \oplus y_i = 1$ if $d_i \in D'$ and $x_i \oplus y_i = 0$ otherwise, for $1 \leq i \leq n$.

Output: Alice and Bob learn γ and δ , respectively, such that $\gamma + \delta \equiv |D'_c| \pmod{N}$, where $|D'_c|$ is the total number of instances for which $x_i \oplus y_i = 1$ and M takes the value c .

1. Alice computes a vector (w_1, \dots, w_m) with $w_i = 1$ if $d_i(M) = c$ and $w_i = 0$ otherwise, for $1 \leq i \leq m$.
 2. Bob computes a vector (w_{m+1}, \dots, w_n) with $w_i = 1$ if $d_i(M) = c$ and $w_i = 0$ otherwise, for $m+1 \leq i \leq n$.
 3. Alice and Bob run SPP four times, as follows:
 - inputs $(\overline{x_1}w_1, \dots, \overline{x_m}w_m)$ and (y_1, \dots, y_m) ; outputs α_1 and β_1 .
 - inputs (x_1w_1, \dots, x_mw_m) and $(\overline{y_1}, \dots, \overline{y_m})$; outputs α_2 and β_2 .
 - inputs $(\overline{x_{m+1}}, \dots, \overline{x_n})$ and $(w_{m+1}y_{m+1}, \dots, w_ny_n)$; outputs γ_1 and δ_1 .
 - inputs (x_{m+1}, \dots, x_n) and $(w_{m+1}\overline{y_{m+1}}, \dots, w_n\overline{y_n})$; outputs γ_2 and δ_2 .
 4. Alice computes $\gamma = \alpha_1 + \alpha_2 + \gamma_1 + \gamma_2$ and Bob computes $\delta = \beta_1 + \beta_2 + \delta_1 + \delta_2$.
-

instances satisfying $(\overline{x_i}w_i)y_i = 1$ and the number of instances satisfying $(x_iw_i)\overline{y_i} = 1$. This is computed as random shares between Alice and Bob using SPP. The complete protocol is shown in Figure 3.

The communication and computation complexity are the same as that of SPP. The privacy of this protocol follows from the privacy of the SPP protocol.

3.3.3 Secure Protocol to Compute Split Entropy

This protocol takes a subset D' of the database D , horizontally partitioned between Alice and Bob, and an attribute A known to both Alice and Bob. The subset D' is represented by two bit vectors P and Q known to Alice and Bob, respectively, such that $p_i \oplus q_i = 1$ if $d_i \in D'$ and $p_i \oplus q_i = 0$ otherwise, for $1 \leq i \leq n$.

The protocol computes random shares of the entropy after splitting D' on the attribute

Protocol 4 Secure Split Entropy

Input: A database D of labeled instances where Alice owns $D_A = (d_1, \dots, d_m)$ and Bob owns $D_B = (d_{m+1}, \dots, d_n)$, an attribute A known to both Alice and Bob (which takes on values a_1, \dots, a_k), and $D' \subseteq D$ represented by bit vectors $P = (p_1, \dots, p_n) \in$ Alice and $Q = (q_1, \dots, q_n) \in$ Bob such that $p_i \oplus q_i = 1$ if $d_i \in D'$, and $p_i \oplus q_i = 0$ otherwise, for $1 \leq i \leq n$.

Output: Random shares of the entropy resulting from splitting D' on attribute A .

1. For $j = 1$ to k

- (a) Compute $D(a_j) \subseteq D'$ in which A has value a_j . $D(a_j)$ is represented as two bit vectors $R \in$ Alice and $S \in$ Bob such that, for $1 \leq i \leq n$,

$$R_i \oplus S_i = \begin{cases} 1 & \text{if } p_i \oplus q_i = 1 \text{ and } d_i(A) = a_j \\ 0 & \text{otherwise} \end{cases}$$

To compute R and S , for each i , Alice and Bob use Yao's protocol where Alice inputs p_i and Bob inputs q_i . If $d_i \in$ Alice, then she inputs $d_i(A)$; otherwise, Bob inputs $d_i(A)$.

- (b) Alice and Bob engage in the **Hamming Distance** protocol of Section 3.3.2 on R and S to compute random shares of $|D(a_j)|$.
- (c) To compute shares of the number p_{jt} of instances of $D(a_j)$ in which the class label is c_t for $1 \leq t \leq g$, Alice and Bob engage in g executions of the **Total Instances with Class Label** protocol of Section 3.3.2.
- (d) They use a secure $x \log x$ protocol to compute random shares of $|D(a_j)| \log |D(a_j)|$, $p_{jt} \log p_{jt}$ for $1 \leq t \leq g$.
- (e) They locally compute random shares of $\text{sum}_j = |D(a_j)| \log |D(a_j)| - \sum_{t=1}^g (p_{jt} \log p_{jt})$.

2. Alice and Bob compute random shares of entropy for the attribute A by locally adding their own shares obtained in Step 1e.

A , which takes on values a_1, \dots, a_k . In the decision tree computation, the entropy after splitting the database D' on attribute A is as shown in Equation (3.1) in Section 3.3.1. Alice and Bob use Yao's protocol to privately compute an XOR sharing representation of $D(a_j) \subset D'$ for $1 \leq j \leq k$. That is, $D(a_j)$ is represented by two bit vectors $R = (R_1, \dots, R_n) \in \text{Alice}$ and $S = (S_1, \dots, S_n) \in \text{Bob}$, where for $1 \leq i \leq n$, $R_i \oplus S_i = 1$ if $p_i \oplus q_i = 1$ and $d_i(A) = a_j$; $R_i \oplus S_i = 0$ otherwise. Using the protocols of Section 3.3.2, Alice and Bob compute random shares of $|D(a_j)|$ and p_{jt} for $1 \leq t \leq g$ and $1 \leq j \leq k$. Random shares of the terms $p_{jt} \log p_{jt}$, for $1 \leq t \leq g$, are computed using a secure $x \log x$ protocol. The complete **Secure Split Entropy** protocol is shown in Figure 4.

The size of the circuit in Step 1a is $O(\log N)$ and this circuit evaluation occurs n times. Hence, the communication complexity is $O(ns \log N)$. The computation complexity is $O(n \log N)$ invocations of OT_1^2 . Step 1b involves two parties jointly computing random shares of $|D(a_j)|$ using the protocol described in Section 3.3.2. Step 1c uses g invocations of the **Total Instances with Class Label** protocol of Section 3.3.2. The $x \log x$ protocol of [94] has a communication complexity of $O(s \log N)$ bits and a computation complexity equal to that of $O(\log N)$ OT_1^2 invocations.

Thus, the total communication complexity of the **Secure Split Entropy** protocol is $O(wkng + (g + n)ks \log N)$. The total computation complexity is the cost of $O((n + g)k \log N)$ OT_1^2 invocations plus the cost of $O(gk)$ **SPP** invocations, which in turn involves a total of $O(gnk)$ encryptions and $O(gk)$ decryptions for Alice and $O(ngk)$ modular exponentiations and $O(gk)$ encryptions for Bob.

The output of the **Secure Split Entropy** protocol is a random sharing of the entropy after splitting D' on the attribute A . Yao's protocol securely computes the subset $D(a_j)$ of the database whose instances take a specific value for the attribute A . This subset is represented

as a random sharing between the two parties. The subprotocols in Section 3.3.2 used to compute the size of the split as random shares between the two parties are secure and do not leak any information. Finally, the $x \log x$ protocol is also secure. Thus, the Secure Split Entropy protocol securely computes random shares of the entropy after splitting D' on the attribute A .

3.3.4 Secure Split Protocol

The Secure Split protocol is used to determine if a given instance is in a subset $D' \subseteq D$ and attribute A_i takes the value v in that instance. The result of this test (0 or 1) is randomly shared between Alice and Bob. Here, inputs v and i are shared between Alice and Bob as $v \equiv a + b \pmod{N}$ and $i = i_1 \oplus i_2$, where a and i_1 are known to Alice and b and i_2 are known to Bob. The inclusion of the given instance in D' is represented by two bits $p \in$ Alice and $q \in$ Bob such that $p \oplus q = 1$ if the instance is in D' and $p \oplus q = 0$ otherwise.

Assuming that the instance belongs to Alice, Alice's inputs to the protocol are the instance (v_1, \dots, v_ℓ) , a , i_1 and p . Bob's inputs are b , i_2 and q . (The case where Bob owns the instance is analogous.) At the end of this protocol, Alice and Bob learn bits b_1 and b_2 , respectively, such that

$$b_1 \oplus b_2 = \begin{cases} 1 & \text{if } v_i = v \text{ and } p \oplus q = 1 \\ 0 & \text{otherwise} \end{cases}$$

In other words, the fact that a given attribute has a specific value in a given instance is available to both Alice and Bob only as random shares.

This protocol has two stages. In the first stage, Alice and Bob use PIX to learn results x and y , respectively, where $(x + y) \equiv v_i \pmod{N}$. In the second stage, Alice has inputs x , a , and p , and Bob has inputs y , b , and q . They use Yao's protocol to learn two bits $b_1 \in$

Alice and $b_2 \in \text{Bob}$ such that

$$b_1 \oplus b_2 = \begin{cases} 1 & \text{if } (x + y) \equiv (a + b) \pmod{N} \text{ and } p \oplus q = 1 \\ 0 & \text{otherwise} \end{cases}$$

The communication and computation complexity of the first stage is that of one invocation of OT_1^ℓ . The second stage of the protocol involves one comparison which is done using Yao's protocol, requiring $O(s \log N)$ bits of communication and $O(\log N)$ invocations of OT_1^2 .

The first stage of the protocol uses PIX, which is secure and produces only random shares as its results. Yao's protocol is also secure and does not leak any information. Thus, the Secure Split protocol is secure.

3.3.5 Secure Protocol to Check if D is Pure

The Subset Purity Checking protocol takes as input a subset D' of a database D and outputs to Bob the value c if all the instances in D' have the same label c or \perp otherwise. In contrast to the purity checking primitive of Section 3.2.3, in this case, Alice and Bob only have an XOR sharing of D' —that is, D' is represented by two bit vectors $P = (p_1, \dots, p_n) \in \text{Alice}$ and $Q = (q_1, \dots, q_n) \in \text{Bob}$ such that $p_i \oplus q_i = 1$ if $d_i \in D'$ and $p_i \oplus q_i = 0$ otherwise.

Let $|D'_c|$ denote the number of instances in D' where the class attribute takes value c . For $1 \leq i \leq g$, Alice and Bob compute random shares α_i^A and α_i^B of $|D'_{c_i}|$ using the Total Instances with Class Label protocol of Section 3.3.2. Alice and Bob then use Yao's protocol to check whether there exists j such that $(\alpha_j^A + \alpha_j^B) \neq 0 \pmod{N}$ and $(\alpha_i^A + \alpha_i^B) \equiv 0 \pmod{N}$ for every $i \neq j$. If so, Bob's output from Yao's protocol is c_j ; otherwise, it is \perp .

Yao's protocol for the comparison described above requires $O(sg \log N)$ bits of communication and $O(g \log N)$ invocations of OT_1^2 . The communication and the computation

complexity is dominated by the g executions of the Total Instances with Class Label protocol. The Total Instances with Class Label protocol returns only random shares to Alice and Bob. Yao's protocol is secure and does not leak any information. Hence, the Subset Purity Checking protocol is secure, in that it does not leak anything other than its output. We note however that this protocol breaks our convention that its output is secret shares. We discuss the resulting privacy implications in Section 3.3.7.

3.3.6 Secure Protocol for Majority Label

The Majority Label protocol is similar to the Subset Purity Checking protocol of Section 3.3.5, except that it always returns the majority class label even if its input database is not pure. In the first stage, for $1 \leq i \leq g$, Alice and Bob using the Total Instances with Class Label protocol of Section 3.3.2 to compute random shares $\alpha_i^A \in \text{Alice}$, and $\alpha_i^B \in \text{Bob}$ of $|D'_{c_i}|$. In the second stage, Alice and Bob use Yao's protocol with inputs α_i^A and α_i^B , respectively, for Bob to learn the index $j = \text{argmax}_{1 \leq i \leq g} |D'_{c_i}|$. Bob computes c_j as the majority label. The complexity of this protocol is the same as the purity protocol. It reveals nothing beyond the majority label c_j to Bob and nothing to Alice.

3.3.7 Performance and Privacy Analysis

Having defined all the subprotocols required by the Private Lazy Decision Tree Data Imputation protocol, we now return to its complexity and privacy.

COMPLEXITY. Alice and Bob communicate with each other to compute the attribute with the maximum information gain at each level of the path. Putting together the cost of all the involved subprotocols, we see that the communication complexity is dominated by $O(\ell^2 nk(wg + s \log N))$. The total computation complexity is that of $O(\ell^2 k(n + g) \log N)$

OT_1^2 invocations, $O(\ell)$ OT_1^ℓ invocations, and $O(\ell^2 kg)$ SPP invocations. The latter involves $O(\ell^2 kgn)$ encryptions and $O(\ell^2 kg)$ decryptions for Alice and $O(\ell^2 kgn)$ modular exponentiations and $O(\ell^2 kg)$ encryptions for Bob.

PRIVACY. We compare this protocol with the one that uses the trusted third party (TTP). When a TTP is used, the two parties send their shares of data to the TTP. The TTP computes the missing value and sends it to Bob, and Alice gets nothing. Both parties receive no other information other than Bob receiving the desired imputed value.

In the **Private Lazy Decision Tree Data Imputation** protocol, the attribute and the split of the database at each level of the path in the decision tree are only available as random shares to Alice and Bob. We have already shown that all the intermediate outputs of the subprotocols are held only as random shares by Alice and Bob and that all the subprotocols do not leak any information beyond Bob learning the final imputation result, with one exception. This exception is that the **Subset Purity Checking** protocol reveals to Bob (and indirectly to Alice, by whether the computation then terminates or continues) whether the split database at each node is pure or not. Because the algorithm gradually splits the database until either the database is pure or all the attributes are exhausted, knowing this information is equivalent to knowing the length of the path. That is, in executing the **Private Lazy Decision Tree Data Imputation** protocol, Bob learns the imputed value, but also both Alice and Bob learn the number of attributes in the path in the decision tree (though not which attributes, nor the values they take), which would not be learned in the TTP solution. The composition result (see Section 3.2.2) implies that they do not learn anything else; that is, **Private Lazy Decision Tree Data Imputation** can be viewed as a private protocol computing an output to Alice consisting of the length of the path in the decision tree and to Bob consisting of the length of the path and the learned value for the missing attribute.

In Section 3.3.8, we describe a modification of the protocol that is completely private, not revealing to Alice and Bob the length of the path.

3.3.8 Enhancing Privacy

As just discussed, the Private Lazy Decision Tree Data Imputation protocol has a minor leak—it reveals the number of nodes in the evaluation path of the decision tree used in the imputation process. In this section, we outline a modification that uses additional secret sharing to eliminate this leak, thus achieving the same level of privacy as in the TTP model.

The modified protocol uses two additional variables: `is_pure` and `majority_class`. We treat `is_pure` as Boolean (with 0 = false and 1 = true). Each of these variables is randomly shared between Alice and Bob, via shares `is_pureA`, `is_pureB`, `majority_classA`, and `majority_classB`. At the end of each iteration, the protocol ensures that $\text{is_pure}^A \oplus \text{is_pure}^B = \text{is_pure}$ and $\text{majority_class} \equiv (\text{majority_class}^A + \text{majority_class}^B) \bmod N$. Initially, `is_pure` is set to false by setting `is_pureA = false` and `is_pureB = false`. If `is_pure` becomes true at the end of some iteration, it means that the protocol has determined that the database D at the beginning of that iteration is pure (all rows have the same class label). At the end of each iteration, the protocol ensures that `majority_class` contains the most frequently occurring class label in database D .

The modified protocol requires changes to Subset Purity Checking and Majority Label (Sections 3.3.5 and 3.3.6). In the modified version of Subset Purity Checking, `is_pureA`, `is_pureB`, `majority_classA` and `majority_classB` are supplied as additional inputs and the protocol checks if `is_pure` is false. If so, the protocol computes random shares of the new values of `is_pure` and `majority_class` and returns these to Alice and Bob. On the other hand, if `is_pure` is true when the subprotocol is invoked, then the values of `is_pure` and `majority_class` do not change,

but a new random sharing of each is computed and sent to Alice and Bob.

In the modified version of Majority Label, the values of is_pure^A , is_pure^B , majority_class^A and majority_class^B are supplied as additional inputs. The subprotocol first checks if is_pure is false. If so, the computation for the majority_class proceeds as usual. The value of majority_class is randomly shared as majority_class^A and majority_class^B . On the other hand, if is_pure is true, the computation of the majority label is skipped. Instead, $(\text{majority_class}^A + \text{majority_class}^B) \bmod N$ is used in place of majority_class , and this is again randomly shared as majority_class^A and majority_class^B .

Given these modifications, the new version of the main protocol runs as many iterations as there are attributes. That is, the iteration of the main loop in the protocol does not end if a pure set D is obtained before all attributes have been used. (Indeed, it could not, as now neither participant knows that this has occurred.) At the end of all iterations, Alice sends the value of majority_class^A to Bob who then uses $\text{majority_class}^A + \text{majority_class}^B \bmod N$ as the imputed value. This eliminates the earlier privacy leak and results in a private protocol for Bob to learn the desired imputed value.

3.4 Other Imputation Protocols

In this section, we briefly describe other private data imputation protocols. As explained in Section 3.1.1, however, the results of the decision tree solution are likely to be better in most settings. Let $\alpha_1, \dots, \alpha_n$ denote the values of the attribute M for which $I \in \text{Bob}$ has a missing value for the instances d_1, \dots, d_n .

MEAN. Because Bob will inevitably be able to compute the sum of Alice's values for attribute M from the mean of their combined values, we can satisfy the privacy requirements

described in Section 3.2.2 with a very simple protocol. Alice computes $a = \sum_{j=1}^t \alpha_j$ and sends a to Bob. Bob computes $b = \sum_{j=(t+1)}^n \alpha_j$ and computes the missing value $I(M)$ as $(a + b)/n$.

MODE. Alice computes the frequencies of the possible values (c_1, \dots, c_g) of M in her database D_A as $\alpha_1, \dots, \alpha_g$ and Bob computes the frequencies in his database D_B as β_1, \dots, β_g . Alice and Bob then use Yao's protocol with inputs $\alpha_1, \dots, \alpha_g$ and β_1, \dots, β_g for Bob to learn the $j = \operatorname{argmax}_{1 \leq i \leq g} (\alpha_i + \beta_i)$. Bob then takes c_j as the missing value.

LINEAR REGRESSION. We describe the protocol for two variables. The extension to multiple regression is straightforward. Suppose x is an independent variable and y is the variable that has a missing value. Linear regression involves fitting the straight line $y = mx + b$, where $m = \frac{\sum_{i=1}^n (x_i y_i) - (\sum_{i=1}^n x_i \sum_{i=1}^n y_i)/n}{\sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2/n}$ and $b = \frac{\sum_{i=1}^n x_i}{n} - m \cdot \frac{\sum_{i=1}^n y_i}{n}$. The operations involved are linear, so Alice and Bob can securely compute shares of m and b . Using these shares, Bob can compute the missing value.

CLUSTERING. Alice and Bob use a secure clustering protocol (such as the one in Chapter 5) to compute shares of m cluster centers, where m is a user-specified parameter. Let $C_i^A = (a_{i1}^A, \dots, a_{ik}^A, p_i^A)$ and $C_i^B = (a_{i1}^B, \dots, a_{ik}^B, p_i^B)$, for $1 \leq i \leq m$, denote the shares of the m cluster centers for Alice and Bob, respectively. Let the instance $I = (x_1, \dots, x_k, x)$, where x denotes the missing value.

Alice and Bob jointly and securely compute the shares of the distance between I and each of the m cluster centers using only the first k coordinates. Let $Z^A = (z_1^A, \dots, z_m^A)$ and $Z^B = (z_1^B, \dots, z_m^B)$ denote Alice's and Bob's shares of the distances, respectively. They use Yao's protocol to compute random XOR shares i_1 and i_2 of the index i such that $z_{i_1}^A + z_{i_2}^B = \min_{1 \leq j \leq m} (z_j^A + z_j^B)$. Bob should learn the missing value as $p_{i_1}^A + p_{i_2}^B$, where $p_{i_1}^A$

and p_i^B are Alice and Bob's shares, respectively, of the missing attribute in cluster C_i . To do this, Alice and Bob then invoke **PIX** twice, once for the vector (p_1^A, \dots, p_m^A) and the other for the vector (p_1^B, \dots, p_m^B) . In both invocations, they supply i_1 and i_2 as input. This gives them random shares of p_i^A and of p_i^B . Alice adds her shares and sends them to Bob for him to compute the missing value.

3.5 Summary

In this chapter, we addressed the problem of computing missing values in a privacy-preserving fashion when the database is horizontally partitioned between two parties. We used a lazy decision tree approach to impute missing values. We presented two versions of the protocol, one of which has a small leak but with lower communication complexity, and another that is fully private. We also demonstrated how to privately perform other common methods of data imputation.

Chapter 4

Privacy-Preserving Distributed k -means Clustering over Arbitrarily Partitioned Data

4.1 Introduction

Data mining is an important part of knowledge discovery that refers to the process of applying specific algorithms for extracting patterns from the data. Well-known data mining tasks include clustering, prediction, association rule mining, and outlier detection [69]. Data mining has been extensively used in fields such as biometrics, including biomedical and DNA data analysis [67], financial data analysis [9], fraudulent pattern analysis, identification of unusual patterns, and analysis of telecommunications data [100].

While much of data mining occurs on data within an organization, it is quite common to use data from multiple sources in order to yield more precise or useful knowledge. However, privacy considerations can prohibit organizations from being willing or able to share their data with each other. Privacy-preserving data mining arose as a solution to this problem by allowing parties to cooperate in the extraction of knowledge without any of the cooperating parties having to reveal their individual data items to each other or any other parties.

In this chapter, we provide a privacy-preserving solution to an important data mining problem, that of clustering data that is shared between two parties. Clustering is a well-studied combinatorial problem [70, 79, 69], and there have been a large number of algorithms developed to solve the various formulations of the problem. The task is to group similar

items in a given data set into *clusters*. The goal is to obtain a clustering that minimizes an *objective function* mapping a given clustering into a numerical value.

We briefly discuss related work to put our solution into context and then describe our contributions more fully.

4.1.1 Related Work

Informally, a clustering algorithm partitions a set of objects into clusters so that all objects within any cluster are “similar”, while objects in different clusters are “dissimilar”. The notions of similarity and dissimilarity are usually based on the set of attributes that defines each object. Clustering has been successfully applied in many domains. In image processing, clustering algorithms are used for image segmentation, which is the problem of distinguishing objects from the background [136]. In bioinformatics, clustering has been used to group genes with similar expression profiles [130]. Astronomers have used clustering to create catalogs of objects in the sky [30]. In this chapter, we present a privacy-preserving clustering protocol.

Privacy-preserving data mining has attracted the attention of many researchers since it was introduced in seminal papers by Agarwal and Srikant [2] and Lindell and Pinkas [94]. Since then a number of algorithms and protocols have been developed for privacy-preserving data mining of both horizontally and vertically partitioned databases.

There are a number of protocols for privacy-preserving clustering. Vaidya and Clifton [133] presented a private protocol for k -means clustering over vertically partitioned data, while the protocol by Jha et al. [80] is for the k -means clustering of horizontally partitioned data. Lin, Clifton and Zhu [93] presented a privacy-preserving clustering algorithm based on EM mixture modelling for horizontally partitioned data. We present in Chapter 5 of

this thesis another privacy-preserving protocol for k -clustering based on a new I/O-efficient clustering algorithm. Prasad and Rangan [116] presented a privacy-preserving version of the BIRCH algorithm for clustering. Bunn and Ostrovsky [16] presented a secure two party k -means clustering protocol for well-separated clusters [112]. The latter two results apply to arbitrarily partitioned data, which will be introduced later in this chapter. (They were published after the work in this chapter was initially published in KDD 2005.) All of the above mentioned achieve privacy in a distributed setting using cryptographic techniques in the secure multiparty computation model. Table 4.1 provides a comparison between some of the existing privacy-preserving clustering protocols.

Privacy-preserving clustering algorithms have also been studied with respect to other privacy models. Oliveira and Zaiane [111] addressed the privacy-preserving data clustering problem using data transformation. Friedman, Wolff and Schuster [54] extended the k -anonymity model and showed how to use it for various data mining applications including clustering. Sakuma and Kobayashi [123] presented a k -means clustering protocol under the privacy-preserving concept called user-centric privacy preservation. In this framework, users can conduct data mining using their private information by storing them in their local storages. After the computation, they obtain only the cluster centers without disclosing private information to others. Recently, Feldman, Fiat, Kaplan and Nissim [50] presented a way to release a differentially private data set on which a user can apply clustering algorithms.

4.1.2 Our Contributions

A cluster is usually represented by its *center* (also called its *centroid* or *mean*), which is the attribute-wise average of all objects in the cluster. A statement for the clustering problem

Algorithm	Applicability	Communication	Privacy
Our Protocol—version 1	arbitrarily partitioned	$O(nk)$	L
Our Protocol—version 2	arbitrarily partitioned	$O(nk)$	F
Vaidya and Clifton [133]	vertically partitioned	$O(nk)$	L
Jha, Kruger and McDaniel [80]	horizontally partitioned	$O(k)$	L
Our Protocol—Chapter 5	horizontally partitioned	$O(k^3)$	F
Bunn and Ostrovsky [16]	arbitrarily partitioned	$O(nk)$	F
Prasad and Rangan [116]	arbitrarily partitioned	$O(nk)$	F

Table 4.1: Comparison of privacy-preserving clustering protocols; in the Privacy column, an entry of L indicates that the protocol leaks candidate cluster centers and an entry of F indicates that the protocol is fully private.

includes some measure of utility, the objective function, which measures the “goodness” of a given clustering. The k -clustering problem takes as an input parameter an integer k that indicates the required number of clusters. The k -means objective function seeks to minimize the sum of the squared distances between objects and their closest cluster centers. The k -means clustering algorithm [52, 96, 99] (also known as Lloyd’s algorithm) is a well-known iterative method that successively refines potential clusters in an attempt to minimize the k -means objective function. Our protocol is based on the k -means clustering algorithm, used in conjunction with the Euclidean metric (also called the L_2 metric) for the distance between data objects. Hence, our protocol works only for numeric attributes.

Our contributions in this chapter are as follows:

- We introduce the notion of arbitrarily partitioned data, which generalizes both horizontally and vertically partitioned data. In arbitrarily partitioned data, different attributes for different objects can be owned by any collaborating party. Although extremely “patchworked” data is unlikely in practice, one advantage of considering arbitrarily partitioned data is that protocols in this model apply both to horizontally and vertically partitioned data, as well as to hybrid data that are mostly, but not completely, vertically or horizontally partitioned.

- Our main contribution is a privacy-preserving protocol for k -means clustering in the setting of arbitrarily partitioned data. Our protocol is efficient and provides cryptographic privacy protection. We also provide an analysis of the performance and privacy of our solution.

In Section 4.2, we provide preliminary definitions that are used in the rest of this chapter and review the k -means clustering algorithm. Section 4.3 describes a first version of our privacy-preserving protocol for k -means clustering when the data is arbitrarily partitioned between two parties. The analysis of the communication complexity and the privacy of the protocol is given in Section 4.4. This protocol reveals the candidate cluster assignments at the end of each iteration. In Section 4.5, we show how to obtain full privacy in our k -means clustering protocol.

4.2 Preliminaries

In this section, we introduce the concept of arbitrarily partitioned data. We also discuss the privacy properties we seek to achieve and briefly review clustering and the k -means clustering algorithm.

4.2.1 Arbitrarily Partitioned Data


In the two-party distributed data setting, two parties (call them Alice and Bob) hold data forming a (virtual) database consisting of their joint data. More specifically, the virtual database $D = \{d_1, d_2, \dots, d_n\}$ consists of n *objects*, or *records*. Each object d_i is described by the values of ℓ numeric attributes. We denote the values of the ℓ attributes of object d_i by $(d_{i,1}, d_{i,2}, \dots, d_{i,\ell})$. When convenient, we sometimes abuse notation and refer to this vector of values as the set $d_i = \{d_{i,1}, d_{i,2}, \dots, d_{i,\ell}\}$. Throughout this chapter, we use n to


denote the size of the database, ℓ to denote the number of attributes, and k to denote the number of clusters.

We introduce in this chapter the concept of *arbitrarily partitioned data*, illustrated in Figure 4.1. In arbitrarily partitioned data, there is not necessarily a simple pattern of how data is shared between the parties. For each d_i , Alice knows the values for a subset of the attributes, and Bob knows the values for the remaining attributes. That is, each d_i is partitioned into disjoint subsets d_i^A and d_i^B such that Alice knows d_i^A and Bob knows d_i^B . We emphasize that the set of attributes whose values are known to Alice for some object d_i does not have to equal the set of attributes whose values are known to Alice for some other object d_j ($i \neq j$). In particular, it is possible that $d_i^A = \emptyset$ or $d_i^A = d_i$. That is, a given object may be “completely owned” by Bob or by Alice. We assume that both parties know the total number n of objects as well as the existence and names of all attributes. Some data values may be known to both parties; even in such cases, we consider such values to be owned by one party, who will be responsible for handling the data value.

Clearly, both horizontally partitioned data and vertically partitioned data can be viewed as specific cases of arbitrarily partitioned data. Specifically, in horizontally partitioned data, each object in the virtual database is “completely owned” by Alice or completely owned by Bob. In vertically partitioned data, each attribute is completely owned by either Alice or Bob, except perhaps for some common “data handle”. As previously mentioned, although extremely patchworked data is unlikely in practice, the generality of this model can make it better suited to practical settings in which data may be mostly, but not completely, vertically or horizontally partitioned.

Our solution also makes use of encryption in various places. We use w to denote the maximum number of bits required to represent any public key encryption and s to denote

 = Attribute names (known to both)

 = Data owned by Alice


 = Data owned by Bob

Figure 4.1: Arbitrarily partitioned data

the maximum number of bits required to represent any symmetric key encryption. All computations are performed in a field \mathbb{F} of size N .

4.2.2 Privacy Properties

Our desired outcome is that clusters are computed on Alice’s and Bob’s joint data. Alice should learn the cluster number for each data object she owns completely, and Bob should learn the cluster number for each data object that he owns completely. For objects that both have attributes for, they should both learn the cluster number. In addition, the parties should either learn random shares (defined in Section 2.2.1) of the final cluster means, or, depending on the desired outcome for the particular application, one or both party should learn the actual final cluster means.

In an ideal world, Alice and Bob would have access to a trusted third party who could perform the necessary calculations. Alice and Bob would securely send their data to this trusted party, who would then compute the cluster to which each object is assigned using the appropriate clustering algorithm and send the desired information to the party that owns the object. However, trusted third parties are hard to find in the real world, and even then, such trust may be misplaced. In this work, we do not rely on a third party. Instead, we provide algorithms by which Alice and Bob can carry out the functionality that would be provided by the trusted third party *without* actually requiring such a party or requiring the parties to send their data to each other.

We assume that both Alice and Bob are *semi-honest*. That is, both parties faithfully follow their specified protocols, but we assume they record intermediate messages in an attempt to infer as much information about the other party’s data as possible. Additional cryptographic techniques, such as zero knowledge proofs, can be used to provide privacy

even against malicious behavior, but typically at a significant performance cost.

As we discuss further in Section 4.4, our first solution leaks some additional information beyond just the cluster numbers, in that it reveals some information about the progress of the computation as it is carried out. This information does not appear to be useful except perhaps in degenerate cases, and some protection can be provided by having the initial choice of candidate cluster centers be chosen randomly. In Section 4.5, we also provide a modified protocol which obtains full privacy.

4.2.3 Clustering and the k -Means Algorithm

We briefly review the k -means clustering algorithm. For a more detailed description, see [96].

We consider a database $D = \{d_1, d_2, \dots, d_n\}$ to be a subset of points in \mathbb{R}^ℓ . We denote the j^{th} coordinate of a point $x \in \mathbb{R}^\ell$ by x_j . Similarly, the j^{th} coordinate of $d_i \in D$ is written as $d_{i,j}$. For two points $x, y \in \mathbb{R}^\ell$, we use $\text{dist}(x, y)$ to denote the Euclidean distance between x and y . A k -clustering of D is a partitioning of D into k disjoint subsets $\{C_1, C_2, \dots, C_k\}$, for some $k \geq 1$. The center c_m of a cluster C_m is the coordinate-wise average of all the points in C_m . That is,

$$c_{m,j} = \frac{1}{|C_m|} \sum_{x \in C_m} x_j.$$

The error sum of squares ESS_m of a cluster C_m is the sum of the squared distances of the points in the cluster to the center of the cluster:

$$\text{ESS}_m = \sum_{x \in C_m} \text{dist}^2(x, c_m).$$

The error sum of squares of a k -clustering is $\sum_{i=1}^k \text{ESS}_i$. The k -means clustering problem seeks a k -clustering of D with the lowest error sum of squares. The k -means algorithm is a heuristic for the k -means clustering problem.

The k -means algorithm proceeds iteratively, successively refining k potential clusters until a specified termination condition is reached. Initially, the algorithm arbitrarily selects k of the objects in D as k initial candidate cluster centers. Each object in D is assigned to the cluster whose center is closest to it. Subsequently, the clusters centers and corresponding cluster assignments are modified with the intention of reducing the average distance of each object to its closest candidate cluster center. In each iteration, each object in the database D is reassigned to the cluster whose candidate center is closest. The current center of each cluster is then recomputed on the basis of the modified set of the objects in the cluster. This iterative process continues until a specified termination condition is met. A commonly used condition terminates the process when the change in the average distance of each object to its closest cluster center is small. Another commonly used condition is the absence of any change in cluster composition. In this solution, we use a third commonly used termination condition, terminating when the change in the centers of all clusters falls below a prespecified threshold value. This version is illustrated in Algorithm 2.

Algorithm 2 k -means clustering algorithm

Input: Database D , integer k

Output: Cluster centers μ_1, \dots, μ_k

1. Arbitrarily select k objects from D as initial cluster centers μ'_1, \dots, μ'_k .
 2. Repeat
 - (a) $(\mu_1, \dots, \mu_k) = (\mu'_1, \dots, \mu'_k)$
 - (b) Assign each object $d_i \in D$ to the cluster whose center is closest.
 - (c) Recompute the centroids of the k clusters as $\mu'_1 \dots \mu'_k$.
- until (μ_1, \dots, μ_k) is close to (μ'_1, \dots, μ'_k)
-

4.3 Privacy-Preserving k -Means Clustering Protocol

We now describe our privacy-preserving k -means clustering protocol. Alice and Bob share a dataset in an arbitrary partition, as explained in Section 4.2.1. They wish to learn the k -means clustering of the virtual database comprising their joint data, without revealing their data to each other or anyone else. The final output of the algorithm should be an assignment of a cluster number between 1 and k to each object. If an object is shared, then both parties learn the assignment; otherwise, only the party who owns the object gets the assignment. The algorithm can terminate with the final mean of each cluster randomly shared by both parties, or if desired, one or both parties can learn the actual final means.

As described in Section 4.2.3, the k -means clustering algorithm is an iterative algorithm that successively refines candidate cluster centers. Initially, k candidate cluster centers are chosen from the data points in D . This can be done in one of several ways: for example, they can be taken as a prespecified set of points, such as the first k data points, they can be chosen randomly by one or the other party, or they can be chosen randomly with randomness introduced by both parties. As we discuss in Section 4.4, the latter method provides the most robustness against certain kinds of potential privacy leaks.

In each iteration of the algorithm, each object is assigned to the closest candidate cluster center. Then, new candidate cluster centers are determined based on the actual centers of the points as they have been assigned. This procedure is repeated until a specified termination condition is reached. Our privacy-preserving distributed version follows the same iterative structure, but protects the data and intermediate values from being learned by the parties. As we now describe, this is done using interaction, privacy-preserving subprotocols, and random sharings.

Protocol 5 Privacy-preserving k -means clustering over arbitrarily partitioned data

Input: Database D with n objects, integer k denoting the number of clusters

Output: Assignment of the cluster numbers to the objects

1. Randomly select k objects from D as initial cluster centers μ_1, \dots, μ_k .
2. Randomly share the cluster centers between Alice and Bob:

$$\text{Alice's share} = (\alpha_1^A, \dots, \alpha_k^A)$$

$$\text{Bob's share} = (\alpha_1^B, \dots, \alpha_k^B)$$

3. Repeat

(a)

$$(\mu_1^A, \dots, \mu_k^A) = (\alpha_1^A, \dots, \alpha_k^A)$$

$$(\mu_1^B, \dots, \mu_k^B) = (\alpha_1^B, \dots, \alpha_k^B)$$

(b) For each d_i in D

- i. Run the secure closest cluster protocol
- ii. Assign to d_i the closest cluster

(c) Alice and Bob securely recompute random shares of the centroids of the k clusters as $(\alpha_1^A, \dots, \alpha_k^A)$ and $(\alpha_1^B, \dots, \alpha_k^B)$ respectively.

until $(\mu_1^A + \mu_1^B, \dots, \mu_k^A + \mu_k^B)$ is close enough to $(\alpha_1^A + \alpha_1^B, \dots, \alpha_k^A + \alpha_k^B)$

In each iteration of the privacy-preserving protocol, the candidate cluster centers are calculated as a random sharing between the two parties. Let ℓ denote the number of attributes and $d_i = (d_{i,1}, \dots, d_{i,\ell})$ for $1 \leq i \leq n$ denote the i^{th} object in the database. Let μ_j^A for $1 \leq j \leq k$ denote Alice's share of the j^{th} mean, μ_j^B for $1 \leq j \leq k$ denote Bob's share. The candidate cluster centers are given by $\mu_j^A + \mu_j^B$ for $1 \leq j \leq k$. For each object d_i , Alice and Bob securely compute the distances $\text{dist}(d_i, \mu_j)$ for $1 \leq j \leq k$, between the object and the k cluster centers. The result of the distance calculation is learned as random shares between Alice and Bob. Using these random shares, they then securely compute the closest cluster for each object in the database.

At the end of each iteration, Alice and Bob learn the cluster assignment for the objects, as follows: If an object d_i is shared by both Alice and Bob, then both of them learn the cluster to which the object belongs. If not, it is revealed only to the party who completely owns d_i . The iterative process is repeated until the change in the centers of all clusters falls below a certain specified threshold value.

When the termination condition is reached, the parties may wish to send each other their cluster center shares so that they can learn the actual cluster centers of the joint data. Whether this is desirable or not depends on the particular application for which the clustering results are intended to be used.

Protocol 5 illustrates the overall privacy-preserving k -means clustering protocol. In the remainder of this section, we describe in detail each of the subprotocols used in our clustering protocol. In Section 4.3.1, we present a secure protocol that computes the cluster assignment for each object. In Sections 4.3.2 and 4.3.3, we present secure protocols for recomputing the cluster centers and for iteration termination, respectively.

4.3.1 Secure Protocol to Compute Closest Cluster

This subprotocol takes an arbitrarily partitioned object and k randomly shared candidate cluster means as its input. It outputs to the appropriate party or parties the closest cluster to which the object belongs. It reveals no other additional information.

Consider an object $d_i = (d_{i,1}, \dots, d_{i,\ell})$. This object can be owned by Alice or Bob or shared by both. Suppose that the object is shared by both Alice and Bob. Without loss of generality, assume $d_{i,p_1}, \dots, d_{i,p_s}$ belong to Alice and the rest of the $\ell - s$ attributes belong to Bob. To compute the closest cluster the protocol first computes the distance between the object d_i to each of the k clusters. That is, for each cluster $1 \leq j \leq k$, we compute the squared distance $\text{dist}^2(d_i, \mu_j)$ between the object d_i and the cluster mean μ_j . The squared distance $\text{dist}^2(d_i, \mu_j)$ is defined by the following equation:

$$\text{dist}^2(d_i, \mu_j) = (d_{i,1} - \mu_{j,1})^2 + (d_{i,2} - \mu_{j,2})^2 + \dots + (d_{i,\ell} - \mu_{j,\ell})^2.$$

Since $\mu_{j,t} = \mu_{j,t}^A + \mu_{j,t}^B$, where $\mu_{j,t}^A$ is Alice's share of the mean and $\mu_{j,t}^B$ is Bob's share of the mean,

$$\begin{aligned} \text{dist}^2(d_i, \mu_j) &= (d_{i,1} - (\mu_{j,1}^A + \mu_{j,1}^B))^2 + \dots + (d_{i,\ell} - (\mu_{j,\ell}^A + \mu_{j,\ell}^B))^2 \\ &= \sum_{m=1}^{\ell} d_{i,m}^2 + \sum_{m=1}^{\ell} (\mu_{j,m}^A)^2 + \sum_{m=1}^{\ell} (\mu_{j,m}^B)^2 + 2 \sum_{m=1}^{\ell} \mu_{j,m}^A \mu_{j,m}^B \\ &\quad - 2 \sum_{m=1}^{\ell} \mu_{j,m}^A d_{i,m} - 2 \sum_{m=1}^{\ell} d_{i,m} \mu_{j,m}^B \end{aligned}$$

To privately compute the term $\sum_{m=1}^{\ell} d_{i,m}^2$ over the shared data, Alice computes the sum that involves the components $d_{i,p_1}, \dots, d_{i,p_s}$, while Bob computes the sum of the rest of the components. The second term $\sum_{m=1}^{\ell} (\mu_{j,m}^A)^2$ is completely computed by Alice and similarly the third term is computed by Bob. They then use a secure scalar product protocol, such as the one described in [60], to compute random shares of the last three terms. Since a sum

involving random shares results in random shares, we have

$$\text{dist}^2(d_i, \mu_j) = \alpha_{i,j} + \beta_{i,j}.$$

Here $\alpha_{i,j}$ and $\beta_{i,j}$ are random shares known to Alice and Bob respectively, where

$$\begin{aligned}\alpha_{i,j} &= \sum_{m=1}^s d_{i,p_m}^2 + \sum_{m=1}^{\ell} (\mu_{j,m}^A)^2 + a_j + c_j + e_j \\ \beta_{i,j} &= \sum_{m=1}^{\ell-s} d_{i,q_m}^2 + \sum_{m=1}^{\ell} (\mu_{j,m}^B)^2 + b_j + d_j + f_j \\ a_j + b_j &= 2 \sum_{m=1}^{\ell} \mu_{j,m}^A \mu_{j,m}^B \pmod{N} \\ c_j + d_j &= -2 \sum_{m=1}^{\ell} \mu_{j,m}^A d_{i,m} \pmod{N} \\ e_j + f_j &= -2 \sum_{m=1}^{\ell} \mu_{j,m}^B d_{i,m} \pmod{N}\end{aligned}$$

and N is the size of the chosen finite field.

Alice has a k -length vector $A = (\alpha_{i,1}, \dots, \alpha_{i,k})$ and Bob has $B = (\beta_{i,1}, \dots, \beta_{i,k})$. They securely compute the index j such that $\alpha_{i,j} + \beta_{i,j}$ is minimum using Yao's circuit evaluation [145]. Since k is typically quite small, particularly in comparison to the number of data items, the overhead this requires should be sufficiently small. The object d_i is assigned to cluster j .

COMMUNICATION AND COMPUTATIONAL COMPLEXITY

For each object d_i , this protocol computes the closest cluster. Each object has ℓ components. Communication is involved when the two parties engage in the secure scalar product computation to compute the distance between an object d_i and each of the cluster centers. For each distance computation, the scalar product protocol is invoked three times between vectors of length ℓ . The communication complexity of each scalar product protocol is $O(w\ell)$.

Hence it requires a communication of $O(kw\ell)$ bits to compute the distance of the object d_i from all the k centers. Note that w denotes the maximum number of bits required for public-key encryption.

The overall communication complexity for one invocation of the closest cluster protocol is $O(k(w\ell + s \log N))$. The estimation of the computational complexity involves counting the number of encryptions, decryptions and exponentiations for Alice and Bob. To compute the closest cluster for each object, the scalar product protocol is executed for each cluster three times between vectors of length ℓ . For each execution of the scalar product protocol, Alice performs ℓ encryptions and one decryption. Bob performs ℓ exponentiations and one encryption. Since there are k clusters, the total computational complexity for computing the closest cluster for an object is $O(k\ell)$ encryptions and $O(k)$ decryptions for Alice and $O(k\ell)$ exponentiations and $O(k)$ encryptions for Bob. The secure scalar product protocol described in [60] uses public key encryption schemes and hence increases the computational complexity. But several optimization tricks are presented in [60] to make the computational complexity tolerable for both Alice and Bob. An experimental analysis of the scalar product protocol between two binary vectors held by two parties is presented by Yang, Wright and Subramaniam [143]. The invocations of Yao's protocol require $O(k \log N)$ invocations of OT_1^2 .

PRIVACY

The above protocol securely computes the index of the closest cluster for every object. The only information revealed is the output, which is the cluster index. The distance between each object d_i to each of the k cluster centers is available only as a random sharing between Alice and Bob. Hence each cannot obtain any information about the other party's data.

For each distance computation, the communication between Alice and Bob occurs through the secure scalar product protocol. The scalar product protocol leaks no information and returns only a random share to both Alice and Bob. Thus the distance function is computed as a random sharing between Alice and Bob, without leaking any information. Finally, Yao's circuit evaluation protocol securely computes the closest cluster. The output of the protocol is an assignment of the closest cluster of a given point. This is the only information available to the parties at the end of the protocol when a trusted third party is used.

4.3.2 Secure Protocol for Recomputing the Mean

At the end of each iteration Alice and Bob learn his or her share of the cluster centers. The next iteration requires recomputing each of the k -cluster centers. Let us assume that Alice has objects $d_{i_1}^A, \dots, d_{i_p}^A$ and Bob has objects $d_{i_1}^B, \dots, d_{i_q}^B$, for each cluster $1 \leq i \leq k$. Each of the d_i^A and d_i^B is an ℓ -tuple, where $d_{i,j}^A$ and $d_{i,j}^B$ each denote the j th coordinate of the respective ℓ -tuple.

Alice calculates the shares s_j and n_j for $1 \leq j \leq \ell$, where $s_j = \sum_{r=1}^p d_{i_r,j}^A$ and n_j denotes the number of objects in $d_{i_1}^A, \dots, d_{i_p}^A$ for which Alice has the values for attribute j . If Alice does not have the value for attribute j for some object d_i^A , she treats it as zero in the computation of the above sum.

Similarly, Bob calculates the shares t_j and m_j for $j, 1 \leq j \leq \ell$, where $t_j = \sum_{r=1}^q d_{i_r,j}^B$ and m_j denotes the number of objects in $d_{i_1}^B, \dots, d_{i_q}^B$ for which Bob has the values for attribute j . Again, if Bob does not have the value for attribute j for some object d_i , he treats it as zero in the computation of the sum. The j th component of the i th cluster center is given by

$$\mu_{i,j} = \frac{s_j + t_j}{n_j + m_j}$$

Since this involves only four values, it can be computed efficiently using Yao’s [145] circuit evaluation protocol as random shares between Alice and Bob. Alternatively, one could use Bunn and Ostrovsky’s division protocol [16], which was presented after the initial publication of our results [76].

COMMUNICATION AND COMPUTATIONAL COMPLEXITY

Each component of the mean requires communication of $O(s \log N)$ bits. Hence the communication complexity to recalculate the means is $O(k\ell s \log N)$. The computational complexity is $O(k\ell \log N)$ invocations of OT_1^2 .

PRIVACY

The protocols we use to recalculate the means are secure and they do not leak any information. Each party learns only a random share of each mean and thus cannot infer any information about the means of the clusters.

4.3.3 Secure Protocol for Termination of Iterations

The k -means clustering algorithm is an iterative algorithm. At the end of every iteration, the cluster centers are recalculated. The iterative process is terminated when there is no substantial change in the cluster means. In our case, the algorithm terminates when the Euclidean distance between the cluster centers between two consecutive iterations is less than a specified value ϵ . Denote Alice’s share of cluster centers at the end of the i th iteration by $\mu_1^{A,i}, \dots, \mu_k^{A,i}$ and Bob’s share by $\mu_1^{B,i}, \dots, \mu_k^{B,i}$. Similarly, denote Alice’s share of cluster centers at the end of the $(i+1)$ st iteration by $\mu_1^{A,i+1}, \dots, \mu_k^{A,i+1}$ and Bob’s share by $\mu_1^{B,i+1}, \dots, \mu_k^{B,i+1}$. Each μ_i is an ℓ -tuple and is denoted by $(\mu_{i,1}, \dots, \mu_{i,\ell})$. For $1 \leq j \leq k$,

we securely compute the square of the distances:

$$\begin{aligned}
& \text{dist}^2(\mu_j^{A,i+1} + \mu_j^{B,i+1}, \mu_j^{A,i} + \mu_j^{B,i}) \\
&= \text{dist}^2((\mu_{j,1}^{A,i+1} + \mu_{j,1}^{B,i+1}, \dots, \mu_{j,1}^{A,i+1} + \mu_{j,1}^{B,i+1}), \\
&\quad (\mu_{j,1}^{A,i} + \mu_{j,1}^{B,i}, \dots, \mu_{j,1}^{A,i} + \mu_{j,1}^{B,i})) \\
&= \sum_{m=1}^{\ell} (\mu_{j,m}^{A,i} + \mu_{j,m}^{B,i})^2 + \sum_{m=1}^{\ell} (\mu_{j,m}^{A,i+1} + \mu_{j,m}^{B,i+1})^2 - \\
&\quad 2 \sum_{m=1}^{\ell} ((\mu_{j,m}^{A,i+1} + \mu_{j,m}^{B,i+1})(\mu_{j,m}^{A,i} + \mu_{j,m}^{B,i})) \\
&= \alpha_j + \beta_j,
\end{aligned}$$

where α_j is Alice's share of the distance and β_j is Bob's share. These are random shares. Alice and Bob have k -length vectors $(\alpha_1, \dots, \alpha_k)$ and $(\beta_1, \dots, \beta_k)$ respectively. They communicate with each other and securely check if $\alpha_j + \beta_j < \epsilon$ for $1 \leq j \leq k$. As before, since the amount of input involved is small, this can be securely and efficiently implemented using Yao's protocol.

COMMUNICATION AND COMPUTATIONAL COMPLEXITY

To check if the algorithm terminates, Alice and Bob have to check that the Euclidean distance between two consecutive iterations is less than ϵ for each of the k cluster centers. This involves executing the scalar product protocol four times to obtain the random shares of the distance vectors and then applying Yao's protocol. It requires a communication of $O(kw\ell)$ bits to compute the distance between two consecutive iterations of all the k centers. Yao's circuit evaluation involves $O(k\ell \log N)$ bits of communication to check if all the k distances are less than ϵ . The total computational complexity is $O(kw\ell)$ encryptions and $O(k)$ decryptions for Alice and $O(k\ell)$ exponentiations and $O(k)$ encryptions for Bob. The use of Yao's protocol requires $O(k\ell \log N)$ invocations of OT_1^2 .

PRIVACY

The scalar product protocol and Yao's circuit evaluation protocol used in this protocol are secure and they leak no information. So the only information that Alice and Bob can obtain at the end of the execution of the protocol is the output whether the iterations can be terminated or not.

4.4 Performance Analysis

Putting all the subprotocols together as shown in Protocol 5 yields the complete privacy-preserving k -means clustering protocol over arbitrarily partitioned data.

COMMUNICATION AND COMPUTATIONAL COMPLEXITY

As described in Section 4.3.1, the two parties need to communicate $O(k(w\ell + s \log N))$ bits to determine the closest cluster for each point. Since there are n objects in the database, the total communication complexity to assign cluster numbers in each iteration is $O(nk(w\ell + s \log N))$ bits. At the end of each iteration, the protocol recomputes the mean of each of the k clusters. Each mean is a ℓ -length vector. The total communication complexity to compute all k -means is $O(s\ell k \log N)$. The k -means clustering algorithm is an iterative algorithm and the iterative process is repeated until the improvement to the approximation are within the threshold. The means at the end of each iteration are shared between Alice and Bob. It takes $O(w\ell k)$ bits of communication for Alice and Bob to securely check the termination criterion at the end of each iteration. Thus, the overall communication complexity for the privacy-preserving k -means clustering protocol is $O(nk(w\ell + s \log N))$ per iteration. It has been shown that for typical data, the number of iterations required is usually quite small.

For each iteration the total computational complexity for computing the cluster centers

is $O(nk\ell)$ encryptions and $O(nk)$ decryptions for Alice and $O(nk\ell)$ exponentiations and $O(nk)$ encryptions for Bob, and $O(kn \log N)$ invocations of OT_1^2 .

PRIVACY

We analyze the privacy of the k -means clustering algorithm on arbitrarily partitioned data. As a privacy ideal, we compare this protocol with one that makes use of a trusted third party (TTP) who receives all the data, locally runs the standard k -means algorithm, and computes the appropriate cluster numbers (and possibly cluster centers) to the parties. If one party completely owns an object, then only that party gets the cluster number assigned to that object. If both parties shares an object then both parties gets the assignment.

In contrast, the privacy-preserving k -means clustering protocol is an interactive protocol. The following information is revealed to Alice and Bob at the end of each iteration:

- The assignment of the cluster number to the objects. If one party holds the object completely, then only that party gets the assignment. Otherwise, both parties get the assignment.
- Each party computes its share of the mean of each cluster.

In the TTP case, the assignment to the clusters are revealed only at the final step. In contrast, in our protocol both parties learn the candidate cluster assignments at the end of each iteration.

This intermediate information can sometimes reveal information about the parties' data. For example, consider a database consisting of n objects, say d_1, \dots, d_n , and suppose that each object has two attributes x and y . Further assume that Alice has the values corresponding to the attribute x and Bob has values corresponding to the attribute y . Suppose

$d_1[x] = d_2[x] = \dots = d_n[x] = 0, d_n[y] = 4$, and $d_1[y]$ through $d_{n-1}[y]$ ranges from 0 to 3. Suppose d_1, d_3 and d_n are chosen as the initial choices for the means ($k = 3$). If one of the clusters in the next iteration contains only d_n , then Alice can guess that $d_n[y]$ is close to 4. But the result of the final iteration may not have a cluster having $d_n[y]$ as a single point. However, the likelihood of examples such as this occurring can be reduced by choosing the initial means at random.

Within each iteration, both parties get their shares of the cluster. They can calculate their shares of the k means. As discussed in Sections 4.3.1 and 4.3.2, the protocols to compute closest cluster and to recalculate the mean do not reveal any additional information other than the assignment of the cluster to the objects and a random share of the means.

We note that our protocol when restricted to vertically partitioned data, is similar to the one given by Vaidya and Clifton [133] when restricted to two parties. The communication complexity and the privacy of our protocol is essentially the same as theirs. Our protocol when restricted to horizontally partitioned data is similar to the one given by Jha, Kruger, and McDaniel's [80]. The communication complexity and the privacy of our protocol is same as theirs. The protocols presented in [133, 80] and our protocol presented in Section 4.3 leak the candidate cluster assignments at the end of each iteration. In Section 4.5, we present a modification of the Protocol 5 which prevents Alice and Bob from learning the candidate cluster assignments.

4.5 Enhancing Privacy

As discussed in Section 4.4, the protocol of Section 4.3 has a leak—it reveals the candidate cluster assignments at the end of each iteration. In this section, we outline a modified version of the protocol which eliminates this leak, thus achieving the same level of privacy

as in the TTP model. This new protocol is the same as Protocol 5, except for Steps 3b and 3c. The basic idea is to randomly share the cluster assignments between the two parties in each iteration and use those random shares to recompute the mean. Let d_i^c denote the cluster assignment of the object d_i at the end of an iteration. We associate with each object d_i a k -bit vector e_i , $1 \leq i \leq n$ where for $1 \leq j \leq k$,

$$e_{i,j} = \begin{cases} 1 & \text{if } d_i^c = j \\ 0 & \text{otherwise} \end{cases}$$

In the modified protocol, Alice and Bob compute as random shares the distance between an object d_i and each of the k cluster centers (Section 4.3.1). Then, they use Yao's circuit evaluation protocol [145] to compute random shares of e_i . Let e_i^A denote Alice's share of e_i and e_i^B denote Bob's share such that $e_{i,j} \equiv e_{i,j}^A + e_{i,j}^B \pmod{N}$. This step ensures that cluster assignments in each iteration are not available to either Alice or Bob.

The next step is to recompute the cluster centers at the end of each iteration, given that the cluster assignment for each object is randomly shared between Alice and Bob. Without loss of generality, assume $d_{i,p_1}, \dots, d_{i,p_s}$ belong to Alice and the rest of the $\ell - s$ attributes belong to Bob. Note that for each $1 \leq j \leq k$,

$$\begin{aligned} \sum_{d_i, \text{ where } d_i^c=j} d_{i,m} &= \sum_{i=1}^n e_{i,j} \cdot d_{i,m} \\ &= \sum_{i=1}^n e_{i,j}^A \cdot d_{i,m} + \sum_{i=1}^n e_{i,j}^B \cdot d_{i,m} \end{aligned}$$

In the first term, Alice computes the sum that involves the components $d_{i,p_1}, \dots, d_{i,p_s}$. Alice and Bob invoke the scalar product protocol to compute the random shares of the rest of the first term. They compute the random shares of the second term similarly. Let us denote the shares of Alice and Bob as s_j and t_j respectively. Given $e_{i,j}^A$ and $e_{i,j}^B$, Alice and

Bob need to compute the total number of d_i for each $1 \leq j \leq k$ such that $d_i^c = j$. Note that for each $1 \leq j \leq k$,

$$\begin{aligned} |\{d_i, \text{where } d_i^c = j\}| &= \sum_{i=1}^n e_{i,j} \\ &= \sum_{i=1}^n e_{i,j}^A + \sum_{i=1}^n e_{i,j}^B \end{aligned}$$

Alice computes the first sum and Bob computes the second. Let us denote their shares as n_j and m_j respectively. This does not involve any communication between Alice and Bob. The random shares of the cluster centers given by

$$\frac{s_j + t_j}{n_j + m_j}$$

are computed using Yao's circuit evaluation protocol [145]. Alternatively, one could use Bunn and Ostrovsky's division protocol [16], which was presented after the initial publication of our results [76].

Yao's circuit evaluation protocol uses $O(sk \log N)$ bits of communication to compute e_i for each $1 \leq i \leq n$. Computing the sums of $d_{i,m}$ requires $O(nw)$ bits of communication for each attribute $1 \leq m \leq \ell$ and for each of the k clusters. Computing the random shares of the cluster centers involves a communication of $O(k s \ell \log N)$. Hence the overall communication complexity is $O(nk \ell (w + s \log N))$, which is the same as that of Protocol 5. The overall communication of this protocol remains $O(nk \ell (w + s \log N))$ when restricted to horizontally or vertically partitioned databases. However, for horizontally partitioned databases, we present in Chapter 5 a different fully private k -clustering algorithm with the overall communication complexity of $O(k^2 \ell (w + s \log N))$.

4.6 Summary

This chapter has two main contributions. First, we introduce the idea of arbitrarily partitioned data, which is a generalization of both horizontally and vertically partitioned data. Protocols in this model can be applied to both horizontally and vertically partitioned data, as well as to data anywhere in between. Second, we provide privacy-preserving k -means clustering algorithm over arbitrarily partitioned data.

Chapter 5

Communication-Efficient Privacy-Preserving Clustering Algorithms

5.1 Introduction

Many of the privacy-preserving clustering protocols available in the literature convert existing clustering algorithms into privacy-preserving protocols. The resulting protocols may either leak intermediate information [133, 80], thereby breaching privacy or have high communication complexity [76, 16, 116]. These protocols, except [116], use k -means whereas [116] uses BIRCH. All of these either have at least linear communication (in the number of records in the database) or do not provide full privacy. One could alternately develop privacy-preserving versions of other k -clustering algorithms such as CLARANS [109] and STREAMLS [63]. However, CLARANS is an in-memory algorithm, and hence does not scale well to large databases. The STREAMLS algorithm depends on repeatedly solving a facility-location problem using a local search algorithm, and then finally applying a linear programming based clustering algorithm. Each of these subroutines (local search and linear programming) can be complicated to perform privately.

In this chapter, we describe a k -clustering algorithm that we specifically designed to be converted into a communication-efficient protocol for private clustering. The resulting protocol does not leak any information about the original data. Our algorithm is also I/O-efficient in that each data item is examined only once, and it only uses sequential access

to the data. We also present a modified form of our k -clustering algorithm that works for data streams.

5.1.1 Our Contributions

In this chapter, we present a simple deterministic algorithm called **ReCluster** for I/O-efficient k -clustering. This algorithm examines each data item only once and uses only sequential access to the data. For fixed k , the time complexity of the algorithm is linear in the number n of items in the database and the space complexity is $O(\log n)$. We present results from the application of the algorithm to synthetic data and realistic data, and compare our results with the well-known iterative k -means clustering algorithm and BIRCH. Our results show that **ReCluster**, on average, is more accurate in identifying cluster centers than the k -means clustering algorithm and compares well against BIRCH. We also experimentally show that **ReCluster** requires substantially less memory than BIRCH. We note that although there have been other clustering algorithms that improve on the k -means clustering algorithm, this is the first for which a sublinear-communication cryptographic privacy-preserving version has been demonstrated. **ReCluster** works only on numeric data. We also present a modified form of the **ReCluster** algorithm called **StreamCluster** which is intended to work on data stream inputs.

We also demonstrate that the **ReCluster** algorithm lends itself well to privacy-preserving computation. Specifically, we present a privacy-preserving version of the **ReCluster** algorithm, for two-party horizontally-partitioned databases. This protocol is communication-efficient and it reveals only the final cluster centers (or the cluster assignments of data) to both parties at the end of the protocol. This protocol does not reveal the intermediate

Algorithm 3 ReCluster

Input: Database D , Integer k

Output: Cluster centers S

1. $S' = \text{RecursiveCluster}(D, 2k)$ // Produce $2k$ clusters
 2. $S = \text{MergeCenters}(S', k)$ // Compress to k clusters
 3. Output S
-

candidate cluster centers or intermediate cluster assignments. Although an interesting clustering algorithm in its own right, ReCluster was explicitly designed to be converted into a communication-efficient privacy-preserving protocol.

We present our k -clustering algorithm in Section 5.3. In Section 5.4, we present experimental results comparing our algorithm with the k -means clustering algorithm and BIRCH. We present the privacy-preserving clustering protocol with performance analysis and comparison to other private clustering algorithms in Section 5.5.

5.2 Preliminaries

Throughout this chapter, we use n to denote the size of the database, ℓ to denote the number of attributes, and k denotes the number of clusters. Our solution also makes use of encryption in various places. We use c to denote the maximum number of bits required to represent any public key encryption and s to denote the maximum number of bits required to denote any symmetric key encryption. All computations are performed in a field \mathbb{F} of size N .

Algorithm 4 RecursiveCluster

Input: Database D , Integer k

Output: Cluster centers S

If ($|D| \leq k$) then $S = D$

Else

1. $D_1 =$ First half of D
2. $D_2 =$ Second half of D
3. $S_1 = \text{RecursiveCluster}(D_1, k)$
4. $S_2 = \text{RecursiveCluster}(D_2, k)$
5. $S = \text{MergeCenters}(S_1 \cup S_2, k)$

End If

Output S

5.3 The k -Clustering Algorithm

In this section we present our new algorithm, **ReCluster**, for k -clustering. We also present an extension of the algorithm to data streams and a distributed (non-private) version of **ReCluster**. **ReCluster** was explicitly designed to be converted into a communication-efficient privacy-preserving protocol. We experimentally show that the cluster centers produced by **ReCluster** are comparable with the ones produced by other well known algorithms such as k -means and BIRCH. Experimental results are given in Section 5.4.

5.3.1 Overview

ReCluster makes the assumption that both parties know the size of the data set before running the algorithm. Our algorithm runs in the typical “divide, conquer and combine” fashion used in algorithms such as MergeSort. Strictly speaking, this strategy would require us to divide the database into two equal halves, recursively produce k cluster centers from each of the halves, and then “merge” these $2k$ centers into the k final centers. However, we take a slightly different approach, similar to that used by Guha et al. in [64]. Instead

of generating k centers from each recursive call, we produce $2k$ cluster centers from each recursive call, and then merge the total of $4k$ centers thus received into $2k$ centers¹. (That is, the RecursiveCluster algorithm is initially invoked with the second input parameter being set to $2k$, where k is the desired number of clusters.) A final post-processing step uses the same merge technique to produce the k final centers from the $2k$ clusters returned from the top-most level of the recursion tree. The top level of ReCluster is presented in Algorithms 3 and 4. The subroutine for merging clusters is described in Algorithm 5. This latter algorithm depends on a notion of merge error which is described in Section 5.3.1. For a data set of n elements, the overall time complexity of the algorithm is $O(nk^2)$.

Error Measure.

The key step in ReCluster is the merging of $4k$ centers into $2k$ centers after the two recursive calls. For this step, we use a variation of the notion of error defined in Ward's algorithm for bottom-up hierarchical clustering [137]. Let C_1 and C_2 be two clusters being considered for a merge at some stage. Associated with each cluster center C is a *weight* (denoted $C.\text{weight}$), which is the number of objects that are assigned to that cluster. In Ward's algorithm the error of $C_1 \cup C_2$ is defined as

$$\text{error}_w(C_1 \cup C_2) = \frac{C_1.\text{weight} * C_2.\text{weight} * \text{dist}^2(C_1, C_2)}{C_1.\text{weight} + C_2.\text{weight}},$$

where $\text{dist}(C_1, C_2)$ is defined as the Euclidean distance between the center of C_1 and the center of C_2 . ReCluster defines the error of the union as

$$\text{error}_r(C_1 \cup C_2) = C_1.\text{weight} * C_2.\text{weight} * \text{dist}^2(C_1, C_2).$$

¹When the number of items in the database cannot be expressed in the form $k2^x$ for some integer x , the base case of the RecursiveCluster function may end up producing clusters with fewer than k centers. This has been experimentally seen to sometimes produce poor clusterings.

Algorithm 5 MergeCenters

Input: Cluster centers S , Integer k

Output: Cluster centers S , such that $|S| = k$

1. While ($|S| > k$)
 - (a) Compute the merge error for all pairs of centers in S .
 - (b) Remove from S the pair with the lowest merge error, and insert the center of the merged cluster, with its weight as the sum of the weights of the pair.
 - End While
 2. Output S
-

In most situations, the pair of clusters that is chosen for merging is the same whether we define error of the union as error_w or as error_r . However, there are conditions where the choices made using the two error measures are different. Consider two cluster pairs (C_p, C_q) and (C_s, C_t) such that $\text{dist}(C_p, C_q) \approx \text{dist}(C_s, C_t)$ and $C_p.\text{weight} * C_q.\text{weight} \approx C_s.\text{weight} * C_t.\text{weight}$, but $C_p.\text{weight} \gg C_q.\text{weight}$ and $C_s.\text{weight} \approx C_t.\text{weight}$. That is, the distance between the cluster centers of the first pair is the same as the distance between the cluster centers of the second pair, but cluster C_p has many more objects in it compared to C_q , while clusters C_s and C_t have about the same number of objects.

If we use error_w to decide which of the two pairs gets merged, we would merge the pair (C_p, C_q) , while error_r would favor the merger of (C_s, C_t) . See Figure 5.1. Our experiments indicate that using error_r in ReCluster makes the algorithm less susceptible to noise, because noise objects (which do not clearly belong to any cluster) resemble low-weight clusters.

Given this definition of error, the compression of a set S of $4k$ clusters into $2k$ clusters is obtained by repeatedly:

1. choosing a pair of clusters C_i and C_j such that the error of their union is minimum among all such pairs,

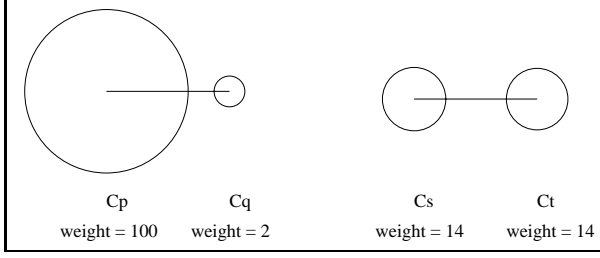


Figure 5.1: The effect of choosing error_r over error_w . ReCluster prefers to merge C_s and C_t , Ward's algorithm would prefer to merge C_p and C_q .

2. deleting from S the clusters C_i and C_j , and
3. inserting into S the new cluster $C_i \cup C_j$.

The weight of this new cluster is $C_i.\text{weight} + C_j.\text{weight}$. See Algorithm 5.

5.3.2 Extension to Data Streams

In this section we modify the ReCluster algorithm to work on data streams. The main idea is to eliminate recursion since the number of data items to be clustered is not known ahead of time.

The algorithm (see Algorithm 6) can be said to be “incrementally agglomerative.” That is, the algorithm merges intermediate clusters without waiting for all the data to be available. At the beginning of each iteration, the algorithm holds a number of intermediate k -clusterings. We associate with each such k -clustering a *level*, a concept similar to the one in [63]. When two k -clusterings at level i are “merged” to form a new k -clustering, this new clustering is said to be at level $(i + 1)$. We also associate with each cluster a *weight*, which is the number of data points in that cluster. Each cluster is represented by its center, which is the mean of all points in it.

In each iteration, the algorithm adds the next k data points as a level-0 intermediate k -clustering. If the algorithm detects two k -clusterings at level i in the list, these are “merged”

Algorithm 6 StreamCluster

Input: Database D , Integer k

Output: Cluster centers S

Repeat as long as data remains:

1. Read the next k items from D as a level-0 k -clustering.
 2. Add the level-0 clustering to the vector V of intermediate clusterings.
 3. While $V[\text{last}].\text{level} = V[\text{last} - 1].\text{level}$
 - (a) $V[\text{last} - 1] = \text{MergeCenters}(V[\text{last} - 1] \cup V[\text{last}], k)$
 - (b) $V[\text{last} - 1].\text{level} = V[\text{last} - 1].\text{level} + 1$
 - (c) Remove $V[\text{last}]$ from V .
 4. If an output is needed, union all intermediate clusterings in V and return output from MergeCenters .
-

(using the MergeCenters routine, see Algorithm 5) into a k -clustering at level $(i + 1)$.

When a k -clustering needs to be output after some points have been read from the data stream, all intermediate k -clusterings at all levels are “merged” using MergeCenters into a single k -clustering.

For a data stream of n elements, the overall time complexity of the algorithm is $O(nk^2)$.

The amount of memory required to store the $O(\log \frac{n}{k})$ intermediate k -clusterings is $O(k \log \frac{n}{k})$.

5.3.3 Distributed k -Clustering Protocol for Two Parties

We next present the distributed version of our protocol, which forms the basis for the privacy-preserving clustering protocol to be presented in Section 5.5. We assume that Alice holds the data set D_1 and Bob holds D_2 . The protocol runs as follows:

1. Alice uses the ReCluster algorithm on the data set D_1 to compute $2k$ intermediate cluster centers.
2. Bob computes $2k$ cluster centers using his data set D_2 .

3. Alice sends her intermediate cluster centers to Bob.
4. Bob takes the union of all the intermediate clusters and uses the `MergeCenters` routine to obtain the k final cluster centers.

5.4 Experimental Results

In this section we present our experimental results that show the ability of `ReCluster` and its extensions to accurately identify clusters, as measured by the error sum of squares. We ran our experiments on a large number of synthetic and realistic data sets. All algorithms were implemented in C++ and experiments were run on a Windows-based personal computer with an AMD 3500+ CPU and 3 GB of main memory. We first describe in Sections 5.4.1 and 5.4.2 the datasets that we use. Results are given in Section 5.4.3 for the basic algorithm and in Section 5.4.4 for the distributed algorithm.

5.4.1 Synthetic Data Generator

To test `ReCluster`'s ability to accurately identify clusters, we generated three types of synthetic data sets, namely Random Centers, Grid data sets, and Offset Grid data sets. We describe these data sets below. These data sets are similar to the ones used in [146, 65, 66, 105]. Each data set consists of points distributed in 2D space. Our synthetic data set generator takes a number of parameters:

- The number k of clusters.
- The range $[n_l, n_u]$ of the desired number of points in each cluster.
- The parameters s_w and s_h which denote the width and the height, respectively, of the space in which data points are generated.

Parameter	Value
Number of clusters, k	5
Min. number of points per cluster, n_l	900
Max. number of points per cluster, n_u	1100
Min. major radius of cluster, r_{al}	30
Max. major radius of cluster, r_{au}	50
Min. minor radius of cluster, r_{bl}	30
Max. minor radius of cluster, r_{bu}	50
Max. X-axis value, s_w	500
Max. Y-axis value, s_h	500
Add noise	y

Table 5.1: Sample parameters for our synthetic data set generator while generating a Random Centers data set

In one collection of data sets that we call Random Centers, the clusters are chosen as ellipses with major radius r_a chosen in the defined range $[r_{al}, r_{au}]$ and minor radius r_b chosen in the defined range $[r_{bl}, r_{bu}]$. (When $r_a = r_b$ we get circular clusters.) The cluster centers are positioned randomly in the space within which points are generated. In the Grid data sets and in the Offset Grid data sets, circular clusters are chosen with radius r , with all clusters having the same radius. The cluster centers are placed on a $\sqrt{k} \times \sqrt{k}$ grid. In the case of the offset grid, the cluster centers are then randomly perturbed with small values in the x and y directions. After the center and radius of each cluster is finalized, the synthetic data generator then generates $n_c \in [n_l, n_u]$ points at random within the cluster. In all cases, in addition to the points generated by the procedure above, we add 5% noise in the form of data points uniformly distributed throughout the data set in order to explore how the algorithm performs with noisy data. Table 5.1 illustrates a set of sample parameters for these data sets. For each data set we ran the k -means algorithm 10 times, each run with a different randomly chosen set of initial cluster centers. We compared the performance of ReCluster against the average performance of the k -means algorithm.

Each synthetic data set used either uniform distributions over some intervals or normal distributions for generating clusters. Our figures do not include any of the normally distributed data since the results were essentially the same as for uniformly distributions. Our experiments show that data ordering has a low effect on the output of our algorithm. This is possibly because the `MergeCenters` routine acts as a global optimizer.

5.4.2 Realistic Data

In order to test the ability of `ReCluster` to accurately identify clusters on large realistic data sets, we chose the network intrusion data set used in the 1999 KDD Cup [85]. We compared the error sum of squares (ESS) of `ReCluster` against `BIRCH`. This data set contains information about five million network connections. The vast majority of these data points are about benign network connections. A small fraction of these connections represent four different kinds of network intrusion attacks. We selected the 10% subset to run our experiments. Each connection record is described in terms of 41 attributes, of which 34 are continuous. We projected out the 7 symbolic attributes for our experiments. Also, we eliminated one outlier record. We set k to 5 in our runs of `ReCluster` and `BIRCH`.

5.4.3 Results for `ReCluster`

We present below the results of our experiments on synthetic data sets and on the network intrusion data set. In order to show the ability of `ReCluster` to accurately identify clusters we compute the error sum of squares for the centers detected by `ReCluster`. For some of the data sets we also visually present the clusters and the detected centers as a demonstration of the clustering algorithm.

In Figure 5.2, we present the results from a typical experiment on a Random Centers data

True centers $\times 10^7$	Recluster centers $\times 10^7$	Average for k -means centers $\times 10^7$
5.52	5.71	7.25
8.14	8.87	11.99
8.77	9.31	14.1
5.68	5.71	7.42
6.13	6.34	7.68
6.59	6.85	8.88
7.48	7.52	13.35
10.04	10.69	14.84
5.65	5.64	11.91
6.55	7.02	8.60

Table 5.2: The error sum of squares for a subset of the Random Centers data sets. In each case the number of clusters was 5, with approximately 10,000 points in each cluster. Noise was added at the rate of 5%.

set, which is a synthetically generated data set with randomly positioned cluster centers. We created 50 such data sets. In 25 of the 50 cases, we distributed data points using a uniform distribution within each cluster, and the remaining data sets used Gaussian distributions. Figure 5.2 shows a data set in which the points were distributed using the uniform distribution. ReCluster does very well in identifying cluster centers, even in the presence of noise. However, in four out of these 25 data sets ReCluster placed the center of a cluster outside the cluster, albeit close to the cluster itself. See Figure 5.3 for such a data set. Also shown in Figure 5.3 are the centers identified in the best run of k -means algorithm. On this particular data set the k -means algorithm failed to identify all cluster centers on four of the 10 runs. In Table 5.2 we present the error sum of squares for ReCluster and k -means on 10 of these Random Centers data sets.

The average running time for ReCluster over all the uniform distribution data sets, which had about 51,000 points on average, was 1.06 seconds, and the average running time for the k -means algorithm was 8.09 seconds.

In Figure 5.4, we present the results from experiments on a Grid data set, which is

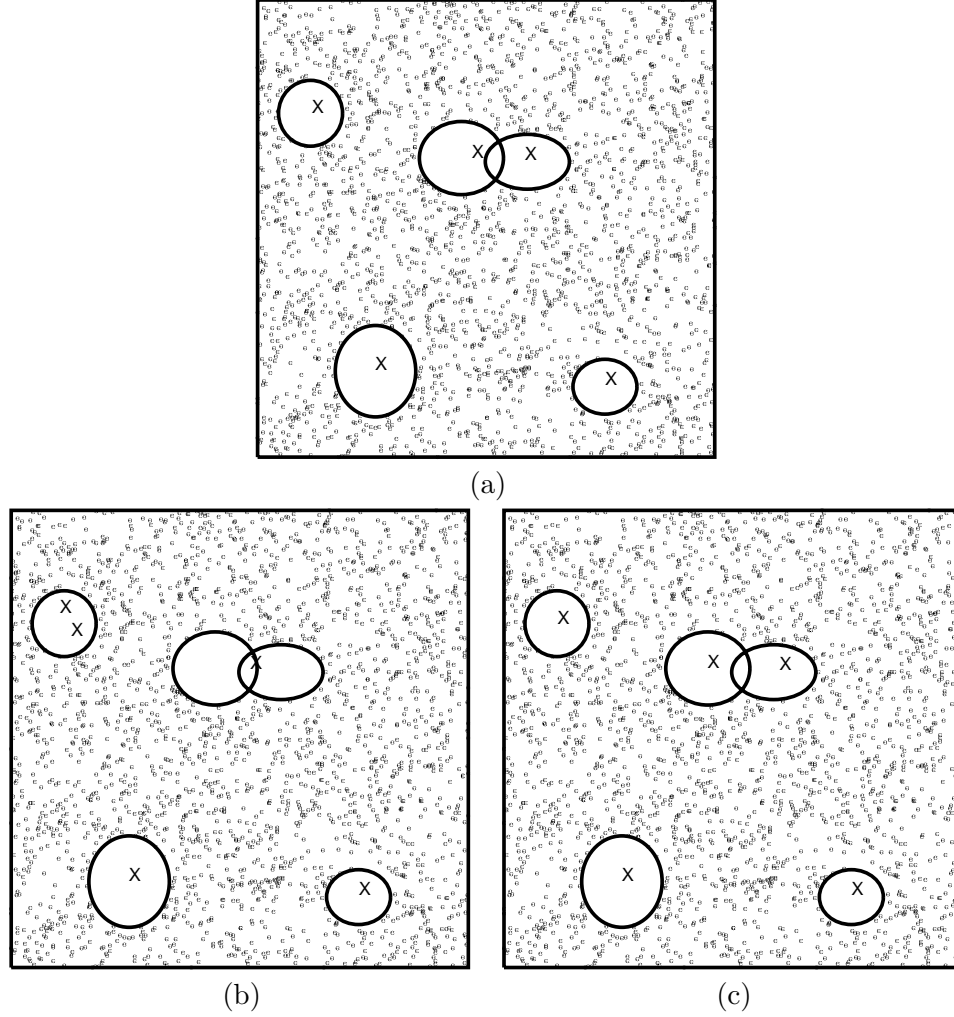


Figure 5.2: This synthetic data set consists of about 10,000 points concentrated around each randomly chosen cluster center plus 5% “noise” points occurring elsewhere. The points concentrated at each chosen cluster center are illustrated by a large open ellipse around each center. The noise points are shown as points. Cluster centers shown are those identified by (a) ReCluster ($\text{ESS} = 5.71 \times 10^7$) (b) the worst run of k -means algorithm ($\text{ESS} = 8.49 \times 10^7$) and (c) the best run of k -means algorithm ($\text{ESS} = 5.48 \times 10^7$) are denoted by x.

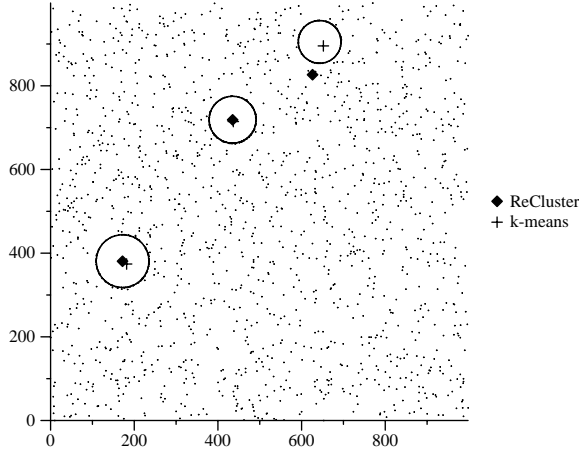


Figure 5.3: An example data set in which ReCluster did not accurately identify all cluster centers. The data set consists of about 10,000 points concentrated around each randomly chosen cluster center plus 5% “noise” points occurring elsewhere. The points concentrated at each cluster center are illustrated by a large open ellipse around each center. The noise points are shown as points. Also shown are the centers identified by the best run of the k -means algorithm denoted as.

a synthetically generated data set with cluster centers positioned on a grid. We ran our experiments on two such data sets and Figure 5.4 represents one of them. As can be seen, ReCluster accurately identifies the centers of all clusters. On the other hand, for this data set not even the best run of the k -means algorithm identified all 25 cluster centers. The worst run of k -means did not identify even half of the cluster centers. In Figure 5.5 we present the results on an Offset Grid data set. ReCluster was quite successful in identifying all cluster centers, which k -means did not do even on its best run.

In Table 5.3 we present the result of our experiment on the Network Intrusion data. As our results indicate, ReCluster does extremely well relative to BIRCH in this data set. BIRCH needs more than 32 times as much memory as used by ReCluster, to find five clusters. Even when given 128 times as much memory as ReCluster, BIRCH does not perform nearly as well as. Perhaps with much more memory BIRCH could perform better.

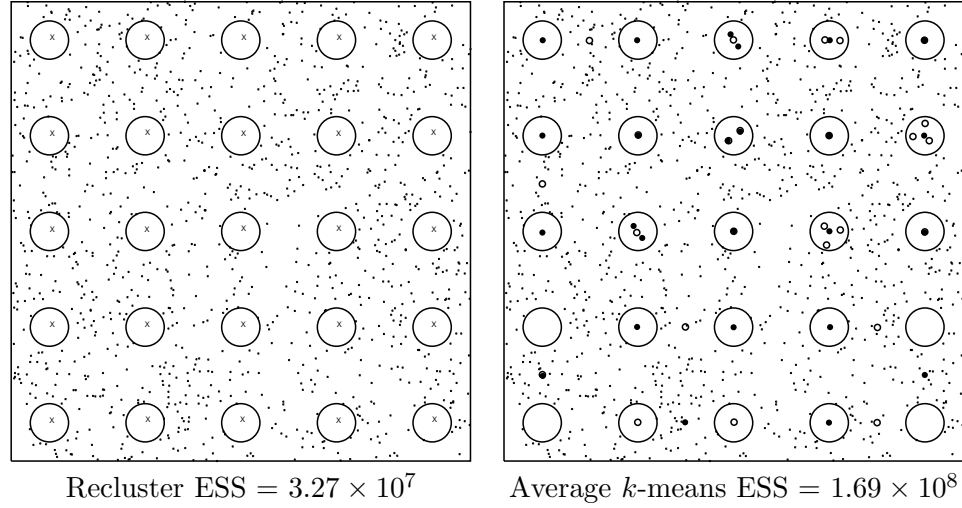


Figure 5.4: Cluster centers identified in a grid data set. The data set consists of about 1000 points concentrated around each grid point plus 5% “noise” points occurring elsewhere. The points concentrated at each grid point are illustrated by a large open circle around each grid point. The noise points are shown as points. On the left are cluster centers identified by ReCluster. On the right are the best (identified by small filled circles) and worst (identified by small open circles) runs of the k -means algorithm.

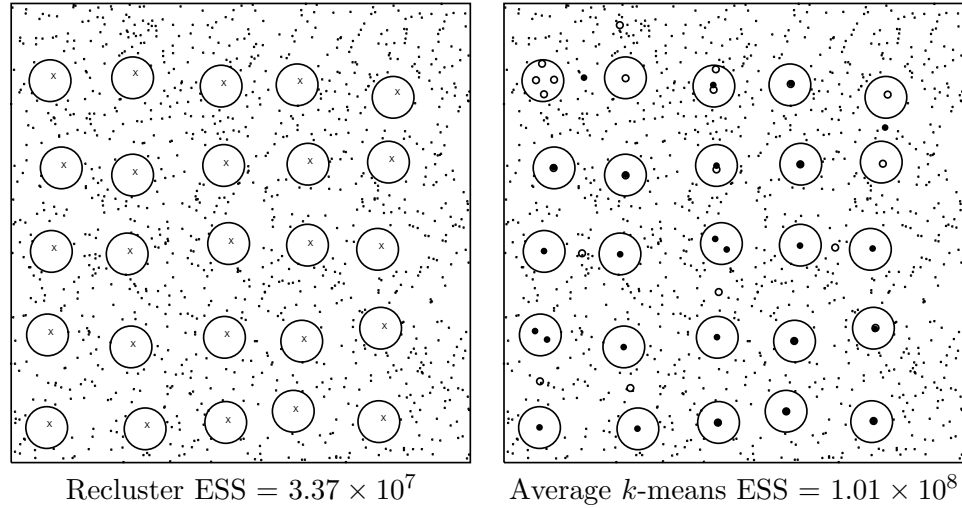


Figure 5.5: Cluster centers identified in an offset grid data set. The data set consists of about 1000 points concentrated around each grid point plus 5% “noise” points occurring elsewhere. The points concentrated at each grid point are illustrated by a large open circle around each grid point. The noise points are shown as points. On the left are cluster centers identified by ReCluster. On the right are the best (identified by small filled circles) and worst (identified by small open circles) runs of the k -means algorithm.

Algorithm	Mem. Allowed ($\times 24000$ bytes)	ESS
ReCluster	1	4.11259E14
BIRCH	1	*
BIRCH	2	*
BIRCH	4	*
BIRCH	32	*
BIRCH	64	4.83018E17
BIRCH	128	4.8299E17

Table 5.3: Comparison of the error sum of squares for the network intrusion 10% data set. An asterisk indicates that BIRCH failed to find five clusters due to insufficient memory.

5.4.4 Results for Distributed ReCluster

The synthetic data set generator can also generate data sets for the two-party distributed k -clustering problem. In addition to the parameters mentioned in Section 5.4.1, the user can input how the clusters are to be apportioned between the two parties, and how much data about one party’s clusters are also known to the other party.

Although similar in its essentials to ReCluster, the distributed clustering protocol in Section 5.3.3 is not identical to ReCluster. The ReCluster algorithm splits the data set into two equal halves, then recursively clusters each half, following which it merges the clusters from both halves. In contrast, the distributed algorithm first operates on each parties’ data independently, regardless of the relative sizes of the data sets hold by the two parties. We make no assumptions about the relative sizes of the data sets held by the two parties, and one party could have far more data than the other. Given this distinction, it is not immediately obvious that the distributed clustering algorithm would have properties similar to that of ReCluster. To test the effectiveness of this distributed algorithm, we ran experiments where the parties had different proportions of the total data. In one set of experiments, the first party had twice as many data points per cluster as the second. In the second set, the first

party had five times as many data points as the second. Within each set, we also varied the “separation” between the two parties so that in some cases, the two parties had knowledge of no clusters in common. That is, some clusters were entirely “owned” by the first party and the others were entirely “owned” by the second. In other cases, each party had some fraction of the data points that belong to clusters that were primarily owned by the other party. In all cases, we compared the results to the k -means clustering algorithm on the combined data. Some typical runs are presented in Figures 5.6 and 5.7. As can be seen, the distributed clustering protocol has performance close to that of k -means.

5.5 Privacy-Preserving k -Clustering Protocol

We now describe our privacy-preserving protocol for k -clustering based on the distributed ReCluster algorithm presented in Section 5.3. Two parties Alice and Bob who own databases $D_1 = \{d_1, \dots, d_m\}$ and $D_2 = \{d_{m+1}, \dots, d_n\}$, respectively, wish to jointly compute a k -clustering of $D_1 \cup D_2$. At the end of the protocol both parties learn the final k cluster centers. The parties learn nothing else. We assume that Alice and Bob are semi-honest.

5.5.1 Overview of the Private Clustering Protocol

We now describe the private protocol. Alice and Bob separately use the ReCluster algorithm to compute $2k$ clusters each from their own shares of the data. Next, they randomly share their cluster centers with each other using the `permute_share` protocol described in detail in Section 5.5.2. At this point, Alice and Bob have random shares of $4k$ cluster centers. They again use the `permute_share` protocol to prevent each party keeping track of the cluster centers across iterations.

The two parties use the secure `merge_clusters` protocol, described in detail in Section

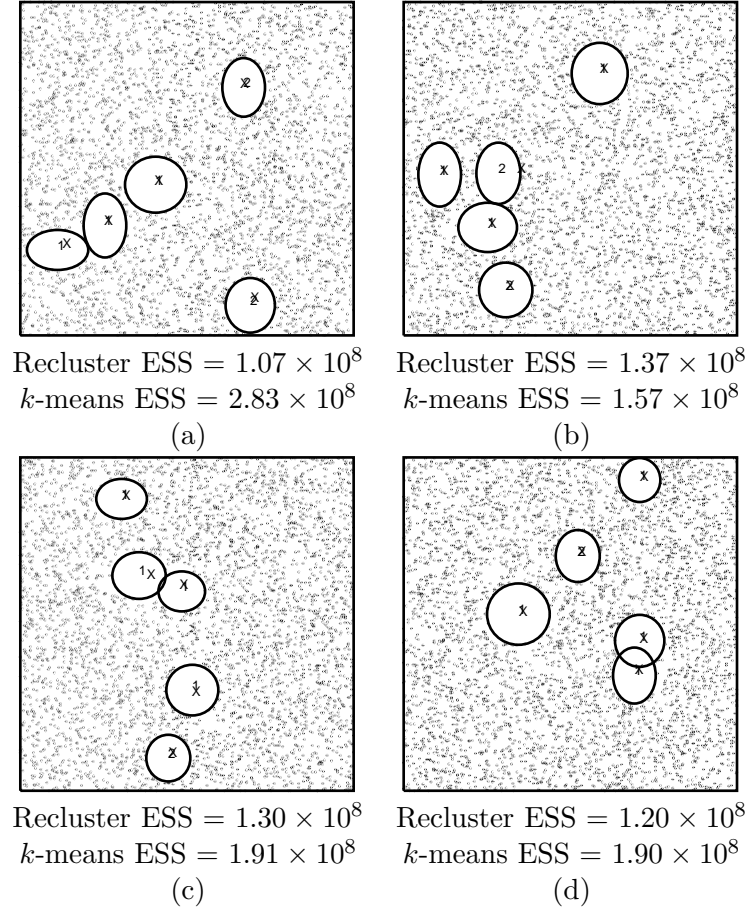


Figure 5.6: Some typical runs of the distributed clustering algorithm. The data sets consists of about 20,000 points concentrated around each randomly chosen cluster center owned by the first party (denoted by 1), 10,000 points concentrated around each randomly chosen cluster center owned by the second party (denoted by 2), plus 5% “noise” points occurring elsewhere. The points concentrated at each cluster center are illustrated by a large open ellipse around each center. The noise points are shown as points. Figures (a) and (b) are example data sets in which the first party “owns” the data for three of the clusters and the second party has data for the two remaining clusters. Figures (c) and (d) are example data sets in which the first party has data for four of the clusters and the second party has data for only one cluster. In Figures (a) and (c) each party has 25% of the data from the other party’s clusters. In Figures (c) and (d) each party has no information regarding the clusters of the other party. In all cases the letter X marks the centers detected by the distributed clustering protocol. Also shown are the error sum of squares value for distributed ReCluster and the average error sum of squares for k -means.

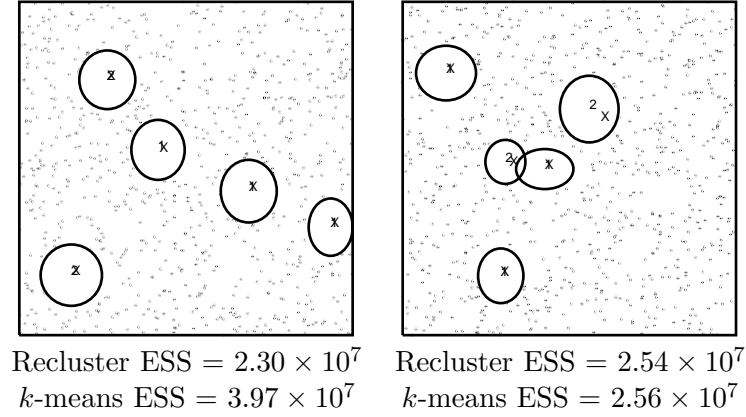


Figure 5.7: Typical runs of the distributed clustering algorithm on more “skewed” data. Clusters owned by the first party have five times as many points as clusters owned by the second party (5000 points on average versus 1000 points on average). The first party’s cluster centers are denoted as 1 and the second party’s cluster centers are denoted as 2. Each party has no data about the clusters owned by the other party. In all cases the letter \mathbf{x} marks the centers detected by the distributed ReCluster protocol. Also shown are the error sum of squares value for distributed ReCluster and the average error sum of squares for k -means.

5.5.3, to iteratively merge the $4k$ clusters into k clusters. If the input to the `merge_clusters` protocol is j cluster centers, then at the end of the protocol, Alice and Bob obtain as output a random sharing of $j - 1$ cluster centers. At the end of $3k$ joint executions of the `merge_clusters` protocol, they obtain a random sharing of k cluster centers. The parties may send each other their cluster center shares so that they obtain the actual cluster centers of the joint data.² This privacy-preserving ReCluster protocol does not leak any intermediate information.

Protocol 6 illustrates the overall privacy-preserving ReCluster protocol. In the rest of this section, we describe in detail the subprotocols used. In Section 5.5.2, we describe the protocol that securely permutes and shares data between two parties. In Section 5.5.3, we present a secure protocol that merges clusters.

²If they wish to learn the assignment of the cluster numbers to the objects they own, they can securely compute the cluster numbers using the shared cluster centers without revealing the cluster centers.

Protocol 6 Privacy-preserving k -clustering protocol

Protocol Private_K_Clustering

Input: Database D consisting of n objects, where Alice owns $D_1 = (d_1, \dots, d_m)$ and Bob owns $D_2 = (d_{m+1}, \dots, d_n)$, and an integer k denoting the number of clusters.

Output: Assignment of cluster numbers to objects

1. Alice computes using the ReCluster algorithm $2k$ cluster centers from her data objects $\{d_1, \dots, d_m\}$ and Bob computes using the ReCluster algorithm $2k$ cluster centers from his data objects $\{d_{m+1}, \dots, d_n\}$.
2. Alice and Bob jointly share the cluster centers computed in Step 1 with each other.
 - (a) Let $(c_1, w_1), \dots, (c_{2k}, w_{2k})$ denote Alice's cluster centers. Here c_i denote the i th center and w_i denotes its corresponding weight. Similarly let $(c_{2k+1}, w_{2k+1}), \dots, (c_{4k}, w_{4k})$ denote Bob's cluster centers.
 - (b) Bob chooses a random permutation ϕ_1 and random values $r_1, \dots, r_{2k}, s_1, \dots, s_{2k}$. They engage in a secure `permute_share` protocol and obtain random shares of Alice's $2k$ cluster centers.
 - (c) Alice chooses a random permutation ϕ_2 and random values $p_1, \dots, p_{2k}, q_1, \dots, q_{2k}$. They engage in a secure `permute_share` protocol and obtain random shares of Bob's $2k$ cluster centers.
 - (d) Now Alice has $(c_1^A, w_1^A), \dots, (c_{4k}^A, w_{4k}^A)$ and Bob has $(c_1^B, w_1^B), \dots, (c_{4k}^B, w_{4k}^B)$. Bob chooses a random permutation ϕ_3 and random values $\alpha_1, \dots, \alpha_{4k}, \beta_1, \dots, \beta_{4k}$. They engage in a secure `permute_share` protocol and obtain random shares of Alice's data.
 - (e) Bob adds the shares he obtains from Step 2d to his data. Alice chooses a random permutation ϕ_4 and random values $\gamma_1, \dots, \gamma_{4k}, \delta_1, \dots, \delta_{4k}$. They engage in a secure `permute_share` protocol and obtain random shares of Bob's data. Alice adds the shares she obtains from Step 2e to her data.

At the end of Step 2, denote Alice's share of $4k$ cluster centers by $(c_1^A, w_1^A), \dots, (c_{4k}^A, w_{4k}^A)$ and Bob's share by $(c_1^B, w_1^B), \dots, (c_{4k}^B, w_{4k}^B)$.

3. Repeat the protocol `merge_clusters` $3k$ times to merge $4k$ clusters into k clusters.
-

5.5.2 Secure Protocol to Share Data

When Alice has a vector of clusters C of the form $((c_1, w_1), \dots, (c_k, w_k))$, this protocol helps Bob to obtain a permuted random share of the vector C . (Here each c_i is a cluster center, and w_i is its weight.) At the beginning of the protocol, Alice and Bob agree on a homomorphic encryption scheme. Bob chooses a random permutation ϕ and a random vector $R = ((r_1, s_1), \dots, (r_k, s_k))$. At the end of the protocol Bob holds $\phi(R)$ and Alice receives $\phi(C + R)$. This protocol is similar to the permutation protocol introduced by Du and Attalah [38]. We briefly describe the protocol in Protocol 7.

COMMUNICATION AND COMPUTATIONAL COMPLEXITY

Alice and Bob send k encrypted pairs to each other. If we assume that it takes c bits to represent each encryption then the total communication complexity is $O(kc)$. The estimation of the computational complexity involves the counting the number of encryptions, decryptions and exponentiations. Alice performs $O(k)$ encryptions and $O(k)$ decryptions, while Bob performs $O(k)$ encryptions. The number of exponentiations required by both Alice and Bob is $O(k)$.

PRIVACY

If Alice and Bob were to use a trusted third party, then at the end of the protocol, Alice learns $\phi(C + R)$ and Bob learns $\phi(R)$. In our privacy-preserving protocol, Alice and Bob communicate using a semantically secure encryption scheme. Thus, Alice learns only her output and nothing else.

Protocol 7 Secure protocol to share data

 Protocol permute_share

Input: Alice has a vector of cluster centers C of the form $((c_1, w_1), \dots, (c_k, w_k))$,
 Bob has a random permutation ϕ and a random vector
 $R = ((r_1, s_1), \dots, (r_k, s_k))$.

Output: Alice obtains $\phi(C + R)$ as output.

1. Alice chooses a key pair (pk, sk) and sends the public key pk to Bob.
 2. Alice computes the encryption of the vector C as $((E(c_1), E(w_1)), \dots, (E(c_k), E(w_k)))$ and sends to Bob.
 3. Bob uses the property of the homomorphic encryption scheme to compute $\phi((E(c_1 + r_1), E(w_1 + s_1)), \dots, (E(c_k + r_k), E(w_k + s_k)))$ and sends to Alice.
 4. Alice decrypts to obtain her share $\phi((c_1 + r_1, w_1 + s_1), \dots, (c_k + r_k, w_k + s_k))$ and Bob's share is $\phi((-r_1, -s_1), \dots, (-r_k, -s_k))$.
-

5.5.3 Secure Protocol to Merge Clusters.

We now describe a protocol that securely merges m clusters into $m - 1$ clusters where the cluster centers are shared between Alice and Bob. Let $\{(c_1^A, w_1^A), \dots, (c_m^A, w_m^A)\}$ denote Alice's share of the cluster centers and $\{(c_1^B, w_1^B), \dots, (c_m^B, w_m^B)\}$ denote Bob's share of the cluster centers. Here c 's denote the cluster centers and w 's denote the weight of the corresponding clusters, where $c_i^A = (a_{i1}^A, \dots, a_{i\ell}^A)$, $c_i^B = (a_{i1}^B, \dots, a_{i\ell}^B)$ for $1 \leq i \leq m$ and ℓ denotes the number of attributes.

Alice and Bob jointly compute the merge error for all pairs of clusters. For two clusters C_i and C_j , for $1 \leq i < j \leq m$, the merge error is given by

$$\begin{aligned}
 \text{error}(C_i \cup C_j) &= C_i.\text{weight} * C_j.\text{weight} * (\text{dist}(C_i, C_j))^2 \\
 &= (w_i^A + w_i^B) * (w_j^A + w_j^B) * (\text{dist}(C_i, C_j))^2 \\
 &= (w_i^A * w_j^A + w_i^B * w_j^B + (w_i^B * w_j^A + w_i^A * w_j^B)) + (\text{dist}(C_i, C_j))^2
 \end{aligned}$$

where

$$(\text{dist}(C_i, C_j))^2 =$$

$$\begin{aligned}
& (\text{dist}((a_{i1}^A + a_{i1}^B, \dots, a_{i\ell}^A + a_{i\ell}^B), (a_{j1}^A + a_{j1}^B, \dots, a_{j\ell}^A + a_{j\ell}^B)))^2 \\
&= \sum_{k=1}^{\ell} ((a_{ik}^A + a_{ik}^B) - (a_{jk}^A + a_{jk}^B))^2 \\
&= \sum_{k=1}^{\ell} (a_{ik}^A - a_{jk}^A)^2 + \sum_{k=1}^{\ell} (a_{ik}^B - a_{jk}^B)^2 + 2 \sum_{k=1}^{\ell} (a_{ik}^A - a_{jk}^A)(a_{ik}^B - a_{jk}^B)
\end{aligned}$$

The first term of the distance function is computed by Alice and the second term is computed by Bob. Alice and Bob use a secure scalar product protocol to compute the random shares of the third term. Similarly the scalar product protocol can be used to compute the random shares of $(w_i^B * w_j^A + w_i^A * w_j^B)$. We have

$$\text{error}(C_i \cup C_j) = e_{ij}^A + e_{ij}^B.$$

where e_{ij}^A is the random share of the error known to Alice and e_{ij}^B is the random share of the error known to Bob, and

$$\begin{aligned}
e_{ij}^A &= (w_i^A * w_j^A) + \alpha_{ij} + \sum_{k=1}^{\ell} (a_{ik}^A - a_{jk}^A)^2 + \gamma_{ij} \\
e_{ij}^B &= (w_i^B * w_j^B) + \beta_{ij} + \sum_{k=1}^{\ell} (a_{ik}^B - a_{jk}^B)^2 + \delta_{ij} \\
\alpha_{ij} + \beta_{ij} &= (w_i^B * w_j^A + w_i^A * w_j^B) \\
\gamma_{ij} + \delta_{ij} &= 2 \sum_{k=1}^{\ell} (a_{ik}^A - a_{jk}^A)(a_{ik}^B - a_{jk}^B).
\end{aligned}$$

All of these computations are done in a finite field of size N .

Alice and Bob have $\left(\frac{m(m-1)}{2}\right)$ -length vectors (e_{ij}^A) and (e_{ij}^B) , respectively, where $1 \leq i < j \leq m$, with m being the number of clusters. They securely compute the indices i and j such that $(e_{ij}^A) + (e_{ij}^B)$ is minimum using Yao's circuit evaluation protocol [145]. Both Alice and Bob learns the indices i and j . Since m^2 is typically quite small, particularly in comparison to the number of data items, the overhead this requires should be fairly small.

The next step is to eliminate the i th and the j th cluster centers and jointly compute the new cluster center and the corresponding weight as random shares. The p th coordinate of the new cluster center is given by

$$\begin{aligned} & \frac{w_i * a_{ip} + w_j * a_{jp}}{w_i + w_j} \\ = & \frac{(w_i^A + w_i^B) * (a_{ip}^A + a_{ip}^B) + (w_j^A + w_j^B) * (a_{jp}^A + a_{jp}^B)}{w_i^A + w_i^B} \end{aligned}$$

This can be computed as random shares using a secure scalar protocol [60] and using Yao's circuit evaluation protocol [145]. Alternatively, instead of using Yao's circuit evaluation protocol one could use Bunn and Ostrovsky's division protocol [16].

COMMUNICATION AND COMPUTATIONAL COMPLEXITY

For each pair of clusters, this protocol computes the merge error, finds the indices for which the error is minimum and recomputes the new cluster center and weight for the merged pair of clusters. Each object has ℓ components. Communication is involved when the two parties engage in the secure scalar product protocol to compute the merge error of each pair of clusters. The scalar product protocol is invoked once between two vectors of length ℓ . The communication complexity for one invocation of the scalar product protocol is $O(c\ell)$. Hence, $O(m^2)$ pairs of error computations involve a communication complexity of $O(m^2c\ell)$, where c is the maximum number of bits required to represent each encryption.

Yao's circuit evaluation protocol [145] is invoked once per comparison and it takes $O(s \log N)$ bits of communication. The total communication for finding the minimum of m^2 elements is $O(m^2s \log N)$. Recomputing the cluster centers and the corresponding weights for the merged clusters involve constant bits of communication. The total communication complexity for one invocation of secure `merge_clusters` protocol is $O(m^2s \log N) + O(m^2c\ell) = O(m^2(s \log N + c\ell))$.

For the `merge_clusters` protocol, a secure scalar product protocol between vectors of length ℓ is executed for each pair of clusters. For each execution of the scalar product, Alice performs ℓ encryptions and one decryption and Bob performs one encryption and ℓ exponentiations. Yao's protocol requires $O(m^2 \log N)$ invocations of OT_1^2 . So, the overall computational complexity for an invocation of the `merge_clusters` protocol is $O(m^2 \ell)$ encryptions and $O(m^2)$ decryptions for Alice, and $O(m^2 \ell)$ exponentiations and $O(m^2)$ encryptions for Bob.

PRIVACY

The `merge_clusters` protocol securely merges m clusters into $m - 1$ clusters without revealing any information. At the beginning of the protocol, the initial cluster centers and the weights are randomly shared between Alice and Bob. The merge error of the cluster pairs are calculated as random shares between Alice and Bob using the scalar product protocol. The scalar product protocol leaks no information and returns only random shares. Yao's circuit evaluation protocol securely computes the cluster pairs that gives a minimum error. The output of the protocol is a random share of the merged $m - 1$ cluster centers.

5.5.4 Overall Privacy and Performance of the Private ReCluster Protocol

PRIVACY

For privacy requirements, we compare this protocol with the one that uses the trusted third party (TTP). When a TTP is used the two parties send the data objects owned by them to the TTP. The TTP computes the cluster centers and send them to both parties. Both parties do not receive any other additional information other than the k cluster centers.

In our privacy-preserving clustering protocol, both parties compute $2k$ clusters each

independently from the data objects owned by them. They communicate with each other when they merge the $4k$ clusters into k clusters. The `merge_clusters` protocol does the merging using secure `permute_share` protocol, the scalar product protocol, and the Yao's protocol. These protocols are secure. They do not leak any information. The cluster centers at the end of the `merge_clusters` protocol are obtained as random shares between the two parties. When the parties need to compute the cluster centers they exchange their shares to each other. All the intermediate results are also available as random shares between the two parties. Neither party can learn any information from the encrypted messages that are communicated since the encryption scheme chosen is semantically secure. Hence the privacy-preserving ReCluster protocol is secure and does not leak any information.

COMMUNICATION AND COMPUTATIONAL COMPLEXITY

Alice and Bob initially compute $2k$ clusters each from their own data. They invoke the `permute_share` protocol four times to obtain a share of each others data. This requires a communication of $O(kc)$ bits where c is the maximum number of bits required to represent each encryption. Alice and Bob jointly invoke the `merge_clusters` protocol $3k$ times to compute k cluster centers from $4k$ cluster centers. Each invocation of the `merge_clusters` protocol involves a communication of $O(k^2c(d+\ell))$ bits, where ℓ is the number of attributes in each row of the table, and d is the maximum number of bits required to represent a database entry. Hence the overall communication complexity is $O(k^3(s \log N + c\ell))$ bits.

In Step 1 of Protocol 6, Alice and Bob independently compute $2k$ clusters from their own data. This involves a time complexity of $O(nk^2)$. The sharing of the data between Alice and Bob in Step 2 requires $O(k)$ encryptions and $O(k)$ decryptions for Alice, and $O(k)$ encryptions and $O(k)$ decryptions for Bob. They also perform $O(k)$ exponentiations.

The `merge_clusters` protocol is invoked $3k$ times. The computational complexity is $O(k^3\ell)$ encryptions, $O(k^3)$ decryptions for Alice, $O(k^3\ell)$ exponentiations and $O(k^3)$ encryptions for Bob, and $O(k^3 \log N)$ invocations of OT_1^2 .

We next compare the performance of our new privacy-preserving protocol with earlier protocols for k -clustering [133, 80, 16, 116], as well as our own protocol presented in Chapter 4. All of these results except [116] were based on the distributed k -means clustering algorithm. The k -means algorithm computes candidate cluster centers in an iterative fashion. The protocols presented in [133, 80] reveal the candidate cluster to which each object has been assigned. As explained in Chapter 4, this intermediate information can sometimes reveal some of the parties' original data, thus breaching privacy. This leak can be prevented by having the cluster assignments shared between the two parties. Bunn and Ostrovsky give a secure protocol that does not leak this information [16]. The protocol presented in Section 4.5 also does not leak any information. However, these protocols have a large communication complexity of $O((n+k)t)$ for horizontally partitioned data, where t is the number of iterations of the k -means algorithm. In the expected setting, where n is very large, this has a substantial impact. Prasad and Rangan [116] presented a privacy-preserving protocol for BIRCH on arbitrarily-partitioned databases. This algorithm, when reduced to the case of horizontally-partitioned databases, results in a communication complexity of $O(n)$, where n is the size of the database.

Our protocol suffers from neither of these drawbacks. Specifically, the communication complexity does not depend on n . Although, the complexity is cubic in k , for large n and small k , our protocol is not significantly slower than the k -means protocol for horizontally partitioned data [80]. Further, it does not leak any intermediate information. We note that hierarchical agglomerative clustering algorithms, such as Ward's, can be made private using

the same techniques used in this chapter. However, the resulting private protocol would have a communication complexity of $\Theta(n^3)$.

5.6 Summary

This chapter makes two major contributions. First, we present a simple deterministic algorithm for I/O-efficient k -clustering. This algorithm examines each data item only once and only uses sequential access to data. We present extensions of the algorithm to data streams. Second, we present a privacy-preserving protocol for k -clustering based on our clustering algorithm. This protocol is communication efficient and does not reveal the intermediate candidate centers or cluster assignments.

Part II

Output Privacy in Data Mining

Chapter 6

Private Inference Control For Aggregate Database Queries

6.1 Introduction

On-line analytical processing systems, commonly known as OLAP systems, store historical data or data related to multiple organizations and are mainly used in knowledge discovery. Aggregate queries play a major role in OLAP systems. OLAP systems are essentially statistical databases that emphasize business applications [129]. Whenever a database is used to discover knowledge, sensitive information about individuals must be protected. In a system that permits aggregate queries to authorized users, no single query may reveal sensitive data. However, an authorized user might invoke a sequence of queries, each of which is under his privileges, but whose results can be combined to infer some additional sensitive information about the data.

Another privacy issue that arises in interactions between a client and a database server is that the client may not want a database server to know what information the client is querying, perhaps because the client wishes to avoid being subjected to targeted marketing or because the queries can reveal important business information. At the same time, the server would like to ensure that the client does not learn information about the database beyond the output of its queries. *Selective private function evaluation* (SPFE), introduced by Canetti et al. [17] provides a solution to these issues. Specifically, it allows a client to interact with a database server to compute a function f of some of the items in the database,

in such a way that the client does not learn anything else about the values in the database and the server does not learn which values were queried.

Applying SPFE to aggregate queries, one might hope that all privacy concerns would be addressed: the client can only make aggregate queries and so does not learn individual items, the server does not learn what queries the client makes, and the client does not learn anything beyond the specified query results. However, this is insufficient to alleviate all privacy concerns. It may be possible to combine the results from multiple queries to reveal other information about the data. As a trivial example, if a client first queries the sum of entries 1 through 10 in the database, and next queries the sum of entries 2 through 10, she can then subtract the two to learn the value of entry 1. In the statistical database literature, this kind of privacy violation is sometimes addressed through *inference control* mechanisms [1]. The database administrator has an *inference control rule* that determines whether a given query is to be allowed or denied based on the query, the history of previous queries, and possibly the database values themselves. For example, in the case of the two SUM queries just discussed, inference control could disallow the second query based on its large intersection with the first query. However, these inference control methods cannot be directly applied to SPFE because they require the server to know the indices being queried.

Note that SPFE can be thought of as an extension to *symmetric private information retrieval* (SPIR) [59]. SPIR allows a client to query individual database elements such that the client learns nothing else about the database and the server does not learn which value is being queried. Thus, SPIR can be thought of as SPFE in which the function is the identity function with a single input. Woodruff and Staddon [141] introduced *private inference control* (PIC) to enable inference control for single database element retrievals. That is, their solutions provide SPIR protocols that ensure that the client only learns query

results for queries that pass a specified inference control rule while still maintaining the other privacy requirements of SPIR.

In this chapter, we introduce aggregate private inference control (APIC) for statistical databases, which applies inference control policies to aggregate queries in a privacy-preserving manner. In our work, a client wants to interact with a database server to compute an aggregate query represented by a function f applied to some of the items in the database. These queries are subject to an inference control rule that determines whether a given query is allowed. The client performs a sequence of such queries. For each query, the client learns the value of the function for that query if and only if the query passes the inference control rule. The server learns nothing about the queries, and the client learns nothing other than the value of the function.

6.1.1 Related Work

Statistical disclosure control methods aim to prevent statistical data from disclosing confidential information about individuals or organizations [89]. While doing so, these techniques should not substantially reduce the usefulness of the data. This is generally achieved by either perturbing the data or by providing a summary of the data so that no individual's identity can be easily identified [113]. Data collected by government agencies such the census bureau are released in two ways: (i) Through microdata files which contain a subset of individual records of the database, and are released to the public after the data entries are perturbed to minimize the risk of disclosure; and (ii) as aggregate data, such as frequency count tables. In this section we present a brief overview of the well known results in this area.

When survey data is released to the public, one of the techniques used to prevent disclosure of sensitive information is to add random noise to the data [87]. Although this may preserve some of the statistical properties of the data, it could compromise the privacy of the participants of the survey [56]. DeWaal and Willenborg [33] presented a statistical disclosure technique that combines global recoding and local suppression. Global recoding combines several categories into one. For example the two categories “actor” and “artist” may be combined into “actor or artist”. In local suppression, certain values corresponding to certain attributes are suppressed. The μ -Argus package [73] was implemented based on this technique. Warner [139] introduced the randomized response technique that allows respondents to respond to sensitive questions while still maintaining their privacy. Later Warner [138] observed that the randomized response technique can also be used to perturb the data in the database to be released. Rosenberg [122] presented a randomized response model that works well for categorical data. Microaggregation involves partitioning the original data into groups and publishing aggregates for each group instead of publishing the actual data [37].

The problem of inference control has been widely studied in regular databases, statistical databases, and even in database systems supporting multilevel security [1, 49, 22, 32, 117]. Statistical databases are particularly vulnerable to these kinds of attacks. Adam and Worthmann provide a comprehensive, but dated, summary of inference control for statistical databases in [1]. Inference control techniques in statistical databases include mechanisms such as controlling query overlap, restricting the query size, and data perturbation [1]. Chin [21] presented various examples that shows how answering queries with small counts may compromise individual’s privacy. His work addressed the security in statistical

databases with small counts. Denning and Scholrer [31] showed that it is possible to construct a “tracker” with which one can construct answers to some of the queries that are not answered by the database mechanism, such as queries pertaining to small groups.

Aiello et al. [3] introduced *priced oblivious transfer* in the context of selling digital goods. Their work addresses certain kinds of inference control, but not in a general context as in [141]. Kenthapadi et al. [86] note that in some cases, the fact of whether a query is allowed or disallowed can itself leak information. To avoid this, they define the notion of *simulatable auditing*, in which query denials provably do not leak information. However, in that work the queries are seen by the server and hence the client’s privacy is not met. The inference control rules used in our paper are dependent only on the query pattern, not on the query results, and therefore meet the requirements of being simulatable and leak-free.

6.1.2 Our Contributions

In this chapter, we present private inference control techniques for database queries over multiple elements. In our setting, the server holds a database $x = x_1, \dots, x_n$ and the client queries the database to compute the function $f(x_{i_1}, \dots, x_{i_k})$. At the end of the protocol, the client receives $f(x_{i_1}, \dots, x_{i_k})$ if the query (i_1, \dots, i_k) passes the inference control rule. Otherwise, the client receives an arbitrary value. In any case, the server learns nothing about the queries themselves, including whether the query was allowed or disallowed. Our contributions can be summarized as follows:

- We present a protocol for private inference control for aggregate queries. We also present two alternate protocols, one of which is more efficient when the client makes a small number of queries, each with a large query size, and the other which is efficient when the client makes a large number of queries.

- We present a protocol for private inference control that uses a more relaxed inference control rule than the one employed in the first set of protocols.
- In addition to the generic protocol that applies to any arbitrary function f , we present simpler protocols for two of the most basic types of aggregate query: the SUM function and the COUNT function.
- We present a protocol for a distributed database that is horizontally partitioned between two servers.

The rest of the chapter is organized as follows. In Section 6.3, we present our primary Aggregate Private Inference Control protocol and its variants. In Section 6.4 we present extensions to our basic protocol, including a protocol that uses a more relaxed inference control rule, simpler protocols for the SUM and COUNT queries, and a protocol for horizontally partitioned databases.

6.2 Preliminaries

In this section, we define our model for aggregate private inference control. We also provide brief reviews of SPFE and of Woodruff and Staddon’s PIC protocols.

6.2.1 Selective Private Function Evaluation

Selective Private Function Evaluation (SPFE) was introduced by Canetti et al. [17] as a generalization of Symmetric Private Information Retrieval (SPIR) [25]. In the SPFE problem, a client interacts with one or more servers holding copies of a database $x = x_1, x_2, \dots, x_n$ to compute $f(x_I)$ where $x_I = (x_{i_1}, x_{i_2}, \dots, x_{i_m})$ for some function f and a set of indices $I = i_1, i_2, \dots, i_m$ chosen by the client. A solution to the problem should satisfy the following privacy requirements:

1. Client Privacy. No adversary that involves using the collusion of up to t servers learns anything about the client's query beyond f and m .
2. Database Privacy. The client learns only the value of $f(x_I)$ and nothing else, even if it arbitrarily deviates from the protocol.

Canetti et al. present three sets of solutions for the SPFE problem:

1. Multi-server protocols based on multivariate polynomial evaluation.
2. Solutions based on private simultaneous message protocols.
3. Solutions based on general secure multiparty computation.

In our work, we add private inference control to the third solution. A brief overview of the third solution follows. The protocol is divided into two phases:

1. The *input selection phase*, where the server and the client obtain an additive secret sharing of the m chosen items x_I . In other words, at the end of this phase, the client and the server respectively receive random elements a_I and b_I such that the random shares add up to x_I .
2. The *secure multiparty computation phase*, where the client and the server invoke a secure MPC protocol to compute $f(x_I)$ using their share of the input computed in the first phase.

6.2.2 Private Inference Control

Private Inference Control (PIC) was introduced by Woodruff and Staddon [141] for queries of a single database element. A PIC protocol enables the server to provide inference control

without knowing the client's queries. We review one of their solutions. The protocol considers a single honest-but-curious server and malicious clients. They assume a "rectangular" database of the form

$$\begin{pmatrix} x_{1,1} & \dots & x_{1,m} \\ \vdots & & \\ x_{n,1} & \dots & x_{n,m} \end{pmatrix}$$

Each query is of the form (i_t, j_t) , where $1 \leq i_t \leq m, 1 \leq j_t \leq n$.

A solution to this problem should satisfy the following security requirements:

1. Client Privacy: The server learns nothing about the client's queries.
2. Server Privacy:
 - (a) Database Privacy: For each query (i_t, j_t) , the client learns only (x_{i_t, j_t}) .
 - (b) Private Inference Control: The client learns (x_{i_t, j_t}) if and only if the query (i_t, j_t) passes the inference control rule.

The protocol makes use of a homomorphic encryption scheme E . The client sends the server encryptions of the query $(E(i_t), E(j_t))$. The server encrypts a secret S using the properties of the homomorphic encryption in such a way that the client can recover S only if the client's query has passed the inference control rule.

Since the server does not know the client's queries, a malicious client can use one query to pass the inference control test but a different query to retrieve values from the database. The server handles this situation by choosing two random numbers $u_{i,j}$ and $v_{i,j}$ for each $x_{i,j}$, and constructs a "masked" database consisting of entries as follows:

$$E(u_{i,j}(j - j_t) + v_{i,j}(i - i_t) + V_t + x_{ij})_{i,j}.$$

Here V_t denotes the secret value, which the client can recover only if the query has passed inference control. The client then uses SPIR to retrieve the (i_t, j_t) entry in the reconstructed database. This succeeds only if the index values used in SPIR match the index values provided in the query itself.

6.2.3 Our Model

In our setting, a server holds the database $x = x_1, \dots, x_n$ and the client queries the database. Each query is specified by a set of indices $I = (i_1, \dots, i_k)$; the client wishes to compute the function $f(x_{i_1}, \dots, x_{i_k})$, which we also denote by $f(x_I)$. For the sake of simplicity, we assume every query has k indices; our solutions can be modified to work with different numbers of indices for different queries. We assume both parties know the function f ; it is possible to avoid the server knowing f by using a universal circuit. We also assume k is known to both parties. The server should not learn anything about the client's queries. The client should learn no more than the output of the function evaluated at the indices chosen by it, and it should only learn this output if its query passes the inference control rule. Furthermore, the server should not learn whether a client's query has passed the inference control rule. In all cases, these requirements take the form of semantic security, so no partial information beyond the specified outputs should be revealed. Specifically, our private inference control protocol should satisfy the following requirements, defined relative to a specified *inference control rule*:

- **Correctness:** When the client and the server follow the protocol, the client obtains $f(x_I)$ if I satisfies the inference control rule.
- **Client Privacy:** The server should not learn anything about the client's queries, including whether or not each query passes the inference control rule. Formally, there

exists a simulator that produces an output distribution which is computationally indistinguishable from the server's view.

- **Server Privacy:** The server's privacy includes two concepts:
 - **Database Privacy:** For each query I that passes the inference control rule, the client learns only $f(x_I)$ and nothing else.
 - **Private Inference Control:** For each query I that does not pass the inference control rule, the client receives an arbitrary value¹.

We note that these requirements imply that the server must apply the inference control rule on the queries without actually knowing the client's queries or learning whether each query passes.

- **Inference Control Rule:** The first inference control rule we consider (in Section 6.3) requires that, when the client makes multiple queries, the set of input indices in the current query should not intersect with any of the input indices of previous queries.

The following example demonstrates the privacy need for such a requirement. If f is the weighted sum function and the client repeats the same query with different weights k , then the client can solve the system of equations to obtain x_I .

In this privacy model, we assume that the server is semi-honest meaning that the server will faithfully follow the protocol but will record all the intermediate messages. We also assume that the client is malicious, meaning the client may deviate arbitrarily from the protocol, change its inputs or abort the protocol. Our protocols only offer weak security [17]

¹We note that if the inference control rule is public and dependent only on the indices of all the queries, then the client knows which queries are allowed and disallowed, and therefore can avoid ever confusing an arbitrary value with a real response.

against a malicious client, meaning that the client only learns some function f' on some sequence of data items. Here the function f' is determined by the client's actions and f' is guaranteed to have the same output size as f .

6.3 Private Inference Control for Aggregate Queries

In this section, we present protocols that enforce inference control while processing queries for statistical information in databases. We assume that all the indices in the client's query are distinct and valid (in the sense that, if they are not, the privacy guarantee may not be met). This last property can, for example, be proved to the server by the client using a zero knowledge proof for each query. (It is an open problem if there is a more efficient way to handle this.) The protocols in this section enforce the following inference control rule on each of the client's queries: the set of indices in the current query should not intersect with any of the sets of indices used in previous queries. Because this rule is overly restrictive, we also consider a relaxed version of this inference control rule which requires that the cardinality of the intersection of the queries be less than some threshold value t . We describe the solution to the relaxed inference control rule in Section 6.4.1. We denote by f a function (such as the sum or average) that the client wants to evaluate in his query.

As mentioned in Section 6.2, we make use of general secure multiparty computation as part of our solution. In particular, in this section, we make use of secure multiparty computation to compute f itself. In this setting, especially when $k \ll n$, this can be done efficiently using the protocols of [17]. The APIC protocols presented in this section have four phases:

1. **Query generation phase:** In this phase, the client sends his query to the server.

Since he does not want the server to know the indices in the query, they are sent in a

“masked” form.

2. **Inference control phase:** In this phase, the server chooses a secret value V and masks it in such a way that the client can retrieve V if and only if the query satisfies the inference control rule.
3. **Query processing phase:** This has two sub-phases, namely
 - **Input selection phase:** The client and the server obtain a simple secret-sharing of x_I .
 - **Secure multiparty computation phase:** The client and the server use their shares of the input computed in the input selection phase along with the secret value V chosen by the server to compute the function $g(x_I, V) = f(x_I) + V$. This is done using the secure multiparty protocol given in [17]. The client receives the value of the function $g(x_I, V)$ and the server receives no output.

This phase is similar to the SPFE solution presented in [17], with the added feature of private inference control.

4. **Answer construction phase:** The client computes V , and hence $f(x_I)$. The client can compute the secret value V if and only if the query passes the inference control rule. The server does not learn whether the client’s query has passed or failed the inference control rule.

The structure of the protocol is to first determine whether the client’s query passes the inference control rule, and then to process the query. Given that the server does not know the client’s queries, a naive way of doing this would allow a cheating client to use one query to pass the inference control rule but a different (and possibly disallowed) query

to retrieve values from the database. To avoid this, our protocol ensures that in such a situation, at the end of the input selection phase, the shares obtained by the client and the server do not add up to x_I , but instead to an arbitrary vector that is independent of the contents of database and unknown to the client. Hence, in this case, the output of the client in the second phase is $f(r_I)$ for some arbitrary r_I . (As previously noted, the client already knows if the inference control rule is not satisfied, so an honest client that does not try to circumvent the inference control rule need never receive an incorrect value.) This is achieved by the server using the masked form of the client's queries in both the inference control phase and in the query processing phase. We present three different APIC protocols in Sections 6.3.1, 6.3.2 and 6.3.3. We present a comparison between the costs of the three protocols in Section 6.3.4.

6.3.1 The First Protocol

In our solution, which is shown in full as Protocol 8, the client and the server agree on a homomorphic encryption scheme. At the beginning of the protocol, the client chooses a public/secret key pair for the chosen encryption scheme and sends the public key to the server.

In the query generation phase, the client sends the encryption of the indices of each of his queries along with a zero-knowledge proof of knowledge that the ciphertexts are well formed. In the inference control phase, the server uses the information obtained during the query generation phase along with the homomorphic property of the encryption scheme to encrypt a secret value V such that the client can compute the secret value V if and only if the query passes the inference control rule. In the query processing phase, our solution makes use of homomorphic encryption to perform oblivious polynomial evaluation (similar

techniques were used in [17, 53]). This allows the client and server to evaluate certain polynomials in a shared way while keeping certain information private.

Theorem 2 *Protocol 8 is a private inference control protocol for aggregate queries.*

Proof: We assume the semantic security of the homomorphic encryption scheme.

Correctness

When the client and the server follow the protocol, an honest client obtains $f(x_I) + V$ where V is the secret key. If the query I passes the inference control rule then the client retrieves the secret key and computes $f(x_I)$ the desired output.

Client Privacy

In our protocol, the server (\mathcal{S}) is assumed to be honest-but-curious. The server's view includes the encryptions sent by the client (\mathcal{C}), the SPIR, and the SPFE interactions with the client. Assuming that the homomorphic encryption scheme used in the protocol is semantically secure, the client's privacy of the protocol depends on the privacy of SPIR and SPFE. Let M_1 denote a simulator for SPIR and M_2 denote a simulator for SPFE [17]. Since we have assumed that the server is honest-but-curious, the input to the simulator is fixed at the beginning of the protocol. The server gets no output at the end of the protocol. Hence, the view of the server in the APIC protocol is the concatenation of the outputs of M_1 and M_2 .

Inference Control

We compare this protocol with one that uses a trusted third party to perform the computation. For every client \mathcal{C} in the real model, there exists a client \mathcal{C}^* in the ideal model (one

Protocol 8 Private Inference Control for Aggregate Queries

Server \mathcal{S} 's Input: Database $D = (x_1, \dots, x_n)$
Client \mathcal{C} 's Input: Queries $Q_j = (i_{j1}, \dots, i_{jk})$, for $j = 1, 2, \dots$
Client \mathcal{C} 's Output: For each query Q_j , the client obtains the value of $f(x_I)$ where $x_I = (x_{i_{j1}}, \dots, x_{i_{jk}})$ iff I passes the inference control rule.
Inference control rule: If Q_t is the current query and Q_1, \dots, Q_{t-1} are the previous queries, then Q_t is allowed if $Q_t \cap Q_1, \dots, Q_t \cap Q_{t-1}$ are empty.

- For Q_1 , \mathcal{C} chooses a key pair (pk, sk) , and sends pk to the server.
- For $j \geq 1$,

1. Query generation:

(a) For query $Q_j = (i_{j1}, \dots, i_{jk})$, \mathcal{C} sends the following encrypted values to \mathcal{S} .

$$\begin{pmatrix} E(i_{j1}) & \dots & E(i_{j1}^{k-1}) \\ \vdots & & \\ E(i_{jk}) & \dots & E(i_{jk}^{k-1}) \end{pmatrix}$$

(b) \mathcal{C} gives a zero-knowledge proof that the ciphertexts in Step 2a are well formed.

2. Inference control:

(a) \mathcal{S} chooses secret values V_1, \dots, V_{j-1} . (For Q_1 , \mathcal{S} sets the secret values as zeros and skips the rest of the inference control steps) For each $1 \leq \ell \leq j-1$, \mathcal{S} generates k^2 random shares $\{y_{m1}^{(\ell)}, \dots, y_{mk}^{(\ell)}\}$, $1 \leq m \leq k$ that add up to V_ℓ .

(b) For \mathcal{C} to learn V_ℓ , for $1 \leq \ell \leq j-1$ if and only if the query passes the inference control rule, \mathcal{S} sends the following $(j-1)k^2$ values to \mathcal{C} :

$$\begin{pmatrix} E((i_{j1} - i_{\ell 1})y_{11}^{(\ell)}) & \dots & E((i_{j1} - i_{\ell k})y_{1k}^{(\ell)}) \\ \vdots & & \\ E((i_{jk} - i_{\ell 1})y_{k1}^{(\ell)}) & \dots & E((i_{jk} - i_{\ell k})y_{kk}^{(\ell)}) \end{pmatrix}$$

(c) \mathcal{C} decrypts all the $(j-1)k^2$ y 's to obtain the secret $V_1, \dots, V_{(j-1)}$. (\mathcal{C} obtains the correct V 's if and only if the inference control rule is satisfied and the client followed the protocol; otherwise the values do not sum to V_ℓ .)

3. Query processing:

(a) **Input selection:**

1. \mathcal{S} chooses a random polynomial $P(u) = s_0 + s_1u + \dots + s_{k-1}u^{k-1}$. For $1 \leq m \leq k$, \mathcal{S} constructs a masked database $E(x_p + P(p) + r_{mp}(p - i_{jm}))$ for $1 \leq p \leq n$, where the r_{mp} 's are random.

2. For each $1 \leq m \leq k$, \mathcal{C} and \mathcal{S} engage in SPIR and client retrieves $E(x_{i_{jm}} + P(i_{jm}))$.

3. \mathcal{C} decrypts and obtains $z_{i_{jm}} = x_{i_{jm}} + P(i_{jm})$, for $1 \leq m \leq k$.

4. \mathcal{C} and \mathcal{S} engage in a secure computation to compute shares of $x_{i_{jm}}$, for $1 \leq m \leq k$:

- \mathcal{S} picks up k random elements q_1, \dots, q_k and computes $E(P(i_{j1}) - q_1), \dots, E(P(i_{jk}) - q_k)$ and sends them to \mathcal{C} .
- \mathcal{C} decrypts and obtains $(P(i_{j1}) - q_1), \dots, (P(i_{jk}) - q_k)$
- \mathcal{C} 's shares are $a_{i_{j1}} = z_{i_{j1}} - (P(i_{j1}) - q_1), \dots, a_{i_{jk}} = z_{i_{jk}} - (P(i_{jk}) - q_k)$
- \mathcal{S} 's shares are $b_{i_{j1}} = -q_1, \dots, b_{i_{jk}} = -q_k$.
- \mathcal{C} and \mathcal{S} 's shares add up to $x_{i_{jm}}$, for $1 \leq m \leq k$.

(b) **Secure multiparty computation:** \mathcal{C} and \mathcal{S} use secure multiparty computation in order for \mathcal{C} to learn the output of the function $g(x_I, V_1, \dots, V_{(j-1)}) = f(x_I) + V_1 + \dots + V_{(j-1)}$. \mathcal{S} receives no output.

4. Answer construction: \mathcal{C} can recover $f(x_I)$ iff he can compute all the secret values.

with the trusted third party (TTP)) such that the view of \mathcal{C} is indistinguishable from the view of \mathcal{C}^* . We describe \mathcal{C}^* 's simulation on j th query.

A simulator for SPIR, when given the code of \mathcal{C} extracts the indices i_1, \dots, i_k . \mathcal{C}^* takes as its input the code of \mathcal{C} and uses the knowledge extractor of the zero-knowledge proof of knowledge to obtain (i'_1, \dots, i'_k) . If the two sets of indices (i_1, \dots, i_k) and (i'_1, \dots, i'_k) are not the same then \mathcal{C}^* chooses random values r_1, \dots, r_k and gives $E(r_1), \dots, E(r_k)$ to \mathcal{C} . If they are the same then as a result of SPIR simulation, \mathcal{C}^* gives \mathcal{C} , $(E(x_{i_1} + P(x_{i_1})), \dots, E(x_{i_k} + P(x_{i_k})))$. \mathcal{C}^* runs through the inference control phase and the rest of the input selection phase as a honest server. If (i'_1, \dots, i'_k) satisfies the inference control rule, \mathcal{C}^* requests the TTP to provide $f(x_{i_1}, \dots, x_{x_k})$ and gives \mathcal{C} , $f(x_{i_1}, \dots, x_{x_k}) + V$ as the output of the SMC.

If a malicious client uses one set of indices for inference control and another set for SPIR then the encryptions obtained by \mathcal{C}^* are encryptions of random values which is also the case in the actual execution. If the inference control rule is not satisfied \mathcal{C}^* 's output is a random value which is also the case in actual execution. In the case where the inference control rule is satisfied, the encryptions received by \mathcal{C}^* from \mathcal{C} is indistinguishable from the ones received by \mathcal{C}^* in the actual execution because of the semantic security of the encryption scheme. Hence, the view of \mathcal{C}^* interacting with a TTP and the view of the \mathcal{C} interacting with a honest server are indistinguishable.

Since our protocol has two phases (the input selection phase and the secure multiparty computation phase), a malicious client can change its share of x_I before passing them as inputs into the multiparty computation stage [17]. In this case, the client ends up computing $f(x_I + \delta)$ rather than computing $f(x_I)$. This does not violate inference control, since the client does not know the entries in the database, and does not know what $f(x_I + \delta)$ corresponds to in terms of entries in the database. Hence, our protocol has weak security

against a malicious client [17]. ■

COMMUNICATION AND COMPUTATIONAL COMPLEXITY

We discuss in this section the cost of the j th query Q_j . Let w denote the maximum number of bits needed to represent an encryption. In the query generation phase, the client transmits $O(wk^2)$ bits. In the inference control phase, the server transmits $O(wjk^2)$ bits. In the query processing phase, the communication cost is dominated by the k executions of SPIR and the cost of one execution of SMC. The total communication cost of one query is $O(wjk^2) + kw \cdot \text{polylog}(n) + \text{cost of SMC}$. This protocol requires 1.5 rounds + round complexity of SMC. (1.5 rounds = message sent by client + one round of the SPIR protocol. The message sent by the server can go in parallel with SPIR.)

Determining the computation complexity involves counting the number of encryptions, decryptions, and exponentiations. The server performs $O(nk)$ encryptions while masking the database and $O(jk^2)$ encryptions for the inference control rule. The client performs $O(k^2)$ encryptions and $O(jk^2)$ decryptions. The encryptions of the polynomial evaluations in query processing stage involve $O(k)$ exponentiations when done naively, but this overhead can be reduced using Horner's method [88]. The polynomial evaluation happens only once and hence the total number of exponentiations required is $O(nk + jk^2)$.

6.3.2 The Second Protocol

The private inference control protocol for aggregate queries presented in Section 6.3.1 is efficient for moderate sized databases and when the length k of each query is small. For large k and large n , however, the protocol is inefficient because it requires encrypting the database k times for each query. In this section, we present a modified solution which avoids

encrypting the database k times.

In this solution, which is shown in full as Protocol 9, the client and the server agree on a homomorphic encryption scheme E . At the beginning of the protocol, the client chooses a public/secret key pair for the chosen encryption scheme and sends the public key to the server. The server chooses a seed s to a pseudo-random function h . This function h is used to mask the database.

The query generation phase uses secure circuit evaluation [145]. It evaluates a circuit which receives as input from the client the indices i_1, \dots, i_k of the current query. The server's input to the circuit is the seed s and the public key pk . The circuit computes random shares of $\{h(s, i_1), \dots, h(s, i_k)\}$. It outputs the client's share, $\{h^C(s, i_1), \dots, h^C(s, i_k)\}$, to the client, and the server's share, $\{h^S(s, i_1), \dots, h^S(s, i_k)\}$, to the server. The circuit also computes and sends to the server the values $\{E(i_1), \dots, E(i_k)\}$.

In the query processing phase the server constructs a masked database D' by setting the i th entry to be $x_i \oplus h(s, i)$. The client and the server engage in SPIR on D' to obtain $x_{i_\ell} \oplus h(s, i_\ell)$, for $1 \leq \ell \leq k$. The client and the server engage in SMC with the client's input as $x_{i_\ell} \oplus h(s, i_\ell) \oplus h^C(s, i_\ell)$ and the server's input as $h^S(s, i_\ell)$, for $1 \leq \ell \leq k$ and a vector of secret values. The inference control phase and the answer construction phase are the same as in Section 6.3.1. The proof of privacy is similar to that of Theorem 2.

COMMUNICATION AND COMPUTATIONAL COMPLEXITY

The communication complexity of the circuit is $\text{poly}(k \log(n))$. In the inference control phase the server transmits $O(wjk^2)$ bits. In the query processing phase the communication cost is dominated by the cost of one execution of SPIR and the cost of one execution of SMC. The total communication cost of one query is $O(wjk^2) + \text{cost of SPIR} + \text{cost of SMC}$.

Here the SPIR protocol takes place on a set of n records each of length w to retrieve k items and its communication complexity is $kw \cdot \text{polylog}(n)$.

The server performs $O(jk^2)$ encryptions for the inference control rule. The server's work is linear in the size of the database. (Note that the server masks the database using the XOR operation once rather than encrypting the database k time as in the protocol presented in Section 6.3.1.) The client performs $O(k^2)$ encryptions and $O(jk^2)$ decryptions.

6.3.3 The Third Protocol

The communication complexity of the APIC protocols presented in Sections 6.3.1 and 6.3.2 depend on the number of past queries made by the client. So the communication overhead in the inference control phase increases linearly in terms of the number of queries made by the client. In this section, we present an APIC protocol that keeps the communication cost of the inference control phase low even as the number of queries increases. A comparison of the costs between all the three approaches is provided in Section 6.3.4. This solution is an extension to the protocol presented in Section 7 of [141]. For consistency we use the same notation as in [141].

In this protocol, the query generation and inference control phases are combined into one phase. The phase uses a secure circuit evaluation protocol [145] to evaluate a circuit which we will now describe. We use a balanced binary tree that has n leaves associated with the elements $\{x_1, \dots, x_n\}$ of the database. We denote the leaves by i where $i \in \{1, 2, \dots, n\}$. Let α denotes the root of the binary tree. Each node of the tree is associated with a key $K(w, z)$ whose computation we will describe shortly. Here w represents the node, and z an integer value. For a leaf node, z takes the value 0 if the corresponding value has never been accessed in any query, and 1 otherwise. In the case of an internal node, z denotes the

Protocol 9 Aggregate Private Inference Control Protocol: A Second Approach

Server \mathcal{S} 's Input: $D = (x_1, \dots, x_n)$ and a seed s to a pseudo-random function h .
Client \mathcal{C} 's Input: Queries $Q_j = (i_{j1}, \dots, i_{jk})$, for $j = 1, 2, \dots$
Client \mathcal{C} 's Output: For each query Q_j , the client obtains the value of $f(x_I)$ where $x_I = (x_{i_{j1}}, \dots, x_{i_{jk}})$ iff I passes the inference control rule.
Inference control rule: If Q_t is the current query and Q_1, \dots, Q_{t-1} are the previous queries, then Q_t is allowed if $Q_t \cap Q_1, \dots, Q_t \cap Q_{t-1}$ are empty.

- For Q_1 , \mathcal{C} chooses a key pair (pk, sk) , and sends pk to the server.

- For $j \geq 1$,

1. **Query Generator:** Let \mathcal{C} denote a secure circuit as described in Section 6.3.2. \mathcal{C} inputs the query indices of Q_j and \mathcal{S} inputs the seed s and the public key pk into the circuit. The circuit outputs $h^C(s, i_{j1}), \dots, h^C(s, i_{jk})$ to \mathcal{C} . The circuit outputs $h^S(s, i_{j1}), \dots, h^S(s, i_{jk})$ and $E(i_{j1}), \dots, E(i_{jk})$ to \mathcal{S} .

2. **Inference Control**

- (a) \mathcal{S} chooses secret values V_1, \dots, V_{j-1} . (For Q_1 , \mathcal{S} sets the secret values as zeros and skips the rest of the inference control steps) For each $1 \leq \ell \leq j$, \mathcal{S} generates k^2 random shares $\{y_{m1}^{(\ell)}, \dots, y_{mk}^{(\ell)}\}$, $1 \leq m \leq k$, that add up to V_ℓ .
- (b) For \mathcal{C} to learn V_ℓ if and only if the query passes the inference control rule, \mathcal{S} sends the following $(j-1)k^2$ values to \mathcal{C} :

$$\begin{pmatrix} E((i_{j1} - i_{\ell 1})y_{11}^{(\ell)}) & \dots & E((i_{j1} - i_{\ell k})y_{1k}^{(\ell)}) \\ \vdots & & \vdots \\ E((i_{jk} - i_{\ell 1})y_{k1}^{(\ell)}) & \dots & E((i_{jk} - i_{\ell k})y_{kk}^{(\ell)}) \end{pmatrix}$$

for $1 \leq \ell \leq j-1$.

- (c) \mathcal{C} decrypts all the $(j-1)k^2$ y 's to obtain the secret $V_1, \dots, V_{(j-1)}$. (\mathcal{C} obtains the correct V 's if and only if the inference control rule is satisfied and the client followed the protocol; otherwise the values do not sum to V_ℓ .)

3. **Query Processing**

- (a) **Input Selection Phase:**

- i. \mathcal{S} constructs a masked database D' whose i th entry is $x_i \oplus h(s, i)$.
- ii. For each $1 \leq m \leq k$, \mathcal{C} and \mathcal{S} engage in SPIR and client retrieves $x_{i_{jm}} \oplus h(s, i_{jm})$
- iii. \mathcal{C} computes his share of input as $x_{i_{jm}} \oplus h(s, i_{jm}) \oplus h^C(s, i_{jm})$ for $1 \leq m \leq k$. \mathcal{S} 's shares are $h^S(s, i_{jm})$ for $1 \leq m \leq k$.

- (b) **Secure Multiparty Computation Phase:** \mathcal{C} and \mathcal{S} use secure multiparty computation in order for \mathcal{C} to learn the output of the function $g(x_I, V_1, \dots, V_{(j-1)}) = f(x_I) + V_1 + \dots + V_{(j-1)}$. The server receives no output.

4. **Answer Construction Phase:** The client can recover $f(x_I)$ if and only if he can compute all the secret values which he can do only if his queries satisfy the inference control rule and the client follows the protocol. Otherwise the client obtains an arbitrary value.

number of times the leaves in the subtree rooted at w have been accessed in past queries.

Let E denote a non-malleable symmetric key encryption scheme [36]. Let sk denote the secret key of the encryption scheme chosen and known only to the server. We compute $K(w, z)$ as the encryption $E_{sk}(w, z)$. For any internal node w along with its children, we say the keys $K(w, z)$ and $\{K(v, m_v) | v \in \text{children}(w)\}$ are *sum-consistent* if $z = \sum_{v \in \text{children}(w)} m_v$. When the client issues a query $\{i_1, \dots, i_k\}$, he inputs the set of keys

$$\pi = \cup_{\ell=1}^k \{K(w, m_w) | w \in \text{sibanc}(i_\ell)\}$$

to the circuit where

$$\text{sibanc}(w) = \text{anc}(w) \cup \{u | \exists v \in \text{anc}(w) \text{ and } u = \text{sib}(v)\},$$

$\text{sib}(v)$ denotes the siblings of v and $\text{anc}(v)$ denotes the ancestors of v .

A malicious user may try to use $K(w, z)$ instead of $K(w, m_w)$ for some integer $z \neq m_w$. We maintain the invariant that when a malicious client replaces $K(w, z)$ for $K(w, m_w)$ then $z < m_w$. (For further discussion, see [141].) The server inputs a seed s to a pseudo-random function h and a key sk to the encryption scheme E . The circuit checks whether π satisfies the following properties:

- For each internal node $w \in \text{anc}(i_\ell)$, $K(w, m_w)$ and $\{K(v, m_v) | v \in \text{children}(w)\}$ are *sum-consistent*, for $1 \leq \ell \leq k$.
- $m_\alpha = jk$
- $K(i_\ell, 0) \in \pi$, for $1 \leq \ell \leq k$. (Inference control rule)

If π satisfies these properties, then the circuit computes random shares of $\{h(s, i_1), \dots, h(s, i_k)\}$

and outputs to the client and the server. If not, the circuit sends random values to the client

and the server. The client also receives the updated keys $\{K(w, m_w + 1) | K(w, m_w) \in \pi\}$. The query processing and answer construction phase are the same as in Section 6.3.2.

Suppose a malicious client wants to query $\{x_{i1}, \dots, x_{ik}\}$ for which some $x_{i\ell}$ was a part of one of the previous queries thus violating the inference control rule. This requires changing some of the keys $K(w, m_w)$ in π to $K(w, z)$ for $m_w \neq z$. By the invariance, $z < m_w$ and hence the first two properties mentioned above cannot hold simultaneously.

In [141], for reject queries the client sends an integer P to the circuit. When the client is honest, P will be of the form $E_{sk}(\text{reject}, z)$. When π does not satisfy the inference control rule the circuit outputs $E_{sk}(\text{reject}, z+1)$ to the client. The server gets no output maintaining user privacy. In our case, the circuit outputs both to the client and the server. When the client's query does not pass the inference control rule the circuit outputs random values to both the client and the server so that the server does not know whether the client's query has passed or failed. The client and the server may engage in the query processing phase but at the end of the protocol the client receives only an arbitrary value. The proof of privacy is similar to the one in Section 7 of [141].

COMMUNICATION AND COMPUTATIONAL COMPLEXITY

The communication complexity of the secure circuit evaluation protocol that tests the inference control rule is $\text{poly}(k \log(n))$. The overall communication complexity is $\text{poly}(k \log(n)) + \text{cost of SPIR} + \text{cost of SMC}$. Here the SPIR protocol takes place on a set of n records each of length w to retrieve k elements and its communication complexity is $kw \cdot \text{polylog}(n)$. The computational complexity is $O(n)$. Both the circuit evaluation and the SPIR takes one round and they can run in parallel. This protocol requires one round more than the round complexity of SMC.

6.3.4 A Comparison

The second and the third APIC protocols presented in Sections 6.3.2 and 6.3.3 use a secure circuit-evaluation protocol for query generation. (The third APIC protocol combines the query generation and inference control phases into one phase.) The first protocol given in Section 6.3.1 avoids the expensive circuit evaluation for the query generation and inference control phases. However, this involves encrypting the database k times in the query processing phase which may be expensive for large databases and for large values of k . (Recall k is the query size.) This protocol works well for moderately sized databases and when the query size is reasonably small.

On the other hand, the second and the third protocols avoid public key encryptions of the database and hence work well for large databases. But the price they pay is the use of a circuit for the query generation and inference control phases. The query processing phase is the same for both of these protocols. The circuit size for both protocols are the same in terms of the O notation given by $(\text{poly}(k \log n))$. But the input size for the first protocol is given by $O(k \log n)$ bits whereas the input size for the third protocol is $O(wk \log n)$, $w \geq \log(nq)$ where q denotes the number of queries made. Since the number of OT_1^2 's depend on the input size, the computational overhead (number of exponentiations) of the third protocol is higher than the second protocol. In the second protocol, the number of bits transmitted by the server in the inference control phase increases linearly in terms of the number of queries made. Therefore, when a client makes fewer queries, the second protocol is more efficient. On the other hand, when a client makes a large number of queries, the third protocol is more efficient in terms of communication complexity.

6.4 Extensions and Special Cases

In this section, we consider various extensions and special cases of the general case.

6.4.1 Relaxing the Inference Control Rule

In this section, we show how to relax the strict requirement that client query indices should have empty intersections.

Our inference control rule in Section 6.3 requires that when the client makes a query, the indices in the current query should not intersect with the indices of previous queries. Here, we consider a relaxed inference control rule: when a client makes a query, the cardinality of the intersection of the queries should be less than some threshold value t . Denote the first two queries as $I_1 = (i_1, \dots, i_k)$ and $I_2 = (j_1, \dots, j_k)$. We describe the protocol for the second query. It can be generalized for multiple queries.

We describe here the modified inference control phase for the protocols described in Sections 6.3.1 and 6.3.2. All the other phases remain the same. In the inference control phase for the second query, the server randomly generates k^2 shares y_{11}, \dots, y_{kk} forming a $(k^2 - t)$ -out-of- k^2 sharing of V . The server sends the following k^2 values to the client:

$$\begin{pmatrix} E((j_1 - i_1)y_{11}) & \dots & E((j_1 - i_k)y_{1k}) \\ E((j_2 - i_1)y_{21}) & \dots & E((j_2 - i_k)y_{2k}) \\ \vdots & & \\ E((j_k - i_1)y_{k1}) & \dots & E((j_k - i_k)y_{kk}) \end{pmatrix}$$

If the cardinality of the intersection of the first and second queries is less than t then the client can recover at least $k^2 - t$ of the y 's and hence reconstruct V . In the APIC protocol described in Section 6.3.3, the relaxed inference control rule can be easily incorporated since

the inference control rule is built into the circuit.

6.4.2 Special Cases: SUM and COUNT

In this section, we present specialized protocols for the SUM and COUNT function with private inference control. The SUM and COUNT functions are very basic and important in data analysis and processing. The specialized protocols, presented below, use the specific nature of the given function instead of using the generic solutions of Section 6.3 in order to obtain simpler solutions that avoid the general circuit evaluation step. We again assume that all the client's queries have k indices and all the indices in each query are distinct. The inference control rule is the same as explained in Section 6.2. Similar to the solution of Section 6.3, the protocols presented in this section ensure that the client does not use one set of indices to pass the inference control rule and another set of indices to engage the server in SPIR to extract values from the database. We present the modification required for the protocol described in Section 6.3.1. A similar modification can be done for the protocols described in Sections 6.3.2 and 6.3.3.

The Sum Function

We present a protocol for computing the sum of a selected subset of the database. \mathcal{C} has a set of indices $I = (i_1, \dots, i_k)$. \mathcal{C} learns $x_{i_1} + \dots + x_{i_k}$ if the inference control rule is passed and the server learns nothing other than the size of the query.

We describe the modified query processing phase of the protocol described in Section 6.3.1. The other phases remain the same. \mathcal{C} and \mathcal{S} interact to perform a SPIR to obtain

$$y_{i_j} = x_{i_j} + P(i_j), 1 \leq j \leq k$$

$$y_{i_1} + \dots + y_{i_k} = x_{i_1} + \dots + x_{i_k} + P(i_1) + \dots + P(i_k)$$

Therefore, if \mathcal{C} learns $P(i_1) + \dots + P(i_k)$, then this implies that \mathcal{C} can learn $x_{i_1} + \dots + x_{i_k}$.

Note that

$$P(i_1) + \dots + P(i_k) = c_0 s_0 + c_1 s_1 + \dots + c_{k-1} s_{k-1}$$

where $c_0 = k$, $c_j = i_1^j + \dots + i_k^j$ for $1 \leq j \leq k-1$, and these c_j 's are all known to \mathcal{C} . \mathcal{C} sends the encryptions of c_0, \dots, c_{k-1} to \mathcal{S} . \mathcal{S} chooses a secret value V and computes $E(c_0 s_0 + c_1 s_1 + \dots + c_{k-1} s_{k-1} + V) = E(P(i_1) + \dots + P(i_k) + V)$ using the homomorphic property of the encryption scheme and sends it to \mathcal{C} . If \mathcal{C} can compute V (inference control rule is satisfied), then \mathcal{C} outputs $x_{i_1} + \dots + x_{i_k}$.

Counting Frequencies

We present a protocol for counting the number of occurrences or the frequency of a keyword in a subset of the database. The keyword and the subset are chosen by the client. At the end of the protocol, the client gets the frequency of the keyword whereas the server gets no output. The server should not gain any knowledge about the query or about the keyword. The only information the server learns is the size of the query.

We describe the modified query processing phase of the protocol described in Section 6.3.1. The other phases remain the same. \mathcal{C} and \mathcal{S} carry out the input selection phase of Protocol 8 to obtain an additive sharing of x_I . Let a_I denote \mathcal{C} 's share and b_I denote \mathcal{S} 's share. \mathcal{C} sends the k encryptions $E(a_{i_j} - w)$ for $1 \leq j \leq k$ to \mathcal{S} . \mathcal{S} computes k encryptions $E(r_j(a_{i_j} + b_{i_j} - w) + V_j)$ for $1 \leq j \leq k$ where r_j 's and V_j 's are random values chosen by \mathcal{S} . \mathcal{S} sends a random permutation of these k encryptions to \mathcal{C} to prevent \mathcal{C} from learning the exact locations in which the matching has occurred. \mathcal{S} also sends k^2 encrypted values $E((i_q - i_j)y_j^{(p)})$, $k+1 \leq q \leq 2k$, $1 \leq j \leq k$ for each $1 \leq p \leq k$ where

$V_j = y_1^{(j)} + \dots + y_k^{(j)}$. \mathcal{C} can recover all k secret values if and only if the query satisfies the inference control rule. The client decrypts all the k encryptions, subtracts the secret values, and outputs the number of zeros.

6.4.3 Horizontally Partitioned Database

In this section, we present a protocol for aggregate private inference control on a horizontally partitioned database (i.e., one in which each database server holds only some of the elements). For notational simplicity, we assume that the database (x_1, \dots, x_n) is shared between two servers \mathcal{S}_1 and \mathcal{S}_2 . (The generalization to more than two servers is immediate.) We assume that \mathcal{S}_1 has m out of the total n records say x_{i_1}, \dots, x_{i_m} and \mathcal{S}_2 has the remaining $n - m$ records. The client queries the servers to compute the function $f(x_I)$ where $I = (i_1, \dots, i_k)$. The servers communicate with each other, but should not need to share their actual data with each other. At the end of the protocol, the client learns the value of the function $f(x_I)$ if and only if the query passes the inference control rule. The servers get no output. The servers learn nothing about the queries other than the function f and the size of the query. The client learns nothing other than the value of the function evaluated at x_I . The client should not know the shares of $f(x_I)$ that came from the respective servers.

To accomplish this, each server pads its partial database with zeros for the indices it does not own to obtain a database of n items.

1. **Query generation:** \mathcal{C} sends the encrypted values described in the query generation phase of Figure 8 to \mathcal{S}_1 , and \mathcal{S}_1 passes on this information to \mathcal{S}_2 .
2. **Inference control:** \mathcal{C} and \mathcal{S}_1 go through the inference control phase of Figure 8.
3. **Query processing:**

- \mathcal{C} carries out the input selection phase of Figure 8 with \mathcal{S}_1 to obtain an additive sharing of x_I . We denote the resulting shares as a_I (client's share) and b_I (\mathcal{S}_1 's share). \mathcal{C} carries out the input selection phase of Figure 8 with \mathcal{S}_2 to obtain an additive sharing of x_I . We denote the resulting shares as c_I (client's share) and d_I (\mathcal{S}_2 's share).
- the client and the two servers carry out the SMC protocol described in [17] with the inputs as $a_I + c_I$, (b_I, V) , and d_I , respectively, where V is a vector of secret values chosen by \mathcal{S}_1 . In this way, the client obtains the value of the function $g(x_I, V) = f(x_I) + V$.

4. **Answer construction:** The client computes V , and hence $f(x_I)$. The client can compute the secret value V if and only if the query passes the inference control rule. The server does not learn whether the client's query has passed or failed the inference control rule.

6.5 Summary

In this chapter, we introduced private inference control for aggregate queries. This extends the notion of private inference control for individual queries introduced by Woodruff and Staddon [141] to the practical setting of aggregate and complex queries over multiple values. The need for inference control is particularly important in the aggregate query setting because it may be possible to use combined queries to extract individual elements from the database, thereby losing the privacy that the restriction to aggregate queries is often used to provide.

It is open whether our solution in Section 6.3.1 can be improved to require only a single masked version of the database, rather than k versions as it now does, in a way that does

not have the additional overhead per query that the solution of Section 6.3.2 does. (The straightforward way of doing this would allow a client to use a different set of indices for passing the inference control than the indices of the resulting query, thereby bypassing the inference control step.) It also remains open to further extend private inference control additional inference control policies, as well as very efficient solutions for particular kinds of aggregate queries. Of particular interest are inference control policies that depend on the return values themselves, not only the indices of the inputs involved. In this case, for maximum privacy to be guaranteed, it would also be necessary to incorporate notions of simulatable auditing [86].

More generally, it would be extremely desirable to have private inference control for general keyword-based queries such as SQL provides (for example, allowing a query to return the average salary for all individuals in the database who reside in a particular zip code).

Chapter 7

A Practical Differentially Private Random Decision Tree Classifier

7.1 Introduction

Recent work in private data analysis by Dwork et al. [43] has radically changed the research landscape by defining a model with a strong definition of privacy that addresses how much privacy loss an individual might incur by being in a database. In the most common setting under this new framework of *differential privacy*, the data owner makes the data available through a statistical database on which only aggregate queries are permitted. The goal is to answer queries while preserving the privacy of every individual in a database, irrespective of any auxiliary information that may be available to the database client. The differential privacy framework has several advantages compared to various other privacy models:

- The privacy guarantee does not depend on the amount of the auxiliary information available to the adversary, unlike models such as k -anonymity and its variants, as well as the private inference control setting of [141] and Chapter 6.
- It achieves privacy at a lower computational cost than secure multiparty computation techniques based on cryptographic primitives that are computationally expensive, and
- it does not assume any prior distribution on the databases.

Using existing differential privacy results, it is possible to create high-level structures such as decision trees using multiple low-level differentially private queries [10, 41]. However, a substantial practical problem arises when high-level structures are created this way. Specifically, if an algorithm makes a large number m of such low-level queries, the privacy guarantee for the high-level structure is reduced by a factor of m . Because of this, for many databases and high-level structures, acceptable levels of privacy in the end result via these methods can only be obtained by sacrificing utility in the high-level structure.

In this chapter, we address the problem of constructing differentially private classifiers using decision trees. Our main result is a differentially private classifier using random decision trees. In the rest of this introduction, we place our results in the context of related work and describe our work in more detail.

7.1.1 Related Work

The notion of differential privacy was developed over a sequence of results in privacy-preserving data mining [34, 44, 10, 43]. Much of the work in the differential privacy framework addresses the problem of issuing noisy answers to low-level queries, the so-called interactive mechanisms. That is, a privacy-preserving database access mechanism would compute the correct answer to a client's query, and then add random noise to this result from a carefully chosen distribution. The client receives only the perturbed result. These low-level queries, such as queries that count the number of rows that match a given predicate, can be used in the construction of differentially private data mining algorithms such as those for decision trees [10].

In the work of Dwork et al. [43], the noise added to the client's query depends only on the query and not on the database itself. Nissim et al. [110] considered some query types in

which the methods of [43] would add too much noise. They introduced a new mechanism that lowers the amount of noise added by tailoring it to both the query and the database instance on which it applies. Dwork and Lei [42] showed several examples in which even instance-based noise can be too high. They addressed this issue by presenting a differentially private sensitivity test that allows a mechanism to answer only those queries that have low sensitivity. The mechanism may fail to answer queries that have high sensitivity. Following the work of [43], a number of differentially private algorithms have been developed [97, 18, 84, 102, 126].

Most of the early results in differential privacy involved the analysis of real-valued functions. McSherry and Talwar [103] introduced the exponential mechanism as a technique for differential privacy that applies to more general functions. They showed that the primary technique in [43] follows as a specific instantiation of their general-purpose mechanism. However, the exponential mechanism is not a computationally efficient method.

In contrast with interactive mechanisms, much less work has been done in the area of differentially private non-interactive mechanisms that output “anonymized” versions of their input databases. The earliest substantial result in this area was by Blum et al. [11]. They showed a differentially private mechanism that, for any concept class with small VC-dimension, outputs a (privacy-preserving) database that is simultaneously useful for all queries in the class. Feldman et al. [50] gave a differentially private algorithm for computing k -median coresets for a database. Kasiviswanathan et al. [84] show that any PAC learning algorithm can be converted to a differentially private PAC learning algorithm, although not necessarily one that runs in polynomial time. They also provide a polynomial time algorithm for differentially private learning of parity functions.

7.1.2 Our Contributions

First, we consider the problem of constructing a differentially private decision tree classifier. We first present experimental evidence that creating a differentially private *ID3* tree using low-level differentially private sum queries does not simultaneously provide good privacy and good accuracy. Specifically, we present results from the application of this algorithm to realistic data and observe that in order to obtain a reasonable privacy guarantee, the privacy parameter for each individual noisy sum query needs to be fairly small. Our experiments show that such a differentially private decision tree gives poor prediction for many databases.

Motivated by this poor performance, we instead take an alternate approach. Our main result is an algorithm for differentially private ensemble classifiers. Using random decision trees [47], our algorithm produces classifiers that have good prediction accuracy without compromising privacy, even for small datasets. In contrast to the privacy-preserving decision tree construction methods presented in [94, 2], these decision trees provide provable privacy guarantees about any individual’s contribution to the final result.

We present results from the application of our differentially private random decision tree algorithm to both synthetic and realistic data. Our experiments demonstrate that our approach yields good prediction accuracy even when the size of the datasets are relatively small. We also extend our results to the case where databases that are periodically updated by appending new data and show experimentally that the resulting differentially private random decision tree classifier handles data updates in a way that has small reductions in accuracy while preserving the privacy guarantee.

We begin in Section 7.2 by describing the differential privacy model in more detail and summarizing relevant existing results. In Section 7.3, we consider the use of low-level differentially private sum queries to create differentially private decision trees and motivate

the necessity of considering a new approach. In Section 7.4, we present an overview of random decision trees, which form the basis of our new approach. In Section 7.5, we show how to construct differentially private random decision trees, our main contribution. We present extensions of private random decision tree classifier in settings that involve data update. We present a procedure to update private random decision trees as data updates are made. We also illustrate how to construct private random decision tree classifiers for databases that are distributed between multiple parties. In Section 7.6, we present experimental analysis of our differentially private random decision tree algorithms.

7.2 Preliminaries

The ϵ -*differential privacy* model introduced by Dwork et al. [43] assures that the removal or addition of a single item in a database does not have a substantial impact on the output produced by a private database access mechanism.

Let $\mathcal{D}_1, \dots, \mathcal{D}_k$ denote domains, each of which could be categorical or numeric. Our database D consists of n rows, $\{x_1, x_2, \dots, x_n\}$, where each $x_i \in \mathcal{D}_1 \times \dots \times \mathcal{D}_k$. Two databases D_1 and D_2 *differ in at most one element* if one is a proper subset of the other and the larger database just contains one additional row [41].

Definition 2 ([41]) *A randomized mechanism \mathcal{M} satisfies ϵ -differential privacy if for all databases D_1 and D_2 differing on at most one element, and all $S \in \text{Range}(\mathcal{M})$,*

$$\Pr[\mathcal{M}(D_1) = S] \leq \exp(\epsilon) * \Pr[\mathcal{M}(D_2) = S] \quad (7.1)$$

The probability is taken over the coin tosses in \mathcal{M} .

Note that smaller values of ϵ correspond to higher levels of privacy. Let f be a function on databases with range \mathbb{R}^m . A standard technique by which a mechanism \mathcal{M} that computes

a noisy version of f over a database D can achieve ϵ -differential privacy is to add noise from a suitably chosen distribution to the output $f(D)$. The magnitude of the noise added to the output depends on the sensitivity of f , defined as follows:

Definition 3 ([41]) *The global sensitivity of a function f is the smallest number $S(f)$ such that for all D_1 and D_2 which differ on at most one element,*

$$\|f(D_1) - f(D_2)\|_1 \leq S(f) \quad (7.2)$$

Let $Lap(\lambda)$ denote the Laplacian distribution with mean 0 and standard deviation $\sqrt{2}\lambda$.

The following theorems are proven in [43, 103].

Theorem 3 ([43]) *Let f be a function on databases with range \mathbb{R}^m . Then, the mechanism that outputs $f(D) + (Y_1, \dots, Y_m)$, where Y_i are drawn i.i.d from $Lap(S(f)/\epsilon)$, achieves ϵ -differential privacy.*

Using this method, smaller values of ϵ imply that more noise is added when query results are returned.

Theorem 4 ([103]) *The sequential application of mechanisms \mathcal{M}_i , each giving ϵ_i -differential privacy, gives $\sum_i \epsilon_i$ -differential privacy.*

Theorem 4 implies that differential privacy is robust under composition, but with an additive loss of privacy for each query made.

7.3 ID3 Trees from Private Sum Queries

Most of the work in the differential privacy framework addresses the problem of issuing noisy answers to low-level queries. These low-level queries, such as count queries, can be

used in the construction of differentially private data mining algorithms such as for decision trees [10]. However, a substantial practical problem arises when higher level structures are created using low-level queries as described in [10]. By Theorem 4, if an algorithm makes q such queries each with a privacy parameter ϵ , the overall privacy guarantee of the structure is $\epsilon' = q\epsilon$. In practice, for ϵ' to be reasonable, one must choose ϵ to be fairly small, which increases the amount of noise added to each low-level query. This can have a significant negative impact on the utility of the high-level structure the user wants to compute.

In this chapter, we study the resulting utility and privacy that occurs when low-level noisy queries are used to construct high-level structures. We first examine the construction of differentially private *ID3* decision trees using noisy sum queries. See Algorithm 7 for the original *ID3* algorithm [120]. We obtain a differentially private *ID3* algorithm by replacing accesses to the training set S in the algorithm with equivalent queries to a differentially private interactive mechanism. We list below the affected statements and how they can be modified with queries to a differentially private mechanism for S .

1. The conditions of the first two **If** statements in function *ID3* access the training set S . These can be evaluated by using two queries, each of global sensitivity 1:
 - (a) one query that asks the mechanism to count the number of rows in S , and
 - (b) one query that asks for the count in S of each value of the class attribute C .

Although it is possible to use the sum of the values obtained for the second query as the size of S (thus avoiding the use of the first query), this sum has higher variance than the result for a direct query for the size of S .

2. The return statement in the the third **If** of function *ID3* refers to the most frequently occurring class value in S .

3. The entropy computation in the return statement of the function `Entropy` needs to know the size of S and sizes of its subsets S_j , for $1 \leq j \leq k$. These can be computed using differentially private queries of global sensitivity 1.
4. The weighted entropy computation in the return statement of the function `AttributeEntropy` needs to know the size of S and the sizes of the subsets S_j , for $1 \leq j \leq m$.

Note that each of the subsets of the training set S used in the algorithm can be referenced using some set of attribute-value pairs.

In order to obtain an acceptable privacy guarantee in the final tree, the privacy parameter needs to be fairly small in each noisy sum query used in the computation of information gain. This requires a large amount of noise to be added to each query result, effectively destroying the correlation between the attributes in the database. As we observe experimentally, the resulting tree has poor accuracy.

We implemented the private *ID3* algorithm and ran our experiments on three datasets from the UCI Machine Learning Repository [4]—namely the **Nursery** dataset, the **Congressional Voting Records** dataset and the **Mushroom** dataset. Accuracy results are given in Table 7.1, which is the percentage of test instances that were classified correctly. (In these cases, the vast majority of the test instances were left unclassified by the trees created, though some instances were also misclassified. The inability of the noisy *ID3* tree to produce labels for test instances can probably be explained by the fact that irrelevant attributes were chosen for testing in many internal nodes of the tree.) In all of these datasets, the overall privacy parameter of the *ID3* tree is set to $\epsilon' = 1$. The privacy parameter ϵ for each noisy sum query is given by ϵ'/q , where q is the number of queries made. For example, in the case of the **Mushroom** dataset, the value of ϵ for each noisy query is

Algorithm 7 The *ID3* decision tree learning algorithm.

```

function ID3( R: a set of independent attributes,
                C: the class attribute,
                S: a training set) returns a decision tree;

  begin
    If S is empty, return a single node with value Failure;
    If S consists of records all with the same value for the class attribute,
      return a single node with that value;
    If R is empty,
      return a single node with value the most frequent of the values
      of the class attribute that are found in records of S;
    Let D be the attribute in R with largest InformationGain(D, S, C);
    Let  $\{d_j | j = 1, 2, \dots, m\}$  be the values of attribute D;
    Let  $\{S_j | j = 1, 2, \dots, m\}$  be the subsets of S consisting
      respectively of records with value  $d_j$  for attribute D;
    Return a tree with root labeled D and arcs labeled
       $d_1, d_2, \dots, d_m$  going respectively to the trees
      ID3(R – {D}, C, S1), ID3(R – {D}, C, S2), ..., ID3(R – {D}, C, Sm);
  end ID3;

function InformationGain( D: an attribute,
                          S: a training set,
                          C: the class attribute) returns information gain;

  begin
    Return (Entropy(S, C) – AttributeEntropy(D, S, C))
  end InformationGain

function Entropy( S: a training set,
                  C: the class attribute) returns entropy of training set;

  begin
    Let  $\{c_j | j = 1, 2, \dots, k\}$  be the values of the class attribute C;
    Let  $\{S_j | j = 1, 2, \dots, k\}$  be the subsets of S consisting
      respectively of records with value  $c_j$  for attribute C;
    Return  $\left( - \sum_{j=1}^k \frac{|S_j|}{|S|} \log \frac{|S_j|}{|S|} \right)$ 
  end Entropy

function AttributeEntropy( D: an attribute,
                           S: a training set,
                           C: the class attribute) returns entropy for attribute;

  begin
    Let  $\{d_j | j = 1, 2, \dots, m\}$  be the values of the attribute D;
    Let  $\{S_j | j = 1, 2, \dots, m\}$  be the subsets of S consisting
      respectively of records with value  $d_j$  for attribute D;
    Return  $\left( \sum_{j=1}^m \frac{|S_j|}{|S|} \text{Entropy}(S_j) \right)$ 
  end AttributeEntropy

```

Data set	# rows	# queries	Global Sensitivity	Accuracy original <i>ID3</i>	Accuracy Private <i>ID3</i>
Nursery	12960	8	8	98.19%	33.92%
Cong. Votes	435	16	16	94.48%	1.77%
Mushroom	8124	22	22	100%	5.85%

Table 7.1: Implementing a privacy-preserving version of *ID3* using low-level differentially private queries produces very poor accuracy on many widely used data sets.

approximately 0.045. As can be seen from Table 7.1, for all three datasets the resulting private *ID3* algorithm has poor utility.

To overcome this poor performance, we take a different approach. We construct a differentially private ensemble classifier using the approach of random decision trees [47]. Instead of adding noise to each of the queries made, this approach computes the entire decision tree and adds noise at the end to the leaves. We show that this gives good utility without compromising privacy. We describe this method in detail in Section 7.5 after first introducing random decision trees.

7.4 Random Decision Trees: An Overview

In most machine learning algorithms, the best approximation to the target function is assumed to be the “simplest” classifier that fits the given data, since more complex models tend to overfit the training data and generalize poorly. Ensemble methods such as Boosting and Bagging [127] combine multiple “base” classifiers to obtain new classifiers. It has been observed that ensemble methods can have significantly lower generalization error than any of the base classifiers on which they are based [127]. The base classifiers used in ensemble methods are usually “conventional” classifiers such as decision trees produced by C4.5, which are computationally expensive. The final step of combining these base classifiers can also be computationally intensive.

However, Fan et al. [47] argue that neither of these steps (creating the classifiers and combining them) need be computationally burdensome to obtain classifiers with good performance. They present a fast and scalable ensemble method that performs better than the base classifiers, and frequently as well as the well-known ensemble classifiers. Counterintuitively, their ensemble classifier uses base classifiers that are created from randomly chosen decision trees, in which attributes for decision tree nodes are chosen at random instead of using a carefully defined criterion. The structure of the decision tree (that is, which attributes are in which internal nodes of the decision tree) is determined even before any data is examined. Data is then examined to modify and label the random tree. The end result based on creating an ensemble of random decision trees is an algorithm that scales well for large databases.

The algorithm to build a single random decision tree is shown in Algorithm 8 (adapted from [47]). The algorithm as presented works only for categorical attributes, though it can easily be extended to continuous-valued attributes by choosing random thresholds for the chosen attribute. The algorithm recursively creates the structure of the tree (**BuildTreeStructure**), and then updates the statistics (**UpdateStatistics**, **AddInstance**) at the leaves by “filtering” each training instance through the tree. Each leaf node of the tree holds T counters, $\alpha[1], \dots, \alpha[T]$, where T is the number of possible labels for training instances. After all the examples have been incorporated into the tree, the algorithm prunes away all internal and leaf nodes that did not encounter any of the examples in the training set. The running time of the algorithm is linear in the size of the database.

The random decision tree classifier is an ensemble of such random decision trees. When a test instance needs to be classified, the posterior probability is output as the weighted sum of the the probability outputs from the individual trees (see Algorithm 9, adapted

Algorithm 8 Random Decision Tree (RDT)

Input: D , the training set, and
 X , the set of attributes.
Output: A random decision tree R

```

 $R = \text{BuildTreeStructure}(X)$ 
UpdateStatistics( $R, D$ )
Prune subtrees with zero counts
return  $R$ 

```

Subroutine $\text{BuildTreeStructure}(X)$

```

if  $X = \emptyset$  then
    return a leaf node
else
    Randomly choose an attribute  $F$  as testing attribute
    Create an internal node  $r$  with  $F$  as the attribute
    Assume  $F$  has  $m$  valid values
    for  $i = 1$  to  $m$  do
         $c_i = \text{BuildTreeStructure}(X - \{F\})$ 
        Add  $c_i$  as a child of  $r$ 
    end for
end if
return  $r$ 

```

Subroutine $\text{UpdateStatistics}(r, D)$

```

for each  $x$  in  $D$  do
    AddInstance( $r, x$ )
end for

```

Subroutine $\text{AddInstance}(r, x)$

```

if  $r$  is not a leaf node then
    Let  $F$  be the attribute in  $r$ 
    Let  $c$  represent the child of  $r$  that corresponds to the value of  $F$  in  $x$ 
    AddInstance( $c, x$ )
else
    /*  $r$  is a leaf node */
    Let  $t$  be the label of  $x$ 
    Let  $\alpha[t] = \#$  of  $t$ -labeled rows that reach  $r$ 
     $\alpha[t] \leftarrow \alpha[t] + 1$ 
end if

```

from [47]).

There are two important parameters to be chosen when using this ensemble method, namely (i) the height h of each random tree, and (ii) the number N of base classifiers. Using simple combinatorial reasoning, Fan et al. [47] suggest that a good choice for the height is $h = m/2$, where m denotes the number of attributes. They also find that a value for N as low as 10 gives good results.

Algorithm 9 Classify

Input: $\{R_1, \dots, R_N\}$, an ensemble of RDTs, and
 x , the row to be classified.

Output: Probabilities for all possible labels

For a tree R_i , let ℓ_i be the leaf node reached by x

Let $\alpha_i[t]$ represent the count for label t in ℓ_i

$$P(t|x) = \sum_{i=1}^N \alpha_i[t] / \left(\sum_{\tau} \sum_{i=1}^N \alpha_i[\tau] \right)$$

return probabilities for all t

The advantage of creating a random tree is its training efficiency as well as its minimal memory requirements. The algorithm uses only one pass over the data to create a random decision tree. In a series of papers, Fan et al. [45, 46] show that the random decision tree algorithm is simple, efficient and accurate. They surmise that the reason for the superiority of their ensemble method is that it optimally approximates for each example its true probability of being a member of a given class—that is, the random decision tree ensembles form efficient implementations of Bayes Optimal Classifiers.

7.5 Differentially Private Random Decision Trees

As was discussed in Section 7.4, the structure of a random decision tree is created without examining the data. Only the counters in the leaves of the tree depend on the input

database. This makes the random decision tree algorithm a potentially good candidate for a differentially private mechanism. Because random decision tree construction already contains randomness, it is possible that the construction method might already be differentially private. However, the given randomized decision tree algorithm does not satisfy the requirements of differential privacy because of the way the pruning step is carried out, as demonstrated by the following counterexample. Consider the databases D_1 and D_2 (that differ in at most one element) in Figure 7.1, shown along with an initially empty randomly generated tree structure. The trees R_1 and R_2 that result from the *deterministic* steps of **UpdateStatistics** and pruning are shown in Figure 7.2. The probability of the tree R_1 being generated from database D_2 is zero.

In this section, we present a modified form of the algorithm that does satisfy ϵ -differential privacy.

7.5.1 Private Random Decision Tree Algorithm

We now present an algorithm for creating a differentially private random decision tree. It is a modification of the original random decision tree algorithm. We begin by eliminating the pruning step that removes “empty” tree nodes. The algorithm thus creates a tree in which all of the leaves are at the same level. The leaf nodes of a random decision tree, then, effectively form a *leaf vector* V of $M \cdot T$ integers, where M is the number of leaf nodes and T is the number of possible labels for instances in the training data. This vector of “counts” is updated by the **UpdateStatistics** function. Effectively, releasing a random decision tree amounts to (i) releasing the structure of the tree followed by (ii) this vector of counts. Since the attributes in the tree nodes are chosen completely at random (even before the data is examined), the structure contains no information about the data. (This is

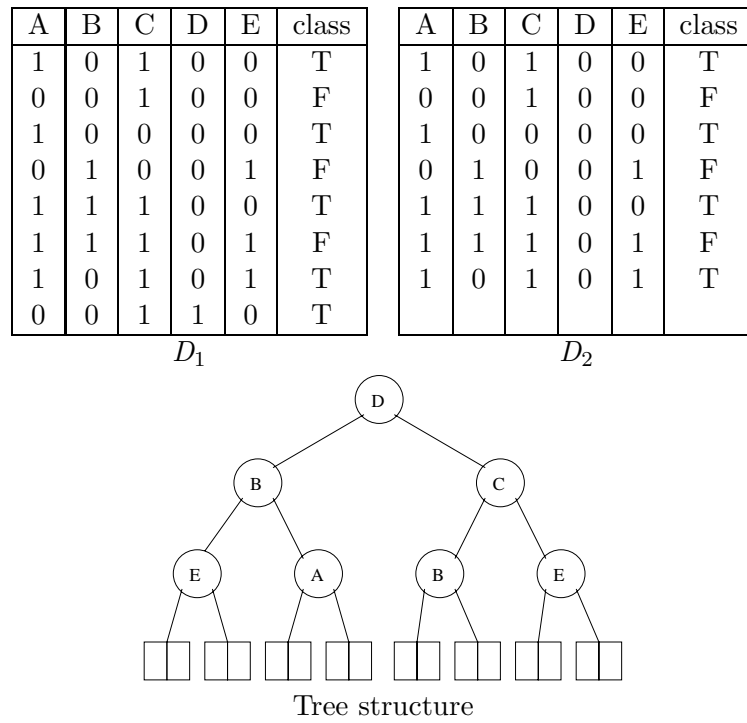


Figure 7.1: RDT Algorithm is not private: Example databases that differ in at most one element and randomly generated tree structure

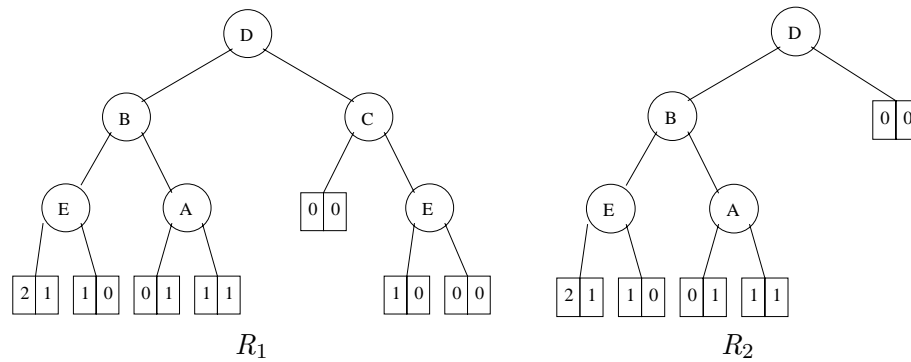


Figure 7.2: RDT Algorithm is not private: Final trees for D_1 and D_2

unlike in conventional decision trees such as *ID3*, where the presence of an attribute in the root indicates its relative predictive capability.) As we show below in Theorem 5, the leaf vector has a global sensitivity of 1. Recall that global sensitivity is defined in Section 7.2. It follows from Theorem 3 that adding $\text{Lap}(\frac{1}{\epsilon})$ noise to each component of V and releasing the resulting noisy vector satisfies ϵ -differential privacy. Note that the values in the leaf vector will no longer be integer counts. The resulting algorithm, shown as Algorithm 10, produces a single differentially private random decision tree. The data owner releases an ensemble of differentially private random decision trees obtained by repeated application of this algorithm.

Algorithm 10 Private-RDT

Input: D , the training set, and
 X , the set of attributes.
Output: A random decision tree R

$R = \text{BuildTreeStructure}(X)$
 $\text{UpdateStatistics}(R, D)$
Add $\text{Lap}(\frac{1}{\epsilon})$ to each component of the leaf vector.
return R

Theorem 5 *The Private-RDT algorithm is ϵ -differentially private.*

Proof Let A denote the **Private-RDT** algorithm. For a database D and a tree R we write $A(D) = R$ to denote the event that algorithm A on input D produces the tree R . It is assumed that the set X of attributes to be use for the tree is implicit in D . For a tree R , we denote the noisy leaf vector of R by $\lambda(R)$.

Consider a fixed random decision tree structure into which no examples have yet been incorporated. Let D_1 and D_2 be two databases differing in at most one element that generate leaf vectors V_1 and V_2 respectively on the tree (before noise is added). The global sensitivity

for the leaf vector of that tree is 1, because V_1 and V_2 must differ in exactly one component by a value of 1.

We need to show that for any tree R the ratio $\frac{P(A(D_1)=R)}{P(A(D_2)=R)}$ is bounded from above by e^ϵ . Since the structure of the random decision tree is generated even before the data is examined, it suffices for us to show that $\frac{P(\lambda(A(D_1))=V)}{P(\lambda(A(D_2))=V)}$ is bounded by e^ϵ , for any leaf vector V . This immediately follows from Theorem 3, taken with the facts that the sensitivity of the noiseless leaf vectors is 1 and the noise added is $\text{Lap}(1/\epsilon)$.

7.5.2 Updating Random Decision Trees

The **Private-RDT** algorithm assumes that all data is available at one time. In the real world data usually arrives in batches. We consider a situation where data is periodically appended to an existing database. The goal is to build a classifier on the combined data each time data gets appended to the existing data. A classifier built on the combined data is likely to be preferred to a classifier built on the new data alone. This kind of *incremental learning* is frequently a challenging problem, even when privacy is not an issue. The solution of rebuilding a classifier from scratch can be a time consuming proposition for large datasets.

In the context of differential privacy, a second problem arises. In order to make use of the composition theorem, the updated classifier must provide a lower privacy guarantee than the initial classifier does. Subsequent updates will worsen the privacy guarantee even further. In this section, we show how private random decision trees can handle data updates in a way that does not lower the privacy guarantee. The tradeoff is a marginal reduction in prediction accuracy when compared with building a random decision tree directly from the combined data.

Let D_1 and D_2 , respectively, represent the old and the new instances of the database.

Let r_1 be a private random decision tree built using D_1 . We present here some possible approaches to update r_1 to include D_2 , and associated problems:

- **(directly incorporate new instances)** One possible option is to incorporate the instances of D_2 into r_1 and then re-release this updated r_1 . However, this can result in a breach of privacy.
- **(recreate leaf vector with $D_1 \cup D_2$)** A second option is to clear out the leaf vector for r_1 , and then run **UpdateStatistics** with $D_1 \cup D_2$. Then add $\text{Lap}(1/\epsilon)$ noise to the leaf vector. By the composition theorem, the resulting private random decision tree will have a privacy guarantee of 2ϵ . In general, k such updates will raise the privacy parameter to $k\epsilon$.
- **(rebuild RDT with $D_1 \cup D_2$)** A third option is to build a new random decision tree from scratch using $D_1 \cup D_2$, and then add noise to the leaf vector. As before, the private guarantee erodes to 2ϵ . For example, if the **Private-RDT** algorithm was run twice with a parameter of $\epsilon = 0.5$, once with the old data alone, and once with the union of the old and new data, the final privacy guarantee would be $\epsilon = 1$.

We can avoid these privacy related issues using the following procedure:

1. Create a clone r'_1 of r_1 and clear out the leaf vector of r'_1 .
2. Use **UpdateStatistics** to insert the rows of D_2 into r'_1 .
3. Add Laplacian noise to the leaf vector of r'_1 .
4. Add the leaf vector of r'_1 to the leaf vector of r_1 and release this updated random decision tree.

Since the leaf vector of r'_1 is based on D_2 alone, and not on any instance of D_1 , the updated

random decision tree continues to satisfy the privacy parameter of ϵ . We present experiments in Section 7.6 to show that the accuracy of r_1 is not substantially reduced after a single update. After a few iterations, the accuracy of the the random decision tree ensemble will be reduced. At that stage, one could build a new ensemble from scratch and then return to more efficient updates.

7.5.3 Private Random Decision Trees for Distributed Databases

In this section, we address the problem of constructing a differentially private random decision tree classifier when the database is distributed among multiple parties. Very little work has done in this area of applying differential privacy in distributed data mining. Beimel et al. [7] examined the idea of combining secure function evaluation and differential privacy. They presented a way of conducting distributed analyses that preserve differential privacy. This work applies to very simple computations such as computing the sum of n distributed bits. Recently, Mironov et al. [104] presented a computational version of differential privacy where privacy guarantees hold only against computationally bounded adversaries. They also presented a differentially private definition of two-party computation and indicated that they have a solution to the two-party Hamming distance problem in a computationally differentially privacy setting. Again, both of the above mentioned work on distributed differential privacy addresses only low-level primitive queries and it is not known how these queries when used to construct a complex structure may lead to the loss of utility.

Private random decision trees on horizontally partitioned data.

A (virtual) database table is horizontally partitioned among a number of parties if each party has a database table with the same set of attributes. The virtual table is the union

of all these tables. We assume that the parties have no rows in common. The parties need to decide on the tree structure ensembles and on ϵ ahead of time.

The solution to this problem is the same as the algorithm for updating random decision trees described in Section 7.5.2. Each update is equivalent to a partition of the database. The experimental results discussed in Section 7.6.2 apply for this scenario. Note that we have presented our experimental results for one update. This implies that those experimental results will hold for two parties. This approach has the following advantages:

- Since each party constructs a classifier from only his or her share of data and publishes it, there is no communication overhead.
- A client who wishes to construct a private classifier can pool the data from publishers of his choice.

Private random decision trees on vertically partitioned data.

A virtual database table is vertically partitioned among a number of parties if each party has data for some fixed set of attributes about individuals that are known to all the parties. The virtual database table is the “join” of all the distributed tables. We consider the case of two parties although the procedure described in this section can be extended to multiple parties. Each party here holds a different set of attributes, except for a common identifier attribute which relates a row in the first database table with a row in the second.

Let D_1 and D_2 denote the vertical partitions of the database D , where D_1 is owned by Alice and D_2 is owned by Bob. Let X_1 denote the set of attributes for the database D_1 and X_2 denote the set of attributes for the database D_2 . Let us assume that both parties have the same class attribute, as is assumed in [40]. We also assume that the new instance to be classified contains the entire set of attributes of the database D . This assumption can be

relaxed with a small loss in utility. The algorithm works as follows:

- Alice computes an ensemble A of random decision trees using algorithm **Private-RDT** with inputs D_1 and X_1 . Alice releases the ensemble A .
- Bob computes an ensemble B of random decision trees using algorithm **Private-RDT** with inputs D_2 and X_2 . Bob releases the ensemble B .
- To classify a new instance run the algorithm **Classify** on the union of the two ensembles A and B .

7.6 Experiments

In this section, we present our experimental results that show that the private random decision tree algorithm achieves good utility in terms of prediction accuracy. We ran three sets of experiments. First, we ran experiments to measure the accuracy of private random decision tree ensembles for various values of the privacy parameter ϵ . Second, we ran experiments to observe the change in the accuracy of random decision tree ensembles when there are batch updates to the data. Third, we ran experiments to measure the utility of the algorithm for vertically partitioned data. We implemented our algorithms in Java using the Weka machine learning framework [140].

7.6.1 Accuracy of Private Random Decision Tree Ensembles

The experiments were run on data sets available from the UCI Machine Learning Repository [4] and from synthetic data that we generated. We restricted ourselves to data sets with only categorical attributes, although extending the implementation to continuous attributes should only take a small amount of additional effort.

Experimental Setup

As noted by Dwork [41], the technique of obtaining differential privacy by adding noise proportional to $\frac{S(f)}{\epsilon}$ yields accurate results only when large data sets are used. For example, consider a histogram query, for which $S(f) = 1$. If the privacy parameter ϵ is set to 0.01, the standard deviation for the Laplacian noise added to each component of the output would be approximately 141, assuming only a single query is made. If q such queries are expected to be made then to use the composition theorem to obtain the same privacy guarantee ϵ the noise added to each query result should be approximately $141q$. If a data set contains only a few hundred or a few thousand rows, this amount of noise would completely overwhelm the underlying “signal.” Since our largest realistic data set contains no more than 13,000 rows, and our largest synthetic data set contains only 16,000 rows, we used larger values of ϵ , namely 0.25, 0.5, 0.75 and 1. As ϵ decreases, the amount of noise to be added must increase, resulting in a decrease in prediction accuracy. The particular choice of ϵ is specific to the application and will be decided by the data owner based on the utility/privacy requirements.

Another important parameter to consider is the number of trees in the ensemble. In the non-private version of random decision trees, increasing the number of trees in the ensemble increases the accuracy of predictions. Our experiments indicate that, for our data sets, using as few as five decision trees in an ensemble produces acceptable accuracy on average. However, the variance in accuracy between runs is higher than is obtained when using more trees. An ensemble with 10 or more trees has better accuracy on average, with lower variance. On the other hand, since one count query is required per random decision tree, creating q trees implies that the per-query privacy parameter needs to be set to $\frac{\epsilon}{q}$. This increases the amount of noise added per query, which negatively impacts prediction accuracy. The **Congressional Voting Records** data set has only 435 rows. Increasing

the number of trees beyond five yielded poor results. For the other data sets, we set the number of trees to 10.

Finally, our experiments indicate that setting the height of the generated random trees to $k/2$, where k is the number of attributes, is not always optimal. Instead, the height we used for a database depended on:

1. the average number of values taken by the attributes of the data set (denoted by b),
2. the number of rows in the database (denoted by n).

Clearly, b is close to the average branching factor of a random decision tree. First, we wanted to ensure that even if the rows were evenly distributed among the leaves of an **RDT**, the leaves would not all be very sparse. Sparse leaves are more susceptible to noise added by the **Private-RDT** algorithm. Hence, it is not advisable to choose a height h such that $b^h \gg n$. If the rows tend to clump together into a small number of leaves, that is usually acceptable, because this would imply that the test set is likely to be similarly distributed. However, if the height is too small there would be too few clumps, and the leaves lose the power of discrimination. The compromise is to choose a value of h close to $\log_b n$. For our experiments, we chose the height $h = \min(\lfloor k/2 \rfloor, (\lfloor \log_b n \rfloor - 1))$.

We performed our experiments on three data sets from the UCI Machine Learning Repository, namely the **Nursery**, **Mushroom** and **Congressional Voting Records** data sets and on three synthetic data sets. Each synthetic data set was generated from a handmade Boolean decision tree. One of these trees is presented in Figure 7.3. We added noise to these data sets by flipping each class label with a probability of 0.05 to make these synthetic data sets more realistic. (This has no bearing on privacy.) See Table 7.2 for data characteristics.

To get a sense of the variation of the prediction accuracies, we present in each case

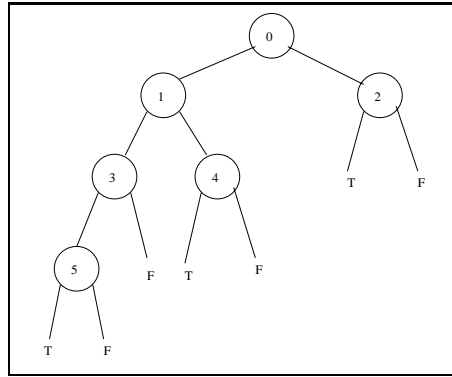


Figure 7.3: The decision tree used to generate synthetic data set 2.

summary statistics over 10 runs of private ensembles of 10 random decision trees (with the exception of the **Congressional Voting Records** data set), for each value of $\epsilon \in \{0.25, 0.5, 0.75, 1\}$. For the sake of consistency, we used the same set of ensembles for every value of ϵ . We also show the prediction accuracy of a non-private implementation of random decision trees (we set ϵ to ∞ and the tree is not pruned). We used trees of height of 4 for **Nursery**, trees of height 5 for **Mushroom**, and trees of height 6 for **Congressional Voting Records**. Prediction accuracy is based on the stratified cross-validation technique available in Weka. The accuracy for each ensemble and each value of ϵ is computed by averaging over 10 runs. We resampled the Laplace Distribution for each run to add noise to the leaf vector. We removed from the **Mushroom** database the attribute that has missing entries. In the **Congressional Voting Records** database, we replaced each missing vote with the majority vote for that bill. We did not compare our results with the standard random decision tree ensemble algorithm with pruning. In a non-private setting, the data owner can release an ensemble with maximal accuracy. But such a release would leak information. In our setting, the data owner releases a random ensemble of random decision trees.

Results

We use box plots to graphically display the five-number summary of the accuracy of **Private-RDT** on each data set and each value of ϵ . The five-number summary includes the observed minimum and maximum accuracies, the lower and upper quartiles and the median. We present in Figures 7.4, 7.5 and 7.6 the box plots of the results of our experiments on the **Nursery**, **Mushroom** and **Congressional Voting Records** data sets. Figures 7.7, 7.8 and 7.9 present the results of our experiments on three synthetic data sets. In almost all experiments on the **Private-RDT** algorithm, whether on realistic data from the UCI Repository or on synthetic data, we see that lower values of ϵ generally result in lowered average accuracy of predictions. This is as expected, since the amount of noise added is inversely proportional to ϵ . However, the drop in average accuracy is gradual and not precipitous.

For the three data sets from the UCI Repository, the reduction in accuracy from $\epsilon = \infty$ to $\epsilon = 1$, though noticeable, is not substantial. The **Mushroom** data set appears to be the least affected by the addition of noise, while the **Congressional Voting Records** data set appears to be the most sensitive. The latter can be partly explained by the smaller size of the data set.

For the three synthetic data sets, just as is the case of the data sets from the UCI repository, there is a gradual reduction in accuracy with the decrease in ϵ . The reduction in privacy appears very slight in relation to the lowering of ϵ . This is because the synthetic data sets are relatively large. A more pronounced reduction can be observed by looking at the minimum accuracy for each value of ϵ .

Overall, as each of these figures show, the private random decision tree ensemble algorithm has good accuracy, even for relatively small data sets.

Data set	# attribs	# rows	# Class labels
Nursery	8	12960	3
Mushroom	22	8124	2
Cong. Votes	16	435	2
Set 1	10	16000	2
Set 2	6	14000	2
Set 3	7	14000	2

Table 7.2: Experimental Data Characteristics

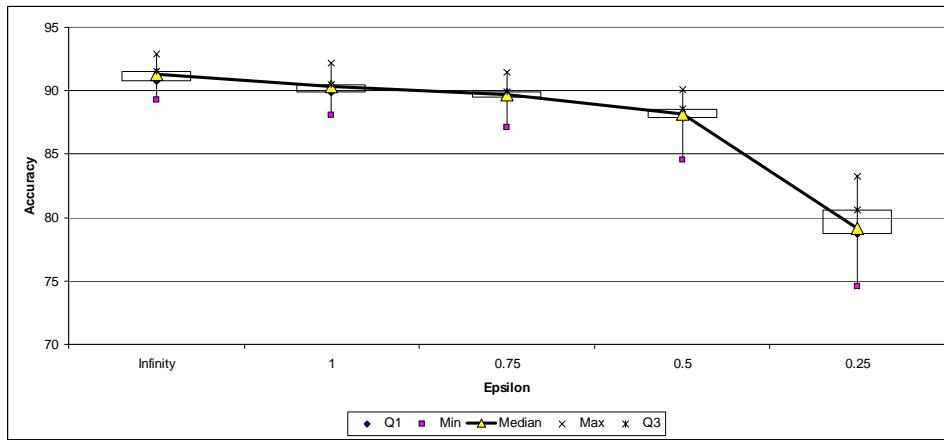
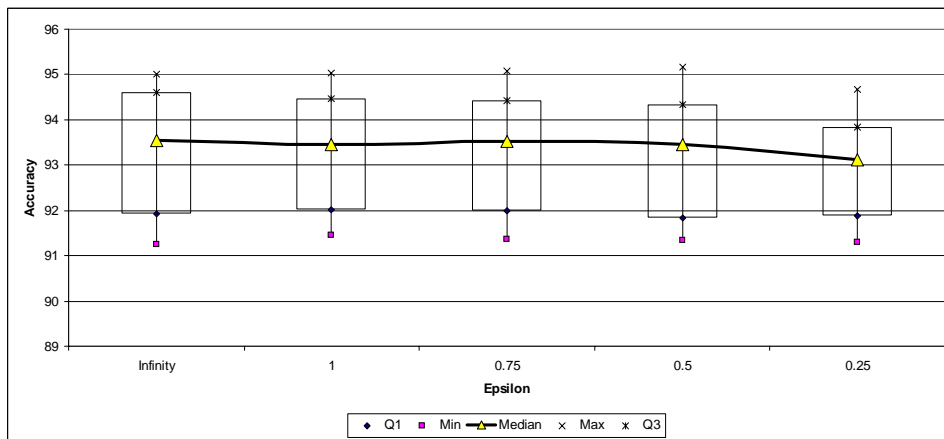
Figure 7.4: Accuracy on the Nursery data set from the UCI Repository. Displayed are the average and maximum accuracy for $\epsilon \in \{0.25, 0.5, 0.75, 1, \infty\}$.

Figure 7.5: Accuracy on the Mushroom data set from the UCI Repository.

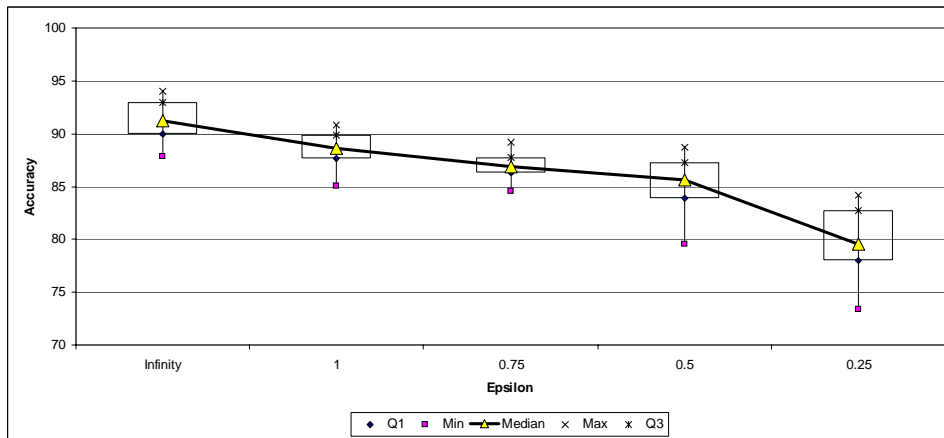


Figure 7.6: Accuracy on the Congressional Voting Records data set from the UCI Repository.

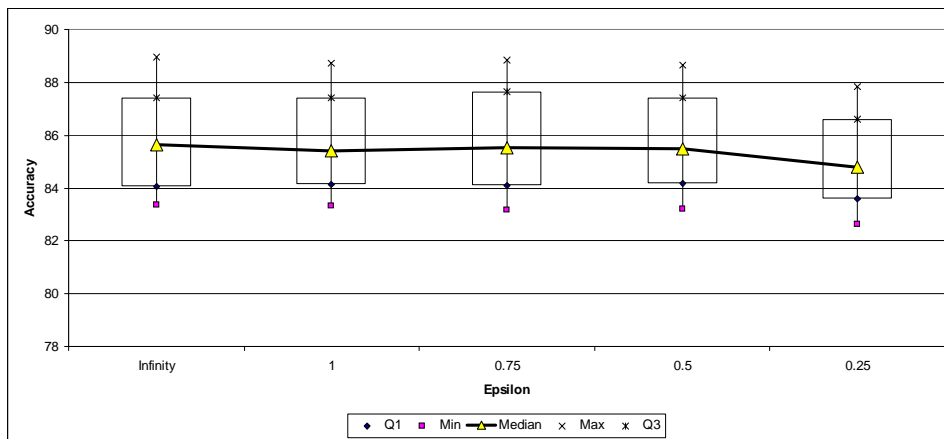


Figure 7.7: Accuracy on synthetic data set 1. The decision tree for this data set (not shown) has 10 attributes.

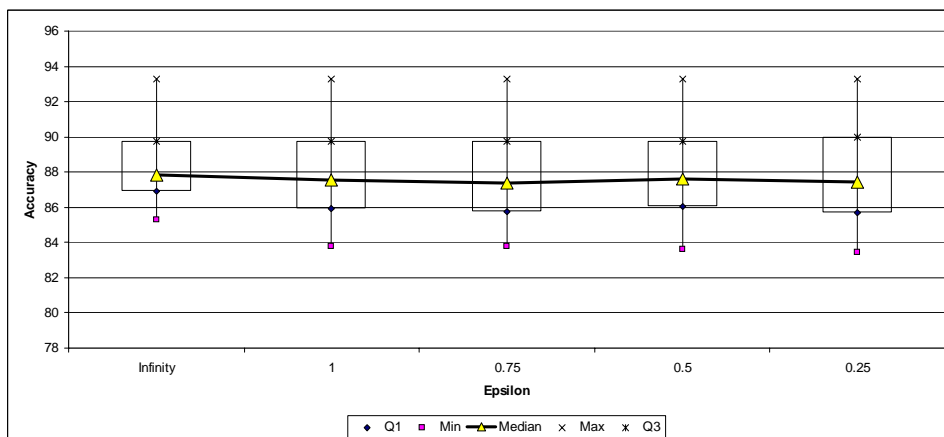


Figure 7.8: Accuracy on synthetic data set 2. See Figure 7.3 for underlying decision tree.

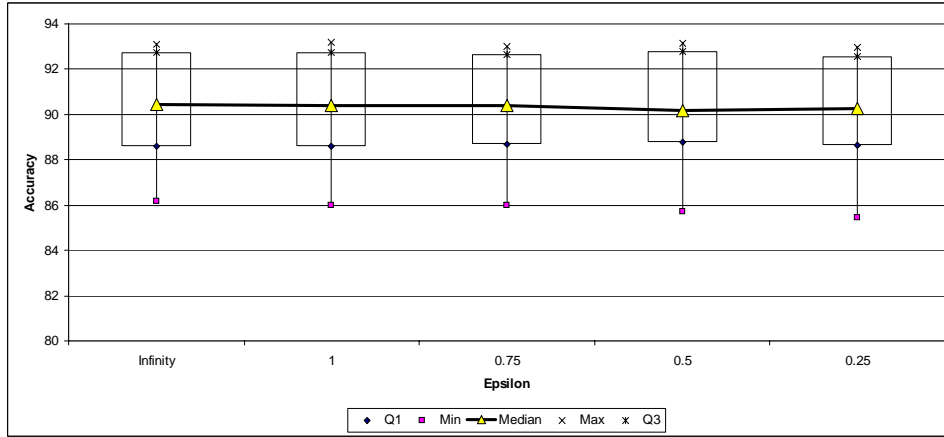


Figure 7.9: Accuracy on synthetic data set 3. The decision tree for this data set (not shown) has 7 attributes.

7.6.2 Updating Random Decision Trees

We ran the algorithm presented in Section 7.5.2 on each of the large data sets from the earlier experiments. We split each data set into two equal halves. The first half was used as D_1 and the second as D_2 . We present the results obtained on **Nursery**, **Mushroom** and on the synthetic data set 1 in Figures 7.10, 7.11 and 7.12 respectively. In each case we compare the accuracy of the ensemble produced by the update algorithm to the ensemble produced by rerunning **Private-RDT** on the union of old and new data.

Our experiments on the private random decision tree update algorithm also show that there is a gradual reduction in accuracy with decreasing values of ϵ . More importantly, the difference in accuracy between this algorithm and the **Private-RDT** algorithm run on the union of old and new data is low. The slightly increased accuracy for some small values of ϵ can be attributed to the fact that we used different ensembles for the various values of ϵ . Running the experiments over a larger set of ensembles will show that accuracy is indeed lowered for smaller values of ϵ .

The value of ϵ for the **Private-RDT** algorithm run on the union of old and new data

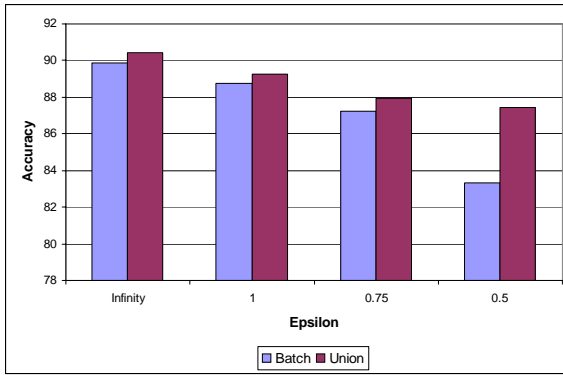


Figure 7.10: Performance of the update algorithm on the Nursery data set.

would be twice the value displayed in Figures 7.10–7.12 because of Theorem 4 (assuming that the data owner had previously released an ensemble based on the old data alone). For example, if the **Private-RDT** algorithm was run twice with $\epsilon = 0.5$, once with the old data alone, and once with the union of the old and new data, the final privacy guarantee would be $\epsilon = 1$. Therefore, to get a fair comparison, one should compare the results for $\epsilon = 1$ for the private random decision tree update algorithm against $\epsilon = 0.5$ for the **Private-RDT** algorithm on the unioned data. The results from the experiments run on the private random decision tree updating algorithm indicate that the algorithm substantially preserves accuracy with the released trees retaining their original (pre-update) levels of privacy.

7.6.3 Private Random Decision Trees on Vertically Partitioned Data

To test the utility of the ensembles produced for vertically partitioned data, we vertically partitioned our datasets by randomly dividing the sets attributes into two disjoint sets. We ran the private RDT algorithm on each partition and computed the union of the ensembles output for each partition. Figure 7.13 presents the accuracy for the **Congressional Voting Records** database.

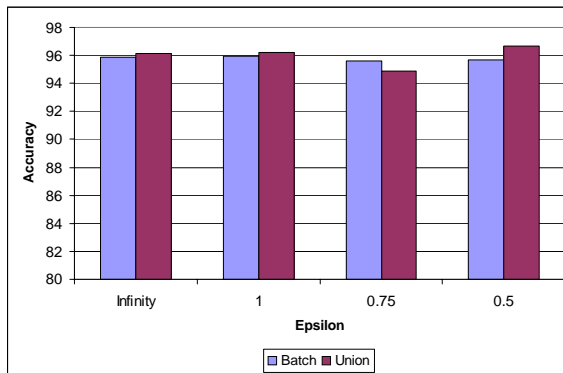


Figure 7.11: Performance of the update algorithm on the Mushroom data set.

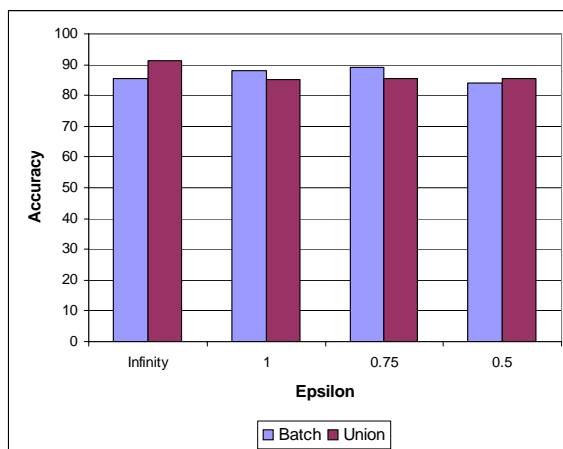


Figure 7.12: Performance of the update algorithm on synthetic data set 1.

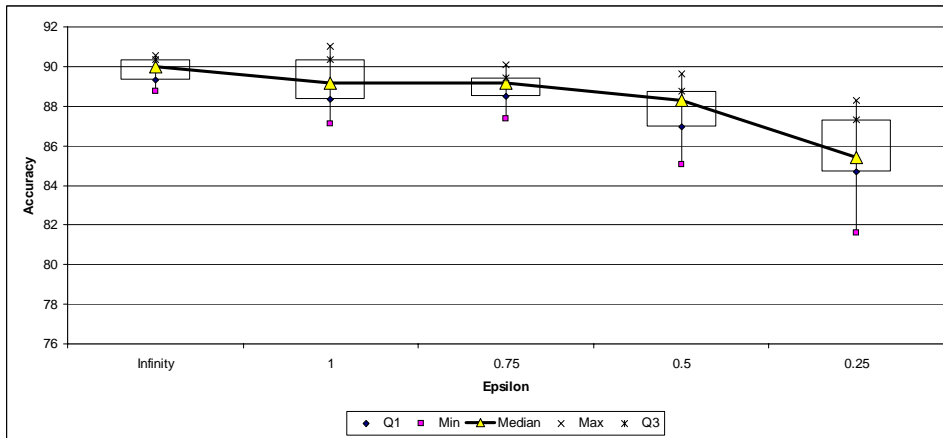


Figure 7.13: Performance of the vertically partitioned database algorithm on randomly partitioned Congressional Voting Records dataset.

7.7 Summary

In this chapter, we presented a differentially private decision tree classifier using the random decision tree approach. We experimentally showed that our approach yields good prediction accuracy even when the size of the database is small. This was possible because the classifier was built from fewer summaries of the database. We also presented a differentially private way of construct a decision tree classifier when new data is periodically appended to an existing database and how to construct a random decision tree classifier when data is horizontally or vertically partitioned between multiple parties. In all of these cases, we experimentally demonstrated that our differentially private random decision tree algorithm produces good utility even for small databases. One possible future direction is to design new machine learning algorithms that need only a small number of summary queries, which could give good accuracy while still maintaining privacy.

Chapter 8

Conclusions

This thesis addresses data privacy in various stages of extracting knowledge embedded in databases. We have addressed the problem of privacy preservation in the two following broad settings: (1) privacy in distributed knowledge discovery, where the data is distributed over multiple sources; and (2) privacy in situations where the output from data mining could itself breach an individual's privacy. Each of the results presented applies to one of two different privacy models, namely, the *secure multiparty computation* (SMC) model and the *differential privacy* model.

Secure multiparty computation involves the collaborative computation of functions based on inputs from multiple parties. The privacy goal is to ensure that all parties receive only the final output without any party learning anything beyond what can be inferred from the output. Within this framework, we addressed the problem of preserving privacy for several problems in the preprocessing and the data mining stages of knowledge discovery in databases. For the preprocessing step, we presented private protocols for the imputation of missing data when the data is shared horizontally between two parties. We introduced the notion of arbitrarily partitioned data as a generalization of both horizontally and vertically partitioned data, and presented a privacy-preserving protocol for k -means clustering of arbitrarily partitioned data. We also presented a new simple k -clustering algorithm that was designed to be converted into a communication-efficient protocol for private clustering.

Even though secure multiparty computation offers the strong privacy guarantee that nothing is leaked beyond what can be inferred from the output, in some cases the output itself might leak information. In the second part of the thesis, we presented privacy-preserving alternatives for data mining algorithms whose outputs themselves breach privacy. In this setting, we presented private inference control protocols for On-line Analytical Processing Systems (OLAP) in the SMC model. We also studied privacy in the data mining step within the framework of differential privacy. In this setting, the goal is to answer queries while preserving the privacy of every individual in the database, irrespective of any auxiliary information that may be available to the database client. Under this privacy model, we constructed a practical differentially private decision tree classifier using random decision trees.

A number of interesting results with privacy and utility guarantees have recently appeared under the differential privacy model. Unfortunately, though theoretically powerful, many of these results appear to be of limited practical utility, especially for constructing complex structures from small- and medium-sized databases. For example, many clinical research databases tend to be relatively small, and these algorithms can be expected to be of little value in such domains. However, there have been a few results in differential privacy that have demonstrably preserved both privacy and utility. An interesting line of work will be in designing general-purpose machine learning algorithms that need a small number of summary queries. The objective is to be able to argue that these summaries can be privatized by the addition of only a small amount of noise. This may lead to differentially private data mining algorithms that are practical and effective for even small-sized databases.

Most models for privacy, irrespective of their strengths, share a common weakness—they assume a “one size fits all approach” to all data. This absence of fine-grained control can impose a substantial limitation in the utility of privacy preserving algorithms. Here are some issues to consider in designing a new privacy model and in creating new privacy infrastructure:

1. Not everyone values privacy the same way. Each individual should be able to set their own privacy threshold. For instance, a person may be willing to accept reduced privacy for their data in exchange for monetary compensation.
2. Not every attribute of an individual is equally private. A person might highly value the privacy of their annual income, but may be less reluctant to reveal their employer.
3. Not every data consumer is equally close to an individual. A person might have lower privacy requirements if the data consumer is their employer, higher privacy requirements if the consumer is a local retailer, and the highest privacy requirement for everyone else.
4. Not every use of private data is the same. A person might be willing to reveal her annual income if it is to be used to compute the average income in her neighborhood, but unwilling to reveal it if it is to be used to find the highest earners in her neighborhood.

Over time, new models for privacy may replace existing ones. But the concept of, and the need for, privacy and privacy research will be longstanding issues.

References

- [1] N. R. Adam and J. C. Worthmann. Security-control methods for statistical databases: A comparative study. *ACM Comput. Surv.*, 21(4):515–556, 1989.
- [2] R. Agrawal and R. Srikant. Privacy-preserving data mining. volume 29, pages 439–450, New York, NY, USA, 2000. ACM.
- [3] W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135, London, UK, 2001. Springer-Verlag.
- [4] A. Asuncion and D. Newman. UCI machine learning repository, 2007.
- [5] G. E. A. P. A. Batista and M. C. Monard. An analysis of four missing data treatment methods for supervised learning. *Applied Artificial Intelligence*, 17(5):519–533, 2003.
- [6] J. Beaumont. On regression imputation in the presence of nonignorable nonresponse. In *Proc. Survey Research Methods Section*, pages 580–585. ASA, 2000.
- [7] A. Beimel, K. Nissim, and E. Omri. Distributed private data analysis: Simultaneously solving how and what. In *CRYPTO 2008: Proceedings of the 28th Annual conference on Cryptology*, pages 451–468, Berlin, Heidelberg, 2008. Springer-Verlag.
- [8] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1988. ACM.
- [9] S. Benninga and B. Czaczkes. *Financial Modelling*. MIT Press, 1997.
- [10] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: The SuLQ framework. In *PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 128–138, 2005.
- [11] A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 609–618, 2008.
- [12] D. Boneh, E. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Second Theory of Cryptography Conference, TCC*, volume 3378 of *LNCS*, pages 325–341, 2005.
- [13] L. Brankovic and H. Giggins. *Security, Privacy, and Trust in Modern Data Management*, pages 167–181. Springer Berlin Heidelberg, 2007.

- [14] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Chapman and Hall, 1984.
- [15] J. Brickell and V. Shmatikov. The cost of privacy: Destruction of data-mining utility in anonymized data publishing. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 70–78, 2008.
- [16] P. Bunn and R. Ostrovsky. Secure two-party k -means clustering. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 486–497, New York, NY, USA, 2007. ACM.
- [17] R. Canetti, Y. Ishai, R. Kumar, M. K. Reiter, R. Rubinfeld, and R. N. Wright. Selective private function evaluation with applications to private statistics. In *PODC '01: Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 293–304, New York, NY, USA, 2001. ACM Press.
- [18] K. Chaudhuri and C. Monteleoni. Privacy-preserving logistic regression. In *NIPS '08: Proc. Neural Information Processing Systems Foundations*, pages 289–296, 2008.
- [19] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, New York, NY, USA, 1988. ACM.
- [20] J. Chen and J. Shao. Nearest neighborhood imputation for survey data. *J. Official Statistics*, 16(2):113–131, 2000.
- [21] F. Chin. Security in statistical databases for queries with small counts. *ACM Trans. Database Syst.*, 3(1):92–104, 1978.
- [22] F. Chin. Security problems on inference control for SUM, MAX, and MIN queries. *J. ACM*, 33(3):451–464, 1986.
- [23] F. Chin and G. Ozsoyoglu. Statistical database design. *ACM Trans. Database Syst.*, 6(1):113–139, 1981.
- [24] F. Chin and G. Ozsoyoglu. Auditing and inference control in statistical databases. *IEEE Trans. Softw. Eng.*, 8(6):574–582, 1982.
- [25] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
- [26] P. Clark and T. Niblett. The CN2 induction algorithm. *Mach. Learn.*, 3(4):261–283, 1989.
- [27] L. Coppola, M. D. Zio, O. Luzi, A. Ponti, and M. Scanu. Bayesian networks for imputation in official statistics: A case study. In *DataClean Conference*, pages 30–31, 2000.
- [28] I. Dångard and M. Jurik. A generalisation, a simplification and some applications of Paillier’s probabilistic public-key system. In *PKC '01: Proceedings of the Fourth International Workshop on Practice and Theory in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, London, UK, 2001. Springer-Verlag.

- [29] I. Davidson and S. S. Ravi. Distributed pre-processing of data on networks of Berkeley Motes using non-parametric EM. In *Proc. of SIAM SDM Workshop on Data Mining in Sensor Networks*, pages 17–27, 2005.
- [30] R. R. de Carvalho, S. G. Djorgovski, N. Weir, U. Fayyad, J. Roden, A. Gray, and K. Cherkauer. Applications of clustering analysis and unsupervised classification algorithms to digitized POSS-II. *Bulletin of the American Astronomical Society*, 26:1372, Dec. 1994.
- [31] D. Denning and J. Schlörer. A fast procedure for finding a tracker in a statistical database. *ACM Trans. Database Syst.*, 5(1):88–102, 1980.
- [32] D. E. Denning. Secure statistical databases with random sample queries. *ACM Trans. Database Syst.*, 5(3):291–315, 1980.
- [33] A. DeWaal and L. Willenborg. Global recodings and local suppressions in microdata sets. *Journal of Official Statistics*, 95:121–132, 1995.
- [34] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210, 2003.
- [35] D. Dobkin, A. Jones, and R. Lipton. Secure databases: Protection against user influence. *ACM Trans. Database Syst.*, 4(1):97–106, 1979.
- [36] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *STOC '91: Proceedings of the 23rd Annual ACM symposium on Theory of computing*, pages 542–552, New York, NY, USA, 1991. ACM Press.
- [37] J. Domingo-Ferrer and J. Mateo-Sanz. Practical data-oriented microaggregation for statistical disclosure control. *IEEE Trans. on Knowl. and Data Eng.*, 14(1):189–201, 2002.
- [38] W. Du and M. Atallah. Privacy-preserving cooperative statistical analysis. In *AC-SAC '01: Proceedings of the 17th Annual Computer Security Applications Conference*, pages 102–112, 2001.
- [39] W. Du, Y. Han, and S. Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proc. of the Fourth SIAM International Conference on Data Mining*, pages 222–233, 2004.
- [40] W. Du and Z. Zhan. Building decision tree classifier on private data. In *Proceedings of the IEEE International Conference on Privacy, Security and Data Mining*, pages 1–8. Australian Computer Society, Inc., 2002.
- [41] C. Dwork. Differential privacy: A survey of results. In *TAMC : Theory and Applications of Models of Computation, 5th International Conference*, pages 1–19, 2008.
- [42] C. Dwork and J. Lei. Differential privacy and robust statistics. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 371–380, 2009.

- [43] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In S. Halevi and T. Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006.
- [44] C. Dwork and K. Nissim. Privacy-preserving datamining on vertically partitioned databases. In *CRYPTO*, pages 528–544, 2004.
- [45] W. Fan. On the optimality of probability estimation by random decision trees. In *AAAI'04: Proceedings of the 19th national conference on Artificial intelligence*, pages 336–341. AAAI Press / The MIT Press, 2004.
- [46] W. Fan, E. Greengrass, J. McCloskey, P. S. Yu, and K. Drummey. Effective estimation of posterior probabilities: Explaining the accuracy of randomized decision tree approaches. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 154–161, Washington, DC, USA, 2005. IEEE Computer Society.
- [47] W. Fan, H. Wang, P. Yu, and S. Ma. Is random model better? On its accuracy and efficiency. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 51, 2003.
- [48] A. Farhangfar, L. Kurgan, and W. Pedrycz. Experimental analysis of methods for handling missing values in databases. In *Intelligent Computing: Th. and Appl. II*, 2004.
- [49] C. Farkas and S. Jajodia. The inference problem: A survey. *SIGKDD Explor. Newsl.*, 4(2):6–11, 2002.
- [50] D. Feldman, A. Fiat, H. Kaplan, and K. Nissim. Private coresets. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 361–370, 2009.
- [51] B. L. Ford. *Incomplete data in sample surveys*. Academic Press, 1983.
- [52] E. Forgey. Cluster analysis of multivariate data: Efficiency vs. interpretability of classification. *Biometrics*, 21:768, 1965.
- [53] M. Freedman, K. Nissim, and B. 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, London, UK, 2004. Springer-Verlag.
- [54] A. Friedman, R. Wolff, and A. Schuster. Providing k -anonymity in data mining. *The VLDB Journal*, 17(4):789–804, 2008.
- [55] J. H. Friedman, R. Kohavi, and Y. Yun. Lazy decision trees. In *Proc. of 13th AAAI and 8th IAAI*, pages 717–724, 1996.
- [56] F. Fuller. Masking procedures for microdata disclosure limitation. *Journal of Official Statistics*, 9:383–406, 1993.
- [57] S. Ganta, S. Kasiviswanathan, and A. Smith. Composition attacks and auxiliary information in data privacy. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, year = 2008.*,

- [58] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 169–178, 2009.
- [59] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *STOC '98: Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 151–160, New York, NY, USA, 1998. ACM Press.
- [60] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen. On secure scalar product computation for privacy-preserving data mining. In *Proceedings of the 7th Annual International Conference in Information Security and Cryptology*, 2004.
- [61] O. Goldreich. *Foundations of Cryptography, Vol II*. Cambridge University Press, 2004.
- [62] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.
- [63] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003.
- [64] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 359–366, 2000.
- [65] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 73–84, New York, NY, USA, 1998. ACM Press.
- [66] C. Gupta and R. L. Grossman. Genic: A single-pass generalized incremental algorithm for clustering. In *SDM 04: SIAM Int. Conf. on Data Mining*, 2004.
- [67] D. Gusfield. *Algorithms on Strings Trees and Strings*. Cambridge University Press, 1997.
- [68] M. Halatchev and L. Gruenwald. Estimating missing values in related sensor data streams. In *Proc. of the Eleventh International Conference on Management of Data*, pages 83–94, 2005.
- [69] J. Han and M. Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann Publishers Inc., 2000.
- [70] J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, Inc., 1975.
- [71] T. Hsu and M. Kao. Security problems for statistical databases with general cell suppressions. In *SSDBM '97: Proceedings of the Ninth International Conference on Scientific and Statistical Database Management*, pages 155–164, 1997.

- [72] M. Hu, S. Salvucci, and M. Cohen. Evaluation of some popular imputation algorithms. In *The Survey Research Methods Section of the ASA*, pages 308–313, 1998.
- [73] A. Hundepool. The CASC project. pages 172–180, 2002.
- [74] G. Jagannathan, K. Pillaipakkamnatt, and R. N. Wright. A practical differentially private random decision tree classifier. In *ICDMW '09: Proceedings of the 2009 IEEE International Conference on Data Mining Workshops*, pages 114–121, Washington, DC, USA, 2009. IEEE Computer Society.
- [75] G. Jagannathan, K. Pillaipakkamnatt, R. N. Wright, and D. Umamo. Communication-efficient privacy-preserving clustering. *Trans. Data Privacy*, 3(1):1–25, 2010.
- [76] G. Jagannathan and R. N. Wright. Privacy-preserving distributed k -means clustering over arbitrarily partitioned data. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 593–599, New York, NY, USA, 2005. ACM.
- [77] G. Jagannathan and R. N. Wright. Private inference control for aggregate database queries. In *ICDMW '07: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, pages 711–716, Washington, DC, USA, 2007. IEEE Computer Society.
- [78] G. Jagannathan and R. N. Wright. Privacy-preserving imputation of missing data. *Data Knowl. Eng.*, 65(1):40–56, 2008.
- [79] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999.
- [80] S. Jha, L. Kruger, and P. McDaniel. Privacy preserving clustering. In S. D. C. di Vimercati, P. F. Syverson, and D. Gollmann, editors, *ESORICS*, volume 3679 of *Lecture Notes in Computer Science*, pages 397–417. Springer, 2005.
- [81] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *Proc. ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02)*, pages 24–31, June 2002.
- [82] M. Kantarcioglu and J. Vaidya. Privacy preserving naive bayes classifier for horizontally partitioned data. In *Workshop on Privacy Preserving Data Mining held in association with The Third IEEE International Conference on Data Mining*, Melbourne, FL, Nov.19-22 2003.
- [83] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 99, Washington, DC, USA, 2003. IEEE Computer Society.
- [84] S. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? In *FOCS '08: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 531–540, Oct. 2008.

- [85] KDD. The third international knowledge discovery and data mining tools competition dataset (KDD99 cup). <http://kdd.ics.uci.edu/databases/kddcup99.html>, 1999.
- [86] K. Kenthapadi, N. Mishra, and K. Nissim. Simulatable auditing. In *PODC '05: Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing*, 2005.
- [87] J. Kim. A Method for Limiting Disclosure in Microdata based on Random Noise and Transformation. *Proceedings of the Section on Survey Research Methods*, pages 303–308, 1986.
- [88] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 3rd ed.* Addison-Wesley, 1998.
- [89] T. D. W. L. Willenborg. *Elements of Statistical Disclosure Control*. Springer, 2001.
- [90] K. Lakshminarayan, S. A. Harp, and T. Samad. Imputation of missing data in industrial databases. *Applied Intelligence*, 11(3):259–275, 1999.
- [91] R. C. T. Lee, J. R. Slagle, and C. T. Mong. Towards automatic auditing of records. *IEEE Trans. Software Eng.*, 4(5):441–448, 1978.
- [92] N. Li, T. Li, and S. Venkatasubramanian. t -closeness: Privacy beyond k -anonymity and l -diversity. In *ICDE '07 : Proc. of IEEE 23rd Intl Conf. on Data Engineering*, pages 106–115, 2007.
- [93] X. Lin, C. Clifton, and M. Zhu. Privacy-preserving clustering with distributed EM mixture modelling. *Knowl. Inf. Syst.*, 8(1):68–81, 2005.
- [94] Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002.
- [95] R. J. A. Little and D. B. Rubin. *Statistical analysis with missing data*. John Wiley & Sons, Inc., USA, 1986.
- [96] S. P. Lloyd. Least squares quantization in PCM. *IEEE Trans. on Info. Theory*, 28:129–137, 1982.
- [97] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 277–286, Washington, DC, USA, 2008. IEEE Computer Society.
- [98] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkatasubramanian. l -diversity: Privacy beyond k -anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3, 2007.
- [99] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–296, 1967.
- [100] R. Mattison. *Data Warehousing and Data Mining for Telecommunication*. Artech Press, 1997.

- [101] T. S. Mayer. Research report series (survey methodology 2002-01). In *Washington DC, U.S. Census Bureau, Statistical Research Division 2002*, 2002.
- [102] F. McSherry and I. Mironov. Differentially private recommender systems: Building privacy into the net. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 627–636, New York, NY, USA, 2009. ACM.
- [103] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS '07: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 94–103, 2007.
- [104] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan. Computational differential privacy. In *CRYPTO '09: Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, pages 126–142, Berlin, Heidelberg, 2009. Springer-Verlag.
- [105] H. Nagesh, S. Goil, and A. Choudhary. Adaptive grids for clustering massive data sets. In *In 1st SIAM International Conference Proceedings on Data Mining*, 2001.
- [106] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 590–599, 2001.
- [107] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *STOC '99: Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 245–254, New York, NY, USA, 1999. ACM Press.
- [108] NASCIO. Think before you dig: Privacy implications of data mining and aggregation, September 2004.
- [109] R. T. Ng and J. Han. CLARANS: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.
- [110] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84, New York, NY, USA, 2007. ACM.
- [111] S. Oliveira and O. R. Zaiane. Privacy preserving clustering by data transformation. In *Proc. of the 18th Brazilian Symposium on Databases*, pages 304–318, 2003.
- [112] R. Ostrovsky, Y. Rabani, L. J. Schulman, and C. Swamy. The effectiveness of Lloyd-type methods for the k -means problem. In *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 165–176, Washington, DC, USA, 2006. IEEE Computer Society.
- [113] J. T. P. Doyle, J. Lane and L. Zayatz. *Confidentiality, Disclosure, and Data Access. Theory and Practical Applications for Statistical Agencies*. Elsevier Science, 2001.
- [114] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *EUROCRYPT 99*, volume 1592 of *LNCS*, pages 223–238. Springer-Verlag, 1999.

- [115] B. Park and H. Kargupta. Distributed data mining: Algorithms, systems, and applications. In *The Handbook of Data Mining*, edited by N. Ye, pages 341–358, 2003.
- [116] P. K. Prasad and C. P. Rangan. Privacy preserving BIRCH algorithm for clustering over arbitrarily partitioned databases. In *ADMA '07: Proceedings of the 3rd international conference on Advanced Data Mining and Applications*, pages 146–157, Berlin, Heidelberg, 2007. Springer-Verlag.
- [117] X. Qian, M. E. Stickel, P. D. Karp, T. F. Lunt, and T. D. Carvey. Detection and elimination of inference channels in multilevel relational database systems. In *SP '93: Proceedings of the 1993 IEEE Symposium on Security and Privacy*, pages 196–205, Washington, DC, USA, 1993. IEEE Computer Society.
- [118] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
- [119] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
- [120] J. R. Quinlan. *Readings in knowledge acquisition and learning: automating the construction and improvement of expert systems*, pages 349–361. Morgan Kaufmann Publishers Inc., USA, 1993.
- [121] M. O. Rabin. How to exchange secrets by oblivious transfer. *Technical Report TR-81, Aiken Computation Laboratory, Harvard University*, 1981.
- [122] M. J. Rosenberg. Multivariable analysis by a randomized response technique for disclosure control. *Ph.D thesis, Univ of Michigan*, 1979.
- [123] J. Sakuma and S. Kobayashi. Large-scale k-means clustering with user-centric privacy preservation. In T. Washio, E. Suzuki, K. M. Ting, and A. Inokuchi, editors, *PAKDD*, volume 5012 of *Lecture Notes in Computer Science*, pages 320–332. Springer, 2008.
- [124] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: *k*-anonymity and its enforcement through generalization and suppression. Technical Report SRI-CSL-98-04, SRI Computer Science Laboratory, 1998.
- [125] W. Sarle. Prediction with missing inputs. In *Proc. of the Fourth Joint Conference on Information Sciences*, pages 399–402, 1998.
- [126] A. D. Sarwate, K. Chaudhuri, and C. Monteleoni. Differentially private support vector machines. *CoRR*, abs/0912.0071, 2009.
- [127] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.
- [128] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
- [129] A. Shoshani. Olap and statistical databases: Similarities and differences. In *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 185–196, New York, NY, USA, 1997. ACM.

- [130] F. D. Smet, J. Mathys, K. Marchal, G. Thijs, B. D. Moor, and Y. Moreau. Adaptive quality-based clustering of gene expression profiles. *Bioinformatics*, 18(5):735–746, 2002.
- [131] L. Sweeney. k -anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.
- [132] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 639–644. ACM Press, 2002.
- [133] J. Vaidya and C. Clifton. Privacy-preserving k -means clustering over vertically partitioned data. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery in data mining*, 2003.
- [134] J. Vaidya and C. Clifton. Privacy-preserving decision trees over vertically partitioned data. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM Press, 2004.
- [135] J. Vaidya and C. Clifton. Privacy-preserving naive Bayes classifier for vertically partitioned data. In *Proc. of the 4th SIAM International Conference on Data Mining*, pages 522–526, 2004.
- [136] O. Veksler. Image segmentation by nested cuts. In *Proc. of IEEE Computer Vision and Pattern Recognition*, pages 339–344, 2000.
- [137] J. H. Ward. Hierarchical grouping to optimize an objective function. *J. Amer. Stat. Assoc.*, 58(2):236–244, 1963.
- [138] S. Warner. The linear randomized response model. *Journal of the American Statistical Association*, 66:884–888, 1965.
- [139] S. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.
- [140] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.
- [141] D. Woodruff and J. Staddon. Private inference control. In *CCS '04: Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 188–197, New York, NY, USA, 2004. ACM Press.
- [142] Z. Yang and R. N. Wright. Privacy-preserving computation of Bayesian networks on vertically partitioned data. *IEEE Trans. on Knowl. and Data Eng.*, 18(9):1253–1264, 2006.
- [143] Z. Yang, R. N. Wright, and H. Subramaniam. Experimental analysis of a privacy-preserving scalar product protocol. *Comput. Syst. Sci. Eng.*, 21(1), 2006.
- [144] A. C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

- [145] A. C. Yao. How to generate and exchange secrets. In *FOCS '86: Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.
- [146] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.

Curriculum Vita

Geetha Jagannathan

Education

- Ph.D. in Computer Science, Rutgers University, New Brunswick, Spring 2010.
- M.S. in Computer Science, Stony Brook University, Stony Brook, NY, Fall 2003.
- Ph.D. in Mathematics, Indian Institute of Technology, Madras, May 1994.
- M.S. in Mathematics, Indian Institute of Technology, Madras, May 1990.
- B.S. in Mathematics, University of Madras, India, May 1988.

Principal Occupations

09/2007—01/2020	<i>Graduate Assistant</i> , Rutgers University, NJ.
01/2004—08/2007	<i>Research Assistant</i> , Stevens Institute of Technology, NJ.
09/2001—12/2002	<i>Teaching Assistant</i> , Stony Brook University, Stony Brook, NY.
10/2000—06/2001	<i>Senior Developer</i> , RightFreight, Inc., New York, NY.
08/1999—10/2000	<i>Postdoctoral Researcher</i> , Department of Physics, Hofstra University, NY.
08/1998—03/1999	<i>Assistant Professor</i> , Indian Institute of Technology, Madras.
08/1997—08/1998	<i>Research Scholar</i> , Chennai Mathematical Institute, India.
08/1994—08/1997	<i>Lecturer</i> , Sri Venkateswara College of Engineering, India.

Publications

1. Anonymizing Databases for Regression, with K. Pillaipakkamnatt and R.N. Wright. In preparation.
2. A Practical Differentially Private Random Decision Tree Classifier, with K. Pillaipakkamnatt and R.N. Wright. Proceedings of the ICDM International Workshop on Privacy Aspects of Data Mining, 2009. Invited to appear as a journal paper in Transactions on Data Privacy.
3. Communication-Efficient Privacy-Preserving Clustering, with K. Pillaipakkamnatt, D. Umamo and R.N. Wright. Trans. Data Privacy 3 (1): 1–25 (2010)
4. Privacy-preserving imputation of missing data, with R.N. Wright. Data and Knowledge Engineering 65(1): 40-56 (2008)

5. A Secure Clustering Algorithm for Distributed Data Streams, with K. Pillaipakkamnatt and D. Umano, Proceedings of the ICDM International Workshop on Privacy Aspects of Data Mining, 2007.
6. Private Inference Control For Aggregate Database Queries, with R. N. Wright, Proceedings of the ICDM International Workshop on Privacy Aspects of Data Mining, 2007.
7. Sum-of-squares heuristics for bin packing and memory allocation, with M.A. Bender, B. Bradley, and K. Pillaipakkamnatt. ACM Journal of Experimental Algorithmics 12: (2007)
8. Privacy-Preserving Data Imputation, with R.N.Wright, Proceedings of the ICDM International Workshop on Privacy Aspects of Data Mining, 2006.
9. A New Privacy-Preserving Distributed k-Clustering Algorithm, with K. Pillaipakkamnatt and R. N. Wright, Proceedings of the 2006 SIAM International Conference on Data Mining, 2006.
10. Privacy-Preserving Distributed k-Means Clustering over Arbitrarily Partitioned Data, with R. N. Wright, Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2005.
11. The Robustness of the Sum-of-Squares Algorithm for Bin Packing, with M.Bender,et al. ALENEX/ANALC 2004: 18-30.
12. Alternans and the onset of ventricular fibrillation, with Harold M. Hastings, et al. Physical Review E. Volume 62, 2000, pp 4043-4048.
13. One the Image System of Certain Line Singularities in the Vicinity of a Circular Cylinder, with A. Avudainayagam. MechanicsResearchCommunications. Volume25,1998,pp 25-32.
14. A Boundary Integral Equation Formulation for the Two Dimensional Oscillating Stokes Flow Past an Arbitrary Body, with A. Avudainayagam. Journal of Engineering Mathematics. Volume 33, 1998, pp 251-258.
15. A Necessary Condition for the Existence of Plane Stokes Flows Around An Ellipse, with A. Avudainayagam. Canadian Applied MathematicsQuarterly. Volume 3, 1995, pp 237-251.
16. Oscillating Line Singularities of Stokes Flows, with A. Avudainayagam. International Journal of Engineering Science. Volume 31, 1995, pp 1295-1299
17. Unsteady Singularities of Stokes Flows in Two Dimensions, with A. Avudainayagam. International Journal ofEngineering Science. Volume 33, 1995, pp 1713-1724.
18. Oscillating Stokes Flows in Two Dimensions, with A. Avudainayagam. Mechanics Research Communications. Volume21, 1994, pp 617-628.