Distributed Protocols for Data Privacy

Zhiqiang Yang

Department of Computer Science

Stevens Institute of Technology

Castle Point on Hudson

Hoboken, New Jersey 07030, USA

zyang@cs.stevens.edu

Committee Members:

Joan Feigenbaum, Computer Science Department, Yale University

Charles Suffel, Mathematics Department, Stevens Institute of Technology

Susanne Wetzel, Computer Science Department, Stevens Institute of Technology

Rebecca Wright, Computer Science Department, Stevens Institute of Technology

July 3, 2007

Distributed Protocols for Data Privacy

Abstract

With the rapid development of the Internet and computer technology, more and more of our activities are carried out on the Internet. Consequently, more and more data related to individuals are collected and used by different parties who are generally distributed over a wide variety of sites. Therefore, the protection of data privacy in such distributed settings is drawing more attention than ever.

In this thesis, we present five techniques for protecting data privacy in different distributed settings by using cryptographic tools. For each of our techniques, we formally give an appropriate definition of data privacy. We also give thorough analysis to show that data privacy is protected properly under certain settings.

We have considered the distributed setting where a data miner or collector wants to collect data or learn models from (potentially) large amount of online respondents, and we have designed three privacy-preserving techniques in this setting. Our first technique enables the data miner to learn certain classification models from respondents' data without even seeing those data. Assuming there is no identifiable information in respondents' data, our second technique provides an efficient technique for data collection such that the data miner can collect respondents' data without the ability to link the data with each respondent. Considering that respondents' data contain some identifiable information, we design our third technique to enable the data miner to collect respondents' data in a privacy-preserving manner.

After data are collected by different parties, those parties may want to share their data in certain ways to benefit each other, e.g., learning certain models from the combination of their data. To protect data privacy in such settings, we then design our fourth privacy-preserving technique for a particular data mining task: learning a Bayesian network from a database vertically partitioned among two parties. In this setting, two parties owning confidential databases wish to learn a Bayesian network from the combination of their databases without revealing anything else about their data to each other. By using cryptographic techniques, we present efficient and

privacy-preserving protocols to construct a Bayesian network on the parties' joint data.

Once the data are collected by different parties, generally those data are stored and maintained in databases. Encryption is a powerful tool to protect data. However, when data are encrypted, performing queries becomes more challenging. To solve this problem, we study efficient methods for queries on encrypted data. Specifically, we show that even if an intruder breaks into the database and observes some interactions between the database and its users, he learns very little about the data stored in the database and the queries performed on the data. In addition to proving security guarantees formally, we provide empirical data for performance evaluations.

Overall, we have provided five techniques in the distributed setting to protect data privacy using cryptographic tools. Our techniques show that data privacy can be properly protected in distributed settings. Our experimental results further demonstrate that our techniques are very efficient and can be deployed across large scales.

Author: Zhiqiang Yang

Advisor: Rebecca N. Wright

Date: March 30, 2007

Department: Computer Science

Degree: Doctor of Philosophy

Acknowledgments

First and foremost, I would like to thank my advisor Rebecca Wright. I have been working with her for four years in my graduate education in Stevens, and her unending patience and encouragement have helped me make this milestone. I give her my deepest respect and thankfulness for her advices, guidance, and immediate help.

During the time of my graduate study, I am very fortunate to have worked with Sheng Zhong, a wonderful collaborator and great friend. His advice and personal supports are crucial for me to finish this dissertation.

I am also very grateful to my committee members: Joan Feigenbaum, Susanne Wetzel, and Charles Suffel. I have benefited a great deal from conversations with them. Especially Thanks Susanne Wetzel for her careful review on the final version of this dissertation. I also want to thank all faculty and staff in the computer science department for their support. In particular, I want to thank Professors David Naumann, Dominic Duggan, Stephen Bloom, George Kamberov, and Adriana Compagnoni for their academic support. I also thank Sherry Dorso and Dawn Garcia for their technical support. My thanks also go to all graduate students and postdocs in computer science department. We shared a lot of happy and unforgettable moments during my life in Stevens. This work was supported by the National Science Foundation under grant number CCR-0331584.

Last but not least, I want to thank my parents and my wife for their support for many years.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

With the rapid development of the Internet, more and more sensitive data are collected, processed, and stored by different parties. If those sensitive data are disclosed to unauthorized parties, data privacy may be breached.

Data privacy is an important issue to many people, businesses, and governments. Generally data privacy means not that the data are hidden from all parties, but rather that there is the ability to reveal selected information to selected parties under certain circumstances without being completely revealed to everyone or for every purpose. In different settings, data privacy has different meanings. For example, patients are willing to share their medical information with their doctors, but not their colleagues. In this example, data privacy means that patients' medical history should only be revealed to their doctors. Otherwise, data privacy is considered breached. Another example is that people may want to release their medical records without their identifiable attributes (e.g., SSN or name) to some research institute for research purposes. In this example, data privacy means that only the selected attributes in medical records are revealed to others.

Privacy definitions specify what condition should be satisfied to protect data privacy. In this work, rather than providing a single definition for data privacy, we give different privacy definitions in different settings. Correspondingly, privacy preservation means that data privacy

is protected under certain specific settings, and the protocols of protecting privacy are called privacy-preserving protocols. Data privacy issues may arise in distributed settings where data flow from one party to another party.

Data mining is a very powerful tool for knowledge discovery from large amounts of data. In data mining applications, a data miner often needs to collect data from or share data with other parties. When a data miner collects data from large sets of respondents, generally each respondent submits her data to the miner in response. Clearly, this can be an efficient and convenient procedure, assuming the respondents are willing to submit their data. However, once a respondent submits her data to the miner, she loses the control of her data, and the privacy of her data is fully dependent on the miner. The respondents' willingness to submit data is also affected by their privacy concerns [31].

In certain circumstances, the purpose of a miner to collect data is to learn specific data mining models, e.g., classification models. To learn those models, the miner does not necessarily need to collect all original data from respondents. In Chapter 3, we provide privacy-preserving protocols to enable a data miner to learn a classification model—naive Bayes classifiers, on which new instances can be classified. In our protocols, data privacy is protected in the sense that the data miner only learns the models rather than each respondent's original data.

Very often, a data miner wants to collect the respondents' data for general purposes such that those data can be used for learning other types of models besides classification models. Based on different types of respondents' data, we differentiate between two cases. In the first case, respondents' data do not contain *explicit identifiable attributes* which can be used to uniquely identify the data subjects, e.g, social security number (SSN). In this case, respondents' privacy can be protected if each respondent can anonymously submit their data to the data miner. That is, each respondent can submit their data in such a way that the miner cannot link each respondent to their original data in the data collection procedure. In Chapter 4, we design privacy-preserving techniques which can enable each respondent to submit her data to the data miner anonymously.

Although respondents' data do not contain explicit identifiable attributes, a set of attributes, e.g., {zip code, sex, date of birth}, may be used to identify the respondents with significant

probability. Such attributed are called *quasi-identifiers*. If quasi-identifiers can be associated with explicit identifiable attributes, perhaps using other public datasets, respondents' privacy can be seriously violated. For example, Sweeney pointed out that one can find out who has what disease using a public database and voter lists by taking advantage of quasi-identifiers [101]. In the second case, we consider that respondents' data contain quasi-identifiers. Techniques called k-anonymization have been proposed to solve the privacy problems caused by quasi-identifiers [99, 92, 101, 100] in the centralized setting where a set of data are stored only in one database. Given the set of data, existing k-anonymization techniques modify some entries in quasi-identifiers such that any value of quasi-identifiers appear at least k times in the dataset. A respondent's privacy is protected because her identity is hidden within k peers. But the existing k-anonymization techniques cannot work in the distributed setting where a data miner wants to collect a k-anonymized set of data in which each record is held by a respondent. In Chapter 5, we present privacy-preserving protocols to protect respondents' privacy in the distributed setting. In our technique, the values of quasi-identifiers are modified in the data collection procedure such that any value of quasi-identifiers appears at least k times in the collected data.

Rather than collecting and mining the data directly from individuals, data mining algorithms are often applied on existing databases. Currently more and more data are distributed and maintained in databases by different parties. For example, different banks have their own databases which store and maintain customers' information. Traditional data mining algorithms are based on the centralized model where the data from different databases are gathered into a central site. This means that different parties have to share their data with each other. However, with privacy concerns about the data, those parties may not willing to share their data with others although they may greatly benefit from the results by mining their joint data. In some cases, regulations and consumer backlash also prevent different organizations from sharing their data with each other. Such regulations include HIPAA [59] and the European privacy directives [86, 87]. As an example, consider a scenario in which one research center maintains a DNA database about a large set of persons, and the records of those persons' medical diagnosis are stored and maintained by a hospital. The research center wants to explore correlations between DNA sequences

and specific diseases. Due to privacy concerns and privacy regulations, the hospital cannot provide any information about individuals' medical records to the research center.

As this example shows, it is worth investigating how to enable data mining across distributed sites in a privacy-preserving manner—i.e., without requiring the sites to share their original data with each other. The elegant and powerful paradigm of secure multiparty computation (SMC) provides generic secure solutions for any probabilistic polynomial-time function [52, 110]: distributed parties can compute a probabilistic polynomial-time function on their private data without revealing their data to others. In principle, mining across distributed data can be solved by these generic solutions. However, the generic solutions are very inefficient when applied to large input data sets. Privacy-preserving data mining provides data mining algorithms in which the goal is to compute or approximate the output of a particular algorithm applied to the joined data, without revealing anything else, or at least anything else sensitive, about the data. Existing privacy-preserving solutions can be categorized into two approaches. One approach adopts cryptographic techniques to provide secure solutions in distributed settings [75]. The other approach randomizes the original data such that the underlying patterns are still kept in the randomized data [9]. In Chapter 6, we provide a privacy-preserving distributed solution for a particular data mining algorithm—Bayesian network learning. Our solution enables learning of Bayesian networks on the combination of two distributed databases without the need to exchange the data in those two databases directly.

Sensitive data are frequently stored and maintained in database systems; these systems are attractive targets for security attacks. To protect sensitive data, database systems generally offer access control mechanisms to restrict access. However, access control cannot ensure that databases are fully immune to intrusion and unauthorized access. If unauthorized users can bypass access control mechanisms provided by database systems, data in the databases are vulnerable to attacks. There have been numerous data breaches reported recently. For instance, MasterCard reported in June 2005 that more than 40 million credit card accounts were exposed and these compromised data could be used to steal funds [33]. Sensitive data in databases may contain personal information, transaction history, and propriety information. Disastrous consequences might befall a company whose database were penetrated by attackers. Therefore,

it is worth studying how to protect sensitive data when the database is subject to unauthorized access.

Encryption is a well-studied technique to protect sensitive data. There is increasing awareness of encryption techniques to protect sensitive data in databases. When sensitive data are properly encrypted, even if the intruder could gain access to the encrypted data, no information or very little information would be leaked if the intruder does not have the encryption key. Clearly, this is desirable for security and privacy considerations. But, the integration of encryption/decryption techniques into database systems cause performance degradation because encryption/decryption computation is computationally expensive. Furthermore, queries on encrypted data are undesirable slow because the indexing techniques, e.g., B+ trees, cannot be used on encrypted data. In Chapter 7, we propose new encryption schemes for databases to protect sensitive data. Our schemes allow efficient methods for certain kinds of queries on the encrypted data. Such methods are very useful in strengthening the protection of sensitive data in databases.

In addition to designing privacy-preserving protocols for those five problems, we have given detailed analysis of the privacy properties of these protocols. We also provide experimental results which further demonstrate that these protocols are efficient.

1.2 Related Work

Secure two-party computation was first introduced by Yao [110], then it was later generalized to secure multiparty computation [52]. Secure two-party computation is a special case of SMC. In distributed settings, SMC provides generic solutions of protecting data privacy to compute any probabilistic polynomial-time function which has input data from multiple parties. In SMC, the target function is first represented as a combinatorial circuit, then each party runs a short protocol on each gate of this circuit. In the end, each party gets the final results of the target function. In SMC, privacy means that no more information is leaked than in an “ideal” model, in which each party sends her input to a trusted third party, who carries out the computation on the received inputs and sends the appropriate results back to each party. The privacy definition

of SMC guarantees that no more information than the final results (which are supposed to be learned by each party) is transferred. We introduce the formal privacy definition of SMC in Chapter 2.

The complexity of SMC depends on the size of the circuit which depends on the size of inputs and the complexity of expressing the function as circuits. Hence, SMC is not a practical solution for computing complex functions on large data sets, and more efficient solutions should be designed for specific distributed computations. Our work constitutes examples of such work. In particular, we design efficient and practical distributed solutions for five different problems. Although SMC is not efficient on large data sets, secure two-party computation of small circuits with small inputs may be practical and efficient [77]. We use secure two-party computation as a building block for some smaller parts of our computation in one of our solutions in Chapter 6.

Recently there is a large and growing body of work on privacy-preserving data mining which enables certain data mining computations while protecting the underlying data [81, 40, 75, 9]. Those solutions can largely be categorized into two approaches: cryptographic approach and randomization approach. The basic idea in randomization approaches is to add random noise to the original data. Randomization approaches provide a methodology of reconstructing distributions from the randomized data. Data privacy is protected by having the original data disguised by random noise. Randomization approaches can be used in the scenario where a data miner wants to mine the data from large sets of respondents. These respondents can simply randomize their data and then submit their randomized data to the miner who can later reconstruct some useful information. Different random perturbation methods have been proposed and applied in different data mining algorithms [9, 5, 42, 90, 41, 38]. Randomization approaches are very efficient, but in general they can induce a tradeoff between privacy of respondents and accuracy of the data mining result: the more privacy each respondent has, the less accurate the mining results are, and vice-versa. Although some randomization approaches lead to good accuracy and good privacy in specific data mining problems, these approaches would produce inaccurate results when used to learn other models. In Chapter 3, we present a privacy-preserving frequency mining protocol to enable a data miner to learn certain data mining models. Compared with randomization approaches, this protocol is very efficient and achieves perfect privacy which

means there is no tradeoff between efficiency and privacy.

To enable learning of specific data mining models from randomized data, we have to design specific randomization approaches. The data which are randomized by different approaches cannot be used to learn any data mining model in the general case. Assuming that respondents' data do not contain identifiable information, we design general-purpose data collection protocols in Chapter 4. Our protocols enable a data miner to learn the original data from large sets of respondents while preserving each respondent's privacy by maintaining each respondent's anonymity. That is the miner cannot link respondent's data to her identity. Since the initial publication of this work [109], an improved protocol that is secure—even given malicious behavior by the participants—has been presented by Brickell and Shmatikov [94].

We also notice that related work in anonymous communication channels could be used to provide an anonymous data collection channel. However, building an anonymous channel is a nontrivial task. General-purpose ways to build anonymous channels include mix networks [23, 85, 91, 65, 54], dining cryptographer networks (DC-nets) [24, 107, 53] and k-anonymous message transmission [10], all of which are still under active study. Compared with mix networks, our methods can be viewed as a (nontrivial) application of certain techniques in their design to the scenario of distributed data collection, which eliminates the burden of introducing a set of mix servers while keeping analogous security guarantees. Compared with DC-nets and k-anonymous message transmission, our solution produces more efficient results for the data collection setting. For example, the communication cost of k-anonymous message transmission is cubic in the number of respondents; in comparison, our solutions have linear communication costs. Systems like Crowds [89] and Hordes [74] provide practical anonymity for web access without assuming an existing anonymous channel. Similarly, our work provides anonymity for data collection without assuming an existing anonymous channel. However, Crowds and Hordes assume that eavesdroppers involved are local (i.e., can only eavesdrop communication within one hop). They do not guarantee anonymity when an adversary is able to eavesdrop communications in the entire system. In contrast, our solutions for data collection can defend against global eavesdroppers.

In the cases where respondents' data contain quasi-identifiers, we use k-anonymization

ideas to design the solutions such that the miner can only collect a k-anonymized version of respondents' data. As a strategy to prevent identity disclosure in data release, k-anonymization was first proposed and analyzed by Samarati and Sweeney [99, 92, 101, 100]. Meyerson and Williams [79] formally studied how to minimize the number of suppressed entries in k-anonymization and showed that it is NP-hard; they then gave approximation algorithms. However, all existing k-anonymization techniques work on the centralized setting where all the data reside in one site, and those techniques do not work in the distributed setting. Our work provides a fully distributed version of k-anonymization technique: the data miner can only learn a k-anonymized version of data from online respondents.

Cryptographic approaches of privacy-preserving data mining work in the distributed setting where data are distributed across several sites. In cryptographic approaches, privacy is quantified by the privacy guarantee of SMC: each party involved in the privacy-preserving distributed protocols should only learn the data mining models which they are supposed to learn, and nothing else. Generally, specific protocols have to be designed for specific computation for the reason of efficiency. Currently specific privacy-preserving distributed protocols have been proposed to address different data mining problems across distributed databases, e.g., association rules mining across multiple databases [103, 67], ID3 trees [75], clustering [104, 64], naive Bayes classification [69, 105], and Bayesian network parameters [78]. Privacy-preserving primitives for simple statistical computations are very useful for designing privacy-preserving distributed data mining algorithms: finding common elements [44, 6], scalar product [21, 12, 103, 98, 44, 48], and computing correlation matrices [76]. The field of distributed data mining provides distributed data mining algorithms for different applications [71, 97, 84, 66], and they may provide privacy-preserving solutions with minor modifications.

According to how the data are distributed, we distinguish two categories—*vertically partitioned data* and *horizontally partitioned data*. By vertically partitioned data, we mean that each distributed database contains some attributes of a record. By horizontally partitioned data, we mean that each distributed site contains the same sets of attributes. In Chapter 6, we provide a privacy-preserving protocol to enable learning of a particular data mining model–

Bayesian networks over data vertically partitioned among two sites. Distributed Bayesian network learning has been addressed for both vertically partitioned data and horizontally partitioned data [26, 25, 108]. However, these algorithms were designed without privacy in mind, and indeed they require parties to share substantial amounts of information with each other. A privacy-preserving technique for learning the parameters of a Bayesian network on vertically partitioned data has been proposed in [78]. However, our solution provides better accuracy, efficiency, and privacy than the solution in [78]. We give a detailed comparison in Chapter 6.

To protect sensitive data in databases, various methods have been proposed in various settings [18, 82, 106, 58, 72, 55]. In particular, encryption has been used as an important technique to protect sensitive data [34]. An analysis of how to encrypt and securely store data in relational database management systems has been given in [62]. Recognizing the importance of encryption techniques, some database vendors have included encryption functionalities in their products [1, 2]. By considering different privacy policies for different data records, Hippocratic databases combine privacy policies with sensitive to prevent unauthorized users from accessing sensitive data [7].

With data stored in an encrypted form, a crucial question is how to perform queries. Hacigumus et al. [56] studied querying encrypted data in the database-as-service (DAS) model where sensitive data are outsourced to an untrusted server [57]. Their solution divides attribute domains into partitions and maps partitions ids to random numbers to achieve privacy. This idea is simple, practical, and elegant. However, it relies on an implicit tradeoff between privacy and efficiency: if the partitions are larger, then less information is leaked, but the database server needs to send more false positives (i.e., data that should not have been in the results of queries) to the user; if the partitions are smaller, then the database server needs to send fewer false positives, but more information is leaked. This issue is further explored in [61]. Furthermore, there is no precise quantification of the information leak given the size of partitions or the amount of communication overhead. In comparison, the solutions we provide in Chapter 7 do not have such a tradeoff; our solutions exhibit strong privacy without wasting communication resources on false positives; our security guarantee is precisely quantified using a cryptographic measure. Another issue is that, although the partition ids in [56] can be used for indexing to speed up

queries, just as is pointed out in [32], such an index can incur inference and linking attacks. In comparison, we give a solution that speeds up queries using metadata and that metadata does not introduce any additional information leakage.

For range queries on numerical data, Agrawal et al. [8] propose a solution which allows convenient indexing. Their solution is built on an encoding which preserves the order of the numerical data in each column. Consequently, if a database intruder observes the encrypted data, he learns the order of all cells in every column, which is a significant amount of information. Nevertheless, no formal analysis is given in [8] to quantify the information leak of the solution. In comparison, we formally show that our solutions reveal only very little (as quantified in Chapter 7) information to a potential intruder.

In the scenario considered in [8, 62], the adversary is modeled to have access to the data storage and has no access to the transmitted messages between the users and the database. In the setting of the DAS model [56], since the database server is untrusted, the server itself is a potential adversary who tries to breach the data privacy. The server has access to all encrypted data and all the transmitted messages between the users and the server. In this sense, the server has the most power to breach data privacy. In comparison, we model that the adversary can have access to all encrypted data in the server, and he also can monitor some transmitted messages between the server and the users. Under our model, we can provide more efficient and secure solutions. We give the details of the attack (or adversary) model, and we also formally prove the security properties of our solutions under the attack model.

In the areas of network security and applied cryptography, the study of "search on encrypted data" is closely related to our work. Specifically, Song et al. [96] propose practical techniques for finding keywords in encrypted files, which allow a user, when given a trapdoor for a keyword, to check the existence of the key word in a file. But their solution needs to scan the entire file sequentially and no provably-secure index technique is provided. A follow-up by Chang and Mitzenmacher [22] has interesting analysis but their solution is restricted to searching for a keyword chosen from a pre-determined set. Boneh et al. present a searchable public key scheme [17]; the scenario they considered is analogous to that of [96] but uses public key encryption rather than symmetric-key encryption. In the same scenario, Goh demonstrates a

method for secure indexes by using Bloom filters [49]. These solutions are possibly useful in searching for keywords in a file; however, it is unclear how to apply them to the problem of efficiently querying encrypted relational databases by providing certain indices. Another piece of related work is by Feigenbaum et al. [43], in which an encryption scheme was proposed to efficiently retrieve tuples from a look-up dictionary by using hash functions. The basic idea is that only if a valid key is provided, can the corresponding tuple be retrieved. Another related work is private information retrieval (PIR) [29, 19, 73] which is designed to hide from the public database what queries a user is making. Compared with our work, the goal of PIR is to protect each user's privacy rather than the data. In symmetric private information retrieval (SPIR) [46], the privacy goal is to also prevent the users from learning extra information in the databases from their queries.

A class of closely related work studies how to measure privacy in different settings. This includes privacy definitions based on confidence intervals [9], based on mutual information [5], and based on priori and posterior knowledge [41, 36]. Privacy definitions motivated by cryptographic notions of confidentiality are given by Gilburd et al. [47] and Dwork and Nissim [39], respectively. Another related research to our work is statistical databases. In statistical databases, it has been studied how to protect individual privacy while allowing information sharing. There is an array of literature on privacy in statistical databases (see surveys [4, 95]). Proposed methods can be categorized into query restriction (e.g., [63, 28]) and data perturbation (e.g., [88, 102, 13, 3]). In particular, the tradeoff between privacy and utility in statistical databases was investigated by Dinur and Nissim [36].

1.3 Our Contributions

We design five protocols in distributed settings to protect data privacy. Our contributions are summarized as follows:

1. Our first protocol enables a data miner to learn classification models without collecting all the original data from each respondent (Details in Chapter 3.

2. Our second protocol, where each respondent's data do not contain identifiable attributes, enables each respondent to anonymously submit their data while the miner cannot link the respondent's identity to their data from the communication channel(Details in Chapter 4).

3. Our third protocol is suitable for the case where each respondent's data contain some identifiable attributes. This solution de-identifies respondents' data in order to protect their privacy(In Chapter 5).

4. We provide privacy-preserving protocols for learning Bayesian networks on vertically partitioned data(Details in Chapter 6).

5. To protect data privacy in databases, we design encryption schemes on relational databases. Our solution supports efficient queries on the encrypted data(Details in Chapter 7).

.

Chapter 2

Cryptographic Preliminaries

To design privacy-preserving protocols, we use some cryptographic tools as the building blocks. In this section, we introduce several such cryptographic fundamentals which are used in the next few chapters to design our protocols.

2.1 Definitions

In this section, we introduce some basic encryption schemes and concepts including the *ElGamal cryptosystem* and the concepts of *ensembles* and *computational indistinguishability*.

Definition 2.1.1 ElGamal Cryptosystem*: The ElGamal cryptosystem is a public encryption scheme. The public key is* (h, G, q, g) *where* G *is a cyclic group of order* q *with the generator* g*,* $h = g^x$ *and* x *is the private key which is randomly chosen from* $[1, q]$*. All computation in the ElGamal scheme is done in the group* G*.*

Under the public key (h, G, q, g)*, the ciphertext of a message* m *(which is the representation of an element of* G*) is encrypted as* $E(m) = (c_1, c_2)$ *where* $c_1 = m \cdot h^r$*,* $c_1 = g^r$ *and* r *is randomly chosen from* $[1, q]$*. To decrypt the ciphertext* (c_1, c_2) *with the private key* x*, the plaintext message* m *can be decrypted as* $m = c_1(c_2^x)^{-1}$*.*

ElGamal encryption is *semantically secure* under the Decisional Diffie-Hellman (DDH) assumption [16]. One family in which DDH is believed to be intractable is the quadratic residue

subgroup Q_p of $\mathbb{Z}_p^*$ where p, q are two primes and $p = 2q + 1$.

In the ElGamal encryption scheme, one cleartext has many possible ciphertexts because of the random value r. ElGamal supports *rerandomization*: a new ciphertext $E'(m)$ of m can be computed from a ciphertext $E(m) = (c_1, c_2)$ as $E'(m) = (c_1 \cdot h_{r'}, c_2 \cdot g_{r'})$ where r' is randomly chosen from $[1, q]$.

Definition 2.1.2 Secret Sharing: *Secret sharing refers to any method by which a* secret *can be shared by multiple parties in such a way that no party knows the secret, but it is easy to construct the secret by combing some parties' shares.*

In a two-party case, Alice and Bob share a value x modulo some appropriate value N, in such a way that Alice holds a, Bob holds b, and x is equal (not just congruent, but equal) to $(a + b) \bmod N$. This is called *additive secret sharing*. An important property of this kind of secret sharing is that if Alice and Bob have shares of x and y, then they can each locally add their shares modulo N to obtain shares of $x + y$.

Shamir secret sharing is a threshold scheme [93]. In Shamir secret sharing, there are N parties and a polynomial P of degree $k - 1$ such that $P(0) = x$ where x is a secret. Each of the N parties holds a point in the polynomial P. Because k points (x_i, y_i) $(1 \leq i \leq k)$ uniquely define a polynomial P of degree $k - 1$, a subset of at least k parties can reconstruct the secret x. But, fewer than k parties cannot construct the secret x. This scheme is also called (N, k) Shamir secret sharing.

Ensembles and computational indistinguishability are two important concepts in proving the security properties of distributed protocols [50, 51].

Definition 2.1.3 Ensembles: *An ensemble indexed by n $(n \in \mathbb{N})$ is sequence of random variables indexed by n. For example, $X = \{X_n\}_{n \in N}$ is an ensemble indexed by n where the X_i's are random variables.*

Definition 2.1.4 Computational Indistinguishability*: Two ensembles, $X = \{X_n\}_{n \in \mathbb{N}}$ and $Y = \{Y_n\}_{n \in \mathbb{N}}$ are indistinguishable in polynomial time if for every probabilistic polynomial-*

time algorithm D, every polynomial $p(\cdot)$, and all sufficiently large n

$$|P_r(D(X_n) = 1) - P_r(D(Y_n) = 1)| < \frac{1}{p(n)}.$$

We write $X \stackrel{c}{\equiv} Y$ to denote that X and Y are computationally indistinguishable.

2.2 Secure Multiparty Computation

Before we introduce secure multiparty computation, we must first make the distinction between *semi-honest* and *malicious* adversaries in multiparty protocols. Semi-honest adversaries follow the protocol exactly but try to learn additional information by analyzing the messages they received during the execution of the protocol. Such adversaries often model attacks that take place only after the execution of the protocol has completed. Malicious adversaries can always execute some arbitrary, malicious operations which can be very damaging to other parties. The malicious adversaries are much more difficult to defend against when designing the protocol.

It is proved that, in the distributed multiparty setting, any probabilistic polynomial-time function can be securely computed by assuming a majority of honest parties. Informally, in the semi-honest model, a protocol *privately* computes a function if whatever can be computed by a subset of parties could be computed from their inputs and all intermediate computing messages. The formal definition is stated as follows [51].

Definition 2.2.1 Privacy w.r.t. Semi-honest Behavior*:*

In the distributed setting, let f: $(x_1, \ldots, x_n) \rightarrow (y_1, \ldots, y_n)$ be a n-ary function, and let π be a n-party protocol for computing f. Let $\overline{x}$ denote $(x_1, \ldots, x_n)$. The view of the i^{th} $(i \in [1, n])$[1] party during an execution of π on $\overline{x}$ is denoted by $\mathsf{view}_i^{\pi}(\overline{x})$ which includes x_i, all received messages, and all internal coin flips. For every subset I of $[1, m]$, denoted by $I = \{i_1, \ldots, i_t\}$, let $f_I(\overline{x})$ denote $(y_{i_1}, \ldots, y_{i_t})$ and $\mathsf{view}_I^{\pi}(\overline{x}) = (I, \mathsf{view}_{i_1}^{\pi}(\overline{x}), \ldots, \mathsf{view}_{i_t}^{\pi}(\overline{x}))$. Let OUTPUT$(\overline{x})$ denotes the output of all parties during the execution of π. We say that π privately computes f if there exists a probabilistic polynomial-time algorithm denoted by S, such that for every

[1]Throughout this paper, $[1, n]$ denotes the set of integers from 1 to n.

$I \subseteq [1, n]$ *it holds that*

$$\{S(x_{i_1}, \ldots, x_{i_t}, f_I(\overline{x})), f(\overline{x}))\} \stackrel{\mathsf{c}}{\equiv} \{\mathsf{view}_I^{\pi}(\overline{x}), \mathit{OUTPUT}(\overline{x})\}$$

This definition states that the view of the parties in I can be simulated from only the parties' inputs and outputs. If the function is privately computed by the protocol, then privacy of each party's input data is protected. In this dissertation, we focus on designing privacy-preserving protocols in the semi-honest model. The formal definition of the security protocol in the malicious model can be found in [51].

Chapter 3

Privacy-Preserving Learning of Classification Model

3.1 Introduction

In this chapter, we consider a scenario in which a data miner surveys a possibly large number of respondents to learn classification models on their data, while the sensitive data of these respondents need to be protected.

To solve this problem, in Section 3.2 we propose a simple cryptographic approach which provides strong privacy for each respondent and does not lose any accuracy at the cost of privacy. Our key technical contribution is a privacy-preserving protocol that allows a data miner to compute frequencies of values or tuples of values in the respondents' data, without revealing the privacy-sensitive part of the data. Unlike general-purpose cryptographic protocols, this method requires no interaction between respondents, and each respondent only needs to send a single flow of communication to the data miner. However, we are still able to ensure that nothing about the sensitive data beyond the desired frequencies are revealed to the data miner. To address the classification-learning problem, in Section 3.3 we use this frequency mining computation to obtain a privacy-preserving naive Bayes classifier learning algorithm.

The scenario we consider here is similar to the scenario of electronic voting [14]. However,

e-voting systems usually assume that the voting authority consists of a group of servers that are threshold-trusted, or that another authority independent from the voting authority also participates in the protocol. Both of these possibilities are not justifiable in our scenario, however. Another difference is that, in our setting, additional specialized concerns in e-voting such as receipt-freedom of the protocol are not an issue. Recent work gives privacy-preserving solutions for mining a naive Bayes classifier across a database horizontally or vertically partitioned into a small number of partitions [69, 105]. In contrast, our solution provides a privacy-preserving method for mining a naive Bayes classifier in a fully distributed setting where each respondent holds one record.

3.2 Privacy-Preserving Frequency Mining

This section describes a privacy-preserving primitive for frequency mining. In Section 3.2.1, we formulate the problem of frequency mining and state our privacy definition. We present our privacy-preserving protocol for frequency mining in Section 3.2.2. We give a security proof for this protocol in Section 3.2.3 and provide experimental results in Section 3.2.4.

3.2.1 Problem Formulation

We consider a very basic problem: there are n respondents $U_1, \ldots, U_n$. Each respondent U_i has a Boolean value d_i. The miner would like to find out how many d_i's are 1's and how many are 0's.

This problem essentially amounts to computing the sum $d = \sum_{i=1}^{n} d_i$ without revealing each d_i. However, for practical reasons, we seek a solution satisfying a few important restrictions:

- Each respondent only sends one flow of communication to the miner; there is no further interaction between the respondent and the miner.
- No respondent communicates with other respondents.

We call this the *reduced interaction* model. Solutions in this model are highly practical

because they do not need communication channels between different respondents or multi-round interaction between any respondent and the miner. In addition, we require that each respondent's d_i is protected as defined below.

We introduced the general definition of privacy in secure multiparty computation in Chapter 1. Our definition of privacy given below can be viewed as a simplification of the general definition in the semi-honest model, where the simplification results from our reduced interaction model. In our definition, we consider the possibility that some corrupted respondents might share their information with the miner in order to help the miner derive the information about private data of honest respondents. We require that no extra information about the honest respondents' values be leaked even if the miner above receives such help from corrupted respondents. In the following definition, we do not consider the problem of respondents sharing their information with each other because, as we discuss after the definition, this will not give them any additional information in the reduced interaction model.

Definition 3.2.1 *Assume that each respondent $U_i (i \in [1, n])$ has the input d_i, private keys x_i, y_i, and public keys X_i, Y_i. Let $d = \sum_{i=1}^{n} d_i$. A protocol for the frequency mining problem protects each respondent's privacy against the miner and up to t corrupted respondents in the semi-honest model if, $\forall I \subseteq [1, n]$ such that $|I| \leq t$, there exists a probabilistic polynomial-time algorithm M such that*

$$\{M(d, [d_i, x_i, y_i]_{i \in I}, [X_j, Y_j]_{j \notin I})\} \stackrel{\mathsf{c}}{\equiv} \{\mathsf{view}_{\mathsf{miner}, \{U_i\}_{i \in I}}([d_i, x_i, y_i]_{i=1}^{n})\}.$$

Here, $\{\mathsf{view}_{\mathsf{miner}, \{U_i\}_{i \in I}}([d_i, x_i, y_i]_{i=1}^{n})\}$ is the joint view of the miner and the t corrupted respondents, $\{U_i\}_{i \in I}$. Intuitively, this definition states that a polynomial-time algorithm M, called a simulator, can simulate what the miner and the corrupted respondents have observed in the protocol using only the final result d, the corrupted respondents' knowledge including their inputs and private keys, and the public keys. This captures the intuition that the miner and the corrupted respondents jointly learn nothing beyond d.

Definition 3.2.1 only addresses privacy in the semi-honest model. However, in our reduced interaction model, a protocol protecting respondent privacy in the semi-honest model actually

also protects respondents' privacy even when the miner and the corrupted respondents are fully malicious (i.e., may deviate arbitrarily from the protocol). This is because these malicious parties cannot have any influence on the honest respondents due to the restrictions of the reduced interaction model. In this sense, our solution provides privacy against malicious parties "for free". We note, however, that the correct completion of the protocol cannot be guaranteed with malicious parties, as they may send garbage or refuse to send anything at all.

3.2.2 Protocol

Our protocol is based on the homomorphic property of a variant of ElGamal encryption [60]. The privacy of our protocol is based on the DDH assumption and the ElGamal cryptosystem which has been described in Chapter 1. The protocol itself uses the mathematical properties of exponentiation, which allows the miner to combine encrypted results received from the respondents into the desired sums.

Let G be a group where ($|G| = q$ for a large prime q), and let g be a generator of G. All computations in this Chapter are carried out in the group G. Suppose that each respondent U_i has two pairs of keys: $(x_i, X_i = g^{x_i}), (y_i, Y_i = g^{y_i})$. We also define

$$X = \prod_{i=1}^{n} X_i, \text{ and } Y = \prod_{i=1}^{n} Y_i.$$

The values x_i and y_i are private keys; X_i and Y_i are public keys. In particular, the protocol requires that all respondents know the values X and Y.

Recall that each respondent U_i holds the Boolean value d_i, and the miner's goal is to learn $d = \sum_{i=1}^{n} d_i$. The privacy-preserving protocol for the miner to learn d is showed in Figure 3.1.

In our protocol, the message m_i sent by respondent U_i is equivalent to the first part of an ElGamal encryption of d_i under a private key $(\sum x_i)y_i$, while the message h_i is the second part is part of an ElGamal encryption of d_i under private key $(\sum y_i)x_i$. When put together, all the respondent messages are combined by the miner to be an encryption of g^d. In order to undo the resulting encryption, the miner must first decrypt to learn r, which is g^d. Then, since even the miner cannot take discrete logarithms, the miner must use trial and error to learn d. Since the

range of possible values of d is not too large, this use of trial and error is feasible.

We note that the security of ElGamal encryption depends on new random values being used for each encryption. In our setting, this means that the x_i and y_i values, and associated X and Y, cannot be reused in different uses of the protocol. However, since these parameters do not depend on the actual data values, they can in general be precomputed off-line before the protocol starts. In particular, if the protocol is to be run many times, many sets of values could be precomputed in advance so that only a single phase of key distribution is required.

$U_i \rightarrow$ miner : $m_i = g^{d_i} \cdot X^{y_i}$;
$h_i = Y^{x_i}$.

miner: $r = \prod_{i=1}^{n} \frac{m_i}{h_i}$;
for $d = 1$ to n
if $g^d = r$ output d.

Figure 3.1: Privacy-Preserving Protocol for Frequency Mining

Theorem 3.2.1 *The protocol for frequency mining presented in Figure 3.1 correctly computes the sum of all respondents' inputs.*

Proof: We show that, in the protocol, when the miner finds $g^d = r$, the value d is the desired sum. Suppose that $g^d = r$. Then:

$$\begin{aligned}
g^d &= \prod_{i=1}^{n} \frac{m_i}{h_i} \\
&= \prod_{i=1}^{n} \frac{g^{d_i} \cdot X^{y_i}}{Y^{x_i}} \\
&= g^{\sum_{i=1}^{n} d_i} \cdot \prod_{i=1}^{n} \frac{(\prod_{j=1}^{n} X_j)^{y_i}}{(\prod_{j=1}^{n} Y_j)^{x_i}} \\
&= g^{\sum_{i=1}^{n} d_i} \cdot \prod_{i=1}^{n} \frac{(g^{\sum_{j=1}^{n} x_j})^{y_i}}{(g^{\sum_{j=1}^{n} y_j})^{x_i}} \\
&= g^{\sum_{i=1}^{n} d_i} \cdot \frac{g^{\sum_{i=1}^{n} \sum_{j=1}^{n} x_j y_i}}{g^{\sum_{i=1}^{n} \sum_{j=1}^{n} y_j x_i}} \\
&= g^{\sum_{i=1}^{n} d_i}
\end{aligned}$$

Thus, $g^d = g^{\sum_{i=1}^{n} d_i}$, and therefore $d = \sum_{i=1}^{n} d_i$, as desired. ■

3.2.3 Protocol Analysis

Next, we establish our privacy guarantee based on the ElGamal encryption scheme which is introduced in Chapter 2. To encrypt a message α using public key X, one computes

$$C = (\alpha X^k, g^k),$$

where k is chosen uniformly at random in $[1, q]$. Our protocol makes use of a *homomorphic* property of a modified version of ElGamal. Specifically, if the message α is changed to g^α before encrypting, then from encryptions $(g^{\alpha_1} X^{k_1}, g^{k_1})$ and $(g^{\alpha_2} X^{k_2}, g^{k_2})$ of α_1 and α_2, respectively, then it is possible to derive an encryption of $\alpha_1 + \alpha_2$, as $(g^{\alpha_1} X^{k_1} \cdot g^{\alpha_2} X^{k_2}, g^{k_1} \cdot g^{k_2}) = (g^{(\alpha_1+\alpha_2)} X^{(\alpha_1+\alpha_2)}, g^{(k_1+k_2)})$.

Under the assumption of the semantic security of ElGamal encryption, we show that our protocol protects each respondent's privacy (even if there are up to $n-2$ respondents colluding with the miner) as long as the ElGamal encryption scheme is secure.

Theorem 3.2.2 *Assuming that all keys have been distributed properly when the protocol starts, the protocol for frequency mining presented in Figure 3.1 protects each honest respondent's privacy against the miner and up to $n-2$ corrupted respondents.*

Proof: To prove that respondents' privacy is protected, we consider the worst case with the maximum number $n-2$ of corrupted respondents. Since our protocol is symmetric in respondents' indices, without loss of generality we assume that $I = \{3, 4, \ldots, n\}$. Recall that, to prove the protocol protects respondent privacy, we need to construct a simulator M that can generate an ensemble indistinguishable from the miner and the corrupted respondents' view using only the final result d, the corrupted respondents' knowledge, and the public keys.

Instead of describing the entire simulator in detail, we give an algorithm that computes the view of the miner and the corrupted respondents in polynomial-time using only d, corrupted respondents' knowledge, public keys, and some ElGamal encryptions. Under the assumption

that the ElGamal encryption is semantically secure, we already know that each ElGamal ciphertext can be simulated. Therefore, combining our algorithm with a simulator for ElGamal ciphertexts, we obtain a complete simulator.

Below is the algorithm that computes the view of the miner and the corrupted respondents. It takes four encryptions as its input: $(u_{11}, v_{11}) = (g^{d_1} \cdot g^{x_1 y_1}, g^{x_1})$, $(u_{12}, v_{12}) = (g^{d_1} \cdot g^{x_2 y_1}, g^{x_2})$, $(u_{21}, v_{21}) = (g^{d_2} \cdot g^{x_1 y_2}, g^{x_1})$, $(u_{22}, v_{22}) = (g^{d_2} \cdot g^{x_2 y_2}, g^{x_2})$. Then it computes m_1, m_2 by the computation

$$m_1' = u_{11} u_{12} Y_1^{\sum_{i \in I} x_i}$$

and

$$m_2' = u_{21} u_{22} Y_2^{\sum_{i \in I} x_i}.$$

It computes h_1, h_2 by the computation

$$h_1' = u_{11} u_{21} X_1^{\sum_{i \in I} y_i} / g^{d - \sum_{i \in I} d_i}$$

and

$$h_2' = u_{12} u_{22} X_2^{\sum_{i \in I} y_i} / g^{d - \sum_{i \in I} d_i},$$

completing the proof. ■

3.2.4 Experimental Results

We implemented our privacy-preserving frequency mining algorithm in C, using the OpenSSL libraries for the cryptographic operations. We ran a series of experiments on a PC with a 2GHz processor and 512MB memory under NetBSD. In our experiments, the length of each cryptographic key is 512 bits. We measured the computing time of the privacy-preserving frequency mining protocol for different numbers of respondents, from $2,000$ to $10,000$ with the stepsize $2,000$. Our experimental results are based on the average of 20 protocol runs.

To set up for the privacy-preserving frequency mining protocol, the key generation time for each respondent is 4.2 seconds. Computing the protocol parameters X and Y for $10,000$

respondents takes 140 milliseconds. As previously noted, these values can be precomputed off-line before the protocol starts.

In the privacy-preserving frequency mining protocol, it takes each respondent only 1 millisecond to prepare her message to the miner, as it requires just a single modular exponentiation. The miner's computation is somewhat longer, but still quite efficient. Figure 3.2 shows the times the miner uses to compute one frequency for different numbers of respondents. For example, for $10,000$ respondents, the miner's computation takes 146 milliseconds.

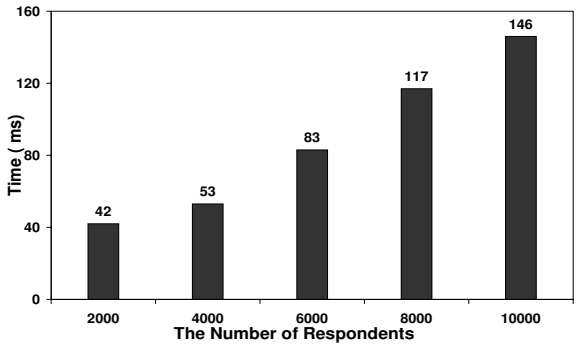


Figure 3.2: Miner's Computation Time for a Single Frequency Calculation

3.3 Privacy-Preserving Learning of Naive Bayes Classifiers

The primitive of frequency mining is simple, but is very useful in data mining applications. In this section, we demonstrate the power of our primitive by showing a privacy-preserving naive Bayes classifier computation in the fully distributed setting (which can be thought of as a horizontally partitioned database in which *each* record is held by a different party).

3.3.1 Problem Formulation

Naive Bayes classifiers have been used in many practical applications, e.g., in text classification and medical diagnosis [37, 80]. They simplify greatly the learning task by assuming that attributes are independent given the class. Although independence of attributes is an unrealistic assumption, naive Bayes classifiers often compete well with more sophisticated models,

even if there is modest correlation between attributes. Naive Bayes classifiers have significant advantages in terms of simplicity, learning speed, classification speed, and storage space.

In (non-private) naive Bayes learning, the miner is given a set of training examples. Each example is an attribute vector of a respondent together with her class label. From these examples the miner learns a classifier that can be used to classify new instances.

We consider the following scenario: there are m attributes, $(A_1, \ldots, A_m)$, and one class attribute V. Without loss of generality, we assume that each attribute A_i ($i \in [1, m]$ has a domain of $\{a_i^{(1)}, \ldots, a_i^{(d)}\}$ and the class attribute V has a domain of $\{v^{(1)}, \ldots, v^{(p)}\}$. We also assume that there are n respondents $(U_1, \ldots, U_n)$, where each respondent U_j has a vector denoted by $(a_{j1}, \ldots, a_{jm}, v_j)$. In the respondent's vector, $(a_{j1}, \ldots, a_{jm}, v_j)$ is an instance of the attribute vector $(A_1, \ldots, A_m)$ and v_j is U_j's class label. In our setting, these data are the training samples from which the miner learns the classifier without learning the samples themselves. The miner surveys all respondents for values based on their data as described in Section 3.3.2, and constructs a classifier to classify a new instance by selecting the most likely class label v:

$$v = \underset{v^{(\ell)} \in V}{\operatorname{argmax}} \Pr(v^{(\ell)}) \prod_{i=1}^{m} \Pr(a_i \mid v^{(\ell)}), \tag{3.3.1.1}$$

where $(a_1, \ldots, a_m)$ is the attributes vector of the new instance.

To learn the naive Bayes classifier, traditionally the miner collects all respondents' data in one central site, and then learns the classifier at that cental site. In our setting, there is a set S of privacy-sensitive attributes where $S \subseteq A = (A_1, \ldots, A_m)$. Formally, for any $j \in [1, n]$, U_j is not willing to reveal any information about the attributes in S to the miner; but she is willing to reveal to the miner all the remaining non-sensitive attributes $A - S$. To protect respondents' privacy and also enable learning classifier, we design a privacy-preserving protocol for naive Bayes learning and apply this protocol on S. (Note that we can apply this protocol to A if All attributes in A are sensitive.)

3.3.2 Protocol

We use the primitive for frequency mining in Section 3.2 as a building block to design a privacy-preserving protocol for naive Bayes learning. We can assume all respondents' sensitive attributes need to be protected. We first rewrite (3.3.1.1) as:

$$v = \underset{v^{(\ell)} \in V}{\operatorname{argmax}} \#(v^{(\ell)}) \prod_{i=1}^{m} \frac{\#(a_i, v^{(\ell)})}{\#(v^{(\ell)})}, \tag{3.3.2.1}$$

where $\#(v^{(\ell)})$ ($\#(a_i, v^{(\ell)})$, resp.) denotes the frequency, or number of occurrences, of attribute value $v^{(\ell)}$ (attribute pair value $(a_i, v^{(\ell)})$, resp.) in all respondents' data. All the frequencies $\#(\cdot)$ in Equation 3.3.2.1 represent the naive Bayes classifier. To learn the classifier, all the miner needs to do is to learn $\#(v^{(\ell)})$ and $\#(a_i^{(k)}, v^{(\ell)})$ for each $i \in S$, each k, and each ℓ. Since the occurrence of $v^{(\ell)}$ or of the pair $(a_i^{(k)}, v^{(\ell)})$ can be denoted by a Boolean value, we can use the primitive presented in Section 3.2. A detailed specification of the protocol is given in Figure 3.3.

3.3.3 Protocol Analysis

For accuracy, we compare our privacy-preserving protocol with a traditional naive Bayes learning algorithm running on all respondents' data without protection of privacy. Suppose that the learning algorithm without privacy protection outputs a classifier c and that our privacy-preserving protocol outputs c'. We claim $c = c'$, which means our protocol does not lose any accuracy at the cost of privacy.

Theorem 3.3.1 *The protocol for naive Bayes learning presented in Figure 3.3 does not lose accuracy.*

Proof: This is straightforward from our protocol specification. Our protocol counts each $\#(v^{(\ell)})$ and each $\#(a_i^{(k)}, v^{(\ell)})$ for $i \notin S$ directly. It uses the privacy-preserving method we presented in Section 3.2 to count each $\#(a_i^{(k)}, v^{(\ell)})$ for $i \in S$. Since the method in Section 3.2 computes frequencies precisely, our protocol outputs exactly the same classifier as a non-privacy-preserving naive Bayes algorithm. ■

Key Setup:

$a_{ji}^{(k,\ell)} \stackrel{\text{def}}{=} 1$ if $(a_{ji}, v_j) = (a_i^{(k)}, v^{(\ell)})$;
0 otherwise.

U_j's private keys:
$(x_{ji}^{(k,\ell)})_{i\in S, 1\le k\le d, 1\le \ell\le p}, (y_{ji}^{(k)})_{i\in S, 1\le k\le d, 1\le \ell\le p}$.

Public keys: $X_{ji}^{(k,\ell)} = g^{x_{ji}^{(k,\ell)}}; Y_{ji}^{(k,\ell)} = g^{y_{ji}^{(k,\ell)}}$.
$(1 \le j \le n, i \in S, 1 \le k \le d, 1 \le \ell \le p)$

$X_i^{(k,\ell)} = \prod_{1\le j\le m} X_{ji}^{(k,\ell)}; Y_i^{(k,\ell)} = \prod_{1\le j\le m} Y_{ji}^{(k,\ell)}$.
$(1 \le j \le n, i \in S, 1 \le k \le d, 1 \le \ell \le p)$

Step 1:

U_j's computation: for $i \in S, 1 \le k \le d, 1 \le \ell \le p$

$$\overline{a}_{ji}^{(k,\ell)} = g^{a_{ji}^{(k,\ell)}} \cdot (X_i^{(k,\ell)})^{y_{ji}^{(k,\ell)}};$$
$$h_{ji}^{(k,\ell)} = (Y_i^{(k,\ell)})^{x_{ji}^{(k,\ell)}}.$$

U_j's message to miner : $(\overline{a}_{ji}^{(k,\ell)}, h_{ji}^{(k,\ell)})_{i\in S, 1\le k\le d, 1\le \ell\le p}$;
$(a_{ji})_{i\notin S}$, v_j.

Step 2:

miner: for $i \in S, 1 \le k \le d, 1 \le \ell \le p$

$$r_i^{(k,\ell)} = \prod_{j=1}^{n} \frac{\overline{a}_{ji}^{(k,\ell)}}{h_{ji}^{(k,\ell)}};$$

for $\#(a_i^{(k)}, v^{(\ell)}) = 1$ to n
if $g^{\#(a_i^{(k)}, v^{(\ell)})} = r_i$ break.

for $i \notin S, 1 \le k \le d, 1 \le \ell \le p$
count $\#(a_i^{(k)}, v^{(\ell)})$.

for $1 \le \ell \le p$
count $\#(v^{(\ell)})$.

output the classifier.

Figure 3.3: Privacy-Preserving Protocol for Naive Bayes Learning

The protocol for naive Bayes learning presented in Figure 3.3 protects each respondent's sensitive data against the miner and up to $n-2$ corrupted respondents since all the privacy-preserving frequency computations are done independently.

Recall that there are n respondents and m attributes and that each (non-class) attribute has a domain of size d, and the class label has a domain of size p. Assume $s = |S|$ is the number of sensitive attributes. It is easy to see that each respondent has a computational overhead—as compared to a non-private solution—of dps ElGamal encryptions. In data mining applications, we usually have $n \gg dps$; thus the computational overhead for each respondent is small. The computation overhead for the miner is $O(dpsn)$ modular exponentiations. The communication

overhead is $dpsn$ ciphertexts.

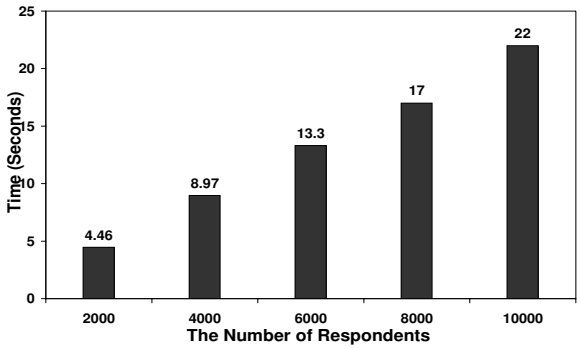


Figure 3.4: Miner's Learning Time for Naive Bayes Classifier vs. Number of Respondents

3.3.4 Experimental Results

The experimental set-up here is the same as described in Section 3.2.4. The Bayes classifier learning algorithm is implemented in C, and uses the frequency mining implementation as subroutine. Again, we ran a series of experiments on a PC with a 2GHz processor and 512MB memory under NetBSD, using 512 bit cryptographic keys. Our experimental results are based on the average of 20 protocol runs. For the Bayes classifier experiments, we assume that each respondent has 10 attributes, that each attribute has 8 nominal values, and that there are 2 classes. We measured the computation time of each respondent and the miner in our privacy-preserving protocol for Bayes classifier learning. Our results show that each respondent needs only 120 milliseconds to compute her message flow to the miner. Figure 3.4 shows the computation times the miner learns a naive Bayes classifier for different numbers of respondents for different number of respondents from $2,000$ to $10,000$ with the step size $2,000$.

Figure 3.5 further studies how the miner's learning time changes when both the respondent number and the attribute number vary: the number of respondents from $2,000$ to $10,000$ with step size $2,000$ and the number of attributes from w to 10 with step size 2. In this experiment, we fix the domain size of each non-class attribute to 4 and the domain size of the class attribute to 2.

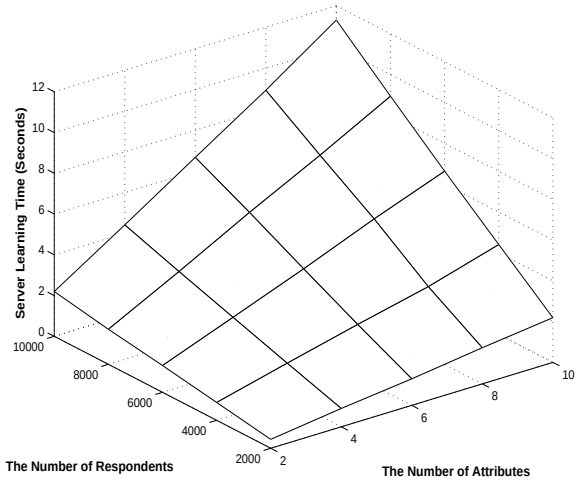


Figure 3.5: Miner's Learning Time for Naive Bayes Classifier vs. Number of Respondents and Number of Attributes

3.4 Discussion

In this chapter, we proposed a privacy-preserving protocol of frequency mining and applied it to naive Bayes learning in the distributed setting. If this problem is solved using randomization techniques, then there is a trade-off between privacy and accuracy. However, our proposed solution enjoys cryptographically strong privacy without losing any accuracy at the cost of privacy. Furthermore, both theoretical analysis and experimental results show that the method itself and its application to naive Bayes learning are very efficient.

The privacy-preserving frequency mining protocol can also be used for any data mining model enabled by frequency computation, e.g., ID3 trees and association rules. These are very practical in some cases, but rely on certain parameters being small. It is worth investigating how to use our frequency mining protocol to learn efficiently other data mining models. A possible direction to the solution of this problem is the combination of our method with randomization techniques to further improve efficiency.

Chapter 4

Anonymity-Preserving Data Collection

4.1 Introduction

The protocol we presented in Chapter 3 allows a miner to learn only the models which can be enabled by frequencies. However, in some cases the miner wants to collect respondents' data for uses that cannot be enabled by frequencies. For example, suppose the miner is a medical researcher who studies the relationship between dining habits and a certain disease. If a respondent does not want to reveal what food she eats and/or whether she has that disease, she may give false information or decline to provide information. Note that even if each respondent's data do not contain or imply any identifiable attribute, the privacy of each respondent cannot be guaranteed if the miner can link the respondent's identity with their submitted data through the communication channel, e.g., by IP address.

If respondents' data do not contain or imply any identifiable attribute, their privacy can be protected if they can submit their data to the miner anonymously. That means the miner cannot link each respondent to her submitted data through the communication channel. We generalize this idea to propose an approach called *anonymity-preserving data collection*. Specifically, we propose that the miner should collect data in such a way that he is unable to link any data to the respondent who provided that data. Furthermore, the data collected is not modified in any way, and thus the miner will have the freedom to apply any suitable mining algorithms to the data.

In Section 4.3, we give a basic solution in the semi-honest model in which the miner and all

respondents are guaranteed to follow the protocol. We also prove the correctness and anonymity properties of the protocol and present experimental results of its efficiency. In Section 4.4, we describe an extended protocol that protects respondents' privacy against a malicious data miner, again with experimental results to measure efficiency.

4.2 Problem Formulation

We consider a scenario of data collection in which there are a data miner and a (potentially large) number of respondents. Each respondent owns a piece of data. In our setting, the goal is to allow the miner to collect respondents' data without being able to determine which piece of data came from whom. Specifically, let N be a small number, typically 20, 50, or 100. We divide the respondents into groups, where each group has N members. In order to make the protocol practical, the miner collects the data from one group at a time, so that only N respondents need to be simultaneously online. Our requirement is that each respondent should be "hidden" in the N respondents in her group. In other words, the miner should get N pieces of data from a group, but should not know which piece came from which group member. Small N can achieve better efficiency, but big N can provide better privacy. Different values of N represent the tradeoff between privacy and efficiency.

In the sequel, we restrict our discussion to one group of respondents and denote the N respondents in this group by $1, \ldots, N$. We assume that there is a private and authenticated communication channel between each respondent and the miner. Although communication channels between respondents may exist in some practical situations, to make the problem as general as possible, we do not assume their existence in our solution. Figure 4.1 illustrates the resulting layered architecture of our solution. Our anonymity-preserving data collection solution runs over any private and authenticated channels between respondents and miner, and the miner can use any data mining or data analysis tools thereafter. We denote by d_i the piece of data owned by respondent i, whose length is bounded by a security parameter κ. There are two possible ways for the miner to violate respondent i's anonymity: either d_i contains some information about respondent i (such as respondent i's identifier, telephone number, or

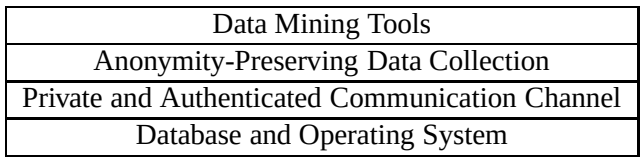


Figure 4.1: System Components

zip code) and by looking at this information the miner is able to associate d_i with respondent i, or during the data collection process the miner observes that d_i comes from respondent i. As noted before, in this chapter we focus on the second possibility. In particular, we assume that d_i does not contain information that can be used to associate it with respondent i; in this case, respondent i will remain anonymous as long as the data collection process preserves her anonymity.

In this case, we allow for the possibility that some of the respondents may be corrupted and collude with the miner. Informally, the data collection process preserves each honest respondent's anonymity if, when we arbitrarily switch the data between honest respondents within their group of N respondents, the miner cannot see any difference in the data collection process (even with the help of dishonest respondents). Let σ be an arbitrary permutation on $\{1, \ldots, N\}$; then $(d_{\sigma(1)}, \ldots, d_{\sigma(N)})$ is a possible result of arbitrarily switching data between respondents. We further require that $\sigma(i) = i$ for any corrupted respondent i, which means only the honest respondents' data are switched. The anonymity requirement is that the miner cannot distinguish the data collection procedure in which the respondents have data $(d_1, \ldots, d_N)$ from the data collection procedure in which respondents have data $(d_{\sigma(1)}, \ldots, d_{\sigma(N)})$.

We give the definition of anonymity in the semi-honest model following the definition introduced in Chapter 1. In the semi-honest model, the miner and the corrupted respondents follow the protocol but attempt to derive private information and violate the anonymity of the honest respondents.

Definition 4.2.1 *A protocol for the data collection problem preserves each honest respondent's anonymity against the miner and $t-1$ corrupted respondents in the semi-honest model if, for any $I \subseteq \{1, \ldots, n\}$ such that $|I| \leq t-1$, for any $(d_1, \ldots, d_N)$, and any permutation σ on*

$\{1, \ldots, N\}$ *such that* $\forall i \in I, \sigma(i) = i$*,*

$$\{\mathsf{view}_{\mathsf{miner},I}(d_1, \ldots, d_N)\} \stackrel{\mathsf{c}}{\equiv} \{\mathsf{view}_{\mathsf{miner},I}(d_{\sigma(1)}, \ldots, d_{\sigma(N)})\}.$$

$\{\mathsf{view}_{\mathsf{miner},I}(d_1, \ldots, d_N)\}$ is the joint view of the miner and the set I of $t-1$ corrupted respondents. Intuitively, Definition 4.2.1 states that the adversary (i.e., the miner and the corrupted respondents) cannot notice any difference in his view if we arbitrarily switch data between the honest respondents. Therefore, the miner and the corrupted respondents jointly learn nothing about which piece of data corresponds to which honest respondent. Clearly, this is consistent with the intuitive understanding of anonymity.

We not only develop a solution in the semi-honest model, but also extend the solution to another standard cryptographic model, the malicious model [51]. First, an anonymity-preserving protocol in the malicious model needs to be anonymity-preserving when all participants follow the protocol; second, when any malicious participant deviates from the protocol, the honest participants must be able to detect this before the anonymity is violated, so that the honest participants can abort the protocol without their anonymity being compromised. In this way, the malicious participants are effectively "forced" to follow the protocol. In Section 4.4, we provide a solution against a malicious miner.

4.3 Basic Solution

In this section, we give a solution to the problem of anonymity-preserving data collection in the semi-honest model. Extensions of this solution to the malicious model are presented in Sections 4.4.

Our goal is that the miner should obtain a random permutation of the respondents' data $(d_1, \ldots, d_N)$, without knowing which piece of data comes from which respondent. To achieve this goal, we use ElGamal encryption together with a *rerandomization* technique and a *joint decryption* technique. Both of these techniques have been used extensively in mix networks, e.g., [85, 91, 65, 54].

Permutation of the order of items means randomly rearranging the order of items. If we rerandomize and permute a sequence of ciphertexts, then we get another sequence of ciphertexts with the same multiset of cleartexts but in a different order. Looking at these two sequences of ciphertexts, the adversary cannot determine any information about which new ciphertext corresponds to which old ciphertext.

In our solution, t of the N respondents act as "leaders". Leaders have the special duty of anonymizing the data. At the beginning of the protocol, all respondents encrypt their data using a public key which is the product of all leaders' public keys. Note that the private key corresponding to this public key is the sum of all leaders' private keys; without the help of all leaders, nobody can decrypt any of these encryptions. The leaders then rerandomize these encryptions and permute them. Finally, the leaders jointly help the miner to decrypt the new encryptions, which are in an order independent of the original encryptions.

For notational convenience, we assume in the sequel that the leaders are respondents 1 through t. In practice, the choice of leaders can be arbitrary or can be dependent on the application.

4.3.1 Protocol

Each respondent i has an ElGamal key pair (x_i, y_i) where x_i is the private key and y_i is the corresponding public key. All ElGamal key pairs use the same parameters (G, g), and all computations are done in the group G. Here, the public key y_i is known to all participants, while the private key x_i is kept secret by respondent i. In the sequel, let $y = \prod_{i=1}^{t} y_i$ and $x = \sum_{i=1}^{t} x_i$. In our protocol, we use this public value y as a public key to encrypt respondent data. Clearly, $y = g^x$. So, this secret value x, which is not known to any individual respondent, is needed to decrypt these encryptions of respondent data. Our protocol has 3 phases:

- Phase 1: Data submission.
  - For $i = 1, \ldots, N$, each respondent i encrypts her data using public key y:

$$C_i \stackrel{\mathsf{def}}{=} (C_i^{(1)}, C_i^{(2)}) = (y^{r_i} d_i, g^{r_i}),$$

where r_i is picked uniformly at random from $[1, q-1]$. Then respondent i sends C_i to the miner.

- Phase 2: t-round anonymization. For $i = 1, \ldots, t$, the miner and the respondents work as follows:

  – At the beginning of the ith round, the miner sends the current values of $(C_1, \ldots, C_N)$ to leader i.

  – Leader i rerandomizes each piece of data and permutes the pieces: for $j = 1, \ldots, N$,

$$\begin{aligned} R_j &\stackrel{\mathsf{def}}{=} (R_j^{(1)}, R_j^{(2)}) \\ &= (C_{\pi_i(j)}^{(1)} \cdot y^{\delta_{\pi_i(j)}}, C_{\pi_i(j)}^{(2)} \cdot g^{\delta_{\pi_i(j)}}), \end{aligned}$$

  where π_i is a random permutation on $\{1, \ldots, N\}$, and each δ_j is picked independently and uniformly from $[1, q-1]$.

  – For $j = 1, \ldots, N$, leader i sets $C_j = R_j$. Then leader i sends $(C_1, \ldots, C_N)$ back to the miner.

- Phase 3: Decryption.

  – The miner sends $(C_1^{(2)}, \ldots, C_N^{(2)})$ to all leaders.

  – Each leader i computes partial decryption: for $j = 1, \ldots, N$,

$$p_{j,i} = (C_j^{(2)})^{x_i}.$$

  – Each leader i sends the miner the partial decryption $(p_{1,i}, \ldots, p_{N,i})$.

  – The miner computes the decryption: for $j = 1, \ldots, N$,

$$d'_j = C_j^{(1)} / \prod_{i=1}^{t} p_{j,i}.$$

4.3.2 Protocol Analysis

Theorem 4.3.1 *If all participants follow the protocol, then the miner's result* $(d'_1, \ldots, d'_N)$ *is a permutation of* $(d_1, \ldots, d_N)$.

Proof: At the end of Phase 1, the miner has received encryptions of $(d_1, \ldots, d_N)$. In Phase 2, these ciphertexts are rerandomized and permuted; therefore, at the end of Phase 2, $(C_1, \ldots, C_N)$ represents the encryptions of a permutation of $(d_1, \ldots, d_N)$. Because

$$\begin{aligned} d'_j &= C_j^{(1)} / \prod_{i=1}^{t} p_{j,i} \\ &= C_j^{(1)} / \prod_{i=1}^{t} (C_j^{(2)})^{x_i} \\ &= C_j^{(1)} / (C_j^{(2)})^{\sum_{i=1}^{t} x_i} \\ &= C_j^{(1)} / (C_j^{(2)})^{x}, \end{aligned}$$

the cleartexts of $(C_1, \ldots, C_N)$ are $(d'_1, \ldots, d'_N)$. Therefore, $(d'_1, \ldots, d'_N)$ is a permutation of $(d_1, \ldots, d_N)$. ■

Theorem 4.3.2 *The protocol of Section 4.3.1 preserves the anonymity of each honest respondent against the miner and up to* $t-1$ *corrupted respondents in the semi-honest model.*

Proof: By contradiction. Suppose that this protocol does not preserve anonymity. We show that this implies a probabilistic polynomial-time algorithm that distinguishes the ElGamal encryptions of two different cleartexts, which contradicts the well-known result that ElGamal is semantically secure.

Clearly it suffices to consider the case in which all the $t-1$ corrupted customers are leaders. If the protocol does not preserve anonymity, then there exist $(d_1, \ldots, d_N)$, a permutation σ on $\{1, \ldots, N\}$ such that $\forall i \in I, \sigma(i) = i$, a probabilistic polynomial algorithm D, and a

polynomial $f()$ such that for infinitely many κ,

$$\Pr[D(\mathsf{view}_{\mathsf{miner},I}(d_1,\ldots,d_N))=1]-\Pr[D(\mathsf{view}_{\mathsf{miner},I}(d_{\sigma(1)},\ldots,d_{\sigma(N)}))=1] > 1/f(\kappa). \tag{4.3.1}$$

Now we use a *hybrid argument* (see Chapter 3 in [50]): since σ is a permutation on $\{1,\ldots,N\}$ such that $\forall i \in I, \sigma(i) = i$, we can decompose it to a number of simple permutations where each simple permutation only switches the order of two elements outside I (that are *not* equal). Formally, there exist permutations $\sigma_1,\ldots,\sigma_m$ ($m < N-(t-1)$) on $\{1,\ldots,N\}$ such that for $j=1,\ldots,m$, $\forall i \in I, \sigma_j(i) = i$ and that

$$\sigma = \sigma_1 \circ \ldots \circ \sigma_m.$$

Defining

$$\mathsf{view}_0 = \mathsf{view}_{\mathsf{miner},I}(d_1,\ldots,d_N),$$

and for $j=1,\ldots,m$,

$$\mathsf{view}_j = \mathsf{view}_{\mathsf{miner},I}(d_{\sigma_1\ldots\sigma_j(1)},\ldots,d_{\sigma_1\ldots\sigma_j(N)}).$$

Then clearly,

$$\mathsf{view}_m = \mathsf{view}_{\mathsf{miner},I}(d_{\sigma(1)},\ldots,d_{\sigma(N)}).$$

By Equation 4.3.1, we know there exists $j \in [0,m]$ such that

$$\Pr[D(\mathsf{view}_j)=1]-\Pr[D(\mathsf{view}_{j+1})=1] > \frac{1}{mf(\kappa)}.$$

The above equation is equivalent to

$$\Pr[D(\mathsf{view}_{\mathsf{miner},I}(d_{\sigma_1\ldots\sigma_j(1)},\ldots,d_{\sigma_1\ldots\sigma_j(N)}))=1] - \Pr[D(\mathsf{view}_{\mathsf{miner},I}(d_{\sigma_1\ldots\sigma_j\sigma_{j+1}(1)},\ldots,d_{\sigma_1\ldots\sigma_j\sigma_{j+1}(N)}))=1] > \frac{1}{mf(\kappa)}.$$

Note a subtle convention of compositions:

$$\sigma_1 \ldots \sigma_j \sigma_{j+1}(i) = \sigma_{j+1}(\sigma_1(\ldots \sigma_j(i))).$$

(If we do not use this convention, we can get the same result by a simple modification of the indices.) Therefore, we can rewrite Equation 4.3.2 as

$$\begin{aligned}\Pr[D(\mathsf{view}_{\mathsf{miner},I}(d_{\ell_1}, \ldots, d_{\ell_N})) = 1] - \Pr[D(\mathsf{view}_{\mathsf{miner},I}(d_{\sigma_{j+1}(\ell_1)}, \ldots, d_{\sigma_{j+1}(\ell_N)})) = 1] \\ > \frac{1}{mf(\kappa)},\end{aligned}$$

where $\ell_i = \sigma_1 \ldots \sigma_j(i)$. Recall that σ_{j+1} only switches the order of two elements outside I that are not equal; suppose that it switches the order of ℓ_α and ℓ_β ($d_{\ell_\alpha} \neq d_{\ell_\beta}$, $\alpha < \beta$, $\alpha, \beta \notin I$). Formally, we have

$$\begin{aligned} d_{\sigma_{j+1}(\ell_\alpha)} &= d_{\ell_\beta}, \\ d_{\sigma_{j+1}(\ell_\beta)} &= d_{\ell_\alpha}, \end{aligned}$$

and that for any $i \neq \alpha$, $i \neq \beta$,

$$d_{\sigma_{j+1}(\ell_i)} = d_{\ell_i}.$$

Below we give a probabilistic polynomial-time algorithm A that distinguishes an ElGamal encryption of d_{ℓ_α} from an ElGamal encryption of d_{ℓ_β}.

On input of ciphertext e, A first computes, using the homomorphic property of ElGamal, another ciphertext e' such that the product of the cleartexts of e and e' is equal to $d_{\ell_\alpha} \cdot d_{\ell_\beta}$: A computes a random encryption of $d_{\ell_\alpha} \cdot d_{\ell_\beta}$ and then divides it by e. Then A rerandomizes e' to get e''. Next, A *simulates* two executions of our protocol; A extracts the view of the adversary generated in each simulated execution and applies D to it. Specifically, during each simulated execution, A simulates the miner using a process that works exactly as described in the protocol. The simulation of other parties is detailed as follows.

In Phase 1 of the protocol, for any i except α and β, A simulates respondent i using a process with input d_{ℓ_i}; the process works exactly as described in the protocol. A simulates respondents α and β with two processes; these two processes do not encrypt their inputs as described in the protocol but directly send out their inputs to the simulated miner. (Note that this has no impact on the view of the adversary because $\alpha, \beta \notin I$; thus, it has no impact on the output of D that we need.) During the first simulated execution, the simulated respondent α starts with ciphertext e and the simulated respondent β starts with e''; during the second execution, the simulated respondent α starts with ciphertext e'' and the simulated respondent β starts with e. Now recall that $t-1$ leaders are dishonest; suppose that the only honest leader is θ.

In Phase 2, the first $\theta - 1$ rounds of anonymization are simulated exactly as described in the protocol. Then A chooses a random permutation ρ on $\{1, \ldots, N\}$ and defines the cleartext output $(d'_1, \ldots, d'_N)$ of the entire simulated protocol as $(d_{\rho(1)}, \ldots, d_{\rho(N)})$. For $i = \theta, \ldots, t-1$, A chooses a random permutation ρ_i on $\{1, \ldots, N\}$ and simulates the ciphertexts $(C_1, \ldots, C_N)$ at the end of round i using random encryptions of $(d_{\rho_i(\rho(1))}, \ldots, d_{\rho_i(\rho(N))})$. Furthermore, A simulates the ciphertexts $(C_1, \ldots, C_N)$ at the end of round t using random encryptions of $(d_{\rho(1)}, \ldots, d_{\rho(N)})$. Then, the corresponding simulated messages and coin flips can be easily computed from these simulated ciphertexts.

In Phase 3, the simulated messages and coin flips can be easily computed from the simulated ciphertexts $(C_1, \ldots, C_N)$ at the end of round t in Phase 2 together with their decryptions $(d_{\rho(1)}, \ldots, d_{\rho(N)})$. Specifically, each message $p_{j,i}$ ($i \neq t$) in the protocol can be simulated using a uniformly random element of G and each $p_{j,t}$ can be simulated using

$$p_{j,t} = \frac{C_j^{(1)}}{d_{\rho(j)} \prod_{i=1}^{t-1} p_{j,i}}.$$

Applying D to the views of the adversary generated in the simulated executions, A can compute

$$o_1 = D(\mathsf{view}_{\mathsf{miner},I}(d_{\ell_1}, \ldots, d_{\ell_{\alpha-1}}, D(e), d_{\ell_{\alpha+1}}, \ldots, d_{\ell_{\beta-1}}, D(e''), d_{\ell_{\beta+1}}, \ldots, d_{\ell_N})),$$

and

$$o_2 = D(\mathsf{view}_{\mathsf{miner},I}(d_{\ell_1}, \ldots, d_{\ell_{\alpha-1}}, D(e''), d_{\ell_{\alpha+1}}, \ldots, d_{\ell_{\beta-1}}, D(e), d_{\ell_{\beta+1}}, \ldots, d_{\ell_N})),$$

where $D(e)$ denotes the decryption of e. If $o_1 = 1$ and $o_2 = 0$, A outputs 1; if $o_1 = 0$ and $o_2 = 1$, A outputs 0; otherwise A outputs a uniformly random bit.

Now we analyze the probabilities of outputing 1 with input ciphertext of d_{ℓ_α} or d_{ℓ_β}. For convenience, let

$$p_1 = Pr[D(\mathsf{view}_{\mathsf{miner},I}(d_{\ell_1}, \ldots, d_{\ell_N})) = 1],$$

and

$$p_2 = \Pr[D(\mathsf{view}_{\mathsf{miner},I}(d_{\sigma_{j+1}(\ell_1)}, \ldots, d_{\sigma_{j+1}(\ell_N)})) = 1].$$

When the input ciphertext is an encryption of d_{ℓ_α}, the probability that we have output equals 1 is

$$\Pr[A(d_{\ell_\alpha}) = 1] = p_1(1 - p_2) + p_1 p_2/2 + (1 - p_1)(1 - p_2)/2.$$

When the input ciphertext is an encryption of d_{ℓ_β}, the probability that we have output equals 1 is

$$\Pr[A(d_{\ell_\beta}) = 1] = p_2(1 - p_1) + p_2 p_1/2 + (1 - p_2)(1 - p_1)/2.$$

Combining the above two equations, we have

$$\begin{aligned}
&\Pr[A(d_{\ell_\alpha}) = 1] - \Pr[A(d_{\ell_\beta}) = 1] \\
&= p_1(1 - p_2) + p_1 p_2/2 + (1 - p_1)(1 - p_2)/2 \\
&\quad - (p_2(1 - p_1) + p_2 p_1/2 + (1 - p_2)(1 - p_1)/2) \\
&= p_1 - p_2 > \frac{1}{mf(\kappa)}.
\end{aligned}$$

The last inequality is due to Equation 4.3.3. However, this contradicts the semantic security of ElGamal. ∎

4.3.3 Experimental Results

In our protocol, the computational overhead of a non-leader respondent is 2 modular exponentiations. The major computational overhead of a leader is $3N + 2$ modular exponentiations. The major computational overhead of the miner is Nt modular multiplications and N modular divisions. The overall communication is at most $(6t + 2)\kappa N$ bits.

To measure the efficiency of our protocol in practice, we implemented it using the OpenSSL libraries and measured the computational overhead. Since the time spent on communication highly depends on the network bandwidth, we did not measure the communication overhead in our experiments. In our experiments, the length of cryptographic keys is 1024 bits. The environment used is the NetBSD operating system running on an AMD Athlon 2GHz processor with 512M memory.

We measure the computation times of the three types of participants: regular (i.e., non-leader) respondents, leaders, and the miner. For each of these times, we measure how it varies with different N and t, where N is from 20 to 100 with stepsize 20 and t is from 5 to 20 with stepsize 5. Our experimental results are based on the average of 20 runs. All our experimental results are consistent with our theoretical analysis.

Figure 4.2 illustrates our measurements of a regular respondent's computation time: it is always about 15ms regardless of N and t. Figure 4.3 illustrates our measurements of a leader's computation time: it is linear in N and does not depend on t. For a typical scenario where $N = 20$, the computation time of a leader is about 0.47 seconds. Figure 4.4 illustrates our measurements of the miner's computation time: it is linear in both N and t. For a typical scenario where $N = 20$ and $t = 3$, the computation time of the miner is about 40ms.

4.4 Malicious Miner

In this section, we extend our solution to the model in which the miner may be malicious. The corrupted respondents are still semi-honest.

Recall that a malicious participant can deviate from the protocol arbitrarily. It is more difficult to preserve anonymity when the miner is malicious. For example, the miner may

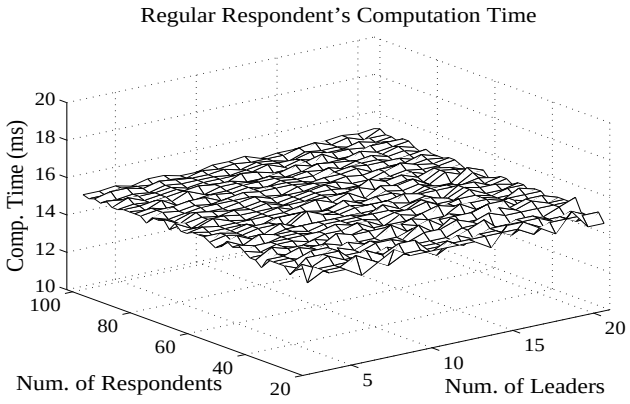


Figure 4.2: Regular Respondent's Computation Time with Semi-honest Participants

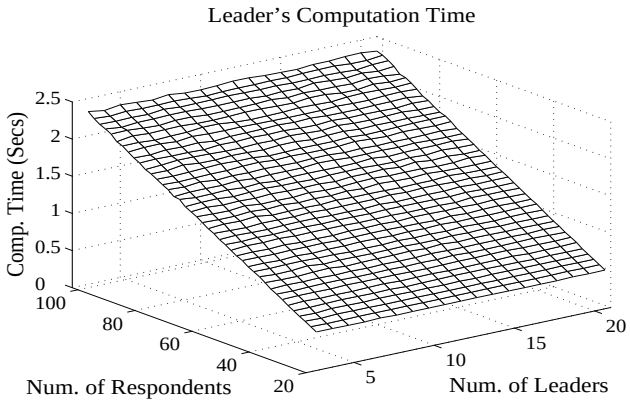


Figure 4.3: Leader's Computation Time with Semi-honest Participants

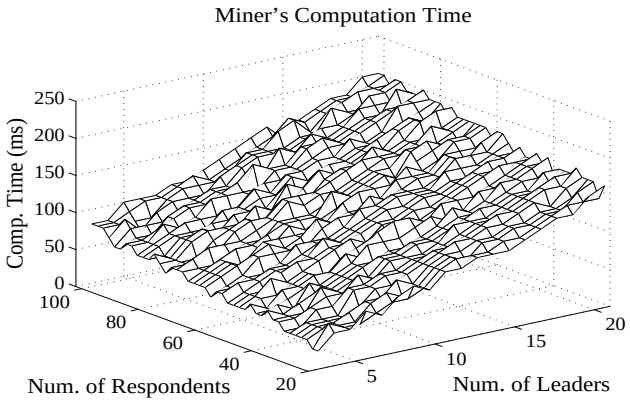


Figure 4.4: Miner's Computation Time with Semi-honest Participants

choose two respondents i and j and replace the encryption of d_j with the encryption of d_i. When the protocol finishes, there will be a piece of data with two copies. If this is the only duplicate, the miner can then easily link this piece of data to respondent i. To disallow such behavior and force the miner to follow the protocol, we use a well known cryptographic tool, *digital signatures*, as we now describe.

Digital Signatures. A digital signature scheme allows each participant, e.g., Alice, to generate a signature on her message using her private key. Anybody can verify this signature using her public key, but it is infeasible for any other party to forge her signature. Formally, we denote by $s = S_{x'}(M)$ a signature on message M using private key x'. We denote by $V_{y'}(M, s)$ the verification function of digital signature using public key y'. For any key pair (x', y') and any message M we have

$$V_{y'}(M, S_{x'}(M)) = \mathsf{accept}.$$

Furthermore, without knowing x it is infeasible to forge a digital signature s such that $V_{y'}(M, s) =$ accept with more than no-negligible probability.

Note that in our solution in the semi-honest model, each message sent from the miner to any respondent originally came from a respondent—the miner only forwards the message. Therefore, if the original sender of the message signs it and the receiver of the message verifies the signature, then a cheating miner who deviates from the protocol can be detected.

4.4.1 Protocol and Analysis

Including the key pair (x_i, y_i), this protocol assumes that each respondent i has another key pair (x'_i, y'_i) such that $y'_i = g^{x'_i}$ (where x'_i is a private key and y'_i is a public key), in addition to the key pair (x_i, y_i) described in Section 4.3.1. This new key pair is used for digital signatures.

- Phase 1: Data submission.
  - For $i = 1, \ldots, N$, each respondent i encrypts her data using public key y:

$$C_i \stackrel{\mathsf{def}}{=} (C_i^{(1)}, C_i^{(2)}) = (y^{r_i} d_i, g^{r_i}),$$

where r_i is picked uniformly at random from $[1, q-1]$. Respondent i signs C_i using private key x'_i:

$$s_i = S_{x'_i}(C_i).$$

Then respondent i sends (C_i, s_i) to the miner.

- Phase 2: t-round anonymization. For $i = 1, \ldots, t$, the miner and the respondents work as follows.

  – At the beginning of the ith round, the miner sends $((C_1, s_1) \ldots, (C_N, s_N))$ to leader i.

  – Leader i checks that she has received N signed messages; if not, she aborts the protocol. Then leader i verifies the signatures: if $i = 1$, for $j = 1, \ldots, N$, she verifies that

  $$V_{y'_j}(C_j, s_j) = \mathsf{accept};$$

  otherwise, for $j = 1, \ldots, N$, she verifies that

  $$V_{y'_{i-1}}(C_j, s_j) = \mathsf{accept};$$

  If any of the above equations does not hold, leader i aborts the protocol.

  – Leader i rerandomizes each piece of data and permutes the pieces: for $j = 1, \ldots, N$,

  $$\begin{aligned} R_j &\stackrel{\mathsf{def}}{=} (R_j^{(1)}, R_j^{(2)}) \\ &= (C_{\pi_i(j)}^{(1)} \cdot y^{\delta_{\pi_i(j)}}, C_{\pi_i(j)}^{(2)} \cdot g^{\delta_{\pi_i(j)}}), \end{aligned}$$

  where π_i is a random permutation on $\{1, \ldots, N\}$, and each δ_j is picked independently and uniformly from $[1, q-1]$.

  – For $j = 1, \ldots, N$, leader i sets $C_j = R_j$ and signs C_j using private key x'_i:

  $$s_j = S_{x'_i}(C_j).$$

Then leader i sends $((C_1, s_1), \ldots, (C_N, s_N))$ back to the miner.

- Phase 3: Decryption.
  - The miner sends $((C_1, s_1), \ldots, (C_N, s_N))$ to all leaders.
  - Each leader checks that she has received N signed messages; if not, she aborts the protocol. Then she verifies the signatures: for $j = 1, \ldots, N$, she verifies that

$$V_{y'_i}(C_j, s_j) = \mathsf{accept}.$$

    If any of the above equations does not hold, the leader aborts the protocol.
  - Each leader i computes partial decryptions: for $j = 1, \ldots, N$,

$$p_{j,i} = (C_j^{(2)})^{x_i}.$$

  - Each leader i sends the miner the partial decryptions $(p_{1,i}, \ldots, p_{N,i})$.
  - The miner computes the decryptions: for $j = 1, \ldots, N$,

$$d'_j = C_j^{(1)} / \prod_{i=1}^{t} p_{j,i}.$$

The only difference between this protocol and the protocol in the semi-honest model is that respondents' messages are signed and then verified. Consequently, when all parties follow the protocol, the miner obtains a permutation of $(d_1, \ldots, d_N)$.

In Phase 1, each respondent submits her message with the corresponding signatures. To try to learn what data are submitted by which respondent, we consider that a malicious miner can drop or tamper with respondents' data when he submits the data to the leaders in Phases 2 and 3. If the miner attempts to change a message received from respondents, the leaders will detect it by checking the number of messages and verifying the signatures. If the leaders detect the miner's malicious behavior, the leaders can immediately abort the protocol so that the miner cannot learn anything about the respondents' data. Hence, this protocol preserves the anonymity of each honest respondent against a malicious miner.

4.4.2 Experimental Results

In this protocol, the computational overhead of a non-leader respondent is 2 modular exponentiations and 1 signing operation. The major computational overhead of a leader is $3N + 2$ modular exponentiations, $N + 1$ signing operations and $2N + 1$ verification operations. The major computational overhead of the miner is Nt modular multiplications and N modular division. The overall communications are at most $(6t + 2)\kappa N + (4t + 1)\kappa' N$ bits, where κ' is the length of a digital signature, typically 512 or 1024 bits. We also implemented this protocol and

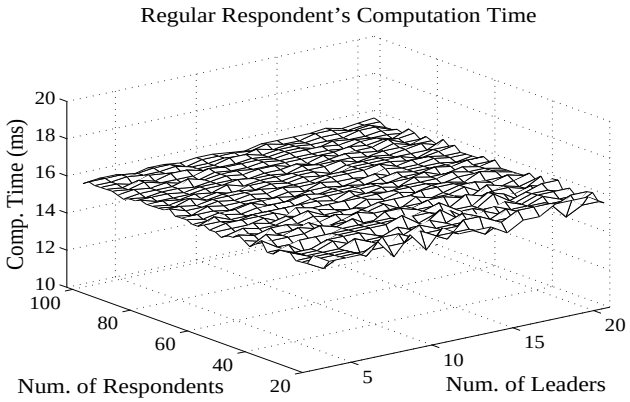


Figure 4.5: Regular Respondent's Computation Time with Malicious Miner

measured the computation times in the environment described in Section 4.3.3. We measure the computation times of the three types of participants: regular (i.e., non-leader) respondents, leaders, and the miner. For each of these times, we measure how it varies with different N and t, where N is from 20 to 100 with stepsize 20 and t is from 5 to 20 with stepsize 5. The digital signature scheme we use is DSA and the length of each signature is 512 bits. Our experimental results are based on the average of 20 runs.

Figure 4.5 illustrates our measurements of a regular respondent's computation time: it is always about 16ms regardless of the values of N and t. Compared with 15ms for the protocol in the semi-honest model, the increase in computational overhead is minimal. Figure 4.6 illustrates our measurements of a leader's computation time: it is linear in N and does not depend on t. For a typical scenario where $N = 20$, the computation time of a leader is about 0.52s, which has a 10% increase over the corresponding overhead of the protocol in the semi-

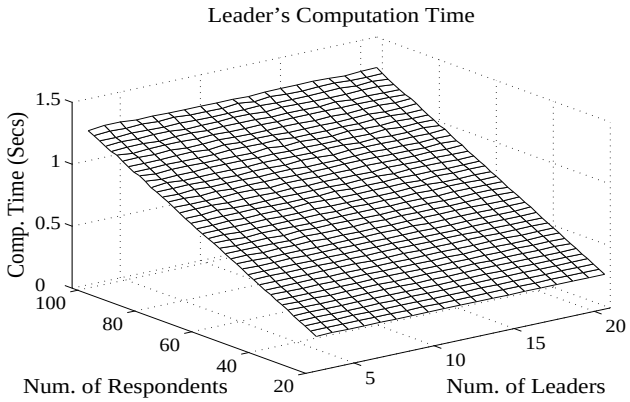


Figure 4.6: Leader’s Computation Time with Malicious Miner

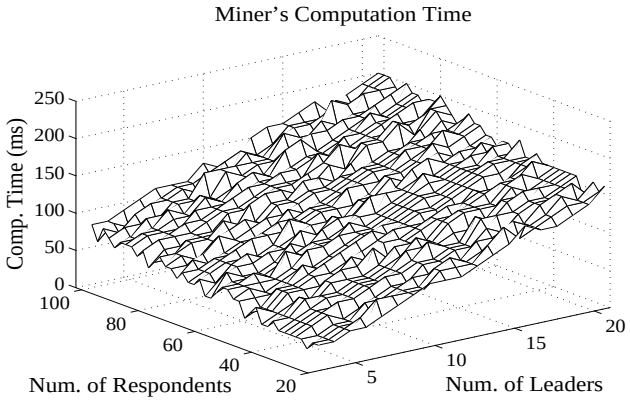


Figure 4.7: Miner Computation Time with Malicious Miner

honest model.

Figure 4.7 illustrates our measurements of the miner's computation time: it is linear in both N and t. For a typical scenario where $N = 20$ and $t = 3$, the computation time of the miner is about 30ms. For the miner, this protocol against a malicious miner adds no additional computational overhead over the protocol against a semi-honest miner.

4.5 Discussion

In this chapter, we present anonymity-preserving data collection, an approach to protect privacy for online data collection. This approach allows a data miner to collect data from a potentially large number of respondents but prevents the miner from finding out which respondent has submitted which piece of data. We present protocols with provable anonymity guarantees, one in the semi-honest model, and one that can tolerate a malicious miner. As confirmed by our experiments, the protocols are very efficient. Since the initial publication of our work on anonymity-preserving data collection, Brickell and Shmatikov have further investigated this problem and designed an efficient protocol against malicious participants [94].

Chapter 5

Fully Distributed k-Anonymization

5.1 Introduction

In the previous chapter, we considered the setting where there is no identifiable attribute (e.g., SSN and name) in each respondent's data. Nevertheless, in some cases, even if there is no explicit identifiable attribute in respondents' data, a set of attributes can still be used to identify an individual with a significant probability, e.g., the set of attributes {date of birth, zip code}. The set of attributes is called a quasi-identifier. If such information can be associated with the corresponding person's identifiers—perhaps using other publicly available databases—then privacy can be seriously violated. For example, Sweeney [101] pointed out that one can find out who has what disease using a public database and voter lists.

Consider a table that provides health information of patients for medical studies, as shown in Table 5.1. Each row of the table consists of a patient's date of birth, zip code, allergy, and history of illness. Although the identifier of each patient does not explicitly appear in this table, a dedicated adversary may be able to derive the identifiers of some patients using the combinations of date of birth and zip code. For example, he may be able to find that his roommate is the patient, who has the allergy to penicillin and a history of pharyngitis.

In the above example, the set of attributes {date of birth, zip code} is the quasi-identifier. We say an attribute is a *quasi-identifier attribute* if it is in the quasi-identifier. In our setting, nonquasi-identifier attributes like allergy and history of illness are called *sensitive attributes*,

Date of Birth	Zip Code	Allergy	History of Illness
03-24-79	07030	Penicillin	Pharyngitis
08-02-57	07028	No Allergy	Stroke
11-12-39	07030	No Allergy	Polio
08-02-57	07029	Sulfur	Diphtheria
08-01-40	07030	No Allergy	Colitis

Figure 5.1: A Table of Health Data

which patients do not want be linked to their identities. The privacy threat we consider here is that an adversary may be able to link the sensitive attributes of some rows to the corresponding patients using the information provided in the quasi-identifiers. A proposed strategy to solve this problem is to make the table k-*anonymous* [92].

In a k-anonymous table, each value of the quasi-identifier appears at least k times (or not at all). Therefore, if the adversary only uses the quasi-identifiers to link sensitive attributes to the identifiers, then each involved entity (patient in our example) is "hidden" in at least k peers. The procedure of making a table k-anonymous is called k-*anonymization*. It can be achieved by *suppression* (i.e., replacing some entries with "*") or *generalization* (e.g., replacing some or all occurrences of "07028" and "07029" with "0702*"). Table 5.2 shows the result of 2-anonymization on Table 5.1. The existing k-anonymization techniques provide privacy protection in the centralized setting where a central trusted party k-anonymizes the whole set of data and then publishes the k-anonymized version of data. The basic idea is that a data table is k-anonymized by changing some attributes such that at least k rows have the same quasi-identifier. As a strategy to prevent identity disclosure in microdata release, k-anonymization was first pro-

Date of Birth	Zip Code	Allergy	History of Illness
*	07030	Penicillin	Pharyngitis
08-02-57	0702*	No Allergy	Stroke
*	07030	No Allergy	Polio
08-02-57	0702*	Sulfur	Diphtheria
*	07030	No Allergy	Colitis

Figure 5.2: 2-Anonymized Table of Health Data

posed and analyzed by Samarati and Sweeney [99, 92, 101, 100]. Meyerson and Williams [79] formally studied how to minimize the number of suppressed entries in k-anonymization and showed that it is NP-hard; they then gave approximation algorithms (which we refer to as MW) for this problem.

We give privacy-enhancing methods for creating k-anonymous tables in a distributed scenario. Specifically, we consider a setting in which there is a set of respondents, each of whom has a row of a table, and a miner, who wants to mine the entire table. Our objective is to design protocols that allow the miner to obtain a k-anonymous table representing the respondent data in such a way that does not reveal any extra information that can be used to link sensitive attributes to corresponding identifiers. We give two different formulations of this problem and we present efficient solutions to both problem formulations. Our solutions use cryptography to obtain provable guarantees of their privacy properties, relative to standard cryptographic assumptions. Our solution to the first problem formulation does not reveal any information about the sensitive attributes outside the k-anonymous part. Our solution to the second problem formulation is not fully private, in that it reveals the k-anonymous result as well as the distances between each pair of rows in the original table. We prove, however, that it does not reveal any additional information. Our protocols enhance the privacy of k-anonymization by maintaining end-to-end privacy from the original respondent data to the final k-anonymous results.

5.2 Problem Formulations

Consider a table with m quasi-identifier attributes, $(s_1, \ldots, s_m)$, and n sensitive attributes, $(a_1, \ldots, a_n)$. Without loss of generality, we assume that there are no other attributes except these $m + n$. Suppose that there are $N + 1$ involved parties: N respondents and one miner, and that all these parties are polynomial-time bounded. For convenience, the miner assigns indices 1 through N to the respondents; in the sequel, by "respondent i" we mean "the respondent with index i". Note that the indices are *not* identifiers because they are arbitrarily assigned by the miner, who does not know the identifiers of the respondents. Each respondent i has a row of the table, which is denoted by $R_i = (s_1^{(i)}, \ldots, s_m^{(i)}, a_1^{(i)}, \ldots, a_n^{(i)})$.

We assume that there are private unidentified channels between each respondent and the miner. That is, the channels are untappable and the miner has no information about which respondent is using which channel, but each channel is used by exactly one respondent. These properties can be provided using standard cryptographic techniques.

We rigorously define our privacy requirement by adapting the standard definition of privacy [51] for cryptographic protocols in the semi-honest model to our setting. In defining our privacy requirement, we assume that before the protocol starts, there exists a global private key for a public key cryptosystem, which is shared among the respondents. Each respondent is preloaded with up to a constant number of shares. Together, the shares form the global private key; each share is only known to its owner. Such a situation can be established without a central authority by a *distributed key generation* protocol [45].

Our overall objective is to enable the miner to obtain a k-anonymized table in a private manner so that he can mine the table. This can be achieved in two ways, described in detail in Sections 5.2.1 and 5.2.2: either we enable the miner to extract the k-anonymous part of the table, or we enable him to obtain a k-anonymized table in which some entries of the quasi-identifier attributes are suppressed.

5.2.1 Formulation 1: Private Extraction of k-Anonymous Part

In the first problem formulation, the miner extracts the k-anonymous part of the table (i.e., the maximum subset of rows that is k-anonymous), but does not learn extra information about the sensitive attributes of the rows outside the k-anonymous part. Consequently, the miner cannot link the sensitive attributes of any row to the corresponding identifiers.

Intuitively, our privacy requirement states that, for each party (miner or respondent), the view of the protocol seen by that party can be simulated by an algorithm that has no knowledge of the sensitive attributes outside the k-anonymous part. This captures the requirement that any individual party cannot learn any extra information about these sensitive attributes by virtue of engaging in the protocol.

We denote by $\mathsf{view}_{\mathsf{miner}}(T)$ ($\mathsf{view}_i(T)$, resp.) the view of the miner (respondent i, resp.)

during an execution with the table

$$\begin{aligned} T &\stackrel{\mathsf{def}}{=} \{R_i : i \in [1, N]\} \\ &= \{(s_1^{(i)}, \ldots, s_m^{(i)}, a_1^{(i)}, \ldots, a_n^{(i)}) : i \in [1, N]\}. \end{aligned}$$

In the following definitions, we denote by $\mathcal{K}(T)$ the k-anonymous part of the table T.

Definition 5.2.1 *A protocol for extracting $\mathcal{K}(T)$ is private if there exist probabilistic polynomial-time algorithms M and $M_1, \ldots, M_N$ such that*

$$\{M(\mathsf{keys}_{\mathsf{miner}}, \mathcal{K}(T), \{(s_1^{(i)}, \ldots, s_m^{(i)}) : i \in [1, N]\})\}_T \stackrel{\mathsf{c}}{\equiv} \{\mathsf{view}_{\mathsf{miner}}(T)\}_T,$$

and that, for any $i \in [1, N]$,

$$\{M_i(\mathsf{keys}_i, R_i, \mathcal{K}(T), \{(s_1^{(i)}, \ldots, s_m^{(i)}) : i \in [1, N]\})\}_T \stackrel{\mathsf{c}}{\equiv} \{\mathsf{view}_i(T)\}_T,$$

where $\mathsf{keys}_{\mathsf{miner}}$ (keys_i, resp.) denotes the miner's (respondent i's, resp.) preloaded key shares (if any). The algorithms M and M_i for $i \in [1, N]$ are called simulators *(for the miner and respondent i, respectively).*

5.2.2 Formulation 2: k-Anonymization by Privately Suppressing Entries

It has been studied how to k-anonymize a table by suppressing entries—ideally suppressing as few as possible [79]. Our second problem formulation supports suppression in our distributed setting. Let $\mathsf{Anonymized}(T)$ denote the output (which is a k-anonymized table) of a protocol that k-anonymizes the table T by suppressing entries. We have an analogous privacy requirement in this case as in Formulation 1, except that the privacy is relative to $\mathsf{Anonymized}(T)$ instead of $\mathcal{K}(T)$ and the quasi-identifier:

Definition 5.2.2 *A protocol for k-anonymization by suppressing entries is private if there exist*

probabilistic polynomial-time algorithms (called simulators*)* M *and* $M_1, \ldots, M_N$ *such that*

$$\{M(\mathsf{keys}_{\mathsf{miner}}, \mathsf{Anonymized}(T))\}_T \stackrel{\mathsf{c}}{\equiv} \{\mathsf{view}_{\mathsf{miner}}(T)\}_T,$$

and that, for any $i \in [1, N]$*,*

$$\{M_i(\mathsf{keys}_i, R_i, \mathsf{Anonymized}(T))\}_T \stackrel{\mathsf{c}}{\equiv} \{\mathsf{view}_i(T)\}_T,$$

where $\mathsf{keys}_{\mathsf{miner}}$ *(*keys_i*, resp.) denotes the miner's (respondent* i*'s, resp.) preloaded key shares (if any).*

In our solution for Formulation 2, we are unable to satisfy the ideal privacy of Definition 5.2.2. General cryptographic solutions exist that could provide ideal privacy, but at much greater computation and communication costs than our solution. Instead, we achieve a relaxed, but well-defined, notion of privacy in which a specified (and presumably small) amount of information is revealed. Formally:

Definition 5.2.3 *Let* $\mathcal{F}(T)$ *be a function of the table* T*. A protocol for* k*-anonymization by suppressing entries* leaks only $\mathcal{F}(T)$ *if there exist probabilistic polynomial-time algorithms (called* simulators*)* M *and* $M_1, \ldots, M_N$ *such that*

$$\{M(\mathsf{keys}_{\mathsf{miner}}, \mathsf{Anonymized}(T), \mathcal{F}(T))\}_T \stackrel{\mathsf{c}}{\equiv} \{\mathsf{view}_{\mathsf{miner}}(T)\}_T,$$

and that, for any $i \in [1, N]$*,*

$$\{M_i(\mathsf{keys}_i, R_i, \mathsf{Anonymized}(T), \mathcal{F}(T))\}_T \stackrel{\mathsf{c}}{\equiv} \{\mathsf{view}_i(T)\}_T,$$

where $\mathsf{keys}_{\mathsf{miner}}$ *(*keys_i*, resp.) denotes the miner's (respondent* i*'s, resp.) preloaded key shares (if any).*

5.3 Protocol for Formulation 1

In this section, we solve the first formulation of the problem. That is, we design a protocol that privately extracts the k-anonymous part of a table. The basic idea of our design is that each respondent encrypts her sensitive attributes using an encryption key that can be derived if and only if there are at least k rows whose quasi-identifiers are equal. Specifically, the key to encrypt the sensitive attributes $(a_1^{(i)}, \ldots, a_n^{(i)})$ is a function of the corresponding quasi-identifier $(s_1^{(i)}, \ldots, s_m^{(i)})$ and it is shared among the respondents with threshold k (which means that at least k respondents are needed to recover the key). Each respondent submits to the miner one share of the key(s) corresponding to her quasi-identifier. As a result, if and only if there are at least k respondents whose quasi-identifiers are equal, the miner is able to recover a key.

To design the encryption scheme we use a $(2N, k)$-Shamir secret sharing of a "seed" key x, where each respondent i has two shares x_{2i-1} and x_{2i} of the seed key: the shares owned by respondent i are $x_{2i-1} = \mathcal{P}(2i-1)$ and $x_{2i} = \mathcal{P}(2i)$. To encrypt the sensitive attributes $(a_1^{(i)}, \ldots, a_n^{(i)})$, we use $(H(s_1^{(i)}, \ldots, s_m^{(i)}))^{x_{2i-1}}$ as the encryption key, where H is a cryptographic hash function. Clearly, this key can be derived if and only if for k or more values of j, $(H(s_1^{(i)}, \ldots, s_m^{(i)}))^{x_j}$ is available. The key share submitted by respondent i is $(H(s_1^{(i)}, \ldots, s_m^{(i)}))^{x_{2i}}$. Consequently, this submitted key share can actually be used in the recovery of any keys used to encrypt the sensitive attributes where the quasi-identifiers are equal to $(s_1^{(i)}, \ldots, s_m^{(i)})$. These keys can be recovered successfully if and only if there are at least k respondents with quasi-identifier equal to $(s_1^{(i)}, \ldots, s_m^{(i)})$. Furthermore, even the respondents having the same quasi-identifier cannot figure out each other's keys because they do not know other respondents' shares of the seed key.

5.3.1 Protocol

Let S be a security parameter, let p, q be two S-bit primes such that $p = 2q + 1$, let G_q be the quadratic residue subgroup of $\mathbb{Z}_p^\times$ (the multiplicative group mod p), and let H be a cryptographic hash function with range G_q.

Before the protocol starts, we assume that a "seed" key $x \in [1, q-1]$ is shared among

respondents using $(2N, k)$-Shamir secret sharing and that each respondent i has two shares, x_{2i-1} and x_{2i}. Specifically, there exists a degree-$(k-1)$ polynomial $\mathcal{P}()$ such that $x = \mathcal{P}(0)$ and $\forall i \in [1, 2N]$, $x_i = \mathcal{P}(i)$.

Data Submission. Respondent i encrypts $(a_1^{(i)}, \ldots, a_n^{(i)})$ using the symmetric key $y_{2i-1} = (H(s_1^{(i)}, \ldots, s_m^{(i)}))^{x_{2i-1}}$. Then she sends the miner the ciphertext together with $(s_1^{(i)}, \ldots, s_m^{(i)})$ and $y_{2i} = (H(s_1^{(i)}, \ldots, s_m^{(i)}))^{x_{2i}}$.

Data Processing. When the miner has collected all respondents' messages, he counts the number of rows (respondents) for each different value of $(s_1, \ldots, s_m)$. If for some value of $(s_1, \ldots, s_m)$ there are k or more rows, then he decrypts the sensitive attributes of these rows by computing the key y_{2j-1}.

Let I be a subset of k such rows and $H(s_1, \ldots, s_m) = h$, then $y_{2j-1} = h^{x_{2j-1}}$. According to Lagrange interpolation, we have

$$x_{2j-1} = \sum_{i \in I} x_{2i} \prod_{\ell \neq i, \ell \in I} (2j - 1 - 2\ell)/(2i - 2\ell)$$

Because of $h^{x_{2j-1}} = y_{2j-1}$, we have

$$y_{2j-1} = \prod_{i \in I} y_{2i}^{\prod_{\ell \neq i, \ell \in I} (2j-1-2\ell)/(2i-2\ell)}.$$

By using respondent j's symmetric key y_{2j-1}, the miner decrypts the sensitive attributes of these rows using the computed keys.

5.3.2 Protocol Analysis

Lemma 5.3.1 *Under the DDH assumption,*

$$\{g_1, g_2, g_3, g_1^{e_1}, g_2^{e_2}, g_3^{\alpha e_1 + \beta e_2}\}_q \stackrel{\mathsf{c}}{\equiv} \{g_1, g_2, g_3, g_1^{e_1}, g_2^{e_2}, g_3^{e_3}\}_q,$$

where g_1, g_2, g_3 are picked uniformly and independently from G_q, and e_1, e_2, e_3 are picked uniformly and independently from $[1, q-1]$.

Proof: Suppose by way of contradiction that the above indistinguishability result does not hold. Then there exist a probabilistic polynomial-time algorithm $\mathcal{D}$ and a polynomial $f()$ such that, for infinitely many q,

$$\begin{aligned} &|\Pr[\mathcal{D}(g_1, g_2, g_3, g_1^{e_1}, g_2^{e_2}, g_3^{\alpha e_1 + \beta e_2}, q) = 1] \\ &\quad - \Pr[\mathcal{D}(g_1, g_2, g_3, g_1^{e_1}, g_2^{e_2}, g_3^{e_3}, q) = 1]| \quad \geq \quad 1/f(S). \end{aligned}$$

Thus, we can construct another polynomial-time algorithm $\mathcal{D}'()$ such that

$$\mathcal{D}'(E_1, E_2, E_3, q, g) \stackrel{\mathsf{def}}{=} \mathcal{D}(g, E_1, g^{e'}, E_2, E_3^{e''}, E_2^{e'(\alpha + \beta e'')}, q),$$

where e', e'' are picked uniformly and independently from $[1, q-1]$. Then clearly, for $\hat{e}_1, \hat{e}_2$ uniformly and independently picked from $[1, q-1]$, we have

$$\begin{aligned} &\mathcal{D}'(g_1^{\hat{e}_1}, g_1^{\hat{e}_2}, g_1^{\hat{e}_1 \hat{e}_2}, q, g_1) \\ &\quad = \quad \mathcal{D}(g_1, g_1^{\hat{e}_1}, g_1^{e'}, g_1^{\hat{e}_2}, g_1^{\hat{e}_1 \hat{e}_2 e''}, g_1^{\hat{e}_2 e'(\alpha + \beta e'')}, q) \\ &\quad = \quad \mathcal{D}(g_1, g_2, g_3, g_1^{\hat{e}_2}, g_2^{\hat{e}_2 e''}, g_3^{\hat{e}_2(\alpha + \beta e'')}, q). \end{aligned}$$

The last identity holds because $g_1^{\hat{e}_1}$ and $g_1^{e'}$ are independent and uniform and we can rename them to g_2 and g_3. We then further rename $\hat{e}_2$ and $\hat{e}_2 e''$ to e_1 and e_2 to obtain

$$\mathcal{D}'(g_1^{\hat{e}_1}, g_1^{\hat{e}_2}, g_1^{\hat{e}_1 \hat{e}_2}, q, g_1) = \mathcal{D}(g_1, g_2, g_3, g_1^{e_1}, g_2^{e_2}, g_3^{\alpha e_1 + \beta e_2}, q).$$

Similarly, we can obtain, for $\hat{e}_1, \hat{e}_2, \hat{e}_3$ uniformly and independently picked from $[1, q-1]$,

$$\begin{aligned}
&\mathcal{D}'(g_1^{\hat{e}_1}, g_1^{\hat{e}_2}, g_1^{\hat{e}_3}, q, g_1) \\
&= \mathcal{D}(g_1, g_1^{\hat{e}_1}, g_1^{e'}, g_1^{\hat{e}_2}, g_1^{\hat{e}_3 e''}, g_1^{\hat{e}_2 e'(\alpha+\beta e'')}, q) \\
&= \mathcal{D}(g_1, g_2, g_3, g_1^{\hat{e}_2}, g_2^{(\log_{g_2} g_1)\hat{e}_3 e''}, g_3^{\hat{e}_2(\alpha+\beta e'')}, q),
\end{aligned}$$

by renaming $g_1^{\hat{e}_1}$ and $g_1^{e'}$ to g_2 and g_3. Further renaming $\hat{e}_2$, $(\log_{g_2} g_1)\hat{e}_3 e''$, and $\hat{e}_2(\alpha + \beta e'')$ to e_1, e_2, and e_3, respectively, we obtain

$$\mathcal{D}'(g_1^{\hat{e}_1}, g_1^{\hat{e}_2}, g_1^{\hat{e}_3}, q, g_1) = \mathcal{D}(g_1, g_2, g_3, g_1^{e_1}, g_2^{e_2}, g_3^{e_3}, q).$$

Therefore,

$$|\Pr[\mathcal{D}'(g_1^{\hat{e}_1}, g_1^{\hat{e}_2}, g_1^{\hat{e}_1\hat{e}_2}, q, g_1) = 1] - \Pr[\mathcal{D}'(g_1^{\hat{e}_1}, g_1^{\hat{e}_2}, g_1^{\hat{e}_3}, q, g_1) = 1]| \geq 1/f(S),$$

an obvious contradiction to the DDH assumption. ■

Theorem 5.3.2 *Under the DDH assumption, the protocol for extracting $\mathcal{K}(T)$ is private.*

Proof: We only need to construct a simulator M for the miner because the respondents do not receive any messages from the miner. The simulator simply outputs the party's data, preloaded key shares (if any) and coin flips.

M picks $x' \in [0, q-1]$ uniformly at random and computes $2N$ Shamir shares of x': $x'_1, \ldots, x'_{2N}$. If respondent i's row is in $\mathcal{K}(T)$, then M computes $y'_{2i-1} = (H(s_1^{(i)}, \ldots, s_m^{(i)}))^{x'_{2i-1}}$, $y'_{2i} = (H(s_1^{(i)}, \ldots, s_m^{(i)}))^{x'_{2i}}$, and encrypts $(a_1^{(i)}, \ldots, a_n^{(i)})$ using the symmetric key y_{2i-1}. M simulates respondent i's message with the above symmetric encryptions $(s_1^{(i)}, \ldots, s_m^{(i)})$ and y'_{2i}.

If respondent i's row is not in $\mathcal{K}(T)$, then M still computes $y'_{2i} = (H(s_1^{(i)}, \ldots, s_m^{(i)}))^{x'_{2i}}$. M simulates respondent i's message with a random ciphertext in the symmetric encryption scheme, $(s_1^{(i)}, \ldots, s_m^{(i)})$, and y'_{2i}.

Based on Lemma 5.3.1, it is conceptually trivial to extend to the indistinguishability between the views of the simulator and the protocol. ■

5.4 Protocol for Formulation 2

In this section, we solve the second formulation of the problem. That is, we provide a protocol that privately k-anonymizes a distributed table by suppressing entries. Our protocol is based on Meyerson and Williams's algorithms (which we refer to as MW) for k-anonymizing a database [79]. Our solution can be viewed as a distributed, privacy-preserving, version of their algorithm. Our protocol provides quantifiable, though not ideal, privacy. Namely, it keeps all information about the suppressed entries private from each individual party, except for revealing the distance between each pair of rows.

Our protocol consists of three phases. In the first phase, the protocol allows the miner to compute the distance between each pair of rows. In the second phase, the miner uses the MW algorithm to compute a *k-partition* of the table. (A k-partition is a collection of disjoint subsets of rows in which each subset contains at least k rows and the union of these subsets is the entire table.) In the third phase, the protocol allows the miner to compute the k-anonymized table. The second phase is a direct computation of part of MW (which relies only on the inter-row distances already known to the miner).

We provide an overview for the more complex first and third phases; we describe all three phases in complete detail later in Section 5.4.1.

Design of Phase 1

The distance between two rows is the number of quasi-identifier attributes in which the rows have different values [79]. We define

$$\sigma_j^{(i,i')} = \begin{cases} 1 & \text{if } s_j^{(i)} = s_j^{(i')} \\ r & \text{if } s_j^{(i)} \neq s_j^{(i')} \end{cases}$$

where r is a random element uniformly picked from an exponentially large prime-order cyclic group. Because the probability of $r = 1$ is negligible, the distance between the ith and i'th rows equals $|\{j : \sigma_j^{(i,i')} \neq 1, j \in [1, m]\}|$ with all but negligible probability. To compute this distance, the miner first computes encryptions of the $\sigma_j^{(i,i')}$s from encryptions of quasi-identifier attributes; then, a respondent rerandomizes and repermutes these encryptions so that the miner does not learn the value of any specific $\sigma_j^{(i,i')}$ when they are decrypted; finally, the respondents jointly help the miner to decrypt the $\sigma_j^{(i,i')}$s.

We once again use the ElGamal encryption scheme in our protocol. To allow the miner to compute encryptions of $\sigma_j^{(i,i')}$s, we use the fact that, because the cyclic group mentioned above is of a prime order, it follows that

$$\sigma_j^{(i,i')} = (s_j^{(i)} / s_j^{(i')})^{e_{i,i',j}}, \tag{5.4.0.1}$$

where $e_{i,i',j}$ is a uniformly random exponent. This technique was first used in [11]. Equation (5.4.0.1) holds because, if $s_j^{(i)} \neq s_j^{(i')}$, then $s_j^{(i)} / s_j^{(i')} \neq 1$. Any element of a prime-order cyclic group not equal to 1 is a generator; and a generator raised to a uniformly random exponent must be a uniformly random element of the cyclic group. Because of the multiplicative homomorphic property of ElGamal, it is easy for the miner to compute the encryption of the $\sigma_j^{(i,i')}$s using the encryptions of the quasi-identifier attributes.

The remaining question is how the respondents jointly help the miner to decrypt ciphertexts of the $\sigma_j^{(i,i')}$s. We use a *threshold cryptography* technique similar to that of Desmedt and Frankel [35]. Assume that the private key is shared among the respondents using a (N, t)-Shamir secret sharing [93], where t is an arbitrary threshold. Then a respondent can compute a "partial decryption" by raising the second component of an ElGamal ciphertext to her share of the private key. To compute the plaintext, the miner only needs to take t partial decryptions and interpolate them.

Design of Phase 3

Let $\mathcal{P}$ be the k-partition computed in the second phase. Let $P_\ell \in \mathcal{P}$. Suppose that the ith row is in P_ℓ. According to MW, respondent i should replace $s_j^{(i)}$ with $*$ if and only if

$$\exists i' \in P_\ell, s_j^{(i)} \neq s_j^{(i')}.$$

With high probability, this is equivalent to

$$\prod_{i' \in P_\ell, i' \neq i} \sigma_j^{(i,i')} \neq 1.$$

Because ElGamal is multiplicative homomorphic, we can easily compute an encryption of $\prod_{i' \in P_\ell, i' \neq i} \sigma_j^{(i,i')}$. Therefore, the remaining technical question is how other respondents jointly help respondent i decrypt it. To achieve this goal, we use the technique of partial decryptions from the first phase; the main difference is that respondent i only needs the help of $t-1$ other respondents, because respondent i herself already has a share of the private key.

5.4.1 Protocol

We now give a detailed description of the entire protocol.As in Section 5.3.1, let S is a security parameter, that p, q are S-bit primes such that $p = 2q + 1$, and that G_q is the quadratic residue subgroup of $\mathbb{Z}_p^\times$. Let $t \in [2, N-1]$ be a threshold. In this section, we assume that N respondents share a private key $x \in [0, q-1]$ using (N, t)-Shamir secret sharing, where respondent i's share is denoted by x_i. Specifically, there exists a degree-$(t-1)$ polynomial $\mathcal{P}()$ such that $x = \mathcal{P}(0)$ and $\forall i \in [1, N]$, $x_i = \mathcal{P}(i)$. We also assume that the corresponding public key $y = g^x$ (where g is a generator of G_q) is known to all involved parties (respondents and the miner).

Phase 1

In this phase, the miner computes the distance between every pair of rows, following the method overviewed earlier.

Submission of Encrypted Quasi-identifier Attributes. Each respondent i encrypts each of her quasi-identifier attributes using ElGamal with public key y (for $j = 1$ to m):

$$\overline{s}_j^{(i)} = (s_j^{(i)} y^{r_{ij}}, g^{r_{ij}}),$$

where r_{ij} is picked uniformly at random from $[1, q-1]$. Then the respondents send all these encryptions to the miner. The ciphertext $\overline{s}_j^{(i)}$ above has two components; we denote the first and the second components by $\overline{s}_j^{(i)}[1]$ and $\overline{s}_j^{(i)}[2]$ respectively.

Computing Encryptions of $\sigma_j^{(i,i')}$. For each pair (i, i'), the miner computes the quotients of their corresponding quasi-identifier attributes for $j = 1$ to m:

$$q_j^{(i,i')} = (\overline{s}_j^{(i)}[1]/\overline{s}_j^{(i')}[1], \overline{s}_j^{(i)}[2]/\overline{s}_j^{(i')}[2]).$$

Then the miner raises the quotients to random powers: (for $j = 1$ to m)

$$p_j^{(i,i')} = ((q_j^{(i,i')}[1])^{e_{i,i',j}}, (q_j^{(i,i')}[2])^{e_{i,i',j}}),$$

where $e_{i,i',j}$ is picked uniformly at random from $[1, q-1]$.

Rerandomization and Repermutation. The miner sends $\{p_j^{(i,i')} : i, i' \in [1, N], i \neq i', j \in [1, m]\}$ to an arbitrary respondent i_0. Respondent i_0 rerandomizes each $p_j^{(i,i')}$ and repermutes $(p_1^{(i,i')}, \ldots, p_m^{(i,i')})$ for each pair (i, i'). Denote the result of the above rerandomization and repermutation operations by $\{u_j^{(i,i')} : i, i' \in [1, N], i \neq i', j \in [1, m]\}$. Then respondent i_0 sends $\{u_j^{(i,i')} : i, i' \in [1, N], i \neq i', j \in [1, m]\}$ back to the miner.

Decrypting $\sigma_j^{(i,i')}$. Consider a set I of t respondents, where $i_0 \notin I$. To each of these t respon-

dents, the miner sends $\{u_j^{(i,i')}[2] : i, i' \in [1, N], i \neq i', j \in [1, m]\}$. For each (i, i', j) such that $i, i' \in [1, N], i \neq i', j \in [1, m]$, each picked respondent $i'' \in I$ raises all elements she receives to the $x_{i''}$th power:

$$v_{j,i''}^{(i,i')} = (u_j^{(i,i')}[2])^{x_{i''}}.$$

Then she sends $\{v_{j,i''}^{(i,i')} : i, i' \in [1, N], i \neq i', j \in [1, m]\}$ back to the miner.

Finally, the miner computes

$$\hat{\sigma}_j^{(i,i')} = u_j^{(i,i')}[1] / \prod_{i'' \in I} (v_{j,i''}^{(i,i')})^{\prod_{\ell \neq i'', \ell \in I} \ell/(\ell - i'')}.$$

Note that $\hat{\sigma}_1^{(i,i')}, \ldots, \hat{\sigma}_m^{(i,i')}$ is nothing but a permutation of $\sigma_1^{(i,i')}, \ldots, \sigma_m^{(i,i')}$. Thus, $|\{j : \sigma_j^{(i,i')} \neq 1, j \in [1, m]\}| = |\{j : \hat{\sigma}_j^{(i,i')} \neq 1, j \in [1, m]\}|$. For each pair (i, i'), the miner computes the distance between ith and i'th rows as $|\{j : \hat{\sigma}_j^{(i,i')} \neq 1, j \in [1, m]\}|$.

Phase 2

In this phase, knowing the pairwise distances of the rows, the miner follows the first part of the MW algorithm to compute a k-partition $\mathcal{P} = \{P_1, \ldots, P_L\}$.

Phase 3

In this phase, the miner computes the k-anonymized table with the help of the respondents, as overviewed earlier.

Computing Encryptions of $\prod_{i' \in \mathcal{P}_l, i' \neq i} \sigma_j^{(i,i')}$. For each $P_\ell \in \mathcal{P}$, each $i \in P_\ell$, each $j \in [1, m]$, the miner computes

$$p_j^{(i)} = \left(\prod_{i' \in P_\ell, i' \neq i} p_j^{(i,i')}[1], \prod_{i' \in P_\ell, i' \neq i} p_j^{(i,i')}[2] \right)$$

.

Decrypting $\prod_{i' \in \mathcal{P}_\ell, i' \neq i} \sigma_j^{(i,i')}$. Then, for each P_ℓ, let I_ℓ be a set of $t - 1$ respondents such that

$i_0 \notin I_\ell$ and $I_\ell \cap P_\ell = \emptyset$. The miner sends $\{p_j^{(i)}[2] : i \in P_\ell, j \in [1,m]\}$ to every respondent in I_ℓ. Each respondent $i' \in I_\ell$ computes (for each $i \in P_\ell$ and each $j \in [1,m]$)

$$w_j^{(i,i')} = (p_j^{(i)}[2])^{x_{i'}},$$

and sends $\{w_j^{(i,i')} : i \in P_\ell, j \in [1,m]\}$ back to the miner.

The miner computes (for each P_ℓ, each $i \in P_\ell$ and each $j \in [1,m]$)

$$z_j^{(i)} = p_j^{(i)}[1] / \prod_{i' \in I_\ell} (w_j^{(i,i')})^{\prod_{i'' \in I_\ell, i'' \neq i'} i''/(i''-i')}.$$

He sends $\{z_j^{(i)} : j \in [1,m]\}$, $\{p_j^{(i)}[2] : j \in [1,m]\}$, and P_ℓ to each respondent i.

For each $j \in [1,m]$, each respondent $i (\in P_\ell)$ computes

$$\hat{z}_j^{(i)} = p_j^{(i)}[2]^{x_i \prod_{i'' \in I_\ell, i'' \neq i} i''/(i''-i)},$$

and compares $\hat{z}_j^{(i)}$ with $z_j^{(i)}$. If they are equal, then respondent i sets $\hat{s}_j^{(i)} = s_j^{(i)}$; otherwise, respondent i sets $\hat{s}_j^{(i)} = *$. Finally, respondent i sends $(\hat{s}_1^{(i)}, \ldots, \hat{s}_m^{(i)}, a_1^{(i)}, \ldots, a_n^{(i)})$ to the miner.

5.4.2 Protocol Analysis

Since our protocol follows exactly the steps of MW, the table computed in our protocol is exactly the same as MW's. In this sense, our protocol correctly computes the k-anonymized table. Our protocol leaks only the distance between each pair of rows. Let $\mathsf{Distance}(T, i, i')$ denote the distance between the ith and i'th rows of table T.

Theorem 5.4.1 *Under the DDH assumption, the protocol of k-anonymization by suppressing entries leaks only* $\{\mathsf{Distance}(T, i, i') : i, i' \in [1,N], i \neq i'\}$.

Proof: We first construct a simulator M for the miner. For the phase of computing pairwise distances, M simulates the first-round messages the miner receives using mN random ElGamal ciphertexts. For the second-round messages the miner receives (which are from i_0), M simulates each $u_j^{(i,i')}$ using a random ElGamal ciphertext ${u'}_j^{(i,i')}$. Then, M picks $i_0'' \in I$; for each

$i'' \neq i''_0$, $i'' \in I$, M simulates the third round message $v_{j,i''}^{(i,i')}$ using a random element $v'^{(i,i')}_{j,i''}$ of G_q. To simulate $v_{j,i''_0}^{(i,i')}$, M first sets the values of $\sigma'^{(i,i')}_j$: for each pair (i, i'), the m variables $\{\sigma'^{(i,i')}_j : j \in [1, m]\}$ have exactly $m - \mathsf{Distance}(T, i, i')$ 1's; M randomly picks this number of variables and sets them to 1 and then sets all the remaining variables randomly. After that, M simulates $v_{j,i''_0}^{(i,i')}$ using

$$v'^{(i,i')}_{j,i''_0} = \frac{u'^{(i,i')}_j[1]}{\sigma'^{(i,i')}_j \prod_{i'' \in I, i'' \neq i''_0} (v'^{(i,i')}_{j,i''})^{\prod_{\ell \neq i'', \ell \in I} \frac{\ell}{\ell - i''}}}.$$

For the phase of computing anonymized data, M simulates the first-round messages the miner receives using $Nm(t-1)$ independent random elements of G_q. M simulates the last-round messages using the rows in $\mathsf{Anonymized}(T)$.

The computational indistinguishability follows from the semantic security of ElGamal encryption, which is well known to hold under DDH [16].

Now we construct a simulator M_i for respondent i. If $i = i_0$, then M_i simulates the messages received by i_0 using $N(N-1)m$ random ElGamal ciphertexts. If $i \in I$, then M_i simulates the messages received by i in the phase of computing pairwise distances using $N(N-1)m$ independent random elements of G_q. If $i \in I_\ell$, then M_i simulates the first-round messages received by i in the phase of computing anonymized data using independent random elements of G_q. For any respondent i, for the last-round messages respondent i receives, M_i simulates each $p_j^{(i)}[2]$ using an independent random element $p'^{(i)}_j$ of G_q; if the (i, j)-entry of the anonymized sensitive attributes is $*$, then M_i simulates $z_j^{(i)}$ using an independent random element of G_q as well; otherwise, M_i simulates $z_j^{(i)}$ using

$$z'^{(i)}_j = (p'^{(i)}_j)^{x_i \prod_{i'' \in I_\ell, i'' \neq i} i''/(i''-i)}.$$

The computational indistinguishability follows immediately from the semantic security of ElGamal encryption. ■

5.5 Discussion

In this chapter, we have studied how to create k-anonymous tables in a distributed scenario without the need for a central authority and while maintaining respondent privacy. We formulated the problem in two ways.

For the first problem formulation, a protocol must extract the k-anonymous part of a table. The major advantage of our protocol is that it is non-interactive—each respondent only sends a single flow of communication to the miner. Therefore, respondents can "submit data and go". Another advantage is that the solution is very efficient. The dominating computational overhead of each respondent is two modular exponentiations; the dominating computational overhead of the miner is kN_k modular exponentiations, where N_k is the number of rows in the k-anonymous part of the table. The limitation of this problem formulation is that it is suitable only if the original table is already close to k-anonymous, as otherwise the subset of the table learned by the miner may not be of sufficient utility.

For the second problem formulation, a protocol k-anonymizes a table by suppressing entries. The advantage of this approach is that it can produce useful results even when the original table is not close to k-anonymous. Our solution to this problem formulation leaks a small amount of information beyond the k-anonymous result—namely, the distance between each pair of rows. Consequently, this approach is a good choice for the applications in which revealing the distances between rows can be tolerated. Our solution is based on one of the existing k-anonymization algorithm. How to design efficient privacy-preserving distributed algorithms of other k-anonymization algorithms is still open.

Chapter 6

Privacy-Preserving Learning of Bayesian Networks on Vertically Partitioned Data

6.1 Introduction

Data mining is a very powerful tool for knowledge discovery from large amounts of data. Traditionally, data mining requires all data to be gathered into a central site where specific mining algorithms can be applied on the joined data. This model works in many data mining settings. However, many data are currently collected and maintained by different parties, so it is undesirable from a privacy perspective to bring all data into a central site from different parties. Privacy-preserving distributed data mining solutions provide data mining algorithms that compute or approximate the output of a particular algorithm applied to the joint data, while protecting other information about the data.

Bayesian networks are powerful data mining tools. Bayesian networks can be used for many tasks, such as hypothesis testing and automated scientific discovery. In this chapter, we present privacy-preserving solutions for learning Bayesian networks on a database vertically partitioned between two parties. Using existing cryptographic primitives, we design several

privacy-preserving protocols. We compose them to compute Bayesian networks in a privacy-preserving manner. Our solution computes an approximation of the existing *K2* algorithm for learning the structure of the Bayesian network and computes the accurate parameters. In our solution, the two parties learn only the final Bayesian network and the order in which network edges were added. Based on the security of the cryptographic primitives used, it is provable that no other information is revealed to the parties about each other's data. More precisely, each party learns no information that is not implied by this output and his or her own input.

6.2 A Brief Review

In this section, we give an introduction to Bayesian networks, the *K2* algorithm for learning a Bayesian network from a set of data, and some cryptographic tools used to design our protocol.

6.2.1 Bayesian Networks

A Bayesian network is a graphical model that encodes probabilistic relationships among variables of interest [30]. This model can be used for data analysis and is widely used in data mining applications. Formally, a Bayesian network for a set V of m variables is a pair $(B_{\mathrm{s}}, B_{\mathrm{p}})$. The *network structure* $B_{\mathrm{s}} = (V, E)$ is a directed acyclic graph whose nodes are the set of variables. The *parameters* B_{p} describe local probability distributions associated with each variable. The graph B_{s} represents conditional independence assertions about variables in V: an edge between two nodes denotes direct probabilistic relationships between the corresponding variables. Together, B_{s} and B_{p} define the joint probability distribution for V. Throughout this chapter, we use v_i to denote both the variable and its corresponding node. We use π_i to denote the parents of node v_i in B_{s}. The absence of an edge between v_i and v_j denotes conditional independence between the two variables given the values of all other variables in the network.

For bookkeeping purposes, we assume there is a canonical ordering of variables and their possible instantiations, which can be extended in the natural way to sets of variables. We denote the jth unique instantiation of V by V_j. Similarly, we denote the kth instantiation of a variable v_i by v_{i_k}. Given the set of parent variables π_i of a node v_i in the Bayesian network structure

B_s, we denote the jth unique instantiation of π_i by ϕ_{ij}. We denote the number of unique instantiations of π_i by q_i and the number of unique instantiations of v_i by d_i.

Given a Bayesian network structure B_s, the joint probability for any particular instantiation V_ℓ of all the variables is given by

$$\Pr[V = V_\ell] \quad = \quad \prod_{v_i \in V} \Pr[v_i = v_{i_k} \mid \pi_i = \phi_{ij}],$$

where each k and j specify the instantiations of the corresponding variables as determined by V_ℓ. The network parameters $B_p = \{\Pr[v_i = v_{i_k} \mid \pi_i = \phi_{ij}] : v_i \in V,\ 1 \leq j \leq q_i,\ 1 \leq k \leq d_i\}$ are the probabilities corresponding to the individual terms in this product. If variable v_i has no parents, then its parameters specify the marginal distribution of v_i; that is, $\Pr[v_i = v_{i_k} \mid \pi_i = \phi_{ij}] = \Pr[v_i = v_{i_k}]$

6.2.2 *K2* Algorithm

Determining the BN structure that best represents a set of data is NP-hard [27], so heuristic algorithms are typically used in practice. One of the most widely used structure-learning algorithms is the *K2* algorithm [30], which we use as the starting point of our distributed privacy-preserving algorithm. The *K2* algorithm is a greedy heuristic approach to efficiently determining a Bayesian network representation of probabilistic relationships between variables from a data set containing observations of those variables.

The *K2* algorithm starts with a graph consisting of nodes representing the variables of interest, with no edges. For each node in turn, it then incrementally adds edges whose addition most increases the score of the graph, according to a specified score function. When the addition of no single parent can increase the score or a specified limit of parents has been reached, this algorithm stops adding parents to that node and moves onto the next node.

In the *K2* algorithm, the number of parents for any node is restricted to some maximum u. Given a node v_i, $\mathsf{Pred}(v_i)$ denotes all the nodes less than v_i in the node ordering. D is a database of n records, where each record contains a value assignment for each variable in V. The *K2* algorithm constructs a Bayesian network structure B_s whose nodes are the variables in

V.

More generally, we define α_{ijk} to be the number of records in D in which variable v_i is instantiated as v_{i_k} and π_i is instantiated as ϕ_{ij}. Similarly, we define α_{ij} to be the number of records in D in which π_i is instantiated as ϕ_{ij}. We note that therefore

$$\alpha_{ij} = \sum_{k=1}^{d_i} \alpha_{ijk} \tag{6.2.2.1}$$

In constructing the BN structure, the *K2* algorithm uses the following score function $f(i, \pi_i)$ to determine which edges to add to the partially completed structure:

$$f(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(d_i - 1)!}{(\alpha_{ij} + d_i - 1)!} \prod_{k=1}^{d_i} \alpha_{ijk}! \tag{6.2.2.2}$$

We refer to all possible α_{ijk} and α_{ij} that appear in Equation 6.2.2.2 as α-*parameters*. The *K2* algorithm [30] is as follows:

Input: an ordered set of m nodes, an upper bound u on the number of parents for a node, and a database D containing n records.

Output: Bayesian network structure B_s (whose nodes are the m input nodes, and whose edges are as defined by the values of π_i at the end of the computation)

For $i = 1$ **to** m

{

$\pi_i = \emptyset$;

$P_{\text{old}} = f(i, \pi_i)$;

KeepAdding = true;

While KeepAdding and $|\pi_i| < u$

{

let z be the node in $\mathsf{Pred}(x_i) - \pi_i$ that maximizes $f(i, \pi_i \cup \{z\})$;

$P_{\text{new}} = f(i, \pi_i \cup \{z\})$;

If $P_{\text{new}} > P_{\text{old}}$

$P_{\text{old}} = P_{\text{new}}$;

$\pi_i = \pi_i \cup \{z\}$;

Else KeepAdding = false;

}

}

6.2.3 Cryptographic Tools

Composition of Privacy-Preserving Protocols. In our solution, we use a composition of privacy-preserving subprotocols in which all intermediate outputs from one subprotocol that are inputs to the next subprotocol are computed as secret shares. In this way, it can be shown that if each subprotocol is privacy-preserving, then the resulting composition is also privacy-preserving [51, 20]. A fully fleshed-out proof of these results requires showing simulators which relate the information available to the parties in the actual computation to the information they could obtain in the ideal model.

Privacy-Preserving Scalar Product Share Protocol. The scalar product of two vectors $\mathbf{z} = (z_1, \ldots, z_n)$ and $\mathbf{z}' = (z'_1, \ldots, z'_n)$ is $\mathbf{z} \cdot \mathbf{z}' = \sum_{i=1}^{n} z_i z'_i$. In a privacy-preserving scalar product share protocol, both parties hold each vector respectively, and both parties learn secret shares of the product result. We only require use of the scalar product protocol for binary data, even if the database consists of non-binary data. This can be done with cryptographic privacy—based on any additive homomorphic encryption scheme [21, 98, 48], such as the Paillier encryption scheme [83], which is secure assuming that it is computationally infeasible to determine composite residuosity classes. The protocol produces two shares' whose sum modulo N (where N is appropriately related to the modulus used in the encryption scheme) is the target scalar product. To avoid the modulus introducing differences in computations, the modulus should be larger than the largest possible outcome of the scalar product.

$\ln x$ and $x \ln x$ Protocols. Lindell and Pinkas designed an efficient two-party privacy-preserving protocol for computing $x \ln x$ [75]. In the protocol, two parties have inputs v_1 and v_2, respec-

tively, and we define $x = v_1 + v_2$. The output for this protocol is that two parties obtain random values w_1 and w_2, respectively, such that $w_1 + w_2 = x \ln x$. With the same techniques, the two parties can also compute secret shares for $\ln x$. Both protocols are themselves privacy-preserving and produce secret shares as their results.

6.3 Problem Formalization

In the ideal model for privacy-preserving Bayesian networks, two parties send their databases to a trusted third party (TTP). The TTP then applies a Bayesian network learning algorithm on the combination of the two databases. Finally, the learned BN model is sent to the two parties by the trusted third party. In the ideal model, the two parties only learn the global BN (their objective) and nothing else. A distributed computation that does not make use of a TTP is then said to be secure if the parties learn nothing about each other's data during the execution of the protocol that they would not learn in the ideal model.

We design a privacy-preserving solution for two parties to learn a BN using a secure distributed computation. The parties should learn nothing more than in the ideal model. In our case, in order to obtain security with respect to an ideal model, we must also allow the ideal model to reveal the order in which an iterative algorithm adds edges to the Bayesian network (as this is revealed to Alice and Bob in our solution). We suppose the parties in our setting are semi-honest adversaries. That is, they correctly follow their specified protocol, but they keep a record of all intermediate computation and passed messages and may use those to attempt to learn information about each other's inputs.

In the distributed two-party setting we consider, a database D consists only of discrete variables, and D is vertically partitioned among Alice and Bob. Alice and Bob hold confidential databases D_A and D_B, respectively, each of which can be regarded as a relational table. Each database has n rows. The variable sets in D_A and D_B are denoted by V_A and V_B, respectively. There is a common ID that links the rows in two databases owned by those two parties. Without loss of generality, we assume that the row index is the common ID that associates the two databases—that is, Alice's rows and Bob's rows represent the same records, in the same order,

but Alice and Bob each have different variables in their respective "parts" of the records. Thus, $D = D_A \bowtie D_B$ where $\bowtie$ represents equal join of two tables. Alice has D_A and Bob has D_B where D_A has the variables $V_A = \{a_1, ..., a_{m_a}\}$ and D_B has the variables $V_B = \{b_1, ..., b_{m_b}\}$. (The sets D_A and D_B are assumed to be disjoint). Hence, $m_a + m_b = m$, and the variable set is $V = V_A \cup V_B$. We assume the domains of databases D are public to both parties. We also assume the variables of interest are those in the set $V = V_A \cup V_B$. That is, Alice and Bob wish to compute the Bayesian network of the variables in their combined database $D_A \bowtie D_B$ without revealing any individual record and ideally not revealing any partial information about their own databases to each other except the information that can be derived from the final Bayesian network and their own database. As noticed earlier, our solution does reveal some partial information, in that it reveals order in which edges were added in the process of structure learning. The privacy of our solution is further discussed in Section 6.6.2.

6.4 Privacy-Preserving Bayesian Network Structure Protocol

In this section, we present a privacy-preserving protocol to learn the Bayesian network structure from a vertically partitioned database. We start in Sections 6.4.1 and 6.4.2 by describing a modification of the *K2* score function and providing some experimental results for it. We describe several new privacy-preserving subprotocols in Sections 6.4.3–6.4.6. In Section 6.4.7, we combine these into our overall privacy-preserving solution for Bayesian network structure. Bayesian network parameters are discussed in Section 6.5.

6.4.1 Score Function

We make a number of changes to the score function that appear not to substantially affect the outcome of the *K2* algorithm, and that result in a score function that works better for our privacy-preserving computation. Because the score function is only used for comparison purposes, we work instead with a different score function that has the same relative ordering. We then use an approximation to that score function. Specifically, we make the following three changes to the score function $f(i, \pi_i)$. We apply a natural logarithm, we take Stirling's approximation, and we

drop some bounded terms, described in more detail below.

First, we apply the natural logarithm to $f(i, \pi_i)$, yielding $f'(i, \pi_i) = \ln f(i, \pi_i)$ without affecting the ordering of different scores:

$$f'(i, \pi_i) = \sum_{j=1}^{q_i} (\ln(d_i - 1)! - \ln(\alpha_{ij} + d_i - 1)!) + \sum_{j=1}^{q_i} \sum_{k=1}^{d_i} \ln \alpha_{ijk}! \tag{6.4.1.1}$$

Next, we wish to apply Stirling's approximation on $f'(i, \pi_i)$. Recall that Stirling's approximation says that for any $\ell \geq 1$, we have $\ell! = \sqrt{2\pi\ell} \left(\frac{\ell}{e}\right)^{\ell} e^{\epsilon_\ell}$, where $\epsilon(\ell)$ is determined by Stirling's approximation and satisfies $\frac{1}{12\ell+1} < \epsilon(\ell) < \frac{1}{12\ell}$. However, if any α_{ijk} is equal to 0, then Stirling's approximation does not apply to $\alpha_{ijk}!$. As a solution, we note that if an α_{ijk} is changed from 0 to 1 in Equation 6.4.1.1, the outcome is unchanged because $1! = 0! = 1$. Hence, we replace any α_{ijk} that is 0 with 1. Specifically, we define $\beta_{ijk} = \alpha_{ijk}$ if α_{ijk} is not 0, and $\beta_{ijk} = 1$ if α_{ijk} is 0. Either way, we define $\beta_{ij} = \alpha_{ij}$. (This is simply so that we may entirely switch to using β's instead of having some β's and some α's). We refer to β_{ijk} and β_{ij} for all possible i, j, and k as *β-parameters*. Replacing α-parameters with β-parameters in Equation 6.4.1.1, we have

$$f'(i, \pi_i) = \sum_{j=1}^{q_i} (\ln(d_i - 1)! - \ln(\beta_{ij} + d_i - 1)!) + \sum_{j=1}^{q_i} \sum_{k=1}^{d_i} \ln \beta_{ijk}!$$

Taking $\ell_{ij} = \beta_{ij} + d_i - 1$, we apply Stirling's approximation to Equation 6.4.1, obtaining:

$$f'(i, \pi_i) \approx q_i \ln (d_i - 1)! + \frac{q_i(d_i - 1)}{2} \ln 2\pi +$$
$$\sum_{j=1}^{q_i} (\sum_{k=1}^{d_i} (\frac{1}{2} \ln \beta_{ijk} + \beta_{ijk} \ln \beta_{ijk} - \beta_{ijk} + \epsilon(\beta_{ijk})) - (\frac{1}{2} \ln \ell_{ij} + \ell_{ij} \ln \ell_{ij} - \ell_{ij} + \epsilon(\ell_{ij}))).$$

Finally, dropping the bounded terms $\epsilon_{\ell_{ij}}$ and $\epsilon_{\beta_{ijk}}$, pulling out $q_i(d_i - 1)$, and setting $\text{pub}(d_i, q_i) = q_i(d_i - 1) + q_i \ln (d_i - 1)! + \frac{q_i(d_i-1)}{2} \ln 2\pi$, we obtain our score function $g(i, \pi_i)$

that approximates the same relative ordering as $f(i, \pi_i)$

$$g(i, \pi_i) = \sum_{j=1}^{q_i}(\sum_{k=1}^{d_i}(\frac{1}{2}\ln \beta_{ijk} + \beta_{ijk}\ln \beta_{ijk}) - (\frac{1}{2}\ln \ell_{ij} + \ell_{ij}\ln \ell_{ij})) + \text{pub}(d_i, q_i). \quad (6.4.1.2)$$

A main component of our privacy-preserving *K2* solution is showing how to compute $g(i, \pi_i)$ in a privacy-preserving manner, as described in the remainder of Section 6.4. First, we provide some experimental results to provide some evidence that f and g produce similar results.

6.4.2 Experimental Results

We tested our score function on two different datasets in order to validate that it produces an acceptable approximation to the standard *K2* algorithm. The first dataset, called the Asia dataset, includes one million instances. It is generated from the commonly used Asia model. The Bayesian network for the Asia model is shown in Figure 6.1. This model has eight variables: Asia, Smoking, Tuberculosis, Lung cancer, Bronchitis, Either, X-ray, and Dyspnoea, denoted by {A, S, T, L, B, E, X, D}.

The second dataset is a synthetic dataset with $10,000$ instances including six variables denoted 0 to 5. All six variables are binary, either true or false. Variables 0, 1 and 3 were chosen uniformly at random. Variable 2 is the XOR of the variables 0 and 1. Variable 4 is the product of the variables 1 and 3. Variable 5 is the XOR of the variables 2 and 4.

On those two datasets, we tested the *K2* algorithms with both score functions f and g. For both the Asia dataset and the synthetic dataset, the *K2* algorithm generates the same structures whether f or g is used as the score function.

We further compare the difference of g and $\ln f$ for both datasets as computed by the *K2* algorithm. (Recall that g is our approximation to $\ln f$.) In total, the *K2* algorithm computes 64 scores on the Asia dataset and 30 scores on the synthetic dataset. Figure 6.2 shows the ratios of each g to the corresponding $\ln f$. The X-axis represents different variables, and the Y-axis represents the ratios of g to $\ln f$ that are computed for choosing the parents at each node. For instance, 14 scores for node D are computed to choose the parents of D. In the Asia model, all g scores

A	0.99	0.01						
S	0.5	0.5						
T	0.1	0.9	0.9	0.1				
L	0.3	0.6	0.7	0.4				
B	0.1	0.8	0.9	0.2				
E	0.9	0.1	0.1	0.01	0.1	0.9	0.9	0.9
X	0.2	0.6	0.8	0.4				
D	0.9	0.1	0.1	0.01	0.1	0.9	0.9	0.9

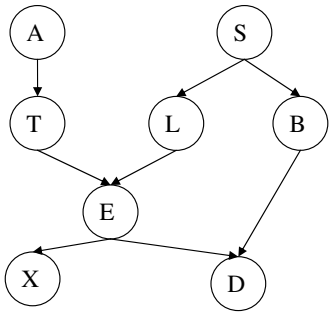


Figure 6.1: The Bayesian Network Parameters and Structure for the Asia Model

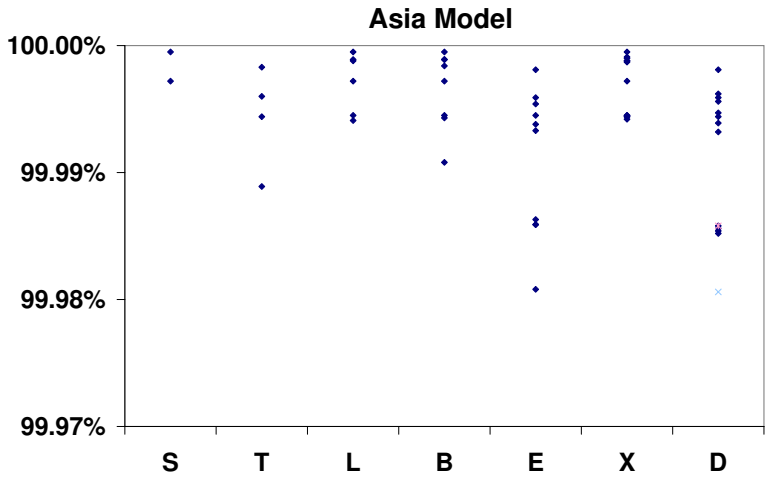


Figure 6.2: The Ratio of g to $\ln f$ in the Asia Model

are within 99.8% of $\ln f$. Kardes et al. [70] have implemented our complete privacy-preserving Bayesian network structure protocol and are currently carrying out additional experiments to test its performance and accuracy.

6.4.3 Privacy-Preserving Computation of α-Parameters

In this section, we describe how to compute secret shares of the α-parameters defined in Section 6.2.2 in a privacy-preserving manner. Recall that α_{ijk} is the number of records in $D = D_A \bowtie D_B$ where v_i is instantiated as v_{i_k} and π_i is instantiated as ϕ_{ij} (as defined in Section 6.2.2), and recall that q_i is the number of unique instantiations that the variables in π_i can take on. The α-parameters include all possible α_{ijk} and α_{ij} that appear in Equation 6.2.2.2 in Section 6.2.2.

Given instantiations v_{i_k} of variable v_i and ϕ_{ij} of the parents π_i of v_i, we say a record in D is *compatible with ϕ_{ij} for Alice* if the variables in $\pi_i \cap V_A$ (i.e., the variables in π_i that are owned by Alice) are assigned as specified by the instantiation ϕ_{ij}, and we say the record is *compatible with v_{i_k} and ϕ_{ij} for Alice* if the variables in $(\{v_i\} \cup \pi_i) \cap V_A$ are assigned as specified by the instantiations v_{i_k} and ϕ_{ij}. Similarly, we say a record is *compatible for Bob* with ϕ_{ij}, or with v_{i_k} and ϕ_{ij}, if the relevant variables in V_B are assigned according to the specified instantiation(s).

We note that α_{ijk} can be computed by determining how many records are compatible for both Alice and Bob with v_i and π_i. Similarly, α_{ij} can be computed by determining how many records are compatible for both Alice and Bob with π_i. Thus, Alice and Bob can determine α_{ijk} and α_{ij} using privacy-preserving scalar product share protocols (see Section 6.2.3) such that Alice and Bob learn secret shares of α_{ijk} and α_{ij}. We describe this process in more detail below.

We define the vector $\mathsf{compat}_A(\phi_{ij})$ to be the vector $(x_1, \ldots, x_n)$ in which $x_\ell = 1$ if the ℓth database record is compatible for Alice with ϕ_{ij}; otherwise, $x_\ell = 0$. We analogously define $\mathsf{compat}_A(v_{i_k}, \phi_{ij})$, $\mathsf{compat}_B(\phi_{ij})$, $\mathsf{compat}_B(v_{i_k}, \phi_{ij})$. Note that given the network structure and i, j, k, Alice can construct $\mathsf{compat}_A(\phi_{ij})$ and $\mathsf{compat}_A(v_{i_k}, \phi_{ij})$, and Bob can construct $\mathsf{compat}_B(\phi_{ij})$ and $\mathsf{compat}_B(v_{i_k}, \phi_{ij})$. Then $\alpha_{ijk} = \mathsf{compat}_A(v_{i_k}, \phi_{ij}) \cdot \mathsf{compat}_B(v_{i_k}, \phi_{ij})$ and $\alpha_{ij} = \mathsf{compat}_A(\phi_{ij}) \cdot \mathsf{compat}_B(\phi_{ij})$. However, the parties cannot in general learn α_{ijk} and α_{ij}, as this would violate privacy.

Note that in the degenerate case, all variables for v_i and π_i belong to one party, who can locally compute the corresponding α-parameters without any interaction with the other party. The following protocol computes α_{ijk} parameters for the general case in which variables including v_i and π_i are distributed among two parties.

Input: D_A and D_B held by Alice and Bob, respectively, Values $1 \leq i \leq m$, $1 \leq j \leq q_i$, and $1 \leq k \leq d_i$, plus the current value of π_i and a particular instantiation ϕ_{ij} of the variables in π_i are commonly known to both parties.

Output: Two secret shares of α_{ijk}.

1. Alice and Bob generate $\mathsf{compat}_A(v_{i_k}, \phi_{ij})$ and $\mathsf{compat}_B(v_{i_k}, \phi_{ij})$ respectively.

2. By taking $\mathsf{compat}_A(v_{i_k}, \phi_{ij})$ and $\mathsf{compat}_B(v_{i_k}, \phi_{ij})$ as two inputs, Alice and Bob execute the privacy-preserving scalar product share protocol of Section 6.2.3 to generate the secret shares of α_{ijk}.

By running the above protocol for all possible combinations i, j and k, Alice and Bob can compute secret shares for all α_{ijk} parameters in Equation 6.2.2.2. Since $\alpha_{ij} = \sum_{k=1}^{d_i} \alpha_{ijk}$, Alice and Bob can compute the secret shares for a particular α_{ij} by simply adding all their secret shares of α_{ijk} together.

Theorem 6.4.1 *Assuming both parties are semi-honest, the protocol for computing α-parameters is privacy-preserving.*

Proof: Because the scalar product share protocol is privacy-preserving, the privacy of each party is protected. Each party only learns secret shares of each α-parameter and nothing else about individual records of the other party's data. ■

6.4.4 Privacy-Preserving Computation of β-parameters

We now show how to compute secret shares of the β-parameters of Equation 6.4.1.2. As described earlier in Section 6.4.3, Alice and Bob can compute secret shares for α_{ijk} and α_{ij}. We denote these shares by $\alpha_{ijk} = a_{ijk} + b_{ijk}$ and $\alpha_{ij} = a_{ij} + b_{ij}$, where a_{ijk}, a_{ij} and b_{ijk}, b_{ij} are secret shares held by Alice and Bob respectively. Since β_{ij} is equal to α_{ij} (by definition), the secret shares of β_{ij} are a_{ij} and b_{ij}.

Recall that $\beta_{ijk} = \alpha_{ijk}$ if α_{ijk} is not 0; otherwise, $\beta_{ijk} = 1$. However, neither Alice nor Bob knows the value of each α_{ijk} because each only has a secret share of each α_{ijk}. Hence, neither of them can directly compute the secret shares of β_{ijk} from α_{ijk}. (The direct exchange of their secret shares would incur a privacy breach.)

We use general secure two-party computation to generate the secret shares of β_{ijk}. That is, Alice and Bob carry out a secure version of the following algorithm. Given that the algorithm is very simple and has small inputs, Yao's secure two-party computation of it can be carried out privately and efficiently [110, 77].

Input: a_{ijk} and b_{ijk} by Alice and Bob.

Output: Rerandomized a_{ijk} and b_{ijk} to Alice and Bob respectively.

If $(a_{ijk} + b_{ijk} == 0)$

Rerandomize a_{ijk} and b_{ijk} s.t. $a_{ijk} + b_{ijk} = 1$;

Else Rerandomize a_{ijk} and b_{ijk} s.t. $a_{ijk} + b_{ijk} = \alpha_{ijk}$;

That is, Alice and Bob's input to the computation are two secret shares of α_{ijk}. They obtain two new secret shares of β_{ijk}.

6.4.5 Privacy-Preserving Score Computation

Our goal in this subprotocol is to privately compute two secret shares of the output of the $g(i, \pi_i)$ score function. There are five kinds of subformula to compute in the $g(i, \pi_i)$ score function:

1. $\ln \beta_{ijk}$
2. $\beta_{ijk} \ln \beta_{ijk}$
3. $\ln (\beta_{ij} + d_i - 1)$
4. $(\beta_{ij} + d_i - 1) \ln(\beta_{ij} + d_i - 1)$
5. pub(d_i, q_i)

To compute two secret shares of $g(i, \pi_i)$ for Alice and Bob, the basic idea is to compute two secret shares of each subformula for Alice and Bob, and then Alice and Bob can add their secret shares of the subformulas together to get the secret shares of $g(i, \pi_i)$. The details of how to compute the secret shares are addressed below.

Since d_i is public to each party, secret shares of $\beta_{ij} + d_i - 1$ can be computed by letting Alice (or Bob) adding $d_i - 1$ to her secret shares of β_{ij} such that Alice holds $a_{ij} + d_i - 1$ and Bob holds b_{ij} as the secret shares for $\beta_{ij} + d_i - 1$. Hence, subformulas 1 and 3 can be written as $\ln a_{ijk} + b_{ijk}$ and $\ln (a_{ij} + d_i - 1) + b_{ij}$. Then the problem of computing secret shares for subformulas 1 and 3 can be reduced to the problem of computing two secret shares for $\ln x$

where x is secretly shared by two parties. The $\ln x$ problem can be solved by the privacy-preserving $\ln x$ protocol of Lindell and Pinkas [75].

Similarly, the problem of generating two secret shares for subformula 2 and 4 can be reduced to the problem of computing secret shares of $x \ln x$ in a privacy-preserving manner, which again is solved by Lindell and Pinkas [75]. In subformula 5, q_i and d_i are known to both parties, so they can be computed by either party.

After computing secret shares for all subformulas 1–5, Alice and Bob can locally add their respective secret shares to compute secret shares of $g(i, \pi_i)$. Because each subprotocol is privacy-preserving and results in only secret shares as intermediate results, the computation of secret shares of $g(i, \pi_i)$ is privacy-preserving.

6.4.6 Privacy-Preserving Score Comparison

In the *K2* algorithm specified in Section 6.2, Alice and Bob need to determine which score is maximum. That is, we require the following privacy-preserving comparison computation:

Input: $(r_{a_1}, r_{a_2}, \ldots, r_{a_x})$ held by Alice and $(r_{b_1}, r_{b_2}, \ldots, r_{b_x})$ held by Bob.

Output: i such that $r_{a_i} + r_{b_i} \geq r_{a_j} + r_{b_j}$ for $1 \leq j \leq x$.

In this case, x is at most $u + 1$, where u is the restriction on the number of possible parents for any node, and in any case no larger than m, the total number of variables in the combined database. Given that generally m will be much smaller than n, this can be privately and efficiently computed using general secure two party computation [110, 77].

6.4.7 Overall Privacy-Preserving Solution for Learning Bayesian Network Structure

Our distributed privacy-preserving structure-learning protocol is shown in Figure 6.3. It is based on the *K2* algorithm, using the variable set of the combined database $D_A \bowtie D_B$, but executes without revealing the individual data values and the sensitive information of each party to the other. Each party learns only the BN structure plus the order in which edges were added (which in turn reveals which edge had maximum score at each iteration).

Input: An ordered set of m nodes, an upper bound u on the number of parents for a node, both known to Alice and Bob, and a database D containing n records, vertically partitioned between Alice and Bob.

Output: Bayesian network structure B_s (whose nodes are the m input nodes, and whose edges are as defined by the values of π_i at the end of the protocol)

Using a fixed ordering of variables in V, Alice and Bob execute the following steps for each node v_i. Initially, each node has no parent. After Alice and Bob run the following steps for node v_i, its parents are the nodes in π_i.

Step 1: Alice and Bob execute the privacy-preserving score protocol to compute the secret shares of $g(i, \pi_i)$ and $g(i, \pi_i \cup \{z\})$ for any $z \in \mathsf{Pred}(v_i) - \pi_i$.

Step 2: Alice and Bob execute the privacy-preserving score comparison protocol to compute which of the scores in Step 1 is maximum.

Step 3: If $g(i, \pi_i)$ is maximum, Alice and Bob go to the next node v_{i+1} to run from Step 1 until Step 3. If one z generates the maximum score in Step 2, then z is added as the parent of v_i s.t. $\pi_i = \pi_i \cup \{z\}$ and Alice and Bob go back to Step 1 with the same node v_i.

Figure 6.3: Structure-learning Protocol

In the original *K2* algorithm, all the variables are in one central site, while in our setting the variables are distributed over two sites. Hence, we must compute the score function across two sites. Remembering that $\ell_{ij} = \beta_{ij} + d_i - 1$, we can see from Equation 6.4.1.2 that the score can be computed from the β-parameters.

Other than the distributed computation of the scores and their comparison, our control flow is as given in the *K2* algorithm. (For efficiency reasons, it is preferable to combine the comparisons that determine which possible parent yields the highest score with the comparison to determine if this score is higher than the current score, but logically the two are equivalent.) Note that this method leaks relative score values by revealing the order in which the edges were added. Formally, in order for the protocol to be considered privacy-preserving, we therefore consider it to be a protocol for computing Bayesian network structure *and the order in which edges were added by the algorithm*.

The protocol does not reveal the actual scores or any other intermediate values. Instead, we use privacy-preserving protocols to compute the secret shares of the scores. We divide the BN structure-learning problem into smaller subproblems, and use the earlier described privacy-

preserving subprotocols to compute shares of the β-parameters (Section 6.4.4) and the scores (Section 6.4.5) in a privacy-preserving way, and to compare the resulting scores in a privacy-preserving way (Section 6.4.6). Overall, the privacy-preserving protocol is executed jointly between Alice and Bob as showed in Figure 6.3. This protocol has been implemented by Kardes et al. [70]. Privacy and performance issues are further discussed in Section 6.6.

Theorem 6.4.2 *Assuming the subprotocols are privacy-preserving, the protocol to compute Bayesian network structure reveals nothing except the Bayesian network structure and order in which the nodes are added.*

Proof: Besides the structure itself, the structure-learning protocol reveals only the order information because each of the subprotocols is privacy-preserving, they are invoked sequentially, and they only output secret shares at each step. ■

6.5 Privacy-Preserving Bayesian Network Parameters Protocol

In this section, we present a privacy-preserving solution for computing Bayesian network parameters on a database vertically partitioned between two parties. Assuming the BN structure is already known, Meng, Sivakumar, and Kargupta presented a privacy-preserving method for learning the BN parameters [78], which we refer to as MSK. In this section, we describe an alternate solution to MSK. In contrast to MSK, our method is more private, more efficient, and more accurate. In particular, our parameter-learning solution provides complete privacy, in that the only information the parties learn about each other's inputs is the desired output, and complete accuracy, in that the parameters computed are exactly what they would be if the data were centralized. In addition, our solution works for both binary and nonbinary discrete data. We provide a more detailed comparison between the two solutions in Section 6.5.2.

As we discuss further in Section 6.6.1, it is possible to run our structure-learning protocol and parameter-learning protocol together for only a small additional cost over just the structure-learning protocol.

6.5.1 Privacy-Preserving Protocol for Learning BN Parameters

Recall the description of Bayesian network parameters in Section 6.2.1. Given Bayesian network structure B_{s}, the network parameters are the conditional probabilities $B_{\mathrm{p}} = \{\Pr[v_i = v_{i_k} \mid \pi_i = \phi_{ij}] : v_i \in V,\ 1 \leq j \leq q_i,\ 1 \leq k \leq d_i\}$. If variable v_i has no parents, then its parameters specify the marginal distribution of v_i; that is, $\Pr[v_i = v_{i_k} \mid \pi_i = \phi_{ij}] = \Pr[v_i = v_{i_k}]$. Note that these parameters can be computed from the α-parameters, as follows:

$$\Pr[v_i = v_{i_k} \mid \pi_i = \phi_{ij}] = \frac{\alpha_{ijk}}{\alpha_{ij}} \tag{6.5.1.1}$$

Earlier in Section 6.4.3, we described a privacy-preserving protocol to compute secret shares of α_{ijk} and α_{ij}. Now, we need to extend this to allow the parties to compute the value α_{ijk}/α_{ij} without sharing their data or revealing any intermediate values such as α_{ijk} and α_{ij} (unless such values can be computed from the BN parameters themselves, in which case revealing them is not considered constitute a privacy breach). We consider three cases separately:

1. **One party owns all relevant variables.** In the degenerate case, one party (say, Alice) owns all of the relevant variables: $\{v_i\} \cup \pi_i$. In this case, she can compute α_{ijk}/α_{ij} locally and announce the result to Bob.

2. **One party owns all parents, other party owns node.** In the next simplest case, one party (again, say Alice) owns all the variables in π_i, and the other party (Bob) owns v_i. In this case, Alice can again directly compute α_{ij} from her own data. Alice and Bob can compute the secret shares of α_{ijk} using the protocol described in Section 6.4.3. Bob then sends his share of α_{ijk} to Alice so she can compute α_{ijk}. (In this case, it is not a privacy violation for her to learn α_{ijk} because knowing α_{ij}, she could compute α_{ijk} from the final public parameter α_{ijk}/α_{ij}.) From α_{ijk} and α_{ij}, Alice then computes α_{ijk}/α_{ij}, which she also announces to Bob.

3. **The general case: the parent nodes are divided between Alice and Bob.** In the general case, Alice and Bob have secret shares for both α_{ijk} and α_{ij} such that $a_{ijk} + b_{ijk} = \alpha_{ijk}$ and $a_{ij} + b_{ij} = \alpha_{ij}$ (where these additions are modular additions in a group depending

on the underlying scalar product share protocol used in Section 6.4.3). Thus, the desired parameter is $(a_{ijk} + b_{ijk})/(a_{ij} + b_{ij})$. In order to carry out this computation without revealing anything about a_{ijk} and a_{ij} to Bob, or b_{ijk} and b_{ij} to Alice, we make use of general secure two-party computation. Note that this is sufficiently efficient here because the inputs are values of size k, independent of the database size n, and because the function to compute is quite simple.

Note that cases (1) and (2) could also be handled by the general case, but the simpler solutions provide a practical optimization as they require less computation and communication. In order to learn all the parameters B_{p}, Alice and Bob compute each parameter for each variable using the method just described above, as demonstrated in Figure 6.4.

Input: A database D is vertically partitioned between Alice and Bob. Alice holds D_A and Bob holds D_B. A global Bayesian network B_{s} over $D_A \bowtie D_B$ is known to both parties.

Output: The parameters B_{p} of the Bayesian network B_{s}.

For each variable v_i in B_{s} and for each possible instantiation of v_i and π_i, Alice and Bob carry out the computation in the appropriate case discussed above. In the general case, this involves:

a. Alice and Bob run the privacy-preserving protocol for computing α-parameters introduced in Section 6.4.3 to learn secret shares of α_{ijk} and α_{ij}.

b. Alice and Bob run a secure two-party computation to compute the desired parameter α_{ijk}/α_{ij}.

Figure 6.4: Parameter-learning Protocol

Theorem 6.5.1 *Assuming the privacy and correctness of the protocol for computing α-parameters is privacy-preserving and the secure two-party computation protocol, the parameter-learning protocol is correct and private.*

Proof: The correctness of that protocol is clear, because the values computed are precisely the desired parameters α_{ijk}/α_{ij}.

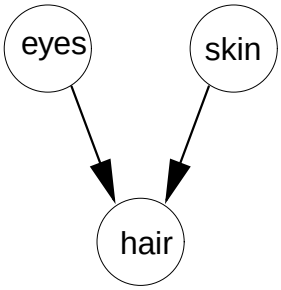
Privacy is protected because in each case, we only reveal values to a party that are either part of the final output or are straightforwardly computable from the final output and its own input.

All other intermediate values are protecting via secret sharing, which reveals no additional information to the parties. ■

6.5.2 Comparison with MSK

For a dataset containing only binary values, the MSK solution showed that the count information required to estimate the BN parameters can be obtained as a solution to a set of linear equations involving some inner products between the relevant different feature vectors. In MSK, a random projection-based method is used to securely compute the inner product.

In this section, we provide a detailed comparison of the privacy, efficiency, and accuracy of our parameter-learning solution and MSK.



eyes: $\Pr[\text{eyes} = \text{brown}] = \Pr[\text{eyes} = \text{blue}] = 1/2$
skin: $\Pr[\text{skin} = \text{fair}] = \Pr[\text{eyes} = \text{dark}] = 1/2$
hair:

hair	eyes, skin			
	brown,fair	blue,fair	brown,dark	blue,dark
blond	1/2	1/2	1/2	1/2
brown	1/2	1/2	1/2	1/2

Figure 6.5: An Example of BN Structure and Parameters

We show that our solution performs better in efficiency, accuracy, and privacy than MSK. The primary difference between our solution and MSK is that MSK computes the parameter probabilities by first computing the counts of the various possible instantiations of nodes and their parents. As we discuss below, MSK approach inherently leaks more information than the parameters alone. In addition, they use a secure "pseudo inner product" to compute those counts, using a method that is less efficient, less accurate, and less private than cryptographic scalar product protocols (such as those discussed in Section 6.2.3).

As we discuss further below, replacing the pseudo inner product of MSK with an appropriate cryptographic scalar product would improve MSK to have somewhat better efficiency than our solution and complete accuracy (as our solution does). Our solution remains more private than the modified MSK, so in some sense this suggests that our solution and the modified MSK

solution represent an efficiency/privacy tradeoff.

Efficiency. Let $d = \max d_i$ be the maximum number of possible values any variable takes on, S be a security parameter describing the length of cryptographic keys used in the scalar product protocol, and u be the maximum number of parents any node in the Bayesian network has. (Thus $u \leq m-1$, and typically $u \ll m \ll n$). Our solution runs in time $O(md^{(u+1)}(Sn+S^2))$. Taking $d = 2$ for purposes of comparison (since MSK assumes the data is binary-valued), this is $O(m2^{(u+1)}(Sn + S^2))$. In contrast, MSK runs in time $O(m(2^{(u+1)} + n^2))$. In particular, for a fixed security parameter S and maximum number u of parents of any node, as the database grows large enough that $S \ll n$, our running time grows linearly in n, while the running time of MSK grows as n^2.

We note that source of the quadratic growth of MSK is their secure pseudo-inner product, as for an input database with n records, it requires the parties to produce and compute with an $n \times n$ matrix. If this step were replaced with an ideally-private cryptographic scalar product protocol such as the one we use, their performance would improve to $O(m(2^{(u+1)} + kn))$, a moderate efficiency improvement over our solution.

Accuracy. Our parameter-learning solution provides complete accuracy, in the sense that we faithfully produce the desired parameters. The secure pseudo-inner product computation of MSK introduces a small amount of computational error. Again, replacing this step with a perfectly accurate cryptographic scalar product can provide perfect accuracy.

Privacy. Our parameter-learning solution provides ideal privacy, in the sense that the parties learn nothing about each other's inputs beyond what is implied by the Bayesian parameters and their own inputs. MSK has two privacy leaks. The first comes from the secure pseudo-inner product computation, but again this could be avoided by using an ideally-private scalar product protocol instead. The second, however, is intrinsic to their approach. As mentioned earlier, they compute the parameter probabilities by first computing the counts of the various possible instantiations of nodes and their parents. As they point out, the probabilities can easily be computed from the counts, so this does not affect the correctness of their computation. However,

the reverse is not true—in general, the counts leak more information than the probabilities, because different counts can give rise to the same probabilities. We illustrate this by a simple example, as shown in Figures 6.5 and 6.6. In this example, Alice owns the variable eyes, while

instantiation			counts	counts
eyes	skin	hair	in DB1	in DB2
brown	fair	blond	2	1
brown	fair	brown	2	1
brown	dark	blond	2	3
brown	dark	brown	2	3
blue	fair	blond	2	3
blue	fair	brown	2	3
blue	dark	blond	2	1
blue	dark	brown	2	1

Figure 6.6: Example Showing That Counts Leak More Information Than Parameters.

Bob owns skin and hair. The Bayesian network, consisting of both the given structure (which we assume is given as part of the input to the problem) and the parameters (which are computed from the input databases), are shown in Figure 6.5.

Figure 6.6 shows two quite different kinds of databases, DB1 and DB2, which are both consistent with the computed Bayesian network parameters and with a particular setting for Alice's values for eyes. Both databases have 16 records. For eyes, half the entries are brown and half are blue; similarly, for skin, half the entries are fair and half are dark. The difference between DB1 and DB2 lies with hair and its relation to the other variables. One can easily verify that both DB1 and DB2 are consistent with the computed Bayesian network parameters and with a particular setting for Alice's values for eyes. Hence, given only the parameters and her own input, Alice would consider both databases with counts as shown in DB1 and with counts as shown in DB2 possible (as well as possibly other databases). However, if additionally given the count information, Alice can determine that either DB1 or DB2 is not possible, substantially reducing her uncertainty about Bob's data values. Although this example is simple and rather artificial, it suffices to demonstrate the general problem.

6.6 Protocol Analysis

We analyze the performance and privacy of the proposed solution in Sections 6.6.1 and 6.6.2.

6.6.1 Performance Analysis

We have presented privacy-preserving protocols for learning BN structure and parameters (Sections 6.4 and 6.5, respectively). Rather than running these sequentially to learn a Bayesian network from a dataset, these two protocols can be combined so that the BN parameters can be computed with a constant overhead over the computation of the BN structure. This is because the secret shares of α_{ijk} and α_{ij} needed in the parameter protocol are already computed in the structure protocol. Hence, the only additional overhead to compute the parameters is the secure two-party computation to divide the shared α_{ijk} by the shared α_{ij}.

Further, we note a few more potential practical optimizations. For example, in order to reduce the number of rounds of communication, the α-parameters can be computed in parallel, rather than in sequence. This allows all the vectors for a given set of variables to be computed in a single pass through the database, rather than multiple passes. Similarly, shares of each α_{ij} need only be computed once, rather than once for each BN parameter. Additionally, if multiple nodes share the same set of parents, the same intermediate values can be reused multiple times.

As discussed above, the dominating overhead of our solution comes from computing the BN structure. Hence, the overall overhead of our solution depends on the database size n, the number m of variables, and the limit u on the number of possible parents for any node. Like the original *K2* algorithm, our Structure Protocol requires computation that is exponential in u (in order to compute the α-parameters for all possible $O(2^u)$ instantiations of the set of parents of a given node). In the *K2* algorithm, the inner loop runs $O(mu)$ times. Each time the inner loop is executed, there are $O(u)$ scores to compute, each requiring $O(m2^u)$ α-parameters to be computed. In our solution, the computation of each α-parameter, including the scalar product share protocol, requires $O(n)$ communication and computation. This is the only place that n comes into the complexity. Everything else, including computing β-parameters from α-parameters, combining β-parameters into the score, and the score comparison, can be done

in computation and communication that is polynomial in m and 2^u.

6.6.2 Privacy Analysis

In our solution, each party learns the Bayesian network, including the structure and the parameters, on the joint data without exchanging their raw data to each other. In addition to the Bayesian network, each party also learns the relative order in which edges are added into the BN structure. While this could be a privacy breach for some settings, it seems a reasonable privacy/efficiency tradeoff that may be acceptable in many settings.

We note that the BN parameters contain much statistical information about each database, so another concern is that even if a privacy-preserving computation of Bayesian network parameters is used, the resulting BN model—particularly when taken together with one party's database—reveals quite a lot of information about the other party's database. That is, the *result* of the privacy-preserving computation may itself leak too much information, even if the computation is performed in a completely privacy-preserving way, a phenomenon discussed nicely by in [68]. To limit this leakage, it might be preferable, for example, to have the parameters associated with a variable v_i revealed only to the party owning the variable. By using a secure two-party computation that gives the result only to the appropriate party, our solution can easily be modified to do this.

Another option would be to have the parties learn secret shares of the resulting parameters, not the actual parameters. This suggests an open research direction, which is to design mechanisms that allow the parties to use the Bayesian network and shared parameters in a privacy-preserving interactive way to carry out classification or whatever task they seek to perform.

6.7 Discussion

Chen, Sivakumar, and Kargupta present efficient solutions for learning Bayesian networks on vertically partitioned data [25, 26]. In their solutions, each party first learns local BN models based on his own data, then sends a subset of his data to the other party. A global BN model is learned on the combination of the communicated subsets. (The computation can be done by

either party). Finally, the BN model is learned by combining the global BN model and each party's local BN model. Those solutions are very efficient both in computation and communication, but obviously they were not designed with privacy in mind, as each party has to send part of his data to the other party. Further, these solutions suffer a tradeoff between the quality of the final BN model and the amount of communicated data: the more of their own data the parties send to each other, the more accurate the final BN model.

By combining our proposed solution with the solutions in [25, 26], we can achieve a new solution which provides privacy as discussed in Section 6.6.2, together with a tradeoff between performance and accuracy. The basic idea is that first each party locally learns a model on his or her own data and chooses the appropriate subset of his or her data according to the methods of [25, 26]. Rather than sending the selected subset of data to the other party, both parties then run our solutions described in Sections 6.4 and 6.5 on the chosen subset of their data to privately learn the global BN model on their data subsets. Finally, each party publishes his or her local BN models, and the parties combine the global BN model with their local models to learn the final BN model following the methods of [25, 26]. This solution suffers a similar tradeoff between performance and accuracy as the solutions of [25, 26], but with improved privacy as parties no longer send their individual data items to each other.

Chapter 7

Privacy-Preserving Queries on Encrypted Data

7.1 Introduction

Databases—which store large amounts of data including sensitive data—are attractive targets for security attacks. Consequently, it is crucial to protect sensitive data stored in a database while maintaining data usability.

Techniques like access control protect sensitive data from unauthorized access. However, those techniques cannot ensure that a database is fully immune to intrusion and unauthorized access. If an intruder can bypass those techniques, e.g., directly having access to the physical files, all data will be disclosed to the intruder. Encryption is a well-studied technique to protect sensitive data [34]. When the sensitive data are (properly) encrypted, even if an intruder could gain access to the data for a while, no information or very little information would be leaked. Nevertheless, it is much less convenient to use encrypted data than to use cleartext data.

Now the question is how can queries be processed in a database that stores encrypted data? If a user fully trusts the database server, she can simply send the encryption key to the database server together with her query. However, since the database may be compromised at some point, revealing the key to the database server is at the risk of leaking all sensitive data to an intruder.

Theoretically, if efficiency was not a concern, a user could retrieve all encrypted data from the database, decrypt the data, and then perform queries on the cleartext data. However, this is clearly impractical when we take efficiency into consideration.

In this chapter, we investigate *efficient* methods for queries on encrypted data. We require the methods to be secure even if the database server may be compromised at some point by an intruder. Such methods are very useful in strengthening the protection of sensitive data in databases.

7.2 Technical Preliminaries

We consider a system as illustrated in Figure 7.1. In this system, data are encrypted and stored in tables. In the front end, when the user has a query, the query is translated to one or more messages. Upon receiving the message(s), the database finds the appropriate encrypted cells and returns them to the front end. Finally, the front end decrypts the received cells. For convenience of presentation, in the sequel, we do not distinguish the human user from the front end program; when we say "user," we mean the human user plus the front end program.

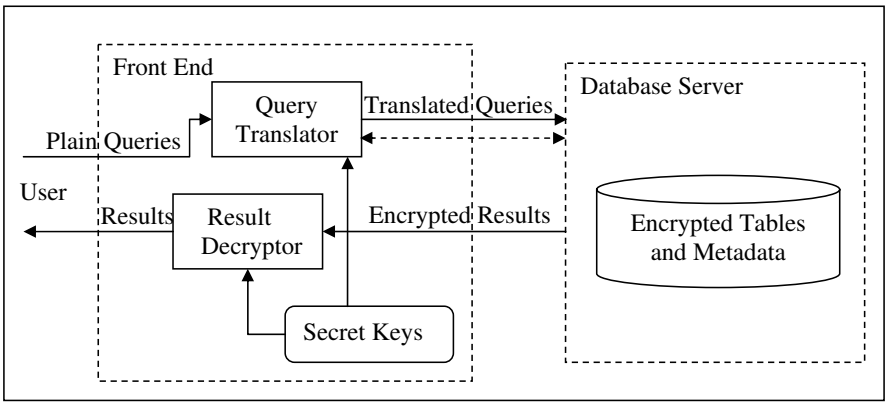


Figure 7.1: Overall Architecture

7.2.1 Trust and Attack Model

In this chapter, we focus on the possible attack of an intruder on the database server. The goal of our work is to ensure an intruder who has complete access to the database server for some time learns very little about the data stored in the database and the queries performed on the data. Our trust and attack model is as follows:

1. We do not fully trust the database server because it may be vulnerable to intrusion. Furthermore, we assume that, once a database intruder breaks into the database, he can observe not only the encrypted data, but can also control the whole database system for a time interval. During the time interval, a number of query messages sent by the user, as well as the database's processing of these queries, can be observed by the intruder. The security guarantees we show are all with respect to such a powerful intruder.

2. We trust the communication channel between the user and the database, as there exist standard protocols to secure the channel, e.g., SSL and IPSec. We also trust the user's front-end program.

3. We require all data and metadata be stored encrypted. This includes user logs and scheme metadata. (Otherwise, these may open the door for intruders.)

The assumption that an intruder can only control the whole database system for a short while (compared with unbounded time) is reasonable. For example, a legal database administrator can always physically shut down the database server once he loses control of the database server.

7.2.2 Table and queries

In this chapter, we focus on a table T and discuss queries performed on T. Suppose T has n rows (i.e., n records) and m columns (i.e., m attributes). We denote by $T_{i,j}$ the cell at the intersection of the ith row and the jth column; we also refer to (i, j) as the *coordinates* of the cell. We denote the ith row by T_i. Each attribute of the table has a finite domain. For the jth attribute A_j, we denote the domain by D_j.

As we have mentioned, we store our tables in an encrypted form. More precisely, for a table T, we store an encrypted table T' in the database, where each $T'_{i,j}$ is an encryption of $T_{i,j}$. Without loss of generality, we assume that each cell $T_{i,j}$ of the plaintext table is a bitstring of at most k_1 bits: $\forall j \in [1, m]$, $D_j \subseteq \{0, 1\}^{k_1}$. (We can always encode any value of an attribute as a bitstring.) When we encrypt a cell, the encryption algorithm uses a random string of k_2 bits.[1] Hence, the input to the encryption algorithm is a k_0-bit string, where $k_0 = k_1 + k_2$. For

[1] As explained in more detail in Section 7.3, the purpose of using a random string is that multiple occurrences of

simplicity (and following the practice of most symmetric encryption schemes), we also assume the output of the encryption algorithm and the encryption key are k_0-bit strings as well.

Suppose that a user intends to perform a query Q on the table T. We mainly consider query protocols that return to the user precisely the set of (encrypted) cells stored in the database that satisfy the condition of the query. We call such query protocols *precise query protocols*. Query protocols which do not fall into this category are briefly discussed in Section 7.7. We denote by $R(Q)$ the set of coordinates of the cells satisfying the condition of query Q—the cells satisfying the condition of query Q are $\{T'_{i,j} : (i,j) \in R(Q)\}$. Clearly, in *any* precise query protocol, if there is a database intrusion of the type discussed in Section 7.2.1, then $R(Q)$ is always revealed to the intruder. This is because the intruder can simply check T' to see which encrypted cells are in the returned result. Therefore, we say $R(Q)$ is the *minimum information revelation* of query Q.

7.2.3 Privacy-Preserving Queries

We give a cryptographic definition of *privacy-preserving query protocols*. In particular, we consider that an intruder may observe up to t queries $Q_1, \ldots, Q_t$, where t is a polynomially bounded function of k_0. We quantify the information leaked by the protocol using a random variable α: we say the protocol only reveals α beyond the minimum information revelation if, after these queries are processed, what the database intruder has observed can be simulated by a probabilistic polynomial-time algorithm using only $\alpha, R(Q)$, and the encrypted table. For simplicity, we only give the formal definition of a privacy-preserving *one-round* query protocol. It is straightforward to extend this definition to multi-round query protocols.

Definition 7.2.1 *(Privacy-Preserving Query) A one-round query protocol* reveals only α beyond the minimum information revelation *if for any polynomial* $\mathsf{poly}()$ *and all sufficiently large* k_0*, there exists a probabilistic polynomial-time algorithm* $\mathcal{S}$ *(called a* simulator*) such that for all* $t < \mathsf{poly}(k_0)$*, all polynomial-size circuit family* $\{\mathcal{A}_{k_0}\}$*, all polynomial* $p()$*, and all* $Q_1, \ldots, Q_t$*,*

a plaintext should lead to different ciphertexts.

$$|\Pr[\mathcal{A}_{k_0}(Q_1,\ldots,Q_t,q_1,\ldots,q_t,T')=1]-$$
$$\Pr[\mathcal{A}_{k_0}(Q_1,\ldots,Q_t,\mathcal{S}(\alpha,R(Q_1),\ldots,R(Q_t),T'))=1]|<1/p(k_0).$$

A query protocol is private if it reveals nothing beyond the minimum information revelation.

The above definition can be viewed as an adaptation of the definition of secure protocol in the semi-honest model. However, note that a secure one-round query protocol as defined here *remains secure even in the case the intruder is fully malicious*, i.e., even when the intruder modifies the database software such that the database deviates from the protocol. The reason is that the database's behavior does not affect the user's behavior in this case.

7.3 Basic Solution

In this section, we give a basic solution for queries of the format "*select ... from* T *where* $A_j = v$," where $v \in D_j$ is a constant. We provide rigorous cryptographic specifications and proofs.

Our basic idea is to encode each cell in a special redundant form. Specifically, for each cell $T_{i,j}$, the encrypted cell $T'_{i,j}$ has two parts: the first part, $T'_{i,j}[1]$, is a simple encryption of $T_{i,j}$ using a block cipher $E()$; the second part, $T'_{i,j}[2]$, is a "checksum" that, together with the first part, enables the database to check whether this cell satisfies the condition of the query or not. There is a secret equation that $T'_{i,j}[1]$ and $T'_{i,j}[2]$ satisfy; this equation is decided by the value of $T_{i,j}$. When the database is given the equation corresponding to value v, it can easily check whether a cell satisfies the condition or not by substituting the two parts of the encrypted cell into the equation.

The remaining question is what equation to choose. We use the following simple equation:

$$E_{f(T_{i,j})}(T'_{i,j}[1]) = T'_{i,j}[2],$$

where f is a function. When the user has a query with condition $A_j = v$, she only needs to send

$f(v)$ to the database so that the database can check, for each i, whether

$$E_{f(v)}(T'_{i,j}[1]) = T'_{i,j}[2]$$

holds. It should be infeasible to derive v from $f(v)$ because otherwise an intruder learns v when observing $f(v)$. To achieve this goal, we define $f(\cdot)$ to be an encryption of v using the block cipher $E(\cdot)$. Additional care needs to be taken when we use the block cipher E. When we make $T'_{i,j}[1]$, in order to prevent the database from knowing which two cells have the same contents, we need to append a random string to $T_{i,j}$ before we apply E. To avoid having the same $f(\cdot)$ for different attributes, we need to append j to $f(\cdot)$ before applying E.

7.3.1 Solution Details

Data Format. Let $E(\cdot)$ be a symmetric encryption algorithm whose key space, plaintext space, and ciphertext space are all $\{0,1\}^{k_0}$. We often use the notation $E_S(M_1, M_2)$ to denote a message (M_1, M_2) encrypted using secret key S, where M_1 (resp., M_2) is either a k_1-bit (resp., k_2-bit) string. We denote the corresponding decryption algorithm by D and we assume that the key generation algorithm simply picks a uniformly random key from the key space $\{0,1\}^{k_0}$.

To create the table T in the database, the user first picks two secret keys s_1, s_2 from $\{0,1\}^{k_0}$ independently and uniformly. The user keeps s_1, s_2 secret. For each cell $T_{i,j}$, the user picks $r_{i,j}$ from $\{0,1\}^{k_2}$ uniformly at random and stores

$$\begin{aligned} T'_{i,j} &\stackrel{\triangle}{=} (T'_{i,j}[1], T'_{i,j}[2]) \\ &= (E_{s_1}(T_{i,j}, r_{i,j}), E_{E_{s_2}(T_{i,j},j)}(E_{s_1}(T_{i,j}, r_{i,j}))). \end{aligned}$$

Query Protocol. Denote by A_j the jth attribute of T. Suppose there is a query *select* $*$ *from* T *where* $A_{j_0} = v$. To carry out this query, the user computes $q = E_{s_2}(v, j_0)$ and sends j_0, q, and $(j_1, \ldots, j_\ell)$ to the database.

For $i = 1, \ldots, n$, the database tests whether $T'_{i,j_0}[2] = E_q(T'_{i,j_0}[1])$ holds. For any i such

that the above equation holds, the database returns $T'_{i,j_1}[1], \ldots, T'_{i,j_\ell}[1]$ to the user. The user decrypts each received cell using secret key s_1, and discards the k_2-bit tail of the cleartext.

In our scheme, note that each encrypted cell with the same plaintext value has a different encryption. Thus if an intruder breaks into the database and sees the encrypted table, he cannot tell whether two cells have the same plaintext value or not.

7.3.2 Security Analysis

We can prove the security of our scheme using standard cryptographic techniques. Recall that for security we need to consider t queries. Suppose the uth query ($1 \leq u \leq t$) is of the format "select $A_{j_{u,1}}$,..., $A_{j_{u,\ell}}$ from T where $A_{j_{u,0}} = v_u$." We show that our basic solution only reveals $j_{1,0}, \ldots, j_{t,0}$ beyond the minimum information revelation. That is, the only extra information leakage by the basic solution is which attributes are tested in the "where" conditions.

The security of our scheme derives from the security of the block cipher we use. In cryptography, secure block ciphers are modeled as *pseudorandom permutations* Here, the encryption key of the block cipher is the random seed for the pseudorandom permutation. For each value of the key, the mapping from the cleartext blocks to the ciphertext blocks is the permutation indexed by the value of the seed. In the following theorem, we assume the block cipher we use satisfies this security requirement.

Theorem 7.3.1 *If the block cipher E is a pseudorandom permutation (with the encryption key as the random seed), the basic protocol reveals only $j_{1,0}, \ldots, j_{t,0}$ beyond the minimum information revelation.*

Proof: We construct a simulator $\mathcal{S}$ as follows. First, let $R_1(Q_u) = \{i : (i,j) \in R(Q_u)\}$ and $R_2(Q_u) = \{j : (i,j) \in R(Q_u)\}$. Then, for any $u \in \{1, \cdots, t\}$, if there exists $u' < u$ such that $j_{u,0} = j_{u',0}$ and that $R_1(Q_u) = R_1(Q_{u'})$, $\mathcal{S}$ sets $\overline{q}_u = \overline{q}_{u'}$. Otherwise, $\mathcal{S}$ chooses $\overline{q}_u$ from $\{0,1\}^{k_0} - \{\overline{q}_{u'} : u' < u \wedge j_{u,0} = j_{u',0}\}$ uniformly at random. Next, for $i = 1$ through n and $j = 1$ through m, $\mathcal{S}$ chooses $\overline{T'}_{i,j}[1]$ uniformly and independently from $\{0,1\}^{k_0}$. For $u = 1$

through t, for each $i \in R_1(Q_u)$, $\mathcal{S}$ computes

$$\overline{T'}_{i,j_{u,0}}[2] = E_{\overline{q}_u}(\overline{T'}_{i,j_{u,0}}[1]).$$

For any pair (i, j) for which $\overline{T'}_{i,j}[2]$ has not been defined, $\mathcal{S}$ chooses $\overline{T'}_{i,j}[2]$ from $\{0,1\}^{k_0}$ uniformly and independently. Finally, $\mathcal{S}$ outputs $\overline{q}_1, \ldots, \overline{q}_t$, $\overline{T'}$. The indistinguishability by polynomial-size algorithms follows from the pseudorandomness of E. ■

In this setting, even if the intruder has access to the whole database, the intruder can learn nothing about the encrypted data. By combining $j_{1,0}, \cdots, j_{t,o}$ with the minimum information revelation, an intruder can attack our solutions by deriving some statistical information about the underlying data or the queries. In Section 7.4, we present a solution that leaks less information to make such attacks more difficult.

7.3.3 Experimental Results

To evaluate the efficiency of our solution in practice, we implemented the basic solution. Our experiments use the Nursery dataset from the UCI machine learning repository [15]. The Nursery dataset is a table with eight categorical attributes and one class attribute. There are $12,960$ records in total. The total number of data cells is about $100,000$. The only change we made to the Nursery dataset is that we added an ID attribute so that the table would have a primary key.

Because the time spent on communication is highly dependent on the network bandwidth, we focus on the computational overhead and ignore the communication overhead. The experimental environment is the NetBSD operating system running on an AMD Athlon 2GHz processor with 512MB memory. For the block cipher, we use the Blowfish symmetric encryption algorithm with a 64-bit block size (i.e., $k_0 = 64$).

Clearly, the overhead to encrypt a database is linear with respect to the size of the database. Specifically, in our experiments it takes only 25 seconds to encrypt the entire Nursery dataset. On average, encrypting a single record requires only 0.25 milliseconds. Figure 7.2 shows the time consumed by four select queries. These queries are select $*$ from Nursery where Parent=usual, where Class=recommend, where Class=very_recom, and where Class=priority.

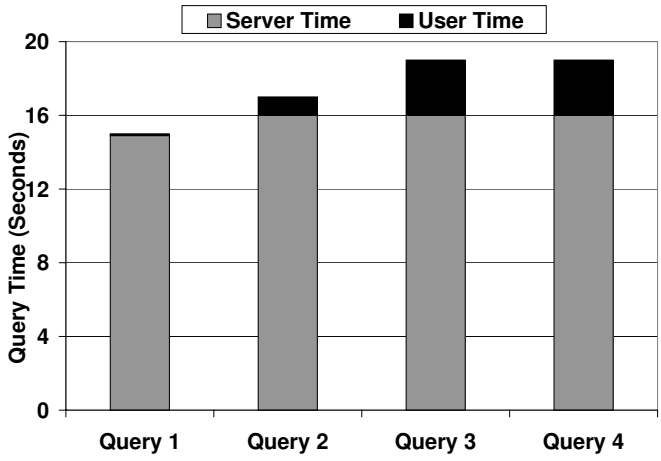


Figure 7.2: Query Time in the Basic Solution

For each query, the database server needs almost the same amount of time for computation (about 16 seconds). The user's computational time depends on the number of returned records from the database. In the first query, only 2 records are returned, and so the computational time by the user is extremely small. In contrast, the last two queries return 4320 and 3266 records, respectively. Therefore, the computational time of the user in each of these two queries is about 4 seconds.

7.4 Solution with Enhanced Security

In this section, we enhance the security of the basic solution so that the query protocol reveals less information. For a broad class of tables, we can show our solution with enhanced security is private.

Recall that the basic solution reveals which attributes are tested in the "where" conditions. Our goal is to hide this information (or at least part of this information). A straightforward way for doing this is to randomly permute the attributes in the encrypted table, in order to make it difficult for a database intruder to determine which attributes are tested.

There remains the question of which distribution to use for the random permutation. If the distribution has a large probability mass on some specific permutations, then the intruder can guess the permutation with a good probability. So the ideal distribution is the uniform distribution. However, if the permutation is chosen uniformly from all permutations of the

attributes, the user needs to "memorize" where each attribute is after the permutation. When the number of attributes is large, this is a heavy burden for the user. To eliminate this problem, we use a pseudorandom permutation, which is by definition indistinguishable from a uniformly random permutation [50]. The advantage of this approach is that it requires the user to only memorize the random seed.

In fact, we note that we do not need to permute all the attributes in the encrypted table. For each (i, j), we can keep $T'_{i,j}[1]$ as defined in the basic solution; we only need to permute the equations satisfied by $T'_{i,j}[1]$ and $T'_{i,j}[2]$because only these equations are tested when there is a query. Specifically, the equation satisfied by $T'_{i,j}[1]$ and $T'_{i,j}[2]$ is no longer decided by the value $T_{i,j}$; instead, it is decided by $T_{i,\pi_S(j)}$, where $\pi_S()$ is a pseudorandom permutation. Consequently, when there is a query whose condition involves attribute A_j, the database actually tests an equation on attribute $A_{\pi_S^{-1}(j)}$.

7.4.1 Solution Details

Data Format. Let $\pi_S()$ be a pseudorandom permutation on $\{1, \ldots, m\}$ for a uniformly random seed $S \in \{0,1\}^{k_0}$. To store the table T in the database, the user first picks secret keys s_1, s_2, s'_2 from $\{0,1\}^{k_0}$ independently and uniformly. The user keeps s_1,s_2, and s'_2 secret. For each cell $T_{i,j}$, the user picks $r_{i,j}$ from $\{0,1\}^{k_2}$ uniformly at random, computes $\hat{j} = \pi_{s'_2}(j)$, and stores

$$\begin{aligned} T'(i,j) &= (T'(i,j)[1], T'(i,j)[2]) \\ &= (E_{s_1}(T_{i,j}, r_{i,j}), E_{E_{s_2}(T_{i,\hat{j}},j)}(E_{s_1}(T_{i,j}, r_{i,j}))), \end{aligned}$$

Query Protocol. Suppose there is a query *select* A_{j_1},...,A_{j_ℓ} *from* T *where* $A_{j_0} = v$. To carry out this query, the user computes $j'_0 = \pi^{-1}_{s'_2}(j_0)$ and $q = E_{s_2}(v, j'_0)$, then sends $j'_0, q, (j_1, \ldots, j_\ell)$ to the database.

For $i = 1, \ldots, n$, the database tests whether $T'_{i,j'_0}[2] = E_q(T'_{i,j'_0}[1])$ holds. For any i such that the above equation holds, the database returns $T'_{i,j_1}[1], \ldots, T'_{i,j_\ell}[1]$ to the user. The user decrypts each received cell using secret key s_1 and discards the k_2-bit tail of the cleartext.

7.4.2 Security Analysis

Again, recall that for security we need to consider t queries, where the uth query ($1 \leq u \leq t$) is of the format "select $A_{j_{u,1}}$,..., $A_{j_{u,\ell}}$ from T where $A_{j_{u,0}} = v_u$." We introduce a new variable that represents whether two queries involve testing the same attribute: for $u, u' \in [1, t]$, we define

$$\epsilon_{u,u'} = \begin{cases} 1 & \text{if } j_{u,0} = j_{u',0} \\ 0 & \text{otherwise.} \end{cases}$$

Using this new variable, we can quantify the security guarantee of our solution with enhanced security. Furthermore, we are able to show that our solution with enhanced security becomes private if the table belongs to a broad class, which we call the *non-coinciding* tables. Intuitively, a table is non-coinciding if any two queries that test different attributes do not have exactly the same result. Below is the formal definition.

Definition 7.4.1 *A table T is* non-coinciding *if for any $j \neq j'$, any $v \in A_j$, $v' \in A_{j'}$,*

$$\{i : T_{i,j} = v\} \neq \{i : T_{i,j'} = v'\}.$$

Theorem 7.4.1 *Suppose that the block cipher $E()$ is a pseudorandom permutation (with the encryption key as the random seed). Then the query protocol with enhanced security reveals only $\epsilon_{u,u'}$ for $u, u' \in [1, t]$. When the table T is non-coinciding, the query protocol with enhanced security is private.*

Proof: We construct a simulator $\mathcal{S}$ as follows. First, recall that $R_1(Q_u) = \{i : (i, j) \in R(Q_u)\}$ and $R_2(Q_u) = \{j : (i, j) \in R(Q_u)\}$. For $u = 1$ through t, if there exists $u' < u$ such that $\epsilon_{u,u'} = 1$ and that $R_1(Q_u) = R_1(Q_{u'})$, $\mathcal{S}$ sets $\overline{q}_u = \overline{q}_{u'}$ and $j'_{u,0} = j'_{u',0}$; if there exists $u' < u$ such that $\epsilon_{u,u'} = 1$ and that $R_1(Q_u) \neq R_1(Q_{u'})$, $\mathcal{S}$ chooses $\overline{q}_u$ from $\{0,1\}^{k_0} - \{\overline{q}_{u'} : u' < u \wedge \epsilon_{u,u'} = 1\}$ uniformly at random and sets $j'_{u,0} = j'_{u',0}$; otherwise, $\mathcal{S}$ chooses $\overline{q}_u$ from $\{0,1\}^{k_0}$ uniformly at random, and $j'_{u,0}$ from $[1, m] - \{j'_{u',0} : u' < u\}$ uniformly at random. Next, for $i = 1$ through n and $j = 1$ through m, $\mathcal{S}$ chooses $\overline{T'}_{i,j}[1]$ uniformly and independently

from $\{0,1\}^{k_0}$. For $u = 1$ through t, for each $i \in R_1(Q_u)$, $\mathcal{S}$ computes

$$\overline{T'}_{i,j'_{u,0}}[2] = E_{\overline{q}_u}(\overline{T'}_{i,j'_{u,0}}[1]).$$

For each pair (i,j) such that $\overline{T'}_{i,j}[2]$ has not been defined, $\mathcal{S}$ chooses $\overline{T'}_{i,j}[2]$ from $\{0,1\}^{k_0}$ uniformly and independently. Finally, $\mathcal{S}$ outputs $\overline{q}_1, \ldots, \overline{q}_t$, $\overline{T'}$. The indistinguishability by polynomial-size circuits follows from the pseudorandomness of $E()$.

When T is non-coinciding, in the above proof we can replace $\epsilon_{u,u'} = 1$ with $R_1(Q_u) = R_1(Q_{u'})$. Because these two conditions are equivalent, this replacement does not change the output of the simulator. On the other hand, because $\epsilon_{u,u'}$ is no longer needed by the simulator, we have shown the protocol is private. ■

7.5 Query Speedup Using Metadata

In the two solutions we have presented, performing a query on the encrypted table requires testing each row of the table. Clearly, this is very inefficient in large-size databases. In this section, we consider a modification to the basic solution that drastically speeds up queries. It is easy to make a similar modification to the solution with enhanced security.

We can improve significantly the efficiency if we are able to replace the sequential search in the basic solution with a binary search. However, our basic solution finds the appropriate rows by testing an equation, while a binary search cannot be used to find the items that satisfy an equation.

To sidestep this difficulty, we add some metadata[2] to eliminate the need for testing an equation: for each cell in the column, we add a tag and a link. The tag is decided by the value of the cell; the link points to the cell. We sort the metadata according to the order of the tags. When there is a query on the attribute, the user sends the appropriate tag to the database so that the database can perform a binary search on the tags. Note that the tags only depend on the encrypted values and do not leak any sensitive information about those data.

[2]The functionality of these metadata is analogous to indices in traditional database systems—to help speed up queries. However, since the structure and usage of these metadata are different from that of traditional indices (e.g., B+-trees), we do not call them indices. Note that indices like B+-trees cannot be used in our scenario.

We illustrate the concept with a simple example, shown in Figure 7.3. Consider a column of four cells with values 977, 204, 403, 155. We compute four tag values based on the corresponding cell values. Suppose that the tags we get are 3, 7, 4, 8. We sort the tags to get a list: 3, 4, 7, 8. After we add links to the corresponding cells, we finally get: (3, link to cell "977"), (4, link to cell "403"), (7, link to cell "204"), (8, link to cell "155").

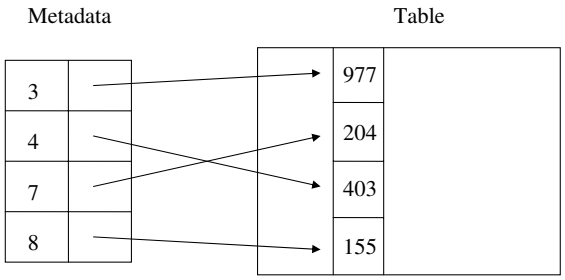


Figure 7.3: Example of Metadata

Nevertheless, there is a question of multiple occurrences of a single value: if multiple cells in the column have the same value, how do we choose the tags for these cells? Clearly, we cannot use the same tag for these cells; otherwise, when the database intruder looks at the tags, he can recognize cells with the same value. In fact, it should be hard for the intruder to find out which tags correspond to the same cell value. On the other hand, it should be easy for the user to derive the entire set of tags corresponding to a single value.

We resolve this dilemma using a two-step mapping. In the first step, we map a cell value $T_{i,j}$ to an intermediate value $H_{i,j}$ whose range is much larger. Then the $H_{i,j}$s are sparse in their range, which means around each value of $H_{i,j}$ there is typically a large "blank" space. Consequently, to represent multiple occurrences of the same cell value $T_{i,j}$, we can use multiple points starting from $H_{i,j}$. In the second step, we map these intermediate points to tags such that the continuous intermediate points become random-looking tags. See Figure 7.4 for an illustration.

In our design, the first step of mapping (from the cell value to the intermediate value) is implemented using an encryption of the cell value (appended with a k_2-bit 0 so that the input to the cipher is k_0 bits), where the encryption key is kept by the user. The second step of mapping (from the intermediate value to the tag) is implemented using another encryption, where the key

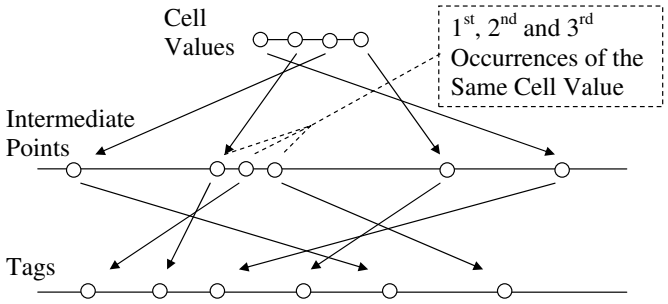


Figure 7.4: Two-Step Mapping from Cell Values to Tags

is again kept by the user. Since the database intruder does not know the two encryption keys, he cannot figure out which cell value corresponds to which tag, or which of the tags correspond to the same cell value. On the other hand, when there is a query, the user can simply send the database the tags for the cell value in the query; then the database can easily locate the rows satisfying the condition of this query.

Note that, for the convenience of queries, we should keep a counter of the occurrences of each cell value; otherwise, when the user has a query, he cannot know how many intermediate values (and thus how many tags) he should compute. Clearly such counters should be encrypted and stored in the database, where the encryption key is kept by the user. Each encrypted counter should be kept together with the corresponding intermediate value (of the first occurrence of the cell value), so that it can be identified by the user. When the database intruder observes the encrypted metadata, he does not know which cell value corresponds to which intermediate value and therefore does not know which cell value corresponds to the encrypted counter.

7.5.1 Solution Details

Metadata Format. To speed up queries on attribute A_j, the user picks keys $s_3, s_4, s_5 \in \{0,1\}^{k_0}$ independently and uniformly. For $i = 1, \ldots, n$, the user computes

$$H_{i,j} = E_{s_3}(T_{i,j}, 0).$$

For each value of each attribute, the user keeps a counter of the number of occurrences. If this is the $c_{i,j}$th occurrence of the value $T_{i,j}$ in the attribute A_j, the user computes

$$I_{i,j} = E_{s_4}((H_{i,j} + c_{i,j}) \bmod 2^{k_0}).$$

When $I'_{i,j}$s have been computed for all is, suppose the final value of the counter is $c_j(v)$ for each value v. Then the user encrypts $c_j(v)$ using secret key k_5:

$$C_j(v) = E_{k_5}(c_j(v), 0).$$

The user stores $L = \{(I_{i,j}, \text{link to row } T'_i)\}_{i\in[1,n]}$ and

$$\begin{aligned} B &\stackrel{\triangle}{=} \{(B_x[1], B_x[2])\}_{x\in[1,\ |\{T_{i,j}:i\in[1,n]\}|]} \\ &= \{(E_{s_3}(v,0), C_j(v))\}_{v\in\{T_{i,j}:i\in[1,n]\}} \end{aligned}$$

in the database as metadata for queries on attribute A_j. Note that L should be sorted in increasing order of $I_{i,j}$. The user keeps s_3, s_4, and s_5 secret.

Query Protocol. Now suppose there is a query *select* A_{j_1},..., A_{j_ℓ} *from* T *where* $A_j = v$. To carry out this query, the user first computes $h = E_{s_3}(v, 0)$, and sends h to the database. The database finds x such that $B_x[1] = h$ and sends the corresponding $C = B_x[2]$ back to the user. The user then decrypts C (and discards the k_2-bit tail) to get $c_j(v)$, the overall number of occurrences of v. For $c = 1, \ldots, c_j(v)$, the user computes

$$I_c = E_{s_4}((h + c) \bmod 2^{k_0}),$$

and sends I_c to the database. Since L is sorted in the increasing order of I_c, the database can easily locate I_c and find the link corresponding to I_c. For each row T'_i pointed by these links, the database sends the encrypted cells $T'_{i,j_1}[1], \ldots, T'_{i,j_\ell}[1]$ to the user. Finally, the user decrypts each received cell using secret key s_1, and discards the k_2-bit tail of the cleartext.

7.5.2 Experimental Results

To evaluate the speedup of our solution, we measure the query time on the same dataset used for testing the basic solution. Figure 7.5 compares the metadata generation time for four different attributes: Class, Finance, Parent, and ID. The metadata generation time depends on not only the number of rows in the table, but also the domain size of the attribute (more precisely, the number of the different values appearing in the attribute). In the attributes we experimented with, Class, Finance, and Parenthave small domain sizes; the metadata generation time for each of them is about 6 seconds. In contrast, generating metadata on ID attribute needs about twice as much time because is the IDattribute has a large domain. Figure 7.6 compares the

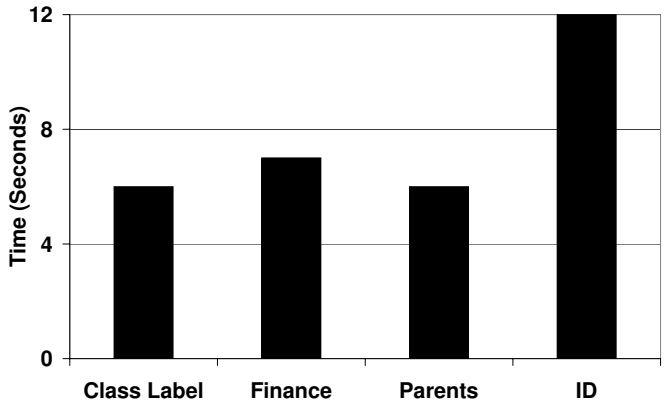


Figure 7.5: Computational Time to Generate Metadata

query time of the basic solution and that of the solution with metadata on the following four queries: to select all records where Class=recommend, where Class=very_recom, where Parent=usual, and where ID=1000. The results of the first and the fourth queries have only 2 and 1 records, respectively. For such queries, the solution with metadata is so fast that the query time can hardly be seen in the figure. The other two queries have more records in their results: the second query has 328 records in its result and our solution with metadata saves about 94% of the query time; the third query has 4320 records in its result and our solution with metadata saves about 79% of the query time. Clearly, the trend is that the solution with metadata gains more in efficiency if there are fewer records in the query result. However, even for a query with a large number of records in the result, the solution with metadata is much faster than the basic

solution.

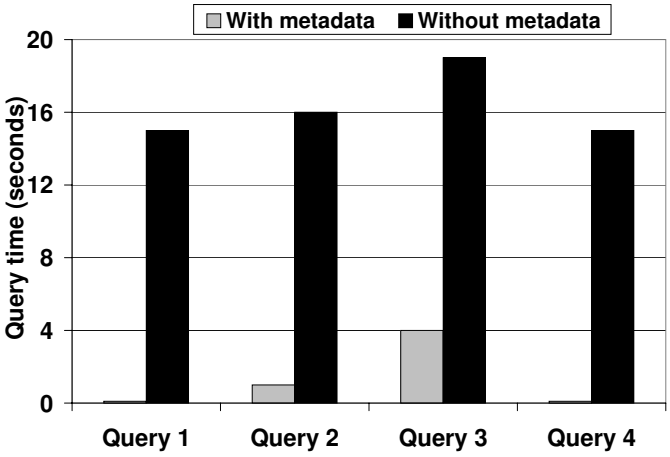


Figure 7.6: Query Time: Solution with Metadata vs. Basic Solution

7.6 Inserting and Deleting Rows

It is relatively easy to insert and delete rows when we use the data format defined in the basic solution (Section 7.3) or the solution with enhanced security (Section 7.4). However, it is much more complex to do so when we have metadata for query speedup (Section 7.5). In this section, we present a method for inserting and deleting rows in an encrypted table with metadata for query speedup. When there is no such metadata, inserting and deleting rows can be achieved using a simplified version of the solution we give.

7.6.1 Solution Overview

To insert a new row, the user sends the database the new row and the intermediate value corresponding to the first occurrence of the cell value in this new row. (In this section, just as in Section 7.5, we focus on cell values in the column with metadata and thus by saying "cell value" we actually mean "cell value in the column with metadata". The database inserts the new row into the encrypted table and sends back the encrypted counter for this cell value. Using the value of this counter, the user computes a new encrypted counter (whose cleartext is the old cleartext counter plus one) and a tag for the cell in the new row. The user sends them to the database, which replaces the old counter with the new one and inserts the tag (together with the

link to the new row) to the tag list.

Now consider the case in which the user would like to delete rows which satisfy a condition. Using the technique in our query protocol, we can allow the database to find these rows. Of course, instead of returning the appropriate cells in these rows to the user, the database should remove these rows. To update the metadata for a deleted row, the user sends the database the intermediate value corresponding to the first occurrence of the cell value in this row. The database sends back the encrypted counter for this cell value. Using the value of this counter, the user computes a new encrypted counter (whose cleartext is the old cleartext counter minus one) and the tag for the last occurrence of the same cell value. The user sends them to the database, who replaces the old counter with the new one. The database removes the link pointing to the row deleted, as well as the corresponding tag; he searches in the list of tags for the tag he received from the user; he replaces the tag found in the list with the tag he just removed.

Note that the above outlined deletion protocol reveals the coordinate of the row that has the last occurrence of the cell value in the deleted row. This information leakage could be avoided by having the user change the keys of the two-step mapping and recompute the entire list of tags.

7.6.2 Solution Details

The solution presented below is for an encrypted table T' with metadata for query speedup on attribute A_j.

Insertion. Suppose there is a insert SQL to table T. The user computes an encrypted row from the given values, as we specify in Section 7.3. We denote the new row by $T_{\hat{i}}$ and the encrypted new row by $T'_{\hat{i}}$. The user computes $h = E_{s_3}(T_{\hat{i},j}, 0)$ and sends $T'_{\hat{i}}, h$ to the database. The database inserts $T'_{\hat{i}}$ to the T'. He finds x such that $B_x[1] = h$, and sends the corresponding $C = B_x[2]$ back to the user. The user then decrypts C (and discards the k_2-bit tail) to get $c_j(T_{\hat{i},j})$, the overall number of occurrences of $T_{\hat{i},j}$. Next, the user computes

$$C' = E_{k_5}(c_j(T_{\hat{i},j}) + 1, 0),$$

$$I' = E_{s_4}((h + c_j(T_{\hat{i},j}) + 1) \bmod 2^{k_0}),$$

and sends C', I' to the database.

The database sets $B_x[2] = C'$ and inserts $(I', \text{link to row } T'_{\hat{i}})$ at the appropriate position in L such that L is still in increasing order of $I_{i,j}$s.

Deletion. Suppose there is a command "delete from T where $A_{j_0} = v$." The user enables the database to find the rows where $A_{j_0} = v$ using the method we presented in Section 7.5. The database removes these rows. We denote a deleted row by $T'_{\hat{i}}$. We now describe how to update the metadata for this deleted row. Updating the metadata for multiple deleted rows can be easily achieved through straightforward extension of this process.

The user computes $h = E_{s_3}(T_{\hat{i},j}, 0)$, and sends h to the database. The database finds x such that $B_x[1] = h$, and sends the corresponding $C = B_x[2]$ back to the user. The user then decrypts C (and discards the k_2-bit tail) to get $c_j(T_{\hat{i},j})$, the overall number of occurrences of $T_{\hat{i},j}$. Next, the user computes $C' = E_{k_5}(c_j(T_{\hat{i},j}) - 1, 0)$ and $I'_c = E_{s_4}((h + c_j(T_{\hat{i},j}) - 1) \bmod 2^{k_0})$. The user sends C', I'_c to the database.

The database sets $B_x[2] = C'$. Then he searches in L for the link pointing to the deleted row. He removes the entry he finds (which includes both the tag and the link). Furthermore, suppose the tag in the removed entry is I'; if $I' \neq I'_c$, the database replaces I'_c with I' in L.

7.7 Imprecise Query Protocols

As we mentioned in Section 7.2, there is a minimum information revelation $R(Q)$ for precise query protocols. In some applications, revealing $R(Q)$ is acceptable; but in other applications, it is not. If revealing $R(Q)$ is not acceptable, we must use imprecise query protocols.

To protect $R(Q)$ against the database intruder, we can design the protocol in such a way that the database returns to the user additional cells which are not really in the result of the query. For example, we can make two simple modifications to our basic solution in Section 7.3:

- For each pair (i, j), we only store the β-bit head (where β is a small constant) of $T'_{i,j}[2]$

in the encrypted table. Therefore, when there is a query, instead of testing

$$T'_{i,j_0}[2] = E_q(T'_{i,j_0}[1]),$$

the database tests whether the stored β-bit head of $T'_{i,j_0}[2]$ is equal to the β-bit head of $E_q(T'_{i,j_0}[1])$. Since β is small, one would expect there to be a good number of rows that does not satisfy the condition of the query but can pass the above test.

- When the user sends the message to the database, she inserts random values (chosen from $[1, m] - \{j_1, \ldots, j_\ell\}$) to the set $\{j_1, \ldots, j_\ell\}$, such that the database will return encrypted cells from more columns.

We can make similar modifications to our other solutions. We can also make other types of modifications to convert precise query protocols to imprecise query protocols. Note that, when we make such extensions to imprecise query protocols, we introduce a trade-off analogous to that of [56]: the more privacy we want to obtain, the more false positive cells the database needs to send to the user, and vice versa.

7.8 Discussion

In this chapter, we investigated privacy-preserving queries on encrypted data. In particular, we presented privacy-preserving protocols for certain types of queries. Although the supported queries are limited, our main goal in this chapter is to provide formal, quantitative (and cryptographically strong) security.

We note that, in general, it is difficult to evaluate the correctness and security of a new design if no quantitative analysis of information leak is given. It is therefore beneficial to introduce quantitative measures of privacy such as those we have introduced. It is our hope that these measures may be useful elsewhere.

Chapter 8

Summary

Throughout this thesis, we have presented five techniques to protect data privacy in distributed settings. we gave thorough analysis to prove that data privacy is protected by these techniques. We also presented experimental results to show how efficient and practical our techniques are.

In the first setting, we designed a technique enabling a data miner to learn the frequencies of combinations of data values without even seeing the respondents' data. The miner can learn the frequencies without gaining other knowledge about the data of individuals. The frequency-learning technique is very powerful and can enable various data mining tasks. In particular, the classification models, e.g., naive Bayes classifiers, can be constructed by using the frequency-computing technique.

In the second setting, we considered a general data collection problem—a data miner wants to collect data from respondents in order to learn any model from them. Under the condition in which each respondent's data are de-identified, we provided a general solution to protect respondents' data, called anonymity-preserving data collection. In this solution, we implemented a transparently anonymous communication channel between each respondent and the miner such that the miner can not link an respondent to his or her data. Our solution can protect each respondent's anonymity in the semi-honest model. Our implementation results show that the solution has good performance and can be deployed over the Internet to enable an organization to anonymously collect users' data on a large scale.

In the third setting, we assume respondents' data do not contain any identifiable information

which can be used to link with their identities. However, in some cases, respondents' data may contain quasi-identifiers which may be used to violate their privacy. Based on the existing k-anonymization solutions developed in the centralized setting, we designed a solution to enhance the privacy protection of k-anonymization in the distributed setting. Our solution can protect respondents' privacy such that each respondent is decoupled from his or her data in the data submission step. This is the first solution that maintains end-to-end privacy between a miner and each respondent.

In the forth setting, we considered a special case where Bayesian networks are learned across two sensitive databases owned by two parties. We started from the existing *K2* algorithm and designed privacy-preserving mining solutions for the task of learning Bayesian networks across two sensitive databases. Our solution enables both parties to learn the Bayesian network, which encodes the relationships among the features in their databases. Such solutions might enable mining of databases at multiple organizations without compromising data privacy or organizational policies. For example, among different banks, mining of multiple databases could result in faster and more accurate detection of fraud.

In the fifth setting, we studied efficient methods for queries on encrypted data. We provided strong, quantitative security guarantees with rigorous proofs. Specifically, the system we designed shows that even if an intruder breaks into the database and observes some interactions between the database and database users, the attacker learns very little about the data stored in the database.

Although we have conducted an intensive study of protecting data privacy in the distributed setting, there are still some open topics. In the distributed settings which we consider, participants may tend to exhibit malicious behavior and breach others' privacy to gain some benefits. The existing cryptographic techniques against malicious adversaries are too expensive to satisfy stringent performance requirements in practical applications. Although we have addressed certain types of adversaries in this thesis, how one may defend against more powerful adversaries in an efficient way is an open question.

Rising from our work on Bayesian networks, the challenge is how to provide a more general privacy-preserving distributed solution for learning the networks, which is not constrained

by certain algorithms and certain score functions. More generally, an interesting direction is to develop a general framework for privacy-preserving distributed data mining such that any distributed data mining problem can be solved in this framework. The rough idea is to first design basic primitives which can be composed to build more complex secure solutions, and then implement the primitives as libraries for further use. But it is a grand challenge and there are many open issues to be solved. For example, what primitives are needed? How can we ensure that the compositions of primitives are still secure? How do we implement and design the APIs of those primitives? Once these problems are solved, the procedure of privacy-preserving distributed data mining can be optimized greatly, and that can further assist adaption of data mining applications in broad areas.

Our work on querying encrypted data supports certain queries. For many practical database applications, more complex queries must be supported. A future research topic is to extend this work to allow more complex queries. Ideally, the extension should maintain strong and quantifiable security while achieving good efficiency for complex queries. The first challenge is how to design cryptographic techniques to secure databases that can both provide powerful functionalities and achieve a quantifiable security guarantee. The second challenge is on how to embed these techniques into current data base management systems and make them transparent to the end users. These open problems deserve further investigation.

Bibliography

[1] Oracle Corporation. Database Encryption in Oracle9i, 2001.

[2] IBM Data Encryption for IMS and DB2 Databases, Version 1.1, 2003.

[3] J. O. Achugbue and F. Y. Chin. The effectiveness of output modification by rounding for protection of statistical databases. *INFOR*, 17(3):209–218, 1979.

[4] N. Adam and J. Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Computer Survey*, 21(4):515–556, 1989.

[5] D. Agrawal and C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 247–255, 2001.

[6] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 86–97, 2003.

[7] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 143–154, 2002.

[8] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2004.

[9] R. Agrawal and R. Srikant. Privacy preserving data mining. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 439–450, 2000.

[10] L. Ahn, A. Bortz, and N. J. Hopper. k-anonymous message transmission. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 122–130, 2003.

[11] W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology - Proceedings of EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135, 2001.

[12] M. Atallah and W. Du. Secure multi-party computational geometry. In *Proceedings of the 7th International Workshop on Algorithms and Data Structures*, pages 165–179, 2001.

[13] L. L. Beck. A security mechanism for statistical databases. *ACM Transactions on Database Systems*, 5(3):316–338, 1980.

[14] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 544–553, 1994.

[15] C. Blake and C. Merz. UCI repository of machine learning databases, 1998.

[16] D. Boneh. The decision Diffie-Hellman problem. In *ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63, 1998.

[17] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology - Proceedings of EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522, 2004.

[18] L. Bouganim and P. Pucheral. Chip-secured data access: Confidential data on untrusted servers. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 131–142, 2002.

[19] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology - Proceedings of EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, 1999.

[20] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[21] R. Canetti, Y. Ishai, R. Kumar, M. Reiter, R. Rubinfeld, and R. Wright. Selective private function evaluation with applications to private statistics. In *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 293–304, 2001.

[22] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *In Proceedings of the 3rd International Conference on Applied Cryptography and Network Security*, pages 442–455, 2005.

[23] D. Chaum. Untraceable electronic mail, return address and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[24] D. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[25] R. Chen, K. Sivakumar, and H. Kargupta. Learning Bayesian network structure from distributed data. In *Proceedings of the SIAM International Data Mining Conference*, 2003.

[26] R. Chen, K. Sivakumar, and H. Kargupta. Collective mining of Bayesian networks from distributed heterogeneous data. *Knowledge and Information Systems*, 6(2):164–187, 2004.

[27] D. M. Chickering. Learning Bayesian networks is NP-complete. *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130, 1996.

[28] F. Y. Chin and G. Ozsoyoglu. Auditing and inference control in statistical databases. *IEEE Transactions on Sofware Engineering*, SE-8(6):113–139, 1982.

[29] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 261–270, 1995.

[30] G. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.

[31] L. Cranor, editor. *Communications of the ACM*42(2), *Special Issue on Internet Privacy*, 1999.

[32] E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 93–102, 2003.

[33] E. Dash. Lost credit data improperly kept, company admits. *New York Times*, 2005.

[34] G. I. Davida, D. L. Wells, and J. B. Kam. A database encryption system with subkeys. *ACM Transactions on Database Systems*, 6(2):312–328, 1981.

[35] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Advances in Cryptology - Proceedings of CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, 1990.

[36] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 202–210, 2003.

[37] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[38] W. Du and Z. Zhan. Using randomized response techniques for privacy-preserving data mining. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 505–510, 2003.

[39] C. Dwork and K. Nissim. Privacy-preserving datamining on vertically partitioned databases. In *Advances in Cryptology - Proceedings of CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, 2004.

[40] V. Estivill-Castro and L. Brankovic. Balancing privacy against precision in mining for logic rules. In *Proceedings of the First International Data Warehousing and Knowledge Discovery*, pages 389–398, 1999.

[41] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 211–222, 2003.

[42] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–228, 2002.

[43] J. Feigenbaum, M. Y. Liberman, and R. N. Wright. Cryptographic protection of databases and software. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 2, pages 161–172, 1990.

[44] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology - Proceedings of EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19, 2004.

[45] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure applications of Pedersen's distributed key generation protocol. In *CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 373–390, 2003.

[46] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pages 151–160, 1998.

[47] B. Gilburd, A. Schuster, and R. Wolff. k-TTP: a new privacy model for large-scale distributed environments. In *Proceedings of the 2004 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 563–568, 2004.

[48] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen. On private scalar product computation for privacy-preserving data mining. In *Proceedings of the Seventh Annual International Conference in Information Security and Cryptology*, pages 104–120, 2004.

[49] E. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216. `http://eprint.iacr.org/2003/216/`.

[50] O. Goldreich. *Foundations of Cryptography*, volume 1. Cambridge University Press, 2004.

[51] O. Goldreich. *Foundations of Cryptography*, volume 2. Cambridge University Press, 2004.

[52] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *Proceedings of the 19th Annual ACM Conference on Theory of Computing*, pages 218–229, 1987.

[53] P. Golle and A. Juels. Dining cryptographers revisited. In *Advances in Cryptology - Proceedings of EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 456–473, 2004.

[54] P. Golle, S. Zhong, D. Boneh, M. Jakobsson, and A. Juels. Optimistic mixing for exit-polls. In *Advances in Cryptology - ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 451–465, 2002.

[55] H. Hacigumus, B. Iyer, and S. Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *Proceedings of the Ninth International Conference on Database Systems for Advanced Applications*, pages 125–136, 2004.

[56] H. Hacigumus, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of data*, pages 216–227, 2002.

[57] H. Hacigumus, B. R. Iyer, and S. Mehrotra. Providing database as a service. In *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[58] J. He and J. Wang. Cryptography and relational database management systems. In *Proceedings of the International Database Engineering and Application Symposium*, 2001.

[59] HIPAA. The health insurance portability and accountability act of 1996, 1998. Available at `www.cms.hhs.gov/hipaa`.

[60] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology - Proceedings of EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, 2000.

[61] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *Proceedings of the 30th International Conference on Very Large Data Bases*, pages 720–731, 2004.

[62] B. Iyer, S. Mehrotra, E. Mykletun, G. Tsudik, and Y. Wu. A framework for efficient storage security in RDBMS. In *Proceedings of the 9th International Conference on Extending Database Technology*, pages 147–164, 2004.

[63] C. H. P. J. M. Kleinberg and P. Raghavan. Auditing boolean attributes. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 86–91, 2000.

[64] G. Jagannathan and R. N. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 593–599, 2005.

[65] M. Jakobsson. Flash mixing. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 83–89, 1999.

[66] E. Johnson and H. Kargupta. Collective, hierarchical clustering from distributed, heterogeneous data. *Lecture Notes in Computer Science*, 1759:221–244, 1999.

[67] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *Proceedings of the ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 24–31, 2002.

[68] M. Kantarcioglu, J. Jin, and C. Clifton. When do data mining results violate privacy? In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 599–604, 2004.

[69] M. Kantarcioglu and J. Vaidya. Privacy preserving naive Bayes classifier for horizontally partitioned data. In *IEEE Workshop on Privacy Preserving Data Mining*, 2003.

[70] O. Kardes, R. S. Ryger, R. N. Wright, and J. Feigenbaum. Implementing privacy-preserving Bayesian-net discovery for vertically partitioned data. In *Proceedings of the ICDM Workshop on Privacy and Security Aspects of Data Mining*, pages 26–34, 2005.

[71] H. Kargupta, B. Park, D. Hershberger, and E. Johnson. Collective data mining: a new perspective towards distributed data mining. *Advances in Distributed and parallel Knowledge Discovery*, pages 133–184, 1999.

[72] J. Karlsson. Using encryption for secure data storage in mobile database systems. Friedrich-Schiller-Universitat Jena, 2002.

[73] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database computationally-private information retrieval. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, page 364, 1997.

[74] B. N. Levine and C. Shields. Hordes - a multicast based protocol for anonymity. *Journal of Computer Security*, 10(3):213–240, 2002.

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

[76] K. Liu, H. Kargupta, and J. Ryan. Multiplicative noise, random projection, and privacy preserving data mining from distributed multi-party data. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):92–106, 2005.

[77] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *Proceedings of 13th Usenix Security Symposium*, 2004.

[78] D. Meng, K. Sivakumar, and H. Kargupta. Privacy-sensitive Bayesian network parameter learning. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, 2004.

[79] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 223–228, 2004.

[80] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[81] D. E. O'Leary. Some privacy issues in knowledge discovery: The OECD personal privacy guidelines. *IEEE Expert*, 10(2):48–52, 1995.

[82] G. Ozsoyoglu, D. Singer, and S. Chung. Anti-tamper databases: Querying encrypted databases. In *Proceedings of the 17th Annual IFIP WG 11.3 Working Conference on Database and Applications Security*, 2003.

[83] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *Advances in Cryptology - Proceedings of EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, 1999.

[84] B. Park, R. Ayyagari, and H. Kargupta. A Fourier analysis-based approach to learn classifier from distributed heterogeneous data. In *Proceedings of the 1st SIAM International Conference on Data Mining*, 2001.

[85] C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *Advances in Cryptology - Proceedings of EUROCRYPT 93*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259, 1993.

[86] E. Parliament. Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of the European Communities*, page 31, 1995.

[87] E. Parliament. Directive 97/66/EC of the European Parliament and of the Council of 15 December 1997 concering the processing of personal data and the protection of privacy in the telecommunications sector. *Official Journal of the European Communities*, pages 1–8, 1998.

[88] S. Reiss. Practical data swapping: The first steps. *ACM Transactions on Database Systems*, 9(1):20–37, 1984.

[89] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[90] S. Rizvi and J. Haritsa. Maintaining data privacy in association rule mining. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

[91] K. Sako and J. Kilian. Receipt-free mix-type voting schemes — a practical solution to the implementation of a voting booth. In *Advances in Cryptology - Proceedings of EUROCRYPT 95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403, 1995.

[92] P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information (abstract). In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, page 188, 1998.

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

[94] V. Shmatikov and J. Brickell. Efficient anonymity-preserving data collection. In *Proceedings of 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[95] A. Shoshani. Statistical databases: Characteristics, problems and some solutions. In *Proceedings of the eighth International Conference on Very Large Data Bases*, pages 208–222, 1982.

[96] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2000.

[97] S. Stolfo, A. Prodromidis, S. Tselepis, W. Lee, D. Fan, and P. Chan. JAM: Java agents for meta-learning over distributed databases. In *Proceedings of Third International Conference on Knowledge Discovery and Data Mining*, pages 74–81, 1997.

[98] H. Subramaniam, R. N. Wright, and Z. Yang. Experimental analysis of privacy-preserving statistics computation. In *Proceedings of the VLDB Worshop on Secure Data Management*, pages 55–66, 2004.

[99] L. Sweeney. Guaranteeing anonymity when sharing medical data, the datafly system. In *Journal of the American Medical Informatics Association*, pages 51–55, 1997.

[100] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):571–588, 2002.

[101] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[102] J. Traub, Y. Yemini, and H. Wozniakowksi. The statistical security of a statistical database. *ACM Transactions on Database Systems*, 9(4):672–679, 1984.

[103] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 639–644, 2002.

[104] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 206–215, 2003.

[105] J. Vaidya and C. Clifton. Privacy preserving naive Bayes classifier on vertically partitioned data. In *Proceedings of 2004 SIAM International Conference on Data Mining*, 2004.

[106] R. Vingralek. A small-footprint, secure database system. In *Proceedings of the 28th International Conference on very Large Data Bases*, 2002.

[107] M. Waidner. Unconditional sender and recipient untraceability in spite of active attacks. In *Advances in Cryptology - Proceedings of EUROCRYPT 89*, volume 434 of *Lecture Notes in Computer Science*, 1989.

[108] K. Yamanishi. Distributed cooperative Bayesian learning strategies. *Information and Computation*, 150(1):22–56, 1999.

[109] Z. Yang, S. Zhong, and R. N. Wright. Anonymity-preserving data collection. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 334–343, 2005.

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