A STUDY OF SEVERAL SPECIFIC SECURE TWO-PARTY COMPUTATION PROBLEMS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Wenliang Du

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2001

To my Father, my Mother and Jing.

ACKNOWLEDGMENTS

Throughout my graduate experience, I have been very fortunate to cross paths with great people, who have had a dramatic impact on my research and my accomplishments.

The person who contributed most to my thesis research was my major advisor Professor Mikhail Atallah. I was very lucky to discuss an interesting problem with him when I was still painfully looking for a thesis topic. It was that discussion that led to the discovery of my thesis topic. Ever since, it has been a privilege and a pleasure to work with him, to talk with him, and to learn from him. His amazing quickness when coming up with ideas has always been inspiring, his insights and suggestions have been invaluable for my work, and his willingness to discuss with me at anytime has been very helpful. I am very grateful to him.

The person who always advised me, supported me and helped me during my five years' graduate research is my other advisor, Professor Eugene Spafford, whose research in the security area was the major reason that attracted me to Purdue University. During these five years, he has provided me with a lot of support and the freedom I needed to be successful in my research; he has taught me a great deal about how to do research, how to think of problems, and most of all, how to become a respected scholar. I am greatly indebted to him.

My graduate studies were conducted in the Center for Education and Research in Information Assurance and Security (CERIAS), formerly known as the Computer Operations, Audit and Security Technology (COAST) Laboratory. I benefited a lot from the research and education environment that CERIAS provided. I thank all of my colleagues in CERIAS, especially Hoi Chang, Tom Daniels, Chapman Flack, Benjamin Kuperman, Mahesh Tripunitara and Diego Zamboni for accompanying me from COAST to CERIAS, and for invaluable discussions during the last five years.

Several people deserve special mention: Professor Aditya Mathur for advising me during the first two years of my research in the security testing area; Professor Jens Palsberg and Professor Sunil Prabhakar for serving on my thesis committee and for reading my thesis; Dr. William J. Gorman in the graduate student office for helping me through a lot of official paper work; Marlene Walls for administrative support; Vince Koser, Susana Soriano and Kent Wert for their excellent systems administration support; Professor Clay Shields for giving me valuable job-hunting advice; Jared Crane, Rajeev Gopalakrishna and Florian Kerschbaum for the research discussions.

Most of all, I would like to thank my beloved wife, my parents and my brothers for their unconditional love, encouragement and support.

Portions of this work were supported by Grant EIA-9903545 from the National Science Foundation, and by the various sponsors of CERIAS; that support is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Du, Wenliang. Ph.D., Purdue University, August, 2001. A Study of Several Specific Secure Two-party Computation Problems. Major Professors: Mikhail J. Atallah and Eugene H. Spafford.

Alice has a private input x (of any data type, such as a number, a matrix or a data set). Bob has another private input y. Alice and Bob want to cooperatively conduct a specific computation on x and y without disclosing to the other person any information about her or his private input except for what could be derived from the results. This problem is a Secure Two-party Computation (STC) problem, which has been extensively studied in the past. Several generic solutions have been proposed to solve the general STC problem; however the generic solutions are often too inefficient to be practical. Therefore, in this dissertation, we study several specific STC problems with the goal of finding more efficient solutions than the generic ones.

We introduce a number of specific STC problems in the domains of scientific computation, statistical analysis, computational geometry and database query. Most of the problems have not been studied before in the literature.

To solve these problems:

- We investigate how *data perturbation* could be used to hide data. Data perturbation hides a datum by adding to it a random number. We show that this technique is effective in preserving privacy.

- We explore how domain specific knowledge can improve the efficiency of the solutions that we develop over the generic solutions that do not consider domain specific knowledge. We show that such knowledge is important in both hiding data and achieving higher efficiency.

- We also introduce a number of common building blocks that are useful in solving secure two-party computation problems in various computation domains.

1. INTRODUCTION

The growth of the Internet opens up tremendous opportunities for cooperative computation, where the answer depends on the private inputs of separate entities. These computations could sometimes occur between mutually untrusting entities. The problem is trivial if the context allows the conduct of these computations by a trusted entity that would know the inputs from all the participants. However, if the context disallows this then the techniques that allow this type of computation without disclosing each entity's private input become relevant and can provide useful solutions.

The above problem is referred to as the Secure Multi-party Computation problem (SMC) in the literature [105]. Generally speaking, a secure multi-party computation problem deals with computing a function on any input in a distributed network where each participant holds one of the inputs, and that no more information is revealed to a participant in the computation than can be inferred from that participant's input and output [54].

In theory, the general secure multi-party computation problem is solvable using circuit evaluation protocols [105, 57, 55]. While this approach is appealing in its generality and simplicity, the communication complexity of the protocol it generates depends on the size of the circuit. This size depends on the size of the input and

on the complexity of expressing F as a circuit. In addition, the solution involves large constant factors in its complexity. Therefore, as Goldreich points out in [55], using the solutions derived by these general results for special cases of multi-party computation can be impractical; special solutions should be developed for special cases for efficiency reasons.

This dissertation is a first step in this direction for several specific computation domains including computational geometry, scientific computation, statistical analysis and database query (approximate). In this study, we mainly focus on the secure two-party computation problems, instead of on the more general secure multi-party computation problems although we believe the extension from two-party to multi-party is a minor modification (for the circuit evaluation protocol, the construction of multi-party protocols for the semi-honest model is a minor modification of the construction used in the two-party case [55]). We will leave such an extension as our future work, and primarily focus on the secure two-party computation problems in this dissertation.

1.1 Contributions

First of all, we defined a new field of research by applying the secure two-party computation concept to specific computation problems in the area of scientific computation, computational geometry, statistical analysis and database query (approximate matching). Although general secure two-party computation problems have been studied extensively, the specific problems defined in this dissertation have not been studied in the literature; this dissertation is the first to introduce and study them.

Second, in this dissertation, we have proposed methods to solve those specific secure two-party computation problems that we have introduced. We have shown that the solutions of our methods are more efficient than the generic solutions (the circuit evaluation protocols).

Third, throughout this study, we have identified a number of building blocks that are commonly used to solve various secure two-party computation problems. We have demonstrated how to use them to solve various problems.

1.2 Thesis Statement

It is possible to solve, in a way more efficient than the general solution, secure two-party computation problems for specific domains such as scientific computation, geometric computation, basic statistical analysis and database query, given that:

- Both parties are semi-honest: informally speaking, they follow the protocol properly but may try to derive the other party's inputs from intermediate results they see.
- Each party's computation is polynomial time-bounded.
- The efficiency is measured using the communication complexity–the amount of data exchanged between the two parties.

1.3 Our Results

1.3.1 Privacy-Preserving Scalar Product Problems

Throughout this study, we find scalar product computation is useful in many situations. Therefore, we have extensively studied the scalar product problem in the

Secure Two-party Computation context. This problem consists of two parties, each having a private vector: they want to jointly compute the scalar product of their private vectors, but neither one wants to disclose its private vector to the other.

We have explored several ways to solve this problem. Our first solution uses an untrusted third party, who does not learn anything about the vectors; our second solution uses the 1-out-of-N Oblivious Transfer protocol to achieve the privacy of the vectors; our third solution uses a homomorphic encryption system to achieve privacy. These three approaches not only provide solutions to the scalar product problem, but also demonstrate three useful techniques in solving secure two-party computation problems.

1.3.2 Scientific Computation Problems

We introduce the Secure Two-Party Scientific Computations (STPSC) problem. The general definition of the STPSC problem is that two or more parties want to conduct a scientific computation based on their private inputs, but neither party is willing to disclose its own input to anybody else. We have further defined three specific STPSC problems, including the Secure Two-Party Linear System of Equations (STP-LSE) problem, the Secure Two-Party Linear Least-Square (STP-LLS) problem, and the Secure Two-Party Linear Programming (STP-LP) problem.

These three STPSC problems all involve a Matrix $M = M_1 + M_2$, where M_1 belongs to one party and M_2 belongs to the other party. The difficulty of these problems is that no one knows M, so there is no easy way to directly use the algorithms intended for the usual environment (without the privacy requirements). We have

devised a technique to transform M to M' in an "one-way fashion" such that for the party who knows M', it is computationally impossible to derive the value of M_i unknown to this party. This transformation also allows the party who knows M' to solve the problem using the usual algorithms.

The approach used here represents one of the important techniques: transforming the two-party problem to a one-party problem such that the algorithms intended for the usual environment can be used directly, while the final output could be derived from the results of the one-party problem.

1.3.3 Geometric Computation Problems

In this dissertation, we have also studied three secure two-party geometric computation problems: the point-inclusion problem, the intersection problem and the closest-pair problem. In the point-inclusion problem, one party has a point and another party has a region; they want to find out whether the point is inside the region or not. In the intersection problem, two parties each have a private shape and they want to know whether these two shapes intersect. In the closest-pair problem, one party has a set of red points in a plane, and the other party has a set of blue points in the same plane; they want to find the closest red-blue pair. In these problems, neither party wants the other party or any third party to know any information about his or her private geometric elements.

The approach used here represents another important technique: computing based on "shared secrets." The computation of geometric computation problems usually contains a procedure consisting of a few steps. It is not difficult to conduct each

individual step securely, but doing this has to disclose the result of each step, a disclosure leading to a privacy compromise. In our approach, we use a secure two-party protocol to conduct the computation of each step, but at the end nobody has full knowledge of the result: namely, both parties have a piece of the result, and from a single piece it is impossible to derive the whole result. The next challenge is how to conduct the subsequent steps based on the "shared secret." We have demonstrated several approaches to achieve this.

1.3.4 Basic Statistical Analysis

Statistical analysis is useful in many areas, but current techniques usually require one to know the whole data set to conduct the analysis. We have studied statistical analysis problems under a different environment, an environment where one cannot obtain the whole data set. More specifically, in this environment, two parties each have a private data set, and they do not want to share the data sets. The parties are interested in learning statistical analysis results based on the joint data set. Such results include mean values, variance, correlation coefficient and linear regression line.

1.3.5 Database Query Based on Approximate Matching

We have also worked on the privacy-preserving remote database query problem. In this problem, one party wants to query a remote database belonging to another party, but neither party is willing to disclose to the other party their own inputs, namely the query and the contents of the database that cannot be derived from the results. Practical protocols for solving such a problem make possible new forms of e-commerce between proprietary database owners and customers with privacy concerns

who seek to interrogate the database. Our research focuses on approximate match, i.e., the result of the database query is the one that most closely matches the query.

We solve the problem using an untrusted third party–a party who should not learn any useful information about either participant's inputs.

1.3.6 Other Secure Two-Party Computation Problems

Apart from the above problems, there are many other secure two-party computation problems that have not been studied before. To facilitate the discovery and the study of these problems, we have proposed a framework that helps one to identify secure two-party computation problems in various computation domains. Using this framework, we have identified a list of problems for further studies.

1.4 Organization of the Dissertation

This dissertation is organized as follows. In Chapter 2, we define some general notations, terminologies, assumptions, and known results that will be used throughout the dissertation. In Chapter 3, we describe protocols for two important building blocks: the Scalar Product Protocol and the Vector Dominance Problem. In Chapter 4, 5, 6 and 7, we study several specific secure two-party computation problems in four different computation domains. In Chapter 8, we describe a list of other specific secure two-party computation problems, and a framework that helps us to identify those problems. Finally, in Chapter 9, we present the conclusions, summarize our findings and experiences, and discuss future directions.

2. PRELIMINARIES

In this chapter we provide some general notations, definitions, and known results that will be used throughout the dissertation. Additional definitions and preliminaries, specific to some parts of the dissertation, appear within the following chapters as appropriate.

2.1 Semi-Honest Model and Malicious Model

In the study of secure two-party and multi-party computation problems, two models are commonly assumed: the *semi-honest* model and the *malicious* model. A semi-honest party is one who follows the protocol properly with the exception that it keeps a record of all its intermediate computations; the record could be used later for deriving additional information about other parties' inputs. In the malicious model, a malicious party does not need to follow the protocol properly, for example, it can substitute its local input and enter the protocol with an input other than the one provided to them.

It has been shown that any protocol secure in the semi-honest model can be transformed into a protocol secure in the malicious model [55]. However, the semi-honest model is not merely an important methodological locus, it may also provide a good model of certain settings. In particular, in reality deviating from the specified program–which may be invoked inside a complex application software–may be more

difficult than merely recording the contents of some communication registers. Thus, whereas totally-honest behavior may be difficult to enforce, semi-honest behavior may be assumed in many settings [55].

2.2 Definitions

SEMI-HONEST PARTY. A *semi-honest* (also termed *passive*) party is one who follows the protocol properly with the exception that it keeps a record of all its intermediate computations in an attempt to later derive additional information. This dissertation discusses secure two-party computation problems with the assumption that all participants are semi-honest.

UNTRUSTED THIRD PARTY. In some two-party protocols, an untrusted third party is introduced for efficiency. An untrusted third party is semi-honest, i.e., it properly executes the protocol and records any intermediate results, but it should not be able to derive the inputs or the results of the protocol from what it has recorded. The untrusted third party does not collude with any party.

PRIVACY. A formal definition of privacy for secure two-party computation is given in [55]. The definition says that a protocol *privately computes* f if whatever a semi-honest *polynomial-time bounded* party can be obtained after participating in the protocol could be essentially obtained from the input and output available to that party. This is stated using the simulation paradigm. Furthermore, it suffices to (efficiently) "simulate the view" of each (semi-honest) party, because anything that can be obtained after participating in the protocol is obtainable from the view.

In this dissertation, we add to the privacy definition the following assumption:

> **Data Perturbation Assumption:** If an input is $x \in \mathcal{X}$, we assume that $x + r$ effectively preserves the privacy of x if r is a secret random number uniformly distributed in a domain $\mathcal{F}$, where $|\mathcal{F}| \gg |\mathcal{X}|$.

In both finite and infinite domains, the data perturbation assumption achieves the same level of privacy as the formal definition in [55]. However in an infinite domain, we do not know how to generate uniformly distributed random numbers; if instead we generate r in a finite domain, say $[-T, T]$, then $x + r$ does reveal some information about x. For example, if $x + r > T$, the information about the range of x is disclosed because one can derive $x > 0$. This is not a problem in a finite computation domain because the result will wrap to a number within the domain. By making the range of r big enough, the data perturbation could approximate the privacy definition from [55].

Even if we know how to generate r in an infinite domain, certain statistic information about x will still be disclosed if x consists of a set of data. For example, if $x_1, \ldots, x_n$ are n numbers in the inputs, $r_1, \ldots, r_n$ are uniformly distributed random numbers, and the addition is performed in an infinite domain, the sum of these x_i's will be disclosed, certain properties of the distribution of x_i's will also be disclosed. Although the disclosure of this type of aggregate information is not acceptable in certain applications, we believe in many situations the participants are more concerned about the disclosure of information of each single item rather than the disclosure of the aggregate information.

Therefore, our definition of privacy is slightly different from the definition in [55], but we believe the degree of privacy provided in our work is still acceptable in many applications.

In implementations, many other factors regarding the random number generation need to be considered, including randomness, precision and the choice of the domains. Because these factors are implementation issues, we do not discuss them in details in this dissertation.

SECURE TWO-PARTY PROTOCOL. A secure two-party protocol is a protocol for solving a secure two-party computation problem. It only involves two parties, and both parties are semi-honest and polynomial-time bounded.

SECURE TWO-PARTY PROTOCOL WITH AN UNTRUSTED THIRD PARTY. A secure two-party protocol with an untrusted third party is a protocol for solving a secure two-party computation problem with the help of an untrusted third party. All three parties are semi-honest and polynomial-time bounded. The inputs and the results of the protocol should not be disclosed to the untrusted third party.

2.3 Secure Multi-Party Computation

Secure Multi-party Computation (SMC) is the problem of evaluating a function to which each party has one secret input, such that the output becomes commonly known while the inputs remain secret. SMC is a very general and powerful notion, and it was preceded by numerous investigations of protocols for a variety of special-

purpose tasks, including secret sharing, bit commitment, coin flipping, mental poker, oblivious transfer, secret exchange, and secret-ballot election.

The first general secure two-party computation protocol result was described by Yao [105]. Assuming the intractability of factoring, Yao showed that every two-party interactive computational problem has a private protocol. Goldreich, Micali, and Wigderson [57] generalized the two-party problem to the multi-party problem, and showed how to weaken the intractability assumption from factoring the existence of any trapdoor permutation. These solutions are all based on the circuit evaluation protocol (in this protocol, the functionality that needs to be computed is represented as a Boolean circuit, and the two parties then jointly evaluate the circuit without disclosing their own inputs of the circuit to the other party). Later Kilian [67] shows how to base circuit evaluation solely on oblivious transfer as a primitive, thereby reducing the oblivious circuit evaluation to the oblivious transfer problem.

2.4 Cryptography Primitives

Circuit Evaluation

Circuit evaluation protocols are used in solving general secure two-party computation problems [105, 57, 55]. Two types of circuit evaluation protocols are well known in the literature. In the first type [57], one party, Alice, "scrambling" the circuit in a manner such that for each input of a gate two random numbers are used to represent 0 and 1, respectively. By getting only the random numbers, it is impossible to tell whether it represents 0 or 1. Each gate also has a table associate with it, such that given the random numbers from the input wire, it is possible to compute the result for

the output wire; however, because the output is also "scrambled" (appears as another random number), it is impossible to figure out whether the result is 0 or 1. After "scrambling" the circuit, Alice sends the circuit to Bob along with her own scrambled inputs (the random numbers corresponding the actual inputs). Bob, after getting the circuit, needs to get the random numbers corresponding to his own input to be able to "evaluate" the scrambled circuit. Bob uses 1-out-of-2 Oblivious Transfer Protocol to get the corresponding random numbers according to his input. The 1-out-of-2 Oblivious Transfer protocol guarantees that Alice does not know which number Bob gets, and thus prevents Alice from knowing Bob's inputs; the protocol also guarantees that Bob only gets one of the numbers, and thereby prevents Bob from deriving Alice's inputs.

The second type [55] is an interactive gate-by-gate evaluation of the circuit for encrypted inputs. The construction takes this Boolean circuit and produces a protocol for evaluating this circuit. The circuit evaluation protocol scans the circuit from the input wires to the output wires, processing a single gate in each *basic step*. When entering each basic step, the parties hold shares of the values of the input wires, and when the step is completed they hold shares of the output wire. Therefore, during the evaluation of any basic step, no intermediate result is disclosed; only at the last step, do two parties put their shares of the output together to compute the final results. The critical part of this protocol is how to compute shares of the output wire from shares of input wires without disclosing one party's share to the other party. This is achieved using 1-out-of-n Oblivious Transfer protocol.

From the way the circuit evaluation protocol works, it is not difficult to see that the communication cost of the protocol is linear to the size of the circuit, which depends on the complexity of expressing the functionality as a circuit. Therefore, for a complex functionality, using the circuit evaluation approach can be impractical. However, we stress that secure computation of small and simple circuits can be practical using the circuit evaluation protocol.

Oblivious Transfer

Kilian [67] showed that the secure two-party computation problem can be reduced to Oblivious Transfer in the semi-honest model, namely when parties are guaranteed to behave according to the protocols.

The notion of Oblivious Transfer (OT) was first introduced by Rabin [88]. In the original OT problem, the sender, Alice, has a secret, the receiver will receive this secret with probability 1/2, and Alice does not know whether or not Bob received it.

The Oblivious Transfer notion has several versions, which were shown to be equivalent to one another by Crépeau [33]. One version is called 1-out-of-2 Oblivious Transfer. It was introduced by Even, Goldreich, and Lempel [42]. 1-out-of-2 Oblivious Transfer is a protocol between two parties, a sender Alice and a receiver Bob. Alice has two bits (b_0, b_1), and Bob wants to get one of the bits b_i of his choice. At the end of the protocol, Alice should not learn anything about i, and Bob should not learn anything about the other bit.

1-out-of-2 Oblivious Transfer was further generalized to 1-out-of-n Oblivious Transfer by Brassard, Crépeau, and Robert [20]. 1-out-of-n Oblivious Transfer protocol

refers to a protocol where at the beginning of the protocol one party, Bob has n inputs $X_1, \ldots, X_n$ and at the end of the protocol the other party, Alice, learns one of the inputs X_i for some $1 \leq i \leq n$ of her choice, without learning anything about the other inputs and without allowing Bob to learn anything about i.

An efficient 1-out-of-n Oblivious Transfer protocol was proposed in [82] by Naor and Pinkas. This protocol invokes 1-out-of-2 Oblivious Transfer protocol $\log n$ times; combining with the PIR protocol [22], this protocol achieves polylogarithmic (in n) communication complexity. This protocol serves as an important building block for our protocols. The idea of using the 1-out-of-n Oblivious Transfer protocol as a building block was pioneered by various researchers such as Goldreich [55], and Naor and Pinkas [82].

Homomorphic Encryption Schemes

We need a public-key cryptosystem with a homomorphic property for some of our protocols. An encryption scheme is *homomorphic* if $E_k(x) * E_k(y) = E_k(x + y)$. Such encryption systems are called *homomorphic* cryptosystems, and examples include the systems by Benaloh [15], Naccache and Stern [80], Okamoto and Uchiyama [84] and Paillier [85]. A good property of homomorphic encryption schemes is that "addition" can be conducted based on the encrypted data without decrypting them.

2.5 Specific Secure Two-Party and Multi-Party Computation Problems

Goldreich pointed out [55] "We view the general solutions as asserting that very wide classes of problems are solvable in principle. However, we do not recommend using the solutions derived by these general results in practice. For example, although

Threshold Cryptography (cf., [50, 36]) is merely a special case of multi-party computation, it is indeed beneficial to focus on its specific." In this section, we review some specific two-party and multi-party computation problems that are related to our work.

Private Information Retrieval

Among various secure multi-party computation problems, the Private Information Retrieval (PIR) problem is one that has been widely studied. The PIR problem consists of devising a protocol involving a user and a database server, each having a secret input. The database's secret input is called the *data string*, an n-bit string $B = b_1 b_2 \ldots b_n$. The user's secret input is an integer i between 1 and n. The protocol should enable the user to learn b_i in a communication-efficient way and at the same time hide i from the database. The trivial solution has an $O(n)$ communication complexity. Much work has been done for reducing this communication complexity [26, 24, 25, 62, 37, 73, 22, 52, 51].

In [26] it was shown that if there is only one server holding the database, then $\Omega(n)$ bits of communication are needed to achieve *information-theoretic* user privacy. However, if there are $k \geq 2$ non-communicating servers, each holding a copy of the database, then there are solutions with sub-linear communication complexity. Unlike the information-theoretic model, computational PIR schemes with sub-linear communication complexity exist using a single server holding the dataset, and relying on some intractability assumptions, as was first proved by Kushilevitz and Ostrovsky [73].

Private Comparison

Two types of comparisons are useful in this dissertation. The first one is to compare whether two numbers (or two datums) are exactly the same; the second one is to compare two numbers, and decide which one is bigger.

The first comparison problem was thoroughly discussed by Fagin, Naor and Winkler in [43] as well as by Naor and Pinkas in [82].

The second comparison problem is called Yao's Millionaire Problem from Yao [104]. Two millionaires wish to know who is richer, without revealing any other information about each other's net worth. The early cryptographic solution by Yao [104] has communication complexity that is exponential in the number of bits of the numbers involved, and the later solutions using general secure multi-party computation techniques also have problems with achieving privacy efficiently. Their advantage, of course, is that they do so without using an untrusted third party. Cachin proposed a much more efficient solution in [21] based on the Φ-hiding assumption; the protocol uses an untrusted third party. The communication complexity of Cachin's scheme is $O(\ell)$, where ℓ is the number of bits of each input number.

Oblivious Evaluation of Polynomials

Naor and Pinkas [82] have studied an oblivious evaluation of polynomials problem, where a polynomial P is known to Bob and he would like to let Alice compute the value $P(\alpha)$ for an input α known to her in such a way that Bob does not learn α and Alice does not gain any additional information about P (except $P(\alpha)$).

The solution is based on the Oblivious Transfer protocol. In the solution, B hides P in a bivariate polynomial $Q(x, y)$, such that $Q(0, \cdot) = P(\cdot)$; A hides α in an univariate polynomial $S(x)$, such that $S(0) = \alpha$. A's plan is to use the univariate polynomial $R(x) = Q(x, S(x))$ to learn $P(\alpha)$: it holds that $R(0) = P(\alpha)$. If the degree of polynomial R is d, once Alice learns $d + 1$ values of the form $(x_i, R(x_i))$, she can compute $R(0)$. Alice can learn these $d + 1$ values using 1-out-of-n Oblivious Transfer Protocol to avoid disclosing information about α to Bob.

Privacy-Preserving Data Mining

The privacy-preserving data mining problem is another specific secure multi-party computation problem that has been discussed in the literature. Two different privacy-preserving data mining problems have been proposed. In Lindell and Pinkas' paper [75], the problem is defined as this: Two parties, each having a private database, want to jointly conduct a data mining operation on the union of their two databases. How could these two parties accomplish this without disclosing their database to the other party, or any third party? In Agrawal's paper [5], the privacy-preserving data mining problem is defined as this: Alice is allowed to conduct data mining operations on a private database owned by Bob. How could Bob prevent Alice from accessing precise information in individual data records, while Alice is still able to conduct the data mining operations? The solution to these two similar problems are quite different: Lindell and Pinkas use secure multi-party computation protocols to solve their problem, while Agrawal uses the data perturbation method.

Cryptographic Tasks

Secure multi-party computation was preceded by numerous investigations of protocols for a variety of special-purpose cryptographic tasks. Secret sharing [95, 98] is a multi-party protocol in which a designated party distributes a message for later recovery by some authorized subcollection of the remaining parties. Bit commitment [19, 81] is a two-party version of secret sharing. Mental poker [106, 45, 29, 31, 32] is a multi-party protocol for producing, and partially applying, a random permutation (i.e., shuffle and deal a deck of cards). Secret ballot election schemes [28, 17] are essentially a special case of secure multi-party computation in which the function is a simple sum of ones and zeros.

3. BUILDING BLOCKS

3.1 Secure Two-Party Permutation Protocols

3.1.1 Secure Two-Party Permutation Problem

Problem 3.1.1. *(Secure Two-Party Permutation Problem)* Alice has a private vector $X = (x_1, \ldots, x_n)$; Bob has a private permutation function π and private vector $R = (r_1, \ldots, r_n)$. Alice needs to get $\pi(X + R)$. Alice should not learn π or the value of any r_i; Bob should not learn the value of any x_i.

To Alice, this is an oblivious permutation, namely Alice has her vector permuted but she does not know how it is permuted. As we will see later, this type of oblivious permutation is useful for solving many secure two-party computation problems.

We have developed two protocols to solve this problem. The first one is based on 1-out-of-n Oblivious Transfer Protocol, and the second one is based on a homomorphic encryption scheme.

3.1.2 Secure Two-Party Permutation Protocol 1

Protocol 3.1.1. *(Secure Two-Party Permutation Protocol 1)*

Inputs: Alice has a secret vector X. Bob has a secret permutation π and a secret vector R.

Outputs: Alice gets $\pi(X + R)$.

1. Alice and Bob agree on two numbers p and m, such that p^m is so big that conducting p^m additions is computationally infeasible.

2. Alice generates m random vectors $V_1, \ldots, V_m$, such that $X = \sum_{j=1}^{m} V_j$.

3. Bob generates m random vectors $R_1, \ldots, R_m$, such that $R = \sum_{j=1}^{m} R_j$.

4. For each $j = 1, \ldots, m$, Alice and Bob conduct the following sub-steps:

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

   (b) Alice sends $(H_1, \ldots, H_p)$ to Bob, where $H_k = V_j$, and the rest of H_i's are random vectors. Because k is a secret number known only to Alice, Bob does not know which one of H_i's is V_j.

   (c) Bob computes $Z_{j,i} = \pi(H_i + R_j)$ for $i = 1, \ldots, p$.

   (d) Using 1-out-of-p Oblivious Transfer Protocol, Alice gets $Z_j = Z_{j,k} = \pi(V_j + R_j)$ from Bob.

5. Alice computes $\sum_{j=1}^{m} Z_j = \pi(\sum_{j=1}^{m}(V_j + R_j)) = \pi(X + R)$.

3.1.3 Secure Two-Party Permutation Protocol 2

This protocol, developed by Kerschbaum [38], is based on a homomorphic public key system. In what follows, we define $E(Z) = (E(z_1), \ldots, E(z_n))$ and $D(Z) = (D(z_1), \ldots, D(z_n))$, for a vector $Z = (z_1, \ldots, z_n)$.

Protocol 3.1.2. *(Secure Two-Party Permutation Protocol 2)*

Inputs: Alice has a secret vector X. Bob has a secret permutation π and a secret vector R.

Outputs: Alice gets $\pi(X + R)$.

1. Alice generates a key pair for a homomorphic public key system and sends the public key to Bob. The corresponding encryption and decryption is denoted as $E(\cdot)$ and $D(\cdot)$.

2. Alice encrypts X and sends the result $E(X)$ to Bob.

3. Bob computes $E(X) * E(R) = E(X + R)$, permutes $E(X + R)$ using π, and then sends the result $\pi(E(X + R))$ to Alice.

4. Alice decrypts $\pi(E(X + R))$ and gets $D(\pi(E(X + R))) = \pi(D(E(X + R)))$ $= \pi(X + R)$.

3.2 Secure Two-Party Scalar Product Protocols

Problem 3.2.1. *(Secure Two-Party Scalar Product Problem)* Alice has a vector $X = (x_1, \ldots, x_n)$ and Bob has a vector $Y = (y_1, \ldots, y_n)$. Alice needs to get the result of $u = X \cdot Y + v = \sum_{i=1}^{n} x_i y_i + v$, where v is a random number that Bob knows. Alice should not be able to derive the result of $X \cdot Y$ or any value of y_i from u and the execution of the protocol; similarily Bob should not be able to derive the result of $X \cdot Y$ or any value of x_i from v and the execution of the protocol.

Instead of computing the usual scalar product, our definition of the problem is slightly different and more general: We assume the goal of the protocol is for Alice

(but not Bob) to get $X \cdot Y + v$ where v is random and known to Bob only (of course without either side revealing to the other the private data they start with). Our protocols can easily be modified to work for the version of the problem where the random v is given ahead of time as part of Bob's data (the special case $v = 0$ puts us back in the usual scalar product definition). The purpose of Bob's random v is as follows: If $X \cdot Y$ is a partial result that Alice is not supposed to know, then giving her $X \cdot Y + v$ prevents Alice from knowing the partial result (even though the scalar product has in fact been performed); later, at the end of the multiple-step protocol, the effect of v can be effectively "subtracted out" by Bob without revealing v to Alice (this should become clearer with example protocols that we later give).

We have developed three protocols to solve this scalar product problem. The first protocol uses an *untrusted* third party, Ursula, with the assumption that Ursula should not collude with either Alice or Bob. Because Ursula is not trusted, so she should not learn the value of either vector X or vector Y from the execution of the protocol. The second protocol and the third protocol do not use a third party. They are based on the 1-out-of-n Oblivious Transfer protocol and the hormomorphic encryption scheme, respectively. The comparison of these three protocols will be discussed following the descriptions of the protocols.

3.2.1 Secure Two-Party Scalar Product Protocol 1

Protocol 3.2.1. *(Secure Two-Party Scalar Product Protocol Using a Untrusted Party)*

Inputs: Alice has a secret vector X, and Bob has a secret vector Y.

Outputs: Alice gets u, and Bob gets v, where $u = X \cdot Y + v$.

1. Alice and Bob agree upon two random vectors R_x and R_y.
2. Bob generates a secret random number v.
3. Alice sends $X + R_x$ and $X \cdot R_y + 2R_x \cdot R_y$ to Ursula.
4. Bob sends $Y + R_y$ and $Y \cdot R_x - R_x \cdot R_y - v$ to Ursula.
5. Ursula computes $u = (X+R_x)\cdot(Y+R_y) - (X\cdot R_y+2R_x\cdot R_y) - (Y\cdot R_x - R_x\cdot R_y - v)$, and gets $u = X \cdot Y + v$.
6. Ursula sends u to Alice.

3.2.2 Secure Two-Party Scalar Product Protocol 2

Consider the following naive solution: Alice sends p vectors to Bob, only one of which is X (the others are arbitrary). Then Bob computes the scalar products between Y and each of these p vectors. At the end Alice uses the 1-out-of-p oblivious transfer protocol to get back from Bob the product of X and Y. Because of the way oblivious transfer protocol works, Alice can decide which scalar product to get, but Bob could not learn which one Alice has chosen. There are many drawbacks to this approach: If the value of X has certain public-known properties, Bob might be able to differentiate X from the other $p-1$ vectors, but even if Bob is unable to recognize X his chances of guessing it is an unacceptably low 1 out of p.

The above drawbacks can be fixed by dividing vector X into m random vectors $V_1, \ldots, V_m$ of which it is the sum, i.e., $X = \sum_{i=1}^{m} V_i$. Alice and Bob can use the above

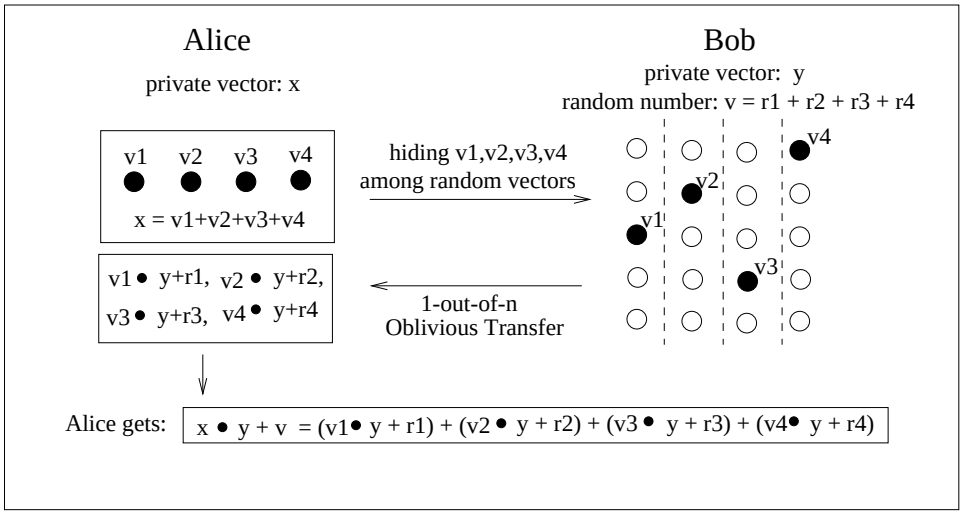

Figure 3.1. Scalar Product Protocol 2

naive method to compute $V_i \cdot Y + r_i$, where r_i is a random number and $\sum_{i=1}^{n} r_i = v$ (see Figure 3.1). As a result of the protocol, Alice gets $V_i \cdot Y + r_i$ for $i = 1, \ldots, m$. Because of the randomness of V_i and its position, Bob could not find out which one is V_i. Certainly, there is 1 out p possibility that Bob can guess the correct V_i, but because X is the sum of m such random vectors, the chance that Bob guesses the correct X is 1 out p^m, which could be very small if we chose p^m large enough.

After Alice gets $V_i \cdot Y + r_i$ for $i = 1, \ldots, n$, she can compute $\sum_{i=1}^{m}(V_i \cdot Y + r_i)$ $= X \cdot Y + v$. The detailed protocol is described in the following:

Protocol 3.2.2. *(Secure Two-Party Scalar Product Protocol 2)*

Inputs: Alice has a vector $X = (x_1, \ldots, x_n)$, and Bob has a vector $Y = (y_1, \ldots, y_n)$.

Outputs: Alice gets u, and Bob gets v, where $u = X \cdot Y + v$.

1. Alice and Bob agree on two numbers p and m, such that p^m is so big that conducting p^m additions is computationally infeasible.

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

3. Bob generates m random numbers $r_1, \ldots, r_m$, and lets $v = \sum_{j=1}^{m} r_j$.

4. For each $j = 1, \ldots, m$, Alice and Bob conduct the following sub-steps:

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

   (b) Alice sends $(H_1, \ldots, H_p)$ to Bob, where $H_k = V_j$, and the rest of H_i's are random vectors. Because k is a secret number known only to Alice, Bob does not know the position of V_j.

   (c) Bob computes $Z_{j,i} = H_i \cdot Y + r_j$ for $i = 1, \ldots, p$.

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

5. Alice computes $u = \sum_{j=1}^{m} Z_j = X \cdot Y + v$.

How is privacy achieved:

- If Bob chooses to guess, his chance of guessing the correct X is 1 out of p^m.

- The purpose of r_j is to add randomness to $V_j \cdot Y$, and thus prevents Alice from deriving information about Y.

3.2.3 Secure Two-Party Scalar Product Protocol 3

Our next solution does not rely on 1-out-of-n Oblivious Transfer cryptography primitive as the previous does, but is instead based on a homomorphic public key

system. In the following discussion, we define $\pi(X)$ as another vector whose elements are random permutation of those of vector X.

We begin with two observations. First, a property of the scalar product $X \cdot Y$ is that $\pi(X) \cdot \pi(Y) = X \cdot Y$, regardless of what π is. Secondly, if Bob sends a vector $\pi(V)$ to Alice, where π and V are known only to Bob, Alice's chance of guessing the position of any single element of the vector V is 1 out of n (n is the size of the vector); Alice's chance of guessing the positions of all of the elements of the vector V is 1 out of $n!$.

A naive solution would be to let Alice get both $\pi(X)$ and $\pi(Y)$ but not π. Let us ignore for the time being the drawback that Alice gets the items of Y in permuted order, and let us worry about not revealing π to Alice: Letting Alice know $\pi(X)$ allows her to easily figure out the permutation function π from knowing both X and $\pi(X)$. To avoid this problem, we want to let Alice know only $\pi(X + R_b)$ instead of $\pi(X)$, where R_b is a random vector known only to Bob. Because of the randomness of $X + R_b$, to guess the correct π, Alice's chance is only 1 out of $n!$. Therefore to get the final scalar product, Bob only needs to send $\pi(Y)$ and the result of $R_b \cdot Y$ to Alice, who can compute the result of the scalar product by using

$$X \cdot Y = \pi(X + R_b) \cdot \pi(Y) - R_b \cdot Y.$$

Now we turn our attention to the drawback that giving Alice $\pi(Y)$ reveals too much about Y (for example, if Alice is only interested in a single element of the vector

Y, her chance of guessing the right one is an unacceptably low 1 out of n). One way to fix this is to divide Y to m random pieces, $V_1, \ldots, V_m$, with $Y = V_1 + \ldots + V_m$; then Bob generates π random permutations $\pi_1, \ldots, \pi_m$ (one for each "piece" V_i of Y) and lets Alice know $\pi_i(V_i)$ and $\pi_i(X + R_b)$ for $i = 1, \ldots, m$. To guess the correct value of a single element of Y, Alice has to guess the correct position of V_i in each one of the m rounds; the possibility of successful guessing becomes 1 out of n^m.

Let us consider the unanswered question: how could Alice get $\pi(X + R_b)$ without learning π or R_b? This can be achieved using the Secure Two-Party Permutation Protocol (Protocol 3.1.2), a protocol based on a homomorphic public key system.

Protocol 3.2.3. *(Secure Two-Party Scalar Product Protocol 3)*

Inputs: Alice has a secret vector X, Bob has a secret vector Y.

Outputs: Alice gets u, and Bob gets v, where $u = X \cdot Y + v$.

1. Bob's set up:

   (a) Bob divides Y to m random pieces, s.t. $Y = V_1 + \ldots + V_m$.

   (b) Bob generates m random vectors $R_1, \ldots, R_m$, let $v = \sum_{i=1}^{m} V_i \cdot R_i$.

   (c) Bob generates m random permutations $\pi_1, \ldots, \pi_m$.

2. For each $i = 1, ..., m$, Alice and Bob do the following:

   (a) Using Secure Two-Party Permutation Protocol (Protocol 3.1.2), Alice gets $\pi_i(X + R_i)$ without learning either π_i or R_i.

   (b) Bob sends $\pi_i(V_i)$ to Alice.

(c) Alice computes $Z_i = \pi_i(V_i) \cdot \pi_i(X + R_i) = V_i \cdot X + V_i \cdot R_i$.

3. Alice computes $u = \sum_{i=1}^{m} Z_i = \sum_{i=1}^{m} V_i \cdot X + \sum_{i=1}^{m} V_i \cdot R_i = X \cdot Y + v$.

How is privacy achieved:

- The purpose of R_i is to prevent Alice from learning π_i.
- The purpose of π_i is to prevent Alice from learning V_i. Although Alice learns a random permutation of the V_i, she does not learn more because of the randomness of V_i.
- If Alice chooses to guess, to successfully guess all of the elements in Y, her chance is $(\frac{1}{n!})^m$.
- Alice's chance of successfully guessing just one elements of Y is 1 out of n^m. For example, to guess the kth element of Y, Alice has to guess the corresponding elements in $\pi_i(V_i)$ for all $i = 1, \ldots, m$. Because for each single i, the possibility is 1 out of n, the total possibility is 1 out of n^m.
- A drawback of this protocol is that the information about $\sum_{i=1}^{n} y_i$ is disclosed because the random permutation does not help to hide this information.

3.2.4 Complexity Analysis

In the following discussion, we assume that d is the number of bits to represent any number in the inputs.

The communication cost (in terms of number of bits sent over the network) of Protocol 3.2.3 is $4m * n * d$, where m is a security parameter (so that $\mu' = n^m$ is large

enough). The communication cost of Protocol 3.2.2 is $p * t * n * d$, where $p \geq 2$ and t are security parameters (so that $\mu'' = p^t$ is large enough). Setting $\mu' = \mu'' = \mu$ for the sake of comparison, the communication cost of Protocol 3.2.3 is $4 \log \mu \frac{nd}{\log n}$ and the communication cost of Protocol 3.2.2 is $\frac{p \log \mu}{\log p} nd$. When n is large, Protocol 3.2.3 is more efficient than Protocol 3.2.2.

The communication cost of Protocol 3.2.1 is $2n * d$, which is much better than the other two protocols. However, this efficiency is achieved by introducing an untrusted and uncolluding third party.

The communication cost of the circuit evaluation protocol is $c * n * d^2$, where c is the number of bits sent over the network in the 1-out-of-n Oblivious Transfer protocol. Although the value of c depends on the specific implementation of the protocol, it is reasonable to assume $c = d$; therefore the communication cost becomes $n * d^3$, which is significantly more expensive than our scalar product protocols.

3.2.5 Applications

The Secure Two-Party Scalar Product Protocol is an important building block. As we will see later, it is used in several of our other protocols, such as secure two-party protocols for computational geometry and statistical analysis.

The similar techniques used for building the Scalar Product Protocol could also be used for building some other protocols, such as the protocol for computing the convolution of two private vectors and so on.

3.3 Secure Two-Party Vector Dominance Protocol

About two decades ago, Yao introduced the "millionaire problem" [104]: Two parties want to determine who is richer without disclosing anything else about their wealth. Several solutions have been proposed in the past to solve this problem, however, none of them are efficient, except the one proposed by Cachin, who has given an elegant practical solution to this problem [21].

Yao's millionaire problem deals with only one comparison, i.e. a comparison between two numbers. However, in many applications, one often encounters situations where one wants to make multiple comparisons without disclosing the result of individual comparisons, or even the statistical information about them. Consider the following situation: Alice has n private numbers $(a_1, \ldots, a_n)$, and Bob has another n private numbers $(b_1, \ldots, b_n)$; Alice and/or Bob want to know whether $a_i > b_i$ is true for all $i = 1, \ldots, n$. However, except for what can be derived from the answer, nobody is allowed to know the other person's private numbers or the comparison result between any a_i and b_i, including the statistical information such as how many a_i's are bigger (or smaller) than their corresponding b_i's, etc.

We consider the above multiple comparison problem as an extension of Yao's Millionaire Problem (a one-dimensional problem) to a multi-dimensional problem. If we consider A and B as vectors, the problem is to actually decide whether A dominates B. Therefore, in this section, we call this problem the *Secure Two-Party Vector Dominance Problem*, or *Dominance Problem* in short.

As we will show later, there are many applications to the dominance problem. For example, in business-to-business bidding, a manufacturer may want to deal with a single supplier that can simultaneously satisfy all of its n requirements (either because there is some coordination required in the production of the n items types, or simply to avoid the bureaucratic overhead of having to deal with multiple suppliers). However, if the supplier cannot satisfy all of its requirements, the manufacturer does not want the supplier to know any information, such as which requirements are satisfied.

Definition 3.3.1. *(Vector Dominance)* Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$; if for all $i = 1, \ldots, n$ we have $a_i > b_i$, then we say that A dominates B and denote it by $A \succ B$.

Definition 3.3.2. *(Secure Two-Party Vector Dominance Problem)* Alice has a vector $A = (a_1, \ldots, a_n)$ and Bob has a vector $B = (b_1, \ldots, b_n)$; Alice and Bob want to find out whether A dominates B or B dominates A or no vector dominates the other. The following privacy requirements should be satisfied:

- Alice should not learn the value of any element of Bob's vector B.
- Bob should not learn the value of any element of Alice's vector A.
- In case that no vector dominates the other, neither Alice nor Bob should learn the relationship between any element of A with any element of B.
- In case that no vector dominates the other, neither Alice nor Bob should learn how many of A's elements are bigger than the corresponding elements in B.

The requirement of not allowing each party to know any partial information about the individual comparisons immediately rules out using the solution to Yao's Millionaire Problem n times, once for each dimension: That would inappropriately reveal the relative ordering of individual a_i, b_i pairs in the case where the answer is "no".

We have developed two protocols to solve the above private vector dominance problem. Both solutions uses Cachin's solution to Yao's Millionaire Problem as a subroutine, but it does so in a non-obvious way, and the overall structure of these solutions are novel and different from Cachin's.

As Cachin [21] points out, the early cryptographic solutions such as Yao's millionaire problem [104] have communication complexity that is exponential in the number of bits of the numbers involved, and the later solutions using general secure multiparty computation techniques also have problems with achieving privacy efficiently. Their advantage, of course, is that they do so without using an untrusted third party. Cachin's work [21], assumes an untrusted third party that can misbehave on its own (for the purpose of illegally obtaining information about Alice's or Bob's private vectors) but does not collude with Alice against Bob or vice-versa. Because we are using Cachin's scheme as a subroutine, we are also implicitly making use of the number-theoretic assumptions made in Cachin's paper [21] (such as the Φ-hiding assumption that was also used in the private information retrieval literature [22]).

The difference of our two protocols is whether an untrusted third party is used in the part of the protocol apart from Cachin's Yao's Millionaire protocol subroutine. The first protocol uses an untrusted third party to achieve privacy, while the

second solution protocol does not. Both of our solutions use Cachin's Yao's Millionaire protocol–a protocol using an untrusted third party–as subroutine, thus making them both as a third-party protocol. However, if a two-party Yao's Millionaire protocol is ever invented, the first protocol will still be a third-party protocol, while the second one will be a two-party protocol. Therefore, we name the first protocol *three-party private vector dominance protocol* and the second one *two-party private vector dominance protocol.*

3.3.1 Secure Two-Party Vector Dominance Protocol 1

Definition 3.3.3. *(Secure Two-Party Vector Dominance Protocol (using a third party))* Secure Two-Party Vector Dominance Protocol (using a third party) is one in which the two parties determine whether one party's secret vector dominates another party's secret vector, with the help of an oblivious third party who, while it does not collude with either one of the two parties against the other, could nevertheless try to illegally acquire information about their secret data. We call this third party Ursula. At the end of the protocol, the following properties must hold:

1. Alice and Bob have determined whether $A \succ B$, $B \succ A$, or neither A nor B dominates the other. However, Ursula has not made such a determination.

2. No party (including the third party) has gained any knowledge about A and B other than the one implied by the previous item. Note that this implies the following:

(a) In the case where neither $A \succ B$ nor $B \succ A$, neither Alice nor Bob knows the relative ordering of any individual a_i, b_i pair (i.e., whether $a_i < b_i$ or not).

(b) Ursula knows nothing about A, B, or the relationships between any elements of A, B.

Overview of the Protocol

A short and oversimplified description of the vector dominance protocol is described here. Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ be Alice's and Bob's inputs, respectively. Because having Alice and Bob compare their vectors directly would inappropriately disclose the ordering information between individual a_i, b_i pairs, instead we ask the oblivious third party, Ursula, to compare numbers derived from Alice's and Bob's numbers in such a way that the comparison reveals no information to Ursula and yet Ursula can perform certain updates that indirectly encapsulate (for A and B) the outcome of the vector-dominance comparison. The idea is not to let Ursula know anything about A, B, or the outcome of the comparison between any a_i and b_i with which it is helping. This implies that Ursula does not find out such statistical aggregate results as how many of the a_i's are larger than the corresponding b_i's, etc.

This objective is achieved by sending Alice's disguised data to Ursula, such that Ursula does not know the actual value of any particular a_i, nor does she know which disguised entry corresponds to a_i. As we know, using protocols for Yao's Millionaire Problem [104, 21] enables two parties to compare their private inputs without disclosing the private inputs to the other party; therefore, after getting Alice's input vector,

Ursula and Bob use Cachin's protocol [21] to compare the disguised elements of Alice's and Bob's input vectors one by one; we actually use a slightly modified version of Cachin's protocol, one where Ursula knows the comparison's outcome while the other party learns nothing. (*Note:* The protocol can easily be modified so it is symmetric in the roles of Alice and Bob, as will be explained later — the symmetric version of it is just as efficient but is notationally more cumbersome and so we postpone discussing it for the sake of a simpler exposition.) Because Ursula does not know which disguised entry corresponds to a particular a_i, she knows nothing about the relationship between a_i and b_i, nor does she know the *dominance* relationship between A and B (whether the entries of one are all larger than the corresponding entries of the other). Moreover, Alice's inputs are disguised in such a way that, to Ursula, half of Alice's disguised inputs are bigger than Bob's corresponding inputs, while another half of Alice's disguised inputs are less than Bob's corresponding inputs, regardless of what the actual relationship between Alice's inputs and Bob's inputs is. Therefore Ursula gains no information about the statistical aggregate results as how many a_i's are larger than the corresponding b_i's.

So Ursula will know the results of the comparisons between disguised items of Alice's and Bob's, but she cannot give these results to Alice and Bob because they can make sense of them to inappropriately glean information: Instead, she has to let Alice and Bob know whether A dominates B, or B dominates A, or neither dominates the other, without Ursula herself knowing anything about the dominance relationship (or lack thereof). This is achieved by having Alice and Bob each generate a set of

pairs of nonces (= random numbers), after which Alice and Bob each share a different set of their own secret nonces with Ursula. Ursula then converts the comparisons' outcomes to a single number formed by the *xor* of a selected set of nonces from the two sets of nonces. It is this number that is finally sent to Alice and Bob by Ursula. The idea is to judiciously construct this number in such a way that Alice and Bob can only verify the dominance relationship between A and B, and nothing else (while Ursula learns nothing).

The above was necessarily an oversimplification of the main ideas of our protocol, and only a look at the details will reveal the subtle intricacies involved.

The Protocol

To make it easier to understand the protocol, we will present a rough outline of it first, omitting many crucial details which are presented following the outline.

Protocol 3.3.1. *(Secure Two-Party Vector Dominance Protocol (using a third party)– Outline)*

The protocol consists of the following stages:

- Alice disguises her vector A and gets A', sends A' to Ursula. (Because of the disguise, Ursula learns nothing about Alice's secret A). Bob disguises B and gets B' but he does *not* send B' to Ursula. A' and B' each have a length m that is larger than n.

  Note: The protocol can easily be modified so it is symmetric in the roles of Alice and Bob, as will be explained later. The reason we present the details of

this version (rather than the symmetric one) is that it results in a considerably less cluttered exposition both conceptually and (especially) notationally.

- Ursula and Bob use a modified version of Cachin's protocol to compare each entry of Ursula's A' to the corresponding entry of Bob's B'; in this modified version of Cachin's protocol, only Ursula knows the outcome of a comparison as opposed to both knowing it in the original version. Because of this protocol, Ursula learns nothing about B', and Bob learns nothing about A' or the comparison results.

- Ursula sends to both Alice and Bob a number h that she computed based on the A'-to-B' comparison outcomes of the previous stage. That number h encapsulates the dominance information between vectors A and B in such a way that it makes no sense to Ursula and yet that can be extracted from it by Alice and Bob.

- Alice and Bob extract the vector-dominance comparison outcome from the number h that they received from Ursula.

Step 1: Setup

Alice and Bob jointly generate $4n$ random numbers $R_1, \ldots, R_{4n}$; they both know all of these $4n$ random numbers.

Step 2: Inputs Disguise

In this step, Alice constructs $A' = (2a_1 + R_1, \ldots, 2a_n + R_n, (2a_1 + 1) + R_{n+1}, \ldots, (2a_n+1)+R_{2n}, -2a_1+R_{2n+1}, \ldots, -2a_n+R_{3n}, -(2a_1+1)+R_{3n+1}, \ldots, -(2a_n+1)+R_{4n})$.

Bob constructs $B' = ((2b_1 + 1) + R_1, \ldots, (2b_n + 1) + R_n,\ 2b_1 + R_{n+1}, \ldots, 2b_n + R_{2n},$ $-(2b_1 + 1) + R_{2n+1}, \ldots, -(2b_n + 1) + R_{3n},\ -2b_1 + R_{3n+1}, \ldots,\ -2b_n + R_{4n})$. Alice and Bob then agree upon a random permutation π of $\{1, 2, \ldots, 4n\}$, and they use π to reorder the entries of A', and the entries of B'. We will next explain the rationale for constructing A' and B' in this way.

Because $a_i > b_i$ if and only if $a_i + R_i > b_i + R_i$, one might as well compare $a_i + R_i$ with $b_i + R_i$ instead of comparing a_i with b_i. As Alice is going to send her disguised inputs to Ursula, she has to encrypt or disguise her data; adding R_i to a_i effectively hides it from Ursula. If that is the only form of disguise that was done, then Alice's A' would be $(a_1 + R_1, \ldots, a_n + R_n)$ and Bob's B' would be $(b_1 + R_1, \ldots, b_n + R_n)$. Although Alice and Bob could rearrange the order of the entries of such an A' and (respectively) B', Alice still cannot send this A' to Ursula and ask Ursula to run a protocol that compares its entries to those of Bob's B', because Ursula would then know for how many indices i the comparison $a_i > b_i$ was true. This is not acceptable. Furthermore, if the mean value of these random numbers is known to Ursula, Ursula can gain statistical information about a_i's. The above drawbacks are addressed by the inclusion in each of A' and B' of the n entries of the form $-a_i + R_{n+i}$ and (respectively) $-b_i + R_{n+i}$. Based on the fact that $a_i > b_i$ and $-a_i > -b_i$ cannot be true or false at the same time (if $a_i \neq b_i$), the additional n entries (involving the $-a_i$ or $-b_i$) are meant to "blind" Ursula from learning how many times $a_i > b_i$, as well as the statistical information about a_i's (the sum of a_i's and $-a_i$'s for $i = 1, ..., n$ is zero). Now, if that is the only form of disguise that was done, Alice's A' would

be $(a_1 + R_1, \ldots, a_n + R_n, -a_1 + R_{n+1}, \ldots, -a_n + R_{2n})$, and Bob's B' would now be $(b_1 + R_1, \ldots, b_n + R_n, -b_1 + R_{n+1}, \ldots, -b_n + R_{2n})$. But Alice still cannot send such an A' for Ursula to use in a protocol comparing its entries to those of Bob's B', and that is because of the *equality* case, i.e., for some i having $a_i = b_i$ (and hence $a'_i = b'_i$). For example, if $a_i = b_i$ for all $i = 1, \ldots, n$, then both $a'_i > b'_i$ and $-a'_i > -b'_i$ are false; therefore if Ursula got $2n$ *false*'s, she would know that Alice and Bob have the exact same vector. Similarly, if $a_i \neq b_i$ for all $i = 1, \ldots, n$, Ursula gets n *true* values and n *false* values; if $a_i = b_i$ for only one i, Ursula gets $n - 1$ *true* values and $n + 1$ *false* values, and so on. The above shows how Ursula would derive the number of equality cases, which is not acceptable.

Notice if $a_i \neq b_i$ for $i = 1, \ldots, n$, then the comparison results will always be n *true* values and n *false* values. Therefore, if we can get rid of the equality case, Ursula will always get n *true* values and n *false* values, which does not disclose any statistical information about the equality cases.

To this end, the following transformation is conducted (for convenience, we assume a_i and b_i for $i = 1, \ldots, n$ are integers; however our scheme can be easily extended to the non-integer case).

We transform a_i to $2a_i$ and $2a_i + 1$; correspondingly, we transform b_i to $2b_i + 1$ and $2b_i$. The transformation has the following good properties: if $a_i = b_i$, then $2a_i < 2b_i + 1$ and $2a_i + 1 > 2b_i$; therefore, there is no equality case. However, this transformation does not affect either "$>$" case or "$<$" case: for example, if $a_i > b_i$, then both $2a_i > 2b_i + 1$ and $2a_i + 1 > 2b_i$ still hold because a_i and b_i are integers.

Observation 3.3.1. *The dominance relationship between* $A = (a_1, \ldots, a_n)$ *and* $B = (b_1, \ldots, b_n)$ *is the same as the dominance relationship between* $A'' = (2a_1, \ldots, 2a_n, 2a_1 + 1, \ldots, 2a_n + 1)$ *and* $B'' = (2b_1 + 1, \ldots, 2b_n + 1, 2b_1, \ldots, 2b_n)$.

By combining the above transformation with the addition of random numbers, Alice's input and Bob's input respectively become the following:

$$\begin{aligned} A' &= (2a_1 + R_1, \ldots, 2a_n + R_n, (2a_1 + 1) + R_{n+1}, \ldots, (2a_n + 1) + R_{2n}, \\ &\quad - 2a_1 + R_{2n+1}, \ldots, -2a_n + R_{3n}, -(2a_1 + 1) + R_{3n+1}, \ldots, -(2a_n + 1) + R_{4n}) \\ B' &= ((2b_1 + 1) + R_1, \ldots, (2b_n + 1) + R_n, 2b_1 + R_{n+1}, \ldots, 2b_n + R_{2n}, \\ &\quad - (2b_1 + 1) + R_{2n+1}, \ldots, -(2b_n + 1) + R_{3n}, -2b_1 + R_{3n+1}, \ldots, -2b_n + R_{4n}) \end{aligned}$$

Theorem 3.3.1. *The comparison of* A' *and* B' *always generates* $2n$ *true values and* $2n$ *false values.*

Proof. Consider the possible cases:

1. If $a_i > b_i$, then $2a_i + R_i > (2b_i + 1) + R_i$, $(2a_i + 1) + R_{n+i} > 2b_i + R_{n+i}$, $-2a_i + R_{2n+i} < -(2b_i + 1) + R_{2n+i}$, $-(2a_i + 1) + R_{3n+i} < -2b_i + R_{3n+i}$, which contributes to 2 *true* values and 2 *false* values.

2. Similarly, $a_i < b_i$ also contributes to 2 *true* values and 2 *false* values.

3. If $a_i = b_i$, $2a_i + R_i < (2b_i + 1) + R_i$, $(2a_i + 1) + R_{n+i} > 2b_i + R_{n+i}$, $-2a_i + R_{2n+i} > -(2b_i + 1) + R_{2n+i}$, $-(2a_i + 1) + R_{3n+i} < -2b_i + R_{3n+i}$, which also contributes to 2 *true* values and 2 *false* values.

Therefore, at the end of the comparison, regardless of what A and B are, there will always be half $(2n)$ *true* values and half *false* values in the results. □

The above theorem indicates that the protocol discloses no statistical information about the relationship between a_i and b_i, including both inequality and equality relationships.

After getting A' and B', Alice and Bob reorder A' and B' using the same random permutation π, thus getting a new $A' = (a'_{\pi(1)}, \ldots, a'_{\pi(4n)})$ and (respectively) a new $B' = (b'_{\pi(1)}, \ldots, b'_{\pi(4n)})$. In what follows, to avoid unnecessarily cluttering the exposition with the $\pi(\cdot)$ notation, we assume that π is the identity permutation (so that $\pi(i) = i$); this is done purely for notational convenience and does not entail any loss of generality.

Step 3: Nonce Pairs Preparation

Alice and Bob each prepares $4n$ pairs of nonces $((q_1, q'_1), \ldots, (q_{4n}, q'_{4n}))$ and (respectively) $((p_1, p'_1), \ldots, (p_{4n}, p'_{4n}))$. They both keep these pairs secret from each other.

The order of these nonce pairs should correspond to the order of the numbers in A' and B', i.e., for $i = 1, \ldots, n$, (q_i, q'_i) corresponds to $2a_i + R_i$, (q_{n+i}, q'_{n+i}) corresponds to $(2a_i + 1) + R_{n+i}$, (q_{2n+i}, q'_{2n+i}) corresponds to $-2a_i + R_{2n+i}$, (q_{3n+i}, q'_{3n+i}) corresponds to $-(2a_i + 1) + R_{3n+i}$.

Alice computes $\alpha_+ = q_1 \oplus \cdots \oplus q_{2n} \oplus q'_{2n+1} \oplus \cdots \oplus q'_{4n}$, and $\alpha_- = q'_1 \oplus \cdots \oplus q'_{2n} \oplus q_{2n+1} \oplus \cdots \oplus q_{4n}$. Alice then sends, as a commitment to α_+ and α_-, a one-way hash

of each of them to Bob. Intuitively, α_+ represents the fact that A' dominates B', and α_- represents the fact that B' dominates A'.

Bob computes $\beta_+ = p_1 \oplus \cdots \oplus p_{2n} \oplus p'_{2n+1} \oplus \cdots \oplus p'_{4n}$, and $\beta_- = p'_1 \oplus \cdots \oplus p'_{2n} \oplus p_{2n+1} \oplus \cdots \oplus p_{4n}$. Bob then sends Alice, as a commitment to β_+ and β_-, a one-way hash of each of them to Alice. Intuitively, β_+ represents the fact that B' dominates A', and β_- represents the fact that A' dominates B'.

Step 4: Evaluation

First, Alice sends her $A' = (a'_1, \ldots, a'_{4n})$ and her nonce pairs $((q_1, q'_1), \ldots, (q_{4n}, q'_{4n}))$ to Ursula; Bob also sends his nonce pairs $((p_1, p'_1), \ldots, (p_{4n}, p'_{4n}))$ to Ursula, but he keeps $B' = (b'_1, \ldots, b'_{4n})$ to himself.

Next, after Ursula initializes h to 0, she and Bob use a modified version of Cachin's protocol for Yao's Millionaire Problem [21] to compare a'_i and b'_i, for each $i = 1, \ldots, 4n$; the modification to Cachin's protocol that is used makes it such that only Ursula knows the outcome of whether $a'_i > b'_i$, as opposed to both knowing the outcome (this can be easily done by skipping the step of "sending h_B to B" in the description of Cachin's protocol as given in [21]). Ursula updates h based on the outcome of this comparison: If $a'_i > b'_i$, she does $h = h \oplus q_i \oplus p'_i$, otherwise she does $h = h \oplus q'_i \oplus p_i$. (Note that if, in the above, we did not use a modified version of Cachin's protocol, then Bob would inappropriately know the outcome of a comparison between an a'_i and a b'_i, i.e., between a_i and b_i.)

After the above is done for all $4n$ pairs a'_i, b'_i, Ursula sends the final h to both Alice and Bob.

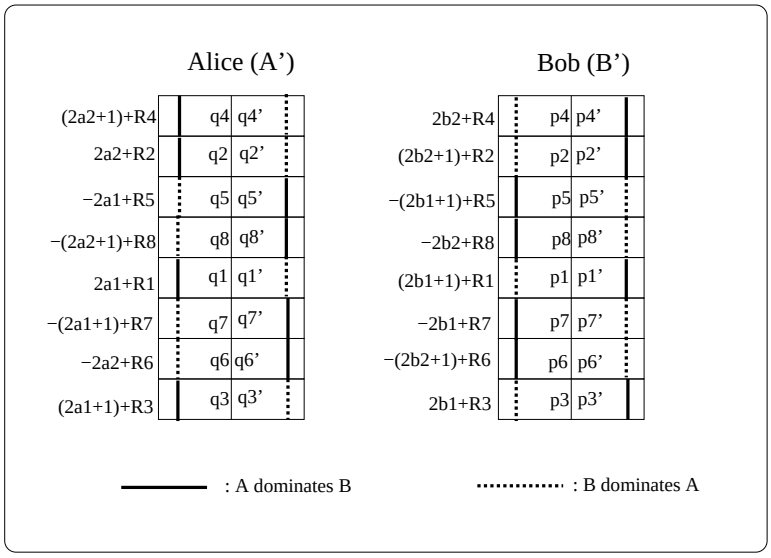

Figure 3.2. Example

Step 5: Result Extraction

Alice sends Bob her α_+ and α_-, and Bob sends Alice his β_+ and β_-, and they each verify that what they received matches the commitments received in Step 3.

Alice and Bob then each computes an "A-dominates" number $h_1 = \alpha_+ \oplus \beta_-$ and a "B-dominates" number $h_2 = \alpha_- \oplus \beta_+$.

Finally, each of Alice and Bob compares h with h_1 and with h_2. If $h = h_1$, then A dominates B; if $h = h_2$, then B dominates A; otherwise, neither one dominates the other.

Figure 3.2 is an example with $n = 2$ and $\pi(1, 2, 3, 4, 5, 6, 7, 8) = (5, 2, 8, 1, 3, 7, 6, 4)$. The nonces marked by the solid lines are those that will be selected by Ursula if $A \succ B$; h_1 is constructed by *xor*'ing all of these and only these solid-line nonces. The nonces marked by the dotted lines are those that will be selected by Ursula if $B \succ A$;

h_2 is constructed by *xor*'ing all of these and only these dotted-line nonces. If neither A dominates B nor B dominates A, then Ursula will end up choosing some of the nonces marked by the solid lines and some of the nonces marked by the dotted lines, which causes $h \neq h_1$ and $h \neq h_2$.

In certain cases, we are only interested in whether A dominates B, and we do not want either party know whether B dominates A; in some other cases, only Alice (or Bob) is allowed to know the dominance result. The above protocol can be easily modified to accommodate these requirements. For example, if we only allow both parties to learn whether A dominates B, we can change the above protocol such that h_1 is computable by both parties and h_2 is not.

Theorem 3.3.2. *The above protocol correctly determines whether (and which) one of A, B dominates the other.*

Proof. It clearly suffices to prove the following three sub-claims.

Sub-claim 1: If there exists $a_i = b_i$, then with overwhelming probability $h \neq \alpha_+ \oplus \beta_-$, and $h \neq \alpha_- \oplus \beta_+$.

Sub-claim 2: If for all $i = 1, \ldots, n$ we have $a_i \neq b_i$ then with overwhelming probability the case of $A \succ B$ is detected by checking whether $h = \alpha_+ \oplus \beta_-$.

Sub-claim 3: If for all $i = 1, \ldots, n$ we have $a_i \neq b_i$ then with overwhelming probability the case of $B \succ A$ is detected by checking whether $h = \alpha_- \oplus \beta_+$.

Proof of Sub-claim 1:

If there exists $a_i = b_i$, we will have $2a_i + R_i < (2b_i + 1) + R_i$ and $(2a_i + 1) + R_{n+i} > 2b_i + R_{n+i}$. Therefore, corresponding to these two comparisons, Ursula will select q'_i,

p_i, q_{n+i} and p'_{n+i} to compute h. However, the corresponding selection by α_+'s for these two comparisons is q_i and q_{n+i}; the corresponding choice by β_-'s for these two comparisons is p'_i and p'_{n+i}. Because with overwhelming probability $q'_i \oplus q_{n+i} \oplus p_i \oplus p'_{n+i} \neq q_i \oplus q_{n+i} \oplus p'_i \oplus p'_{n+i}$, we have $h \neq \alpha_+ \oplus \beta_-$. Although it is theoretically possible for equality to "coincidentally" occur as a chance occurrence of the *xor*'ing of the wrong set of nonces; choosing each nonce to be large enough easily decreases the probability of such an occurrence to almost zero.

Similarly, we can prove $h \neq \alpha_- \oplus \beta_+$.

Proof of Sub-claim 2: First, we have $\alpha_+ = q_1 \oplus \cdots \oplus q_{2n} \oplus q'_{2n+1} \oplus \cdots \oplus q'_{4n}$, and $\beta_- = p'_1 \oplus \cdots \oplus p'_{2n} \oplus p_{2n+1} \oplus \cdots \oplus p_{4n}$. According to the protocol, if $A \succ B$, we have $h = q_1 \oplus p'_1 \oplus \cdots \oplus q_{2n} \oplus p'_{2n} \oplus q'_{2n+1} \oplus p_{2n+1} \oplus \cdots \oplus q'_{4n} \oplus p_{4n}$, therefore $h = \alpha_+ \oplus \beta_-$.

Now let us consider the other direction. Suppose $h = \alpha_+ \oplus \beta_-$, therefore $h = q_1 \oplus p'_1 \oplus \cdots \oplus q_{2n} \oplus p'_{2n} \oplus q'_{2n+1} \oplus p_{2n+1} \oplus \cdots \oplus q'_{4n} \oplus p_{4n}$. Because h is constructed from $q_1, \ldots, q_{4n}$, $q'_1, \ldots, q'_{4n}$, $p_1, \ldots, p_{4n}$, and $p'_1, \ldots, p'_{4n}$, with overwhelming probability, h must be constructed exactly by (q_1, p'_1), $\ldots$ (q_{2n}, p'_{2n}), (q'_{2n+1}, p_{2n+1}), $\ldots$ (q'_{4n}, p_{4n}), which means $a_i > b_i$ and $-a_i < -b_i$ for $i = 1, \ldots, n$, indicating that A dominates B. As in Sub-claim 1, here too it is theoretically possible to fail because of a chance occurrence of the *xor*'ing of the wrong set of nonces: The next sub-section shows how easily the probability of this can be brought to almost zero by using suitably long nonces.

Proof of Sub-claim 3: Similar to the proof of Sub-claim 2. □

Making the Protocol Symmetric

The above protocol can easily be modified so it is symmetric in the roles of Alice and Bob, in the following way. Instead of Alice sending to Ursula all of her A' and Bob sending Ursula none of his B', Alice and Bob agree on a subset I of $\{1, \ldots, 4n\}$. Then Alice sends Ursula I together with the entries of A' whose indices are in I, i.e., $\{a'_i \ : \ i \in I\}$. What Bob sends Ursula are, of course, the entries of B' whose indices are not in I. It is important that Alice receives i with each a'_i or b'_i that she receives, so she knows the position they occupied in A' or (respectively) B'. The rest of the protocol is easily modified accordingly: Whereas the a'_i values received by Ursula are treated by the rest of the new protocol in the same way as in the original protocol we described earlier, for the b'_i values received by Ursula the new protocol effectively reverses the roles that Alice and Bob played in the original protocol.

Communication Complexity Analysis

According to [21], the communication complexity of Cachin's protocol is $O(\ell)$, where ℓ is the number of bits of each input number. Therefore, if all input numbers are in $[0, 2^\ell]$, the communication cost for using Cachin's protocol $O(n)$ times is $O(\ell n)$.

Therefore, the total communication cost (including sending nonces, α_+, α_-, β_+, β_-, h, A', and the $O(n)$ round of using Cachin's protocol) is $O(\ell n)$, i.e., it is linear in the number of bits needed to represent Alice's and Bob's input vectors.

3.3.2 Secure Two-Party Vector Dominance Protocol 2

The Protocol

We first give an outline of the protocol, then discuss each step in details.

Protocol 3.3.2. *(Secure Two-Party Vector Dominance Protocol)*

Inputs: Alice has a vector A, Bob has a vector B.

1. Inputs Disguise: Using the same disguise technique as the one used in the three-party protocol, Alice gets the disguised input $A' = (a'_1, \ldots, a'_{4n})$, and Bob gets the disguised input $B' = (b'_1, \ldots, b'_{4n})$. Let v_A be a number corresponding to the situation of A dominating B, where

$$V_A = (\overbrace{1, \ldots, 1}^{2n}\overbrace{0, \ldots, 0}^{2n}).$$

2. Private Permutation: Bob generates a random permutation π and a random vector R, then computes $B'' = \pi(B' + R)$, $V'_A = \pi(V_A)$. Alice, using Secure Two-Party Permutation Protocol (Protocol 3.1.2), gets $A'' = \pi(A' + R)$.

3. Yao's Millionaire Comparison: Alice and Bob use Yao's Millionaire protocol as subroutine to compare A''_i with B''_i, for $i = 1, \ldots, 4n$, where A''_i (resp., B''_i) is the ith element of vector A'' (resp., B''). At the end, Alice gets the result $U = \{u_1, \ldots, u_{4n}\}$, where $u_i = 1$ if $A''_i > B''_i$, otherwise $u_i = 0$.

4. Dominance Testing:: Alice and Bob use a private comparison protocol to compare U with V'_A: If $U = V'_A$, then A dominates B; otherwise, A does not

dominate the B. (Note: this step is necessary for the point-inclusion protocol, but must be skipped for the intersection protocol.)

Outputs: If the Dominance Testing step needs to be skipped, Alice outputs U and Bob outputs V'_A. Otherwise, Alice outputs the dominance testing result.

Step 1: Inputs Disguise

This step is fully discussed in Vector Dominance Protocol (with a third party). For convenience, we assume a_i and b_i for $i = 1, \ldots, n$ are integers; however our scheme can be easily extended to the non-integer case. The disguised inputs are the followings:

$$\begin{aligned} A' &= (2a_1, \ldots, 2a_n, (2a_1+1), \ldots, (2a_n+1), \\ &\quad -2a_1, \ldots, -2a_n, -(2a_1+1), \ldots, -(2a_n+1)) \qquad (3.1) \\ B' &= ((2b_1+1), \ldots, (2b_n+1), 2b_1, \ldots, 2b_n, \\ &\quad -(2b_1+1), \ldots, -(2b_n+1), -2b_1, \ldots, -2b_n) \qquad (3.2) \end{aligned}$$

The purpose of the inputs disguise is to get the same number of $a'_i > b'_i$ situations as that of $a'_i < b'_i$ situations; therefore, nobody knows how many a_i's are larger than b_i's and vice versa. The disguise is based on the fact that if $a_i > b_i$, then $2a_i > 2b_i + 1$, $(2a_i + 1) > 2b_i$, $-2a_i < -(2b_i + 1)$, and $-(2a_i + 1) < -2b_i$, which generates two $>$'s, and two $<$'s.

Step 2: Private Permutation

This step is fully discussed in the Secure Two-Party Permutation Protocol (Protocol 3.1.2).

Step 3: Yao's Millionaire Comparison

Alice now has $A'' = \pi(A' + R) = (a''_1, \ldots, a''_{4n})$, Bob has $B'' = \pi(B' + R) = (b''_1, \ldots, b''_{4n})$. They can use Yao's Millionaire Protocol to compare each a''_i with b''_i. Actually it is a one-side (asymmetric) version of it because only Alice learns the result. So at the end of this step, Alice gets $U = (u_1, \ldots, u_{4n})$, where for $i = 1, \ldots, 4n$, $u_i = 1$ if $a'_i > b'_i$, otherwise $u_i = 0$.

Step 4: Dominance Testing

Because V_A is exactly what U should be if vector A dominates B, we only need to find out whether $U = V_A$. Alice cannot just send U to Bob because it will allow Bob to find out the relationship between a_i and b_i for each $i = 1, \ldots, n$. So we need a way for Alice and Bob to determine whether Alice's U equals Bob's V_A without disclosing each person's private input to the other person.

This comparison problem is well studied, and was thoroughly discussed by Fagin, Naor, and Winkler [43]. Several methods for it were discussed in [43, 82]. For example, the following is part of the folklore:

Protocol 3.3.3. *(Equality-Testing Protocol)*

Inputs: Alice has U, Bob has V_A.

Outputs: $U = V_A$ iff $E_B(E_A(U)) = E_A(E_B(V_A))$.

1. Alice encrypts U with a commutative encryption scheme, and gets $E_A(U)$; Alice sends $E_A(U)$ to Bob.

2. Bob encrypts $E_A(U)$, and gets $E_B(E_A(U))$; Bob sends the result back to Alice.

3. Bob encrypts V_A, gets $E_B(V_A)$; Bob sends $E_B(V_A)$ to Alice.

4. Alice encrypts $E_B(V_A)$, gets $E_A(E_B(V_A))$.

5. Alice compares $E_B(E_A(U))$ with $E_A(E_B(V_A))$.

3.3.3 Complexity Analysis

According to [21], the communication complexity of Cachin's protocol is $O(\ell)$, where ℓ is the number of bits of each input number. Therefore, if all input numbers are in $[0, 2^\ell]$, the communication cost for using Cachin's protocol $O(n)$ times is $O(\ell n)$.

In Protocol 3.3.1, the total communication cost (including sending nonces, α_+, α_-, β_+, β_-, h, A', and the $O(n)$ round of using Cachin's protocol) is $O(\ell n)$; In Protocol 3.3.2, the total communication cost is also $O(\ell n)$; i.e., the communication cost of both protocols is linear in the number of bits needed to represent Alice's and Bob's input vectors.

3.3.4 Applications

In this section, we describe some specific applications of Secure Two-Party Vector Dominance Protocol.

Multi-Commodity Private Bidding and Auction

Bidding often involves multiple items in an "all-or-nothing" fashion: You are not interested in getting the rental car in isolation of the airline fare, cruise line, hotel room, etc. In business-to-business bidding, a manufacturer may want to deal with a single supplier that can simultaneously satisfy all of its n requirements (either because

there is some coordination required in the production of the n items types, or simply to avoid the bureaucratic overhead of having to deal with multiple suppliers).

This problem can be described as the following: Alice wants to buy n items (numbered 1 to n) from Bob but only if the cost of the ith item is less than a_i *for all* $i \in \{1, \ldots, n\}$, while Bob is willing to sell to A but only for more than (respectively) $b_1, \ldots, b_n$; that is, if for some item i we do not have $a_i > b_i$ then no other item j will be bought even if it does satisfy $a_j > b_j$. The protocol should not reveal to Alice or Bob *anything* other than whether they have a deal.

The problem is exactly a dominance problem: Alice and Bob each have a vector of dimensionality n, and the goal of the protocol is for Alice and Bob to determine whether Alice's vector *dominates* Bob's vector, i.e., whether $a_i > b_i$ for all $i \in \{1, \ldots, n\}$. Therefore, the problem can be solved using Vector Dominance Protocol.

Privacy-Preserving Negotiation

Bob wants to buy a product P, and he has a few requirements on this product, but because these requirements are usually business secrets, he does not want to disclose these requirements. Alice, on the other hand, has a new product that she wants to sell to Bob; however the parameters (or features) of this new product are also business secrets, and if Bob does not buy the product, Alice will not disclose those parameters to Bob. How could Alice and Bob decide whether they have a match without disclosing their secrets to the other party?

We can represent Bob's requirement using a range, say (x_i, y_i) for the ith feature $(i = 1, \ldots, n)$. We also represent the parameters of the Alice's product as $(p_1, \ldots, p_n)$. The task is to find if both $p_i > x_i$ and $p_i < y_i$ are *true* for all $i = 1, \ldots, n$. If the result is *no*, both parties learn nothing else except this answer itself.

The problem can be easily transformed to a dominance problem: to decide whether $(p_1, -p_1, \ldots, p_n, -p_n)$ dominates $(x_1, -y_1, \ldots, x_n, -y_n)$. Our Vector Dominance Protocol could be used to solve this problem.

Ancestor-Descendent Relationship in a Tree

Alice and Bob both know a tree, Alice knows a node A, and Bob knows a node B. Alice and Bob want to know whether A and B have an ancestor-descendent relationship. If they do not have a such relationship, nobody should learn the other party's node or the relative location of the other party's node, such as A is at the left side of B etc.

This problem can also be reduced to the dominance problem: Let us use (x_{pre}, x_{post}) to represent a node in the tree, where x_{pre} is the pre-order number of the node, and x_{post} is the post-order number of the node. Node $A = (a_1, a_2)$ is an ancestor of node $B = (b_1, b_2)$ if both $a_1 < b_1$ and $a_2 > b_2$ are *true*. Therefore finding the ancestor-descendent of node A and B is equivalent to finding whether $(-a_1, a_2)$ dominates $(-b_1, b_2)$.

3.4 Chapter Summary

In this chapter, we have described various protocols to solve two important problems: Secure Two-Party Scalar Product Problem and Secure Two-Party Vector Dominance Problem. These protocols will be used as building blocks throughout this dissertation.

4. SECURE TWO-PARTY SCIENTIFIC COMPUTATIONS

Linear systems of equations problems, linear least squares problems [74] or linear programming problems [90] are three scientific computation problems that have proved valuable for modeling many and diverse types of problems in planning, routing, scheduling, assignment, and design. Industries that make use of these problems and their extensions include banking, transportation, energy, telecommunications, and manufacturing of many kinds. Although these problems have been well studied in the literature, their current solutions rarely extend to the situation in which multiple parties want to jointly conduct the computations based on the private inputs.

For instance, Alice has k linear equations in n unknown variables x_i; Bob has $n-k$ linear equations in the same n unknown x_i. Alice and Bob want to find the solution $(x_1, \ldots, x_n)$ that satisfies the combined n linear equations. This problem can be easily solved if Alice can give her equations to Bob or vice versa, but, if the equations owned by each party are valuable proprietary data that neither party is willing to disclose to the other the problem can no longer be solved using the traditional methods, such as Gaussian elimination and LU factorization, because these methods assume that one who conducts the computation knows all the inputs, an assumption that is not true any more in the secure two-party computation situation. We need to find solutions that allow Alice and Bob to jointly solve their combined n linear equations while not disclosing each person's private equations to the other.

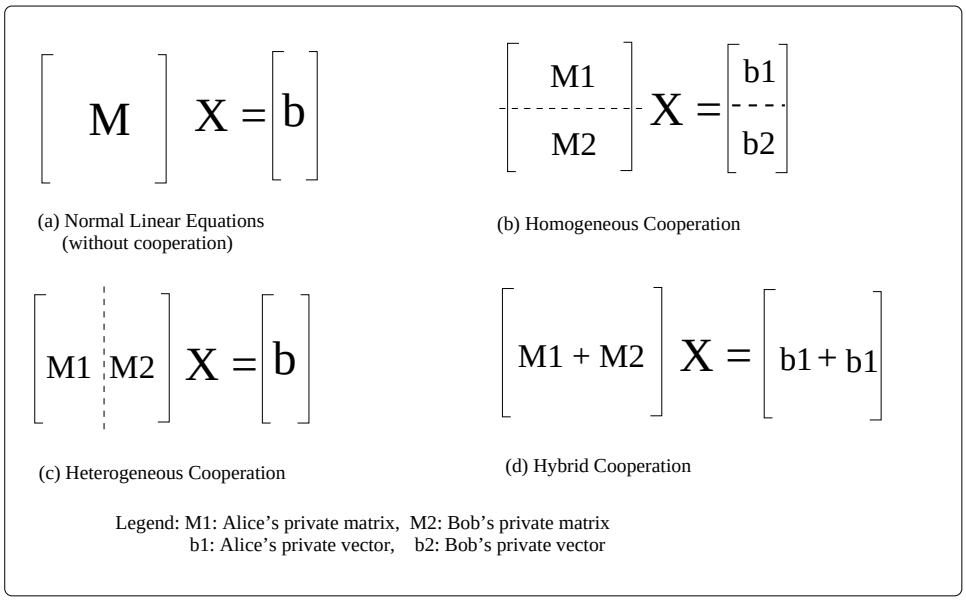

Figure 4.1. Various Ways of Cooperation

In this chapter, we introduce a new problem, the Secure Two-Party Scientific Computations (STPSC) problem. The general definition of the STPSC problem is that two or more parties want to conduct a scientific computation based on their private inputs, but neither party is willing to disclose its own input to anybody else. We have further defined several specific STPSC problems, including the Secure Two-Party Linear System of Equations (STP-LSE) problem, the Secure Two-Party Linear Least-Square (STP-LLS) problem, and the Secure Two-Party Linear Programming (STP-LP) problem, all of which involve a matrix.

There are several ways to share a matrix. Depending on how such a matrix is shared by Alice and Bob, the problems could appear in a variety of forms. Figure 4.1 describes three different types of cooperation.

Figure 4.1(b) depicts homogeneous cooperation, in which each party provides its own equations; Figure 4.1(c) depicts heterogeneous cooperation, in which the two parties have to jointly specify each single equation; Figure 4.1(d) depicts hybrid cooperation, in which the two parties cooperate in an arbitrary way. (b) and (c) are more meaningful cooperations than (d) in real life, and they are two special cases of problem (d). We have developed a protocol to solve problem (d): $(M_1+M_2)x = b_1+b_2$, where matrix M_1 and vector b_1 belong to one party, matrix M_2 and vector b_2 belong to the other party. At the end of the protocol, both parties know the solution x while nobody knows the other party's private inputs. Based on this protocol and similar techniques, we have solved STP-LSE problems, STP-LLS problems, and STP-LP problems.

4.1 Secure Two-Party Linear System of Equations Problem

Problem 4.1.1. *(STP-LSE)* Alice has a matrix M_1 and a vector b_1; Bob has a matrix M_2 and a vector b_2; M_1 and M_2 are $n \times n$ matrices, and b_1 and b_2 are n-dimensional vectors. Without disclosing their private inputs to the other party, Alice and Bob want to solve the linear equation

$$(M_1 + M_2)x = b_1 + b_2.$$

The Protocol Without concerning privacy, a straightforward solution would be to ask one party (say Bob) to send his M_2 and b_2 to the other party, Alice. This however does not work if Bob is concerned about the privacy of his data. Bob cannot simply

send M_1 and b_1 to Alice; he has to disguise the data in a way such that Alice cannot derive the original data from the disguised data.

Our solution is based on the fact that the solution to the linear equations $(M_1 + M_2)x = b_1 + b_2$ is equivalent to the solution to the linear equations $P(M_1+M_2)QQ^{-1}x = P(b_1 + b_2)$, if P and Q are invertible. If Alice knows $M' = P(M_1 + M_2)Q$ and $b' = P(b_1 + b_2)$, she can solve the linear equation problem: $M'\hat{x} = b'$, and thus gets the final solution $x = Q\hat{x}$. But how can Alice know M' and b' without knowing the value of M_2 and b_2? To solve this problem, Bob generates two invertible random $n \times n$ matrices P and Q; then Alice and Bob use secure protocols (will describe them later) to get Alice (and only Alice) to learn the value of $P(M_1 + M_2)Q$ and $P(b_1 + b_2)$. However, Alice will not learn the value of PM_1Q, PM_2Q, Pb_1, Pb_2, much less P, Q, M_2, or b_2.

After Alice gets $M' = P(M_1 + M_2)Q$ and $b' = P(b_1 + b_2)$, she can solve the linear equations $M'\hat{x} = b'$ by herself, and then send the solution $\hat{x}$ to Bob, who can compute the final solution $x = Q\hat{x}$. Finally Bob sends the solution to Alice. Although we do not prevent disruption of the entire computation if Alice or Bob misbehaves, we do allow Alice to detect the case where Bob learns the correct answer but does not allow Alice to learn the correct answer. For example, after getting the actual solution, with an evil mind, Bob may decide not to tell Alice the actual solution x. He can do this without being caught because he can send an arbitrary vector to Alice, who has no way to verify whether the received vector is the actual solution or not. This is not fair to Alice. To achieve the fairness, Alice should request Bob to send back a vector

$v = M_2x - b_2$ along with the solution x. This vector does not give Alice any more power to derive Bob's data because if Bob is honest, Alice will know the value of $M_2x - b_2$ anyway because of $(M_1 + M_2)x = b_1 + b_2$. But if Bob still wants to cheat, he has to find two vectors x' and v', such that $M_1x' - b_1 = v'$. Without knowing M_1 and b_1, Bob cannot find these two vectors. The protocol is described in the following:

Protocol 4.1.1. *(STP-LSE)*

Inputs: Alice has a matrix M_1 and a vector b_1; Bob has a matrix M_2 and a vector b_2. M_1 and M_2 are $n \times n$ matrices; b_1 and b_2 are n-dimensional vector.

Outputs: Alice and Bob both get the solution x.

1. Bob generates two invertible random $n \times n$ matrices P and Q.

2. Alice and Bob use a secure protocol (described later) to evaluate $M' = P(M_1 + M_2)Q$. Only Alice knows the result M'.

3. Alice and Bob use a secure protocol (described later) to evaluate $b' = P(b_1 + b_2)$. Only Alice knows the result b'.

4. Alice solves the linear equations $M'\hat{x} = b'$. If the solution does not exist, Alice tells Bob, then terminates the protocol. If the solution exists, Alice sends the solution $\hat{x}$ to Bob.

5. Bob computes $x = Q\hat{x}$ and $v = M_2x - b_2$, then sends both vectors x and v to Alice.

6. Alice checks whether x is the actual solution by verifying whether $||(M_1x - b_1) + v||$ equals zero (or close to zero within the acceptable range if computation errors are inevitable).

Private Evaluation of $M' = P(M_1 + M_2)Q$

To privately evaluate M', Alice could send p matrices to Bob, with one of the matrices being M_1 and the rest of the matrices being random; however, Bob does not know which one is M_1. Then Bob computes the $P(H_i + M_2)Q$ for each matrix H_i he receives. At the end Alice uses the 1-out-of-N oblivious transfer protocol to get back from Bob one and only one of the results, the result of $M' = P(M_1 + M_2)Q$. Because of the way the 1-out-of-N oblivious transfer protocol works, Alice can decide which result to get, but Bob cannot learn which one Alice has chosen. However there is one drawback in this approach: if the value of M_1 has certain publicly-known properties, Bob might be able to differentiate M_1 from the other random vectors. More seriously, after Bob finally gets the solution x, it only takes him p^2 tries to find both M_1 and b_1.

The above drawback can be fixed by dividing the matrix M_1 into m random matrices $X_1, \ldots, X_m$, with $M_1 = \sum_{i=1}^{m} X_i$. Alice and Bob can use the same method as the one described above to compute $P(X_i + M_2)Q$. As a result of the protocol, Alice gets $P(X_i + M_2)Q$ and Bob only knows one of the p vectors is X_i, but because of the randomness of X_i, Bob cannot find out which one is X_i. Certainly, there is an 1 out of p possibility that Bob could find the correct X_i by guessing, but since M_1 is

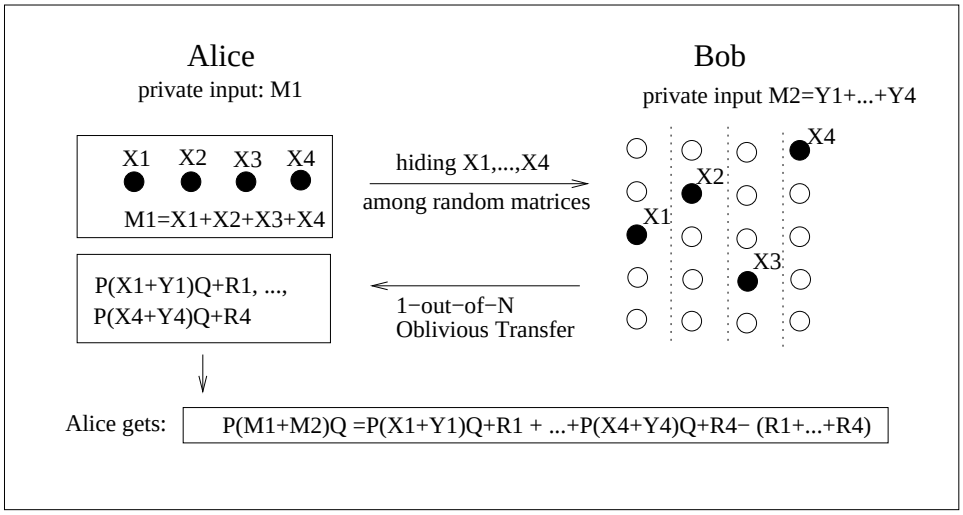

Figure 4.2. Private Evaluation of $P(M_1 + M_2)Q$

the sum of m such random matrices, the chance that Bob guesses the correct M_1 is 1 out of p^m, which could be very small if we chose p^m large enough.

However, knowing the values of $P(X_i + M_2)Q$ for $i = 1, \ldots, m$ might make it easier for Alice to figure out the value of M_2, therefore, Bob also needs to disguise the results of $P(X_i + M_2)Q$. One way to do this is to divide M_2 to m random matrices $(Y_1, \ldots, Y_m)$ as well, each time Bob returns the values of $P(X_i + Y_i)Q + R_i$ for $i = 1, \ldots, m$, where R_i's are also random matrices.

After Alice gets $P(X_i + Y_i)Q + R_i$ for $i = 1, \ldots, m$, she can sum them up and get $P(M_1 + M_2)Q + \sum_{i=1}^{m} R_i$. Bob can send the result of $\sum_{i=1}^{m} R_i$ to Alice who can then get $P(M_1 + M_2)Q$. Figure 4.2 explains how the protocol works. The detail of the protocol is described in the following:

Protocol 4.1.2. *(Private Evaluation of $P(M_1 + M_2)Q$)*

Inputs: Alice has a Matrix M_1; Bob has a Matrix M_2 and two random invertible matrices P and Q.

Outputs: Alice gets $M' = P(M_1 + M_2)Q$.

1. Alice and Bob agree on two numbers p and m, such that p^m is so big that conducting p^m additions is computationally infeasible.

2. Alice generates m random matrices $X_1, \ldots, X_m$, such that $M_1 = X_1 + \ldots + X_m$.

3. Bob generates m random matrices $Y_1, \ldots, Y_m$, such that $M_2 = Y_1 + \ldots + Y_m$.

4. For each $j = 1, \ldots, m$, Alice and Bob conduct the following sub-steps:

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

   (b) Alice sends the following sequence to Bob:

   $$(H_1, \ldots, H_p)$$

   where $H_k = X_j$, and the rest of the sequence are random matrices. Because of the secrecy of k, Bob cannot know the position of X_j.

   (c) Bob computes $P(H_i + Y_j)Q + R_j$ for each $i = 1, \ldots, p$, where R_j is a random matrix.

   (d) Using 1-out-of-p Oblivious Transfer Protocol, Alice gets back the result of

   $$P(H_k + Y_j)Q + R_j = P(X_j + Y_j)Q + R_j.$$

5. Bob sends $\sum_{j=1}^{m} R_j$ to Alice.

6. Alice computes $M' = \sum_{j=1}^{m}(P(X_j + Y_j)Q + R_j) - \sum_{j=1}^{m} R_j = P(M_1 + M_2)Q$.

Intuitively, Alice preserves her privacy by both dividing her matrix M_1 to p random matrices that are further hidden among many other random matrices, and by getting the results back using the 1-out-of-N oblivious transfer protocol. Bob's privacy is preserved by the 1-out-of-N oblivious transfer protocol, random matrices Y_i's and R_i's.

Private Evaluation of $b' = P(b_1 + b_2)$

This protocol is similar to the protocol of evaluating M' and the security property can be proved similarly.

Protocol 4.1.3. *(Private Evaluation of $P(b_1 + b_2)$)*

Inputs: Alice has a vector b_1; Bob has a vector b_2 and a random invertible matrix P.

Outputs: Alice gets $b' = P(b_1 + b_2)$.

1. Alice and Bob agree on two numbers p and m, such that p^m is so big that conducting p^m additions is computationally infeasible.

2. Alice generates m random vectors $x_1, \ldots, x_m$, such that $b_1 = x_1 + \ldots + x_m$.

3. Bob generates m random vectors $y_1, \ldots, y_m$, such that $b_2 = y_1 + \ldots + y_m$.

4. For each $j = 1, \ldots, m$, Alice and Bob conduct the following sub-steps:

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

   (b) Alice sends the following sequence to Bob:

   $$(h_1, \ldots, h_p)$$

   where $h_k = x_j$, and the rest of the sequence are random vectors. Because of the secrecy of k, Bob cannot know the position of x_j.

   (c) Bob computes $P(h_i + y_j) + r_j$ for each $i = 1, \ldots, p$, where r_j is a random vector.

   (d) Using the 1-out-of-p Oblivious Transfer protocol, Alice gets back the result of

   $$P(h_i + y_j) + r_j = P(x_j + y_j) + r_j$$

5. Bob sends $\sum_{j=1}^{m} r_j$ to Alice.

6. Alice computes $b' = \sum_{j=1}^{m}(P(x_j + y_j) + r_j) - \sum_{j=1}^{m} r_j = P(b_1 + b_2)$.

4.2 Secure Two-Party Linear Least-Squares Problem

The linear system of equations problem consists of n equations of n unknown variables. There are situations where we have more equations to satisfy than the number of unknown variables. Most often, we cannot satisfy all of these equations, but we may find a solution that can satisfy them as best as we can. This problem

is called the linear least-squares problem. We solve the Secure Two-Party Linear Least-Squares problem (STP-LLS) in this section.

Problem 4.2.1. *(STP-LLS)* Alice has a matrix M_1 and a vector b_1, and Bob has a matrix M_2 and a vector b_2, where M_1 and M_2 are $m \times n$ matrices ($m > n$), and b_1 and b_2 are m-dimensional vectors. Without disclosing their private inputs to the other party, Alice and Bob want to solve the linear equations

$$(M_1 + M_2)x = b_1 + b_2.$$

As there are more conditions (equations) to be satisfied than degrees of freedom (variables), it is unlikely that they can all be satisfied. Therefore, they want to attempt to satisfy the equations as best as they can–that is, make the size of the residual vector r as small as possible. The component of r is defined in the following:

$$r_j = c_j - \sum_{i=1}^{n} a_{ji} x_i$$

where a_{ji} are the entries in the new matrix $M = M_1 + M_2$, c_j are the entries in the new vector $b = b_1 + b_2$. The least-squares criterion is the use of the Euclidean (or least-squares) norm for the size of r; that is, minimize

$$\sqrt{\sum_{j=1}^{m} r_j^2} = ||r||_2.$$

Solution: The linear least squares problem $Mx = b$ can be expressed in a linear system:

$$M^T Mx = M^T b$$

that contains n linear equations in the n unknowns x_i, hence can be solved using the usual methods for the linear equations problem, such as the the Gaussian elimination method and the Cholesky method. Such an approach to solve the least-squares problem is called the normal equations approach because $M^T Mx = M^T b$ are normal equations.

In the Secure Two-Party Linear Least-Squares problem, $M = M_1 + M_2$, $b = b_1 + b_2$, so we have $M^T M = M_1^T M_1 + M_1^T M_2 + M_2^T M_1 + M_2^T M_2$ and $M^T b = M_1^T b_1 + M_1^T b_2 + M_2^T b_1 + M_2^T b_2$.

Therefore, the linear equations $M^T Mx = M^T b$ becomes the following:

$$(M_1^T M_1 + M_1^T M_2 + M_2^T M_1 + M_2^T M_2)x$$

$$= (M_1^T b_1 + M_1^T b_2 + M_2^T b_1 + M_2^T b_2).$$

Using the Matrix-Vector Product protocol and the Matrix Product protocol (both protocols will be described next), Alice and Bob can get the following:

$$V_1 + V_2 = M_1^T M_2$$

$$W_1 + W_2 = M_2^T M_1$$

$$v_1 + v_2 = M_1^T b_1$$

$$w_1 + w_2 = M_2^T b_2$$

where matrices V_1, W_1, vectors v_1 and w_1 are known only to Alice; matrices V_2, W_2, vectors v_2 and w_2 are known only to Bob. Let $M_1' = M_1^T M_1 + V_1 + W_1$, $M_2' = M_2^T M_2 + V_2 + W_2$, $b_1' = M_1^T b_1 + v_1 + w_1$, $b_2' = M_2^T b_2 + v_2 + w_2$, we have

$$(M_1' + M_2')x = b_1' + b_2'$$

where M_1' and M_2' are $n \times n$ matrices, and b_1' and b_2' are vectors of length n; M_1' and b_1' are known only to Alice, and M_2' and b_2' are known only to Bob. This is a STP-LSE problem. It can be solved using the STP-LSE protocol described in 4.1.

Protocol 4.2.1. *Matrix Product Protocol* Alice has a private matrix A, Bob has a private matrix B. At the end of the protocol, Alice gets R_a, and Bob gets R_b, where $R_a + R_b = AB$, R_a and R_b are random matrices.

Inputs: Alice has a private matrix A, Bob has a private matrix B.

Outputs: Alice gets R_a, and Bob gets R_b, such that $R_a + R_b = AB$.

1. Alice and Bob agree on two numbers p and m, such that p^m is so big that conducting p^m additions is computationally infeasible.

2. Alice generates m random matrices $X_1, \ldots, X_m$, such that $M_1 = X_1 + \ldots + X_m$.

3. For each $j = 1, \ldots, m$, Alice and Bob conduct the following sub-steps:

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

   (b) Alice sends the following sequence to Bob:

   $$(H_1, \ldots, H_p)$$

   where $H_k = X_j$, and the rest of the sequence are random matrices. Because of the secrecy of k, Bob cannot know the position of X_j.

   (c) Bob computes $H_i B - R_j$ for each $i = 1, \ldots, p$, where R_j is a random matrix, and each element of the matrix $H_i B - R_j$ is within field F.

   (d) Using the 1-out-of-N Oblivious Transfer protocol, Alice gets back

   $$H_k B - R_j = X_j B - R_j.$$

4. Alice gets $R_a = \sum_{j=1}^{m} (X_j B - R_j) = AB - \sum_{j=1}^{m} R_j$, and Bob gets $R_b = \sum_{j=1}^{m} R_j$.

Protocol 4.2.2. *Matrix-Vector Product Protocol* Alice has a private matrix A, Bob has a private vector b. At the end of the protocol, Alice gets r_a, and Bob gets r_b, where $r_a + r_b = Ab$, R_a and R_b are random vectors.

The protocol is similar to the Matrix Product protocol. Replace each occurrence of matrix B in the Matrix Product protocol with the vector b; replace the random matrix R_j with the random vector r_j for $j = 1, \ldots, m$; also replace the matrix R_a with the vector r_a, and R_b with r_b.

Based on the Matrix Product Protocol and the Matrix-Vector Product Protocol, we have the following Secure Two-Party Linear Least-Square Protocol:

Protocol 4.2.3. *(STP-LLS)*

Inputs: Alice has a private matrix M_1 and a private vector b_1; Bob has a private matrix M_2 and a private vector b_2.

Outputs: Alice and Bob both get the solution x.

1. Using the Matrix-Vector product protocol and the Matrix product protocol, Alice gets V_1, W_1, v_1, and w_1; Bob gets V_2, W_2, v_2, and w_2; where, U_i and W_i are matrices, v_i and w_i are vectors, and $V_1 + V_2 = M_1^T M_2$, $W_1 + W_2 = M_2^T M_1$, $v_1 + v_2 = M_1^T b_1$, $w_1 + w_2 = M_2^T b_2$.

2. Alice computes $M_1' = M_1^T M_1 + V_1 + W_1$ and $b_1' = M_1^T b_1 + v_1 + w_1$.

3. Bob computes $M_2' = M_2^T M_2 + V_2 + W_2$ and $b_2' = M_2^T b_2 + v_2 + w_2$.

4. Alice and Bob use the STP-LSE protocol to solve $(M_1' + M_2')x = b_1' + b_2'$.

The linear least-squares problem is normally used in regression and mathematical modeling. Consider building an investment model for a financial organization. One example is to model customers' investment as a function of age. In such a case the bank knows or believes there are n different factors–all related to the age–that influence the customers' decision on investment, and the bank wants to build a mathematical model according to these n factors. Formally speaking, the bank wants to

find out the function $b(t) = \sum_{i=1}^{n} x_i f_i(t)$, where t is the variable representing the age, and $f_i(t)$ express the different age factors.

Suppose now that the bank takes a large number of observation from the data it collected, and obtains values b_j for t values t_j, $j = 1, \ldots, m$, and $m > n$. The problem of building such a mathematical model is to solve the following linear least-square system:

$$d_j = \sum_{i=1}^{n} f_i(t_j) x_i, j = 1, \ldots, m.$$

There are times when one financial organization does not have sufficient data to build such a mathematical model. It thereby needs to cooperate with another financial organization, that also wants to benefit from such cooperation. So both financial organizations would contribute their own data toward building such a model. Because this type of data usually consists of proprietary information that none of the financial organizations are willing to disclose to the others, these two financial organizations need to find a way to build the mathematical model without violating their privacy constraints. They can use the STP-LLS protocol.

4.3 Secure Two-Party Linear Programming Problem

The STP-LSE and STP-LLS problems are problems involving equations, which usually represent a set of constraints. But equations are not the only forms of constraints. Most often we will see another type of constraints, the inequalities, which exist in many linear programming problems [78]. In this section, we solve the secure two-party cooperative linear programming problem (STP-LP).

Problem 4.3.1. *(Secure Two-Party Linear Programming Problem)* Alice has a matrix M_1 and a vector b_1, and Bob has a matrix M_2 and a vector b_2, where M_1 and M_2 are $m \times n$ matrices, and b_1 and b_2 are m-dimensional vectors. Alice and Bob also know a vector c^T of size n. Alice and Bob want to optimize (minimize or maximize) the value of $c^T \cdot x$, without disclosing their private inputs to the other party. The solution $x = (x_1, \ldots, x_n)$ should satisfy the following linear constraints:

$$(M_1 + M_2)x \leq b_1 + b_2, x \geq 0.$$

Solution: Without the privacy concerns, the above problem is a linear programming problem, a problem of finding the optimum (maximum or minimum) value of a linear function subject to a number of linear constraints on the variables. The problem has many applications in real life, especially in solving optimization problems. As usual, the existing solutions–such as the simplex method–always assume the one who conducts the operation knows all the inputs, an assumption that does not hold in the secure two-party computation situation. Therefore, current solutions to the linear programming problem cannot be used directly to solve the STP-LP problem.

Similar to the solution to the STP-LSE problem, our solution is based on the fact that $P(M_1 + M_2)Q\ Q^{-1}x \leq P(b_1 + b_2)$, and $Q^{-1}x \geq 0$ if $(M_1 + M_2)x \leq b_1 + b_2$ and $x \geq 0$ (the elements of P and Q^{-1} should all be positive). Let $M' = P(M_1 + M_2)Q$, $b' = P(b_1 + b_2)$, $\hat{x} = Q^{-1}x$, and $c'^T = c^T Q$, we then have a new linear programming problem: optimize $c'^T\hat{x}$, where $\hat{x}$ is subject to $M'\hat{x} \leq b'$, $\hat{x} \geq 0$. We will prove later

that if $\hat{x}$ is the solution to this problem, $x = Q\hat{x}$ must be the solution to the original problem that optimizes $c^T x$.

Protocol 4.3.1. *(STP-LP)*

Inputs: Alice has a matrix M_1 and a vector b_1; Bob has a matrix M_2 and a vector b_2. Alice and Bob both know a vector c. M_1 and M_2 are $m \times n$ matrices; b_1 and b_2 are m-dimensional vectors; c is n-dimensional vectors.

Outputs: Alice and Bob both get the solution x.

1. Bob generates a random $m \times m$ matrix P and an invertible random $n \times n$ matrix Q, such that all of the elements in P and Q^{-1} are positive (actually, Bob generates Q^{-1} first, then gets Q).

2. Alice and Bob use Protocol 4.1.2 to compute $M' = P(M_1 + M_2)Q$, such that only Alice learns M'.

3. Similarly, Alice and Bob use Protocol 4.1.3 to compute $b' = P(b_1 + b_2)$, such that only Alice learns b'.

4. Bob computes $c'^T = c^T Q$, then sends c'^T to Alice.

5. Alice solves the the following linear programming problem: optimize $c'^T \hat{x}$, where $\hat{x}$ is subject to $M'\hat{x} \leq b'$, $\hat{x} \geq 0$. Alice then sends the solution $\hat{x}$ to Bob.

6. Bob computes $x = Q\hat{x}$, then sends x to Alice.

Theorem 4.3.1. Let M', M_1, M_2, P, Q, b', b_1, b_2, c, and c' be the matrices or vectors defined in the STP-LP protocol; let LP_1 and LP_2 be defined as the followings:

- LP_1: minimize $c'^T\hat{x}$, where $\hat{x}$ is subject to $M'\hat{x} \leq b'$, $\hat{x} \geq 0$.

- LP_2: minimize $c^T x$, where x is subject to $(M_1 + M_2)x \leq b_1 + b_2$, $x \geq 0$.

Suppose $\hat{x}$ is the solution to the linear programming problem LP_1, then $x = Q\hat{x}$ must be the solution to the linear program problem LP_2:

Proof. (By contradiction): Assume $x = Q\hat{x}$ is not the optimized solution for LP_2, then there is a vector y, such that $c^T y < c^T x$, where $(M_1 + M_2)y \leq b_1 + b_2$, and $y \geq 0$. Let $\hat{y} = Q^{-1}y$. Because $c'^T\hat{y} = c^T Q Q^{-1} y = c^T y < c^T x$ and $c^T x = c'^T\hat{x}$ imply $c'^T\hat{y} < c'^T\hat{x}$, $(M_1 + M_2)y \leq b_1 + b_2$ implies $P(M_1 + M_2)Q\hat{y} \leq P(b_1 + b_2)$, and $y \geq 0$ implies $\hat{y} \geq 0$ (remember P and Q^{-1}'s elements are all positive), we have $M'\hat{y} \leq b'$, $\hat{y} \geq 0$, and $c'^T\hat{y} < c'^T\hat{x}$. Therefore, $\hat{y}$ is a better solution to the LP_1 problem than $\hat{x}$. This is contradicted to the fact that $\hat{x}$ is the optimized solution to the LP_1 problem. □

A drawback of the above STP-LP protocol is its unfairness. In the last step of the protocol, Bob could decide not to send the actual solution x to Alice. Although we do not prevent disruption of the entire computation if Alice or Bob misbehaves, we do want to allow Alice to detect the case where Bob learns the correct answer but does not allow Alice to learn the correct answer. One way to solve this problem is to conduct the protocol for a second time, with Alice and Bob switching the roles; any mismatch between the two solutions could indicate the presence of dishonesty.

In some situation, the vector c^T could also be shared between Alice and Bob, which is to say, $c^T = c_1^T + c_2^T$, where c_1^T belongs to Alice, and c_2^T to Bob. The STP-LP protocol could be easily extended to handle this situation: instead of learning $c^T Q$ in

step 4, Alice learns $(c_1^T + c_2^T)Q$ by conducting with Bob the Matrix-Vector Product protocol whose privacy property prevents Alice from learning anything about either c_2^T or Q.

4.4 Protocol Efficiency

A Comparison to Generic Solutions

In this section, we will compare the communication cost of our approach with that of the general solution (the circuit evaluation approach).

For the STP-LSE problem (and also for the STP-LLS problem because it can be reduced to the STP-LSE problem), assume the size of the matrix M is $n \times n$, d is the maximum length to represent a number in the input domain, and c is the communication cost of the 1-out-of-n Oblivious Transfer protocol used in the circuit evaluation protocol. Assume that the Gaussian elimination method is used in both the STP-LSE protocol and the general solution.

We know that the cost of Gaussian elimination takes $O(n^3)$ multiplication operations. And as a rough estimate, the size of a secure circuit for a single multiplication is about $O(d^2)$. Therefore, the total size of the circuit to conduct the Gaussian elimination is $O(c * n^3 * d^2)$.

In the STP-LSE protocol, the cost of communication is $O(\mu * n^2)$, where μ is the security parameter. As the difficulty to compromise security is $O(2^\mu * n^2 * d)$ (n^2 is introduced by the multiplication of a matrix and a vector, and 2^μ is introduced by the oblivious transfer), setting $\mu = 256$ is reasonably secure. Therefore the cost of communication $O(\mu * n^2 * d)$ is better than $O(c * n^3 * d^2)$ when n and d are large.

To compare the STP-LP protocol with the general solution, we assume that the size of the matrix M is $m \times n$, where m is the number of inequalities, and n is the number of variables. We also assume that the Simplex method is used in solving the Linear Programming problem. It is well known that the Simplex algorithm is not a polynomial time algorithm. There are examples by Klee-Minty [69] of Simplex problems that are exponential time, though for all practical purposes, problems in the real world are usually low-degree polynomial time.

As our STP-LP protocol does not emulate such a circuit, the cost of communication, has nothing to do with the Simplex algorithm. The cost of the STP-LP protocol is $O(\mu * n * m * d)$, which is much better than the general solution, whose communication cost is exponential to the number of inequalities. Although there are some other ways to solve the linear programming problem, such as the Karmarkar's algorithm [65], the communication cost of them is still significantly worse than $O(\mu * n * m * d)$.

4.5 Applications

The Linear systems of equations problem, the linear least-square problem and the linear programming problem are three important scientific computation problems, which are widely used in many areas such as banking, manufacturing, and telecommunications. The following are two examples of situations where our protocols are needed.

- Two financial organizations plan to cooperatively work on a project for mutual benefit. Each of the organizations would like its own requirements being

satisfied (usually, these requirements are modeled as linear equations or linear inequalities). However, the requirements reflect the organizations' financial situations, economic statistics, strategic plans, customer's portfolio holdings, and their projections (of interest and inflation rates, and of the future evolution of certain commodity prices). These are valuable proprietary data that nobody is willing to disclose to other parties, or even to a "trusted" third party. How could these two financial organizations cooperate on this project?

- Two companies A and B are investigating an opportunity for a partnership. Company A's goal is to optimize the cost of a manufacturing process. As part of the partnership, company B will conduct part of the process. Because of this, A does not know B's constraints on that part of the process, unless B tells A, nor does B know A's constraints. Usually, the constraints reflect information about the company's resource, strategic plans, cost information, and business decisions. They are so critical that both companies try every measure to protect them. Considering that the partnership is not formed yet, B is afraid that, if the partnership eventually falls through, the information it provides to A might be used by A for B's disadvantage. With such a concern, B really does not feel comfortable to give its information to any other company, and neither does A. How could these two companies find out the benefit of a potential partnership without risking their private information?

4.6 Chapter Summary and Future Work

In this chapter, we have defined a set of secure two-party scientific computation problems: a secure two-party linear system of equations problem, a secure two-party linear least-square problem, and a secure two-party linear programming problem. We have developed protocols to solve these problems.

Rice points out that using $M^T Mx = M^T b$ to solve the linear least-square problem is not always the best approach, because it introduces the ill-conditioned matrix $M^T M$–the condition number of $M^T M$ is the condition number of M *squared* [90]. In the case where the condition number of $M^T M$ is too bad, the solution might be random numbers unrelated to the original problem. In those cases, other approaches– such as the Gram-Schmidt Orthogonalization approach and the Orthogonal Matrix Factorization approach– are better than the normal equations approach. Developing protocols to solve the least-square problem using these approaches is an avenue we could pursue in future work.

There are some other interesting scientific computation problems that need to be studied in future work, such as how to compute *eigenvalues*, *eigenvectors*, *determinants*, *conditions*, and *factorization* of a matrix in the secure two-party computation situation.

5. SECURE TWO-PARTY COMPUTATIONAL GEOMETRY PROBLEMS

In this chapter we investigate how various computational geometry problems could be solved in a cooperative environment, where two parties need to solve a geometric problem based on their joint data, but neither wants to disclose its private data to the other party. Some of the problems we solve in this framework are:

Problem 5.0.1. *(Point-Inclusion)* Alice has a point z, and Bob has a polygon P. They want to determine whether z is inside P, without revealing to each other any more than what can be inferred from that answer. In particular, neither of them is allowed to learn information about the relative position of z and P such as whether z is at the northwest side of P, or whether z is close to one of the borders of P, etc.

Problem 5.0.2. *(Intersection)* Alice has a polygon A, and Bob has a polygon B; they both want to determine whether A and B intersect (not where the intersection occurs).

Problem 5.0.3. *(Closest Pair)* Alice has M points in the plane, Bob has N points in the plane. Alice and Bob want to jointly find two points among these $M + N$ points, such that their mutual distance is smallest.

Problem 5.0.4. *(Convex Hulls)* Alice has M points in the plane, Bob has N points in the plane. Alice and Bob want to jointly find the convex hulls for these $M + N$

points; however, neither Alice nor Bob wants to disclose any more information to the other party than what could be derived from the result.

All of the above problems, as well as other computational geometry problems, are special cases of the general Secure Multi-party Computation problem [105, 57, 55]. In this chapter, we investigate how these specific SMC problems could be solved in a way more efficient than the circuit evaluation technique. Our study is a first step in this direction for the area of computational geometry.

5.1 Secure Two-Party Point-Inclusion Problem

We will look at how the point-location problem is solved in a straightforward way without worrying about the privacy concern. The computation cost of this straightforward solution is $O(n)$. Although we know the computation cost of the best algorithm for the point-location problem is only $O(\log n)$, we are concerned that the "binary search" nature of that solution might lead to the disclosure of partial information. Therefore, for a preliminary result, we focus on the $O(n)$ solution. The algorithm works as follows:

1. Find the leftmost vertex l and the rightmost vertex r of the polygon.

2. Decide whether the point $p = (\alpha, \beta)$ is above all the edges on the lower boundary of the polygon between l and r.

3. Decide whether the point p is below all the edges on the upper boundary of the polygon between l and r.

4. If the above two tests are both true, then the point is inside the polygon, otherwise it is outside (or on the edge) of the polygon.

If we use $f_i(x, y) = 0$ for the equation of the line boundary of the polygon, where $f_i(x, y) = 0$ for $i = 1, \ldots, m$ represent the edges on the lower part of the boundary and $f_i(x, y) = 0$ for $i = m + 1, \ldots, n$ represent the edges on the upper part of the boundary, then our goal is to decide whether $f_i(\alpha, \beta) > 0$ for all $i = 1, \ldots, m$ and $f_i(\alpha, \beta) < 0$ for all $i = m + 1, \ldots, n$.

The Protocol

First, we need to find a way to compute $f_i(\alpha, \beta)$ without disclosing Alice's $p : (\alpha, \beta)$ to Bob or Bob's f_i to Alice. Moreover, no party should learn the result of $f_i(\alpha, \beta)$ for any i because that could disclose the relationship between the location and the edge. As $f_i(\alpha, \beta)$ is a special case of scalar product, we can use the Secure Two-Party Scalar Product Protocol to solve this problem. In this protocol, we will let both parties share the result of $f_i(\alpha, \beta)$, namely, one party will have u_i, the other party will have v_i, and $u_i = f_i(\alpha, \beta) + v_i$; therefore nobody learns the value of $f_i(\alpha, \beta)$, but they can find out whether $f_i(\alpha, \beta) > 0$ by comparing whether $u_i > v_i$, which could be done using Yao's Millionaire Protocol [104, 21].

However, we cannot use Yao's Millionaire Protocol for each (u_i, v_i) pair individually because that would disclose the relationship between u_i and v_i, thus reveal too much information. In fact, all we want to know is whether $(u_1, \ldots, u_n)$ dominates

$(v_1, \ldots, v_n)$. This problem can be solved using the *Vector Dominance Protocol* (Protocol 3.3.2).

Based on the Scalar Product Protocol and Vector Dominance Protocol, we have the following Secure Two-Party Point-Inclusion Protocol:

1. Bob generates n random numbers $v_1, \ldots, v_n$.

2. Alice and Bob use the Scalar Product Protocol to compute $u_i = f_i(\alpha, \beta) + v_i$, for $i = 1, \ldots, m$ and compute $u_i = -f_i(\alpha, \beta) + v_i$ for $i = m+1, \ldots, n$. According to the scalar product protocol, Alice will get $(u_1, \ldots, u_n)$ and Bob will get $(v_1, \ldots, v_n)$. Bob will learn nothing about u_i and (α, β); Alice will learn nothing about v_i and the function $f_i(x, y)$.

3. Alice and Bob use the *Vector Dominance Protocol* to find out whether vector $A = (u_1, \ldots, u_n)$ dominates $B = (v_1, \ldots, v_n)$. According to the Vector Dominance Protocol, if A does not dominate B, no other information is disclosed.

Claim 1. *If $A = (u_1, \ldots, u_n)$ dominates $B = (v_1, \ldots, v_n)$, then the point $p = (\alpha, \beta)$ is inside the polygon; otherwise, the point is outside (or on the edge) of the polygon.*

5.2 Secure Two-Party Intersection Problem

Two polygons intersect if (1) one polygon is inside another, or (2) at least one edge of a polygon intersects with one edge of another polygon. As (1) can be decided using the Point-Inclusion Protocol, we only focus on (2).

We will first look at how the intersection problem could be solved in a straightforward way ($O(n^2)$) without worrying about the privacy concern. For the same reason

we decide not to use the more efficient $O(n)$ algorithm because of concern about partial information disclosure. The algorithm works as follows:

1. For each pair (e_i, e'_j), decide whether e_i intersects with e_j, where e_i is an edge of polygon A and e'_j is an edge of polygon B.

2. If there exists an edge $e_i \in A$ and an edge $e'_j \in B$, such that e_i intersects with e'_j, then A and B intersect.

We use $f_i(x, y), (x_i, y_i), (x'_i, y'_i)$, for $i = 1, \ldots, n_a$, to represent each edge of the polygon A, where $f_i(x, y)$ is the equation of the line containing that edge, (x_i, y_i) and (x_i, y_i) represents the two endpoints of the edge. We use $g_i(x, y)$, for $i = 1, \ldots, n_b$, to represent each edge of the polygon B.

The Protocol

During the testing of whether two edges intersect with each other, obviously, nobody should learn the result of each individual test; otherwise, he knows which of his edges intersects with the other party's polygon. In our scheme, Alice and Bob conduct these n^2 tests, but nobody knows the result of each individual test. Instead, they share the results of each test, namely each of them gets a seemingly-random piece of the result. One has to obtain both pieces to know the result of each test. At the end, all these shared pieces are put together in a way that only a single result is generated, to show only whether the two polygon boundaries intersect or not.

First, let us see how to conduct such a secure two-party test of the intersection. Assume Alice has a edge $f_1(x, y) = 0$, where $f_1(x, y) = a_1x + b_1y + c_1$, and $a_1 \geq 0$;

the two endpoints of the edge are (x_1, y_1) and (x'_1, y'_1). Bob has a line $f_2(x, y) = 0$, where $f_2(x, y) = a_2x + b_2y + c_2$, $a_2 \geq 0$; the two endpoints of the edge are (x_2, y_2) and (x'_2, y'_2). According to the geometries, f_1 and f_2 intersect if and only if f_1's two endpoints (x_1, y_1), (x'_1, y'_1) are on the different sides of f_2, and f_2's two endpoints (x_2, y_2) and (x'_2, y'_2) are on the different sides of f_1. In another words, f_1 and f_2 intersect if and only if one of the following expressions is true:

- $f_1(x_2, y_2) > 0 \wedge f_1(x'_2, y'_2) < 0 \wedge f_2(x_1, y_1) > 0 \wedge f_2(x'_1, y'_1) < 0$
- $f_1(x_2, y_2) > 0 \wedge f_1(x'_2, y'_2) < 0 \wedge f_2(x_1, y_1) < 0 \wedge f_2(x'_1, y'_1) > 0$
- $f_1(x_2, y_2) < 0 \wedge f_1(x'_2, y'_2) > 0 \wedge f_2(x_1, y_1) > 0 \wedge f_2(x'_1, y'_1) < 0$
- $f_1(x_2, y_2) < 0 \wedge f_1(x'_2, y'_2) > 0 \wedge f_2(x_1, y_1) < 0 \wedge f_2(x'_1, y'_1) > 0$

We cannot let either party know the results of $f_1(x_2, y_2)$, $f_1(x'_2, y'_2)$, $f_2(x_1, y_1)$, or $f_2(x'_1, y'_1)$ (in the following discussion, we will use $f(x, y)$ to represent any of these expressions). According to the Scalar Product Protocol, we can let Alice know the result of $f(x, y) + r$, and let Bob know r, where r is a random number generated by Bob. Therefore, nobody knows the actual value of $f(x, y)$, but Alice and Bob can still figure out whether $f(x, y) > 0$ by comparing $f(x, y) + r$ with r.

Let $u_1 = f_1(x_2, y_2) + r_1$, $u'_1 = f_1(x'_2, y'_2) + r'_1$, $u_2 = f_2(x_1, y_1) + r_2$, and $u'_2 = f_2(x'_1, y'_1) + r'_2$. Alice has (u_1, u'_1, u_2, u'_2) and Bob has (r_1, r'_1, r_2, r'_2). Then f_1 and f_2 intersect if and only if one of the following expressions is true:

- $u_1 > r_1 \wedge u'_1 < r'_1 \wedge u_2 > r_2 \wedge u'_2 < r'_2$

- $u_1 > r_1 \wedge u_1' < r_1' \wedge u_2 < r_2 \wedge u_2' > r_2'$
- $u_1 < r_1 \wedge u_1' > r_1' \wedge u_2 > r_2 \wedge u_2' < r_2'$
- $u_1 < r_1 \wedge u_1' > r_1' \wedge u_2 < r_2 \wedge u_2' > r_2'$

Our next step is to compute each of the above expressions. As before, nobody should learn the individual comparison results, only the aggregate. Let us use E to denote any one of the above expressions. Using the Vector Dominance Protocol, we can get Alice to know a random piece t, and Bob to know another random piece s, such that E is true if and only if $t = s$.

Now Alice has $4 * n^2$ numbers $(t_1, \ldots, t_{4n^2})$, Bob has $(s_1, \ldots, s_{4n^2})$. We want to know whether there exists an $i = 1, \ldots, 4n^2$, such that $t_i = s_i$. Although there are some other approaches to achieve this, we believe using the circuit evaluation protocol is efficient in this case, because the size of the circuit is small (linear in the number of the items). The security of the circuit evaluation protocol guarantees that only the final results–`yes` or `no`–will be disclosed; nobody learns any other information, such as how many t_i's equal to s_i's, and which $t_i = s_i$.

The following is the outline of the protocol:

1. Let $m = n_a * n_b$.
2. For each pair of edges, perform the following sub-protocol. Suppose the index of this edge pair is i, for $i = 1, \ldots, m$; suppose $(f, (x_1, y_1), (x_1', y_1')) \in A$ and $(g, (x_2, y_2), (x_2', y_2')) \in B$ are two edges.

(a) Using the scalar product protocol, Alice gets $U = (u_1, u_1', u_2, u_2')$, and Bob gets $R = (r_1, r_1', r_2, r_2')$, where $u_1 = f(x_2, y_2) + r_1$, $u_1' = f(x_2', y_2') + r_1'$, $u_2 = g(x_1, y_1) + r_2$, and $u_2' = g(x_1', y_1') + r_2'$.

(b) Using the Vector Dominance Protocol, Alice gets $t_{i,1}, t_{i,2}, t_{i,3}, t_{i,4}$, and Bob gets $s_{i,1}, s_{i,2}, s_{i,3}, s_{i,4}$.

3. Alice has $(t_{1,1}, t_{1,2}, t_{1,3}, t_{1,4}, \ldots, t_{m,1}, t_{m,2}, t_{m,3}, t_{m,4})$, and Bob has $(s_{1,1}, s_{1,2}, s_{1,3}, s_{1,4}, \ldots, s_{m,1}, s_{m,2}, s_{m,3}, s_{m,4})$. Alice and Bob uses circuit evaluation method to find out whether there exists $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, 4\}$, such that $t_{i,j} = s_{i,j}$.

5.3 Secure Two-Party Closest Pair Problem

Suppose Alice has m red points $a_1, \ldots, a_m$, and Bob has n blue points $b_1, \ldots, b_n$. To find the closest red-blue pair, Alice and Bob compute the distance between all of the $m * n$ red-blue pairs. We now show how they compute the distance $s_{i,j}$ between $a_i = (x_a, y_a)$ and $b_j = (x_b, y_b)$ without compromising privacy.

To preserve privacy, neither Alice nor Bob should learn the actual distance $s_{i,j}$, but they can "share" $s_{i,j}$, namely Alice knows $u_{i,j}$ and Bob knows $v_{i,j}$, such that $u_{i,j} = s_{i,j}^2 + v_{i,j}$. This can be achieved using the Scalar Product Protocol because of the following:

$$\begin{aligned} s_{i,j}^2 &= (x_a - x_b)^2 + (y_a - y_b)^2 = (x_a^2 + y_a^2) - (2x_a x_b + 2y_a y_b) + (x_b^2 + y_b^2) \\ &= (x_a^2 + y_a^2, -2x_a, -2y_a, 1) \cdot (1, x_b, y_b, x_b^2 + y_b^2) \end{aligned}$$

Therefore, after using the Scalar Product Protocol for mn times, Alice gets $U = \{u_{i,j} \mid i = 1, \ldots, m,\ j = 1, \ldots, n\}$, while Bob gets $V = \{v_{i,j} \mid i = 1, \ldots, m,\ j = 1, \ldots, n\}$; the next step is to find the minimum distance, namely Alice and Bob need to find α, β, such that

$$u_{\alpha,\beta} + v_{\alpha,\beta} = min\{u_{i,j} + v_{i,j} | u_{i,j} \in U, v_{i,j} \in V\}.$$

This can be achieved using FindMin Protocol, which will be described later.

Protocol 5.3.1. *(Secure Two-Party Closest Pair Protocol)*

Inputs: Alice has m red points $(a_1, \ldots, a_m)$; Bob has n blue points $(b_1, \ldots, b_n)$.

Outputs: Alice gets a_α, and Bob gets b_β, such that the distance between a_α and b_β is the smallest among all red point and blue point pairs.

1. Bob generates $m * n$ random numbers $V = \{v_{i,j} \mid i = 1, \ldots, m,\ j = 1, \ldots, n\}$.

2. For each red point and blue pair (a_i, b_j), Alice and Bob use the Scalar Product Protocol to compute $u_{i,j} = s_{i,j}^2 + v_{i,j}$, where $i = 1, \ldots, m,\ j = 1, \ldots, n$, and $s_{i,j}$ is the distance between a_i and b_j.

3. Alice and Bob conduct FindMin Protocol on inputs (U, V) to find α, β, such that $u_{\alpha,\beta} + v_{\alpha,\beta} = min\{u_{i,j} + v_{i,j} \mid u_{i,j} \in U,\ v_{i,j} \in V\}$.

5.3.1 Find Minimum Protocol (FindMin)

The goal is for Alice (but not Bob) to find the minimum element of vector $S = (s_1, \ldots, s_N)$ where $S = U - V$, Alice has $U = (u_1, \ldots, u_N)$, and Bob has $V =$

$(v_1, \ldots, v_N)$. Neither Alice nor Bob should learn order information about the s_i's (e.g., the s_i's sorted order, or even that $s_8 < s_3$). The following protocol is bad in that it reveals such order information between the s_i's to both Alice and Bob, but otherwise it does succeed in letting Alice know the minimum element value s_k, without revealing that s_k to Bob (although Bob does find out k).

Preliminary (bad) Protocol for Finding the Minimum

1. Alice mimics the standard minimum algorithm and, whenever that algorithm requires that s_i be compared to s_j, Alice gives Bob the ordered pair i, j and they do the following:

   (a) Alice computes $u = u_i - u_j = s_i - s_j + v_i - v_j$, and Bob computes $v = b_i - b_j$. (Note that $s_i > s_j$ iff $u < v$.)

   (b) They compare their respective values u and v using Yao's millionaire protocol [104].

   Note. This can be made an asymmetric version of Yao's millionaire problem, in which Alice but not Bob knows the outcome of the comparison, but Bob nevertheless finds out the outcome based on his observation of future pairs of indices that Alice will send him (if $s_i < s_j$ then index i will reappear but not index j).

2. Let k be the index of the minimum. Alice obtains the desired s_k by getting v_k from Bob, e.g. through 1-out-of-N oblivious transfer [82].

Note. We will explain below how we can prevent Bob from learning k through his observations of the pairs i, j that Alice sends him. Furthermore, oblivious transfer will then no longer be needed to hide k from Bob.

As noted earlier, the above protocol is bad because it reveals too much information during the computation of the minimum. Before running the above protocol Alice and Bob, using a random permutation π that is unknown to both Alice and Bob, could permute the entries of A, and in the same way permute the entries of B (in effect implicitly permuting the entries of S). But that would not be enough; either one of them could find the secret permutation by comparing the elements of their permuted vector to those of their original vector. To prevent this, they have to be also "blinded" by adding a random numbers r_i to each element and u_i and v_i, where r_i is also unknown to both Alice and Bob. The way we achieve the random permutation effect is by using a permutation π that is the composition of two random permutations π_A and π_B, where the former is known to Alice (but not Bob), and the latter is known to Bob (but not Alice). The "additive blinding" of u_i and v_i is by using an $r_i = r'_i + r''_i$ where r'_i is known to Alice (but not Bob), and r''_i is known to Bob (but not Alice). The permutation and blinding are done first by Alice, then again by Bob. To perform additive blinding, we need a special public key cryptosystem that has the following property: $E_k(x) \cdot E_k(y) = E_k(x + y)$. Such systems are called *homomorphic* cryptosystems and examples include the systems by Benaloh [15], Naccache and Stern [80], Okamoto and Uchiyama [84], and Paillier [85].

Protocol for Permuting and Additive Blinding

Inputs: Alice has U, Bob has V, Alice has a random permutation π_A and a random vector R' known to her but not to Bob.

Output: Bob obtains the set of N values of the form $v_i + r'_i$ in a permuted order according to π_A (which he does not know). (Note that Alice trivially has the set of N values of the form $U_i + r'_i$ in a permuted order according to π_A, because she knows π_A and R'.)

1. Alice and Bob each generate a key pair for a homomorphic public key system and exchange their public keys. In what follows $E_A(\cdot)$ denotes encryption with Alice's public key, and $D_A(\cdot)$ decryption with Alice's private key (similarly for $E_B(\cdot)$ and $D_B(\cdot)$).

2. Bob encrypts each entry $(v_1, \ldots, v_N)$ using his public key and sends $V' = (E_B(v_1), \ldots, E_B(v_N))$ to Alice.

3. Alice does the following:

   (a) She computes $\theta_i = v'_i \cdot E_B(r'_i) = E_B(v_i + r'_i)$, for $i = 1, \ldots, N$.

   (b) She permutes, according to π_A, the order of the θ_i's: Let V'' denote the vector of permuted θ_i's.

   (c) Alice sends V'' to Bob.

4. Bob decrypts the entries of V'', obtaining the set of N values of the form $v_i + r'_i$ in a permuted order according to π_A (which he does not know). Note that Alice

trivially has the set of N values of the form $u_i + r'_i$ in a permuted order according to π_A (which she does know).

Suppose Alice and Bob run the above "permute and blind" protocol once, and then follow it with the earlier-given preliminary ("bad") "find minimum" protocol in which the permuted-and-blinded data is used instead of the original vectors U and V. Unlike before, Bob now learns nothing of the relative orderings of the s_i's, and does not learn the index k for which s_k is minimum. But Alice still learns too much (because she knows both π_A and R'). This is easily fixed: After running the above "permute and blind" protocol, we run it again, on the already permuted-and-blinded data, but with roles of Alice and Bob interchanged. As a result, Alice and Bob end up with doubly-permuted and doubly-blinded data: Permuted according to π_A followed by π_B, the composition of which is unknown to both of them (because Alice knows only π_A and Bob knows only π_B), and additively blinded according to a random vector that is unknown to both of them because it is the sum of two random vectors the first of which is known only to Alice and the second only to Bob. In other words, Bob now has a random permutation π of the set of N values $v_i + \hat{r}_i$, and Alice has the same permutation π of the set of N values $u_i + \hat{r}_i$, and neither Alice nor Bob know π or $\hat{r}_i$. It is finally safe to use the earlier-given (and no longer bad!) preliminary protocol.

5.4 Protocol Efficiency

A Comparison to Generic Solutions.

The motivation of this research, i.e. designing specific solutions for each specific problems, is to reduce the communication cost. Therefore, in this section, we will compare the communication cost of our approach with that of the general solutions (the circuit evaluation approach). In what follows, d is the number of bits of any number in the inputs, c is the communication cost of 1-out-of-n Oblivious Transfer Protocol used in the circuit evaluation protocol.

For the Point-Inclusion problem, our protocol has communication complexity of $O(nd)$, which is independent of the size of the circuit. The communication complexity of the general solution is linear to the size of the circuit. Because such a circuit includes $2n$ multiplication, the size of circuit is at least $O(cnd^2)$ (circuit multiplication is quadratic in the length of the inputs).

For the Intersection problem, our protocol has communication complexity of $O(n^2d)$ while the general solution has $O(cn^2d^2)$ communication complexity.

For the Closest Pair problem, our protocol has communication complexity of $O(mnd)$, whereas the general solution has $O(cmnd^2)$ communication complexity.

5.5 Applications

The following two scenarios describe some potential applications of the problems we have discussed in this chapter.

1. Company A decided that expanding its market share in some region will be beneficial after costly market research; therefore A is planning to do this. However A is aware of that another competing company B is also planning to expand its market share in some region. Strategically, A and B do not want to compete against each other in the same region, so they want to know whether they have a region of overlap? Of course, they do not want to give away location information because not only does this information cost both companies a lot of money, but it can also cause significant damage to the company if it were disclosed to other parties. For example, a larger competitor can immediately occupy the market there before A or B even starts; or some real estate company can actually raise its price during the negotiation if they know A or B is interested in that location. Therefore, they need a way to solve the problem while maintaining the privacy of their locations.

2. A country decides to bomb a location x in some other country; however, A does not want to hurt its relationship with its friends, who might have some areas of interests in the bombing region: for example, those countries might have secret businesses, secret military bases, or secret agencies in that area. Obviously, A does not want to disclose the exact location of x to all of its friends, except the one who will definitely be hurt by this bombing; on the other hand, its friends do not want to disclose their secret areas to A either, unless they are in the target area. How could they solve this dilemma? If each secret area is represented by a secret polygon, the problem becomes how to decide whether A's secret point

is within B's polygon, where B represents one of the friend countries. If the point is not within the polygon, no information should be disclosed, including the information such as whether the location is at the west of the polygon, or within certain longitude or latitude. Basically it is "all-or-nothing": if one will be bombed, it knows all; otherwise it knows nothing.

5.6 Chapter Summary and Future Work

In this chapter, we have considered several secure two-party computational geometry problems and presented some preliminary work for solving such problems.

In the protocols for the geometry problems discussed in this chapter, we do not use the most efficient algorithms for them because of the concern about information disclosure. In our future work, we will study how to take advantage of those efficient solutions without degrading privacy.

6. SECURE TWO-PARTY STATISTICAL ANALYSIS AND PRIVACY-PRESERVING SURVEY PROBLEMS

In this chapter we investigate how various statistical analysis problems could be solved in a cooperative environment, where two parties need to conduct statistical analysis on the joint data set. We call the new problems *secure two-party* statistical analysis problems.

Basic statistic analysis operations consist of computing the mean value of a data set, the standard deviation, the correlation coefficient between two different features, the regression line and so on. If one knows the full data set, one can use the standard equations described in most of the fundamental statistics books to conduct the analysis. However, in a cooperative environment, one might need to conduct statistical analysis without being able to know the full data set because of privacy constraints. The following examples illustrate this kind of situation:

- A school wants to investigate the relationship between people's intelligence quotient (IQ) scores and their annual salaries. The school has its students' IQ scores, but does not have students' salary information; therefore the school needs to cooperate with companies that hire the students, but those companies are not willing to disclose the salary information. On the other hand, the school cannot give students' IQ scores to their employers either.

- Two retail companies A and B each have a data set about their own customers' buying behaviors. Both companies want to conduct statistical analysis on their joint data set for mutual benefit. As these two companies are competitors in the market, they do not want to disclose the detailed customers' information to the other company, but they feel comfortable disclosing only the aggregate information.

The standard statistical analysis methods cannot easily extend to solve the above problems; we need methods that support statistical analysis in a privacy-preserving manner. The goal of this chapter is to develop protocols for this type of cooperative statistical analysis.

There are two common ways of cooperation in practice. For example, suppose X and Y are two different features of a sample, and they are both used for a statistical analysis. In a cooperative environment, sometimes both cooperating parties can observe both X and Y features of the sample, while at some other time, one party can only observe X feature of the sample, and the other party can only observe Y feature of the same sample. The difficulties of cooperation in these situations are different. Therefore, based on whether the two cooperating parties could observe the same features of a sample or not, we formalized two different models for the secure two-party statistical analysis: the heterogeneous model and the homogeneous model. As we will show later, the solutions to these two different models are different.

An interesting and more general case of the homogeneous secure two-party statistical analysis is the privacy-preserving survey problem: To conduct a survey, an

interviewer sends out questions to many interviewees; each interviewee is supposed to send answers back to the interviewer, who then conducts certain statistical analysis on the samples collected. However, the interviewees do not want to disclose the answers to the interviewer or other interviewees because the answers contain private information. In this chapter, we present a solution to this problem.

6.1 Secure Two-Party Statistical Analysis Problem

6.1.1 Statistical Analysis Background

Without loss of generality, throughout this chapter, we will use a data set D of size n that only consists of two different features x and y, where $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$.

As a preliminary study on the topic of secure two-party statistical analysis, we only focus on several basic statistical analysis, which are reviewed in the the following:

- Mean Value: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.

- Correlation Coefficient between x and y: Correlation coefficient measures the strength of a linear relationship between x and y, namely the degree to which larger x values go with larger y values and smaller x values go with smaller y values. Correlation coefficient r is computed using the following equation:

$$\begin{aligned} r &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \\ &= \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sqrt{(\sum_{i=1}^{n} x_i^2 - n\bar{x}^2)(\sum_{i=1}^{n} y_i^2 - n\bar{y}^2)}}. \end{aligned}$$

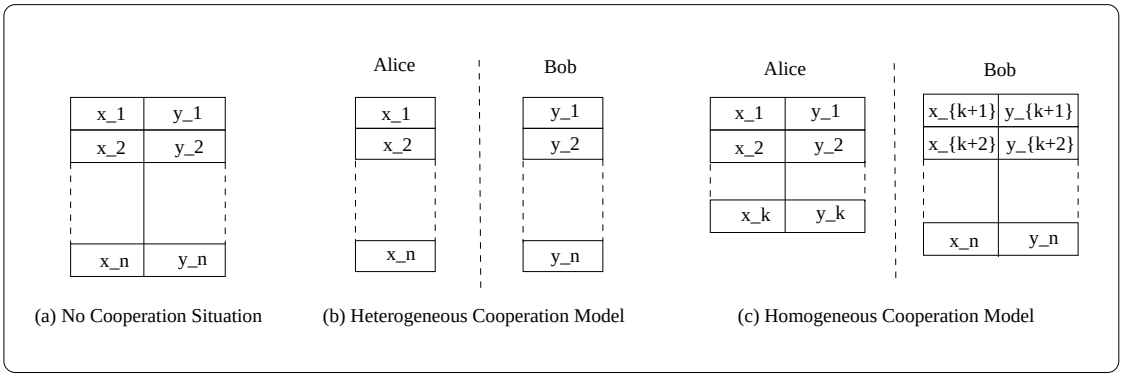

Figure 6.1. Two Models of Cooperation

- Linear Regression Line: The purpose of linear regression is to find the line that comes closest to your data. More precisely, the linear regression program finds values for the slope and intercept that define the line that minimizes the sum of the squares of the vertical distances between the points and the line. The linear regression line is represented by the following equation: $y = bx + (\bar{y} - b\bar{x})$, where

$$b = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2}.$$

6.1.2 Two Models of Cooperation

There are many ways two parties could cooperate in performing statistical analysis; Figure 6.1 describes two ways of cooperation that are common in practice. The first one is the heterogeneous cooperation model (Figure 6.1.b). In this model, each party holds different features of a data set. For example, if the whole data set consists

of employees' salaries and ages, in a heterogeneous model, Alice could hold the salary information while Bob holds the age information.

The second form of cooperation is the homogeneous cooperation model (Figure 6.1.c). In this model, both party hold the same features, but each party holds a different subset of the data set. For instance, in a homogeneous model, Alice could hold department A's employee information while Bob holds department B's employee information.

Both of the above cooperation models are quite common in practice. In this chapter, we have formally defined secure two-party statistical analysis problems corresponding to these cooperation models, and have developed protocols for those problems.

6.1.3 Heterogeneous Model

Problem 6.1.1. *(Secure Two-Party Statistical Analysis Problem in Heterogeneous Model)* Alice has a data set $D_1 = (x_1, \ldots, x_n)$, and Bob has another data set $D_2 = (y_1, \ldots, y_n)$, where x_i is the value of variable x, and y_i is the corresponding value of variable y. Alice and Bob want to find out the following:

1. *correlation coefficient* r between x and y.

2. *regression line* $y = bx + (\bar{y} - b\bar{x})$.

Correlation Coefficient Let $u = \sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}$, and $v = \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}$. To compute the correlation coefficient r, we have the following equations:

$$
\begin{aligned}
r &= \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2\sum_{i=1}^{n}(y_i-\bar{y})^2}} \\
&= \sum_{i=1}^{n}\frac{(x_i-\bar{x})}{u}\frac{(y_i-\bar{y})}{v} \\
&= (\frac{x_1-\bar{x}}{u},\ldots,\frac{x_n-\bar{x}}{u})\cdot(\frac{y_1-\bar{y}}{v},\ldots,\frac{y_n-\bar{y}}{v}).
\end{aligned}
$$

This indicates that the task of computing the correlation coefficient is reduced to a secure two-party scalar product problem. It can be computed using the Scalar Product Protocol (Protocol 3.2.2 or 3.2.3).

Linear Regression Line Let $w = \sum_{i=1}^{n} x_i^2 - n\bar{x}^2$. Because computing w only requires the value of variable x, it can be calculated by Alice alone. Therefore, we can use the following equations to compute the slope of the linear regression line:

$$
\begin{aligned}
b &= \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}x_i^2 - n\bar{x}^2} \\
&= (\frac{x_1-\bar{x}}{w},\ldots,\frac{x_n-\bar{x}}{w})\cdot(y_1-\bar{y},\ldots,y_n-\bar{y}).
\end{aligned}
$$

This indicates that the task of computing b is also reduced to a secure two-party scalar product problem, and thus can be solved using the Scalar Product Protocol (Protocol 3.2.2 or 3.2.3). The details of the protocol are described in the following:

Protocol 6.1.1. *(Secure Two-Party Statistical Analysis Protocol in Heterogeneous Model)*

Inputs: Alice has a data set $D_1 = (x_1, \ldots, x_n)$, and Bob has another data set $D_2 = (y_1, \ldots, y_n)$.

Outputs: Alice and Bob get r and b.

1. Alice computes $\bar{x}$, $u = \sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}$, and $w = \sum_{i=1}^{n} x_i^2 - n\bar{x}^2$.

2. Bob computes $\bar{y}$ and $v = \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}$.

3. Alice and Bob use the Scalar Product Protocol (Protocol 3.2.2 or 3.2.3) to compute

$$\begin{aligned} r &= (\frac{x_1 - \bar{x}}{u}, \ldots, \frac{x_n - \bar{x}}{u}) \cdot (\frac{y_1 - \bar{y}}{v}, \ldots, \frac{y_n - \bar{y}}{v}) \\ b &= (\frac{x_1 - \bar{x}}{w}, \ldots, \frac{x_n - \bar{x}}{w}) \cdot (y_1 - \bar{y}, \ldots, y_n - \bar{y}). \end{aligned}$$

6.1.4 Homogeneous Model

Problem 6.1.2. *(Secure Two-Party Statistical Analysis Problem in Homogeneous Model)* Alice has a data set $D_1 = ((x_1, y_1), \ldots, (x_k, y_k))$, and Bob has another data set $D_2 = ((x_{k+1}, y_{k+1}), \ldots, (x_n, y_n))$, where x_i is the value of variable x, and y_i is the corresponding value of variable y. Alice and Bob want to find out the following:

1. *mean value* $\bar{x}$ (resp., $\bar{y}$).

2. *correlation coefficient* r between x and y.

3. *regression line* $y = bx + (\bar{y} - b\bar{x})$.

Let us first consider the above problem under the following privacy constraint:

> **Privacy Constraint A:** Alice does not want to disclose the information about D_1 other than the aggregate information including $\sum_{i=1}^{k} x_i$, $\sum_{i=1}^{k} x_i^2$, $\sum_{i=1}^{k} y_i$, $\sum_{i=1}^{k} y_i^2$, and $\sum_{i=1}^{k} x_i y_i$. Accordingly, Bob does not want to disclose the information about D_2 other than the aggregate information including $\sum_{i=k+1}^{n} x_i$, $\sum_{i=k+1}^{n} x_i^2$, $\sum_{i=k+1}^{n} y_i$, $\sum_{i=k+1}^{n} y_i^2$, and $\sum_{i=k+1}^{n} x_i y_i$.

Under Privacy Constraint A, computing mean value is trivial because both parties know $\sum_{i=1}^{n} x_i$ and $\sum_{i=1}^{n} y_i$. After getting $\bar{x}$ and $\bar{y}$, computing the correlation coefficient and the linear regression line is straightforward according to the following equations:

$$
\begin{aligned}
r &= \frac{(\sum_{i=1}^{k} x_i \cdot y_i + \sum_{i=k+1}^{n} x_i \cdot y_i) - n * \bar{x}\bar{y}}{\sqrt{(\sum_{i=1}^{k} x_i^2 + \sum_{i=k+1}^{n} x_i^2) - n\bar{x}^2}\sqrt{(\sum_{i=1}^{k} y_i^2 + \sum_{i=k+1}^{n} y_i^2) - n\bar{y}^2}} \\
b &= \frac{(\sum_{i=1}^{k} x_i \cdot y_i + \sum_{i=k+1}^{n} x_i \cdot y_i) - n * \bar{x}\bar{y}}{(\sum_{i=1}^{k} x_i^2 + \sum_{i=k+1}^{n} x_i^2) - n\bar{x}^2}.
\end{aligned}
$$

Now let us consider the same problem under a more strict privacy constraint:

> **Privacy Constraint B:** Alice and Bob do not want to disclose too much information about their data; more specifically, they do not want to disclose any more information than what can be derived from $\bar{x}$, $\bar{y}$, r and b. This implies that Alice can disclose $\sum_{i=1}^{k} x_i$ and $\sum_{i=1}^{k} y_i$ to Bob, and

Bob can disclose $\sum_{i=k+1}^{n} x_i$ and $\sum_{i=k+1}^{n} y_i$ to Alice because those can be derived from $\bar{x}$ and $\bar{y}$.

Under this privacy constraint, computing the mean value is still trivial, but computing the correlation coefficient r and the linear regression line is not. In what follows, we demonstrate how to compute r (the linear regression line can be computed similarly).

Let $a_1 = \sum_{i=1}^{k} x_i \cdot y_i - k\bar{x}\bar{y}$, $b_1 = \sum_{i=k+1}^{n} x_i \cdot y_i - (n-k)\bar{x}\bar{y}$, $a_2 = \sum_{i=1}^{k} x_i^2 - k\bar{x}^2$, $b_2 = \sum_{i=k+1}^{n} x_i^2 - (n-k)\bar{x}^2$, $a_3 = \sum_{i=1}^{k} y_i^2 - k\bar{y}^2$, and $b_3 = \sum_{i=k+1}^{n} y_i^2 - (n-k)\bar{y}^2$. Note that a_i is only known to Alice, and b_i is only known to Bob. We have

$$\begin{aligned} r^2 &= \frac{(a_1+b_1)^2}{(a_2+b_2)(a_3+b_3)} \\ &= \frac{(a_1^2+2a_1b_1+b_1^2)}{(a_2a_3+b_2a_3+a_2b_3+b_2b_3)}. \end{aligned}$$

By using the Scalar Product Protocol, we can let Alice learn u_1 and u_2, and let Bob learn v_1 and v_2, where $u_1+v_1 = a_1^2+2a_1b_1+b_1^2$ and $u_2+v_2 = a_2a_3+b_2a_3+a_2b_3+b_2b_3$. Now the question becomes how to compute $\frac{u_1+v_1}{u_2+v_2}$.

Problem 6.1.3. *(Division Problem)* Alice has u_1 and u_2; Bob has v_1 and v_2. Alice and Bob want to compute $z = \frac{u_1+v_1}{u_2+v_2}$. Alice should not learn v_1 or v_2; Bob should not learn u_1 or u_2.

In the following protocol, we first let Bob generate two random numbers r_1 and r_2; then we let Alice (only Alice) get the result of $z_1 = r_1(u_1+v_1)$, $z_2 = r_2(u_2+v_2)$,

and $r = \frac{r_2}{r_1}$. Therefore, Alice can compute $z = \frac{rz_1}{z_2} = \frac{u_1+v_1}{u_2+v_2}$. If r_1 and r_2 are both real numbers, Alice could not learn v_1 (resp., v_2) from z_1 (resp., z_2).

Protocol 6.1.2. *(Division Protocol)*

Input: Alice has u_1 and u_2; Bob has v_1 and v_2.

Output: Alice and Bob both get the result of $z = \frac{u_1+v_1}{u_2+v_2}$.

1. Bob generates two non-zero random numbers r_1 and r_2, and sends $r = \frac{r_2}{r_1}$ to Alice.

2. Alice and Bob use the Scalar Product Protocol on $(u_1, 1)$ and (r_1, r_1v_1) to get $z_1 = r_1(u_1 + v_1)$.

3. Alice and Bob use the Scalar Product Protocol on $(u_2, 1)$ and (r_2, r_2v_2) to get $z_2 = r_2(u_2 + v_2)$.

4. Alice computes $z = r\frac{z_1}{z_2} = \frac{u_1+v_1}{u_2+v_2}$, and sends it to Bob.

Protocol 6.1.3. *(Secure Two-Party Statistical Analysis Protocol in Homogeneous Model)*

Inputs: Alice has a data set $D_1 = ((x_1, y_1), \ldots, (x_k, y_k))$, Bob has another data set $D_2 = ((x_{k+1}, y_{k+1}), \ldots, (x_n, y_n))$.

Outputs: Alice and Bob both get $\bar{x}$, $\bar{y}$, r and b.

1. Alice sends $\sum_{i=1}^{k} x_i$ and $\sum_{i=1}^{k} y_i$ to Bob.

2. Bob sends $\sum_{i=k+1}^{n} x_i$ and $\sum_{i=k+1}^{n} y_i$ to Alice.

3. Alice and Bob both compute $\bar{x}$ and $\bar{y}$.

4. Alice computes $a_1 = \sum_{i=1}^{k} x_i \cdot y_i - k\bar{x}\bar{y}$, $a_2 = \sum_{i=1}^{k} x_i^2 - k\bar{x}^2$, and $a_3 = \sum_{i=1}^{k} y_i^2 - k\bar{y}^2$.

5. Bob computes $b_1 = \sum_{i=k+1}^{n} x_i \cdot y_i - (n-k)\bar{x}\bar{y}$, $b_2 = \sum_{i=k+1}^{n} x_i^2 - (n-k)\bar{x}^2$, and $b_3 = \sum_{i=k+1}^{n} y_i^2 - (n-k)\bar{y}^2$.

6. Using the Scalar Product Protocol, Alice gets u_1 and u_2, while Bob gets v_1 and v_2, where $u_1 + v_1 = a_1^2 + 2a_1b_1 + b_1^2$ and $u_2 + v_2 = a_2a_3 + b_2a_3 + a_2b_3 + b_2b_3$.

7. Using Division Protocol, Alice and Bob get $r^2 = \frac{u_1+v_1}{u_2+v_2}$ and $b = \frac{a_1+b_1}{a_2+b_2}$.

6.2 Privacy-Preserving Survey Problem

Survey is an important technique to collect information for analysis purposes. Sometimes, the survey contains sensitive questions, such as "how many security break-in's did your company have in the last month," "please choose from the following the most common successful attacks your company is subject to, ..." and "please tell us the numbers of machines in your company that are running Windows NT, Linux, and Solaris, respectively." Interviewees who want to participate in the survey also want to keep the answer to these questions confidential.

Currently, in these survey situations, two common strategies are adopted: The first approach is to assume the trustworthiness of the interviewer, or to assume the existence of a trusted third party. The second commonly used approach is to use an anonymous technique: each interviewee sends back his answers anonymously to the

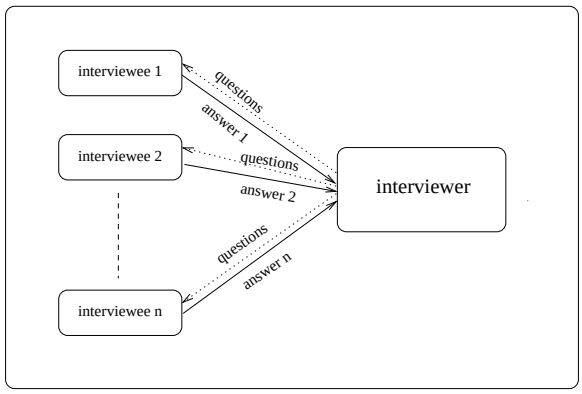

Figure 6.2. Survey Models

interviewer using a hard copy or using any of the anonymous communication protocol proposed in the literature [101, 89]. However, this straightforward use of anonymous reply does not guarantee that the results come from the intended interviewees; anyone else who knows the ongoing survey can make arbitrary answers and anonymously send it back to the interviewer. This renders the results more or less untrusted. Of course, more sophisticated approaches could be used to solve the above drawback, but we suspect that it might require communications among the interviewees–an undesirable requirement in the real life.

In this dissertation, we have developed a method (the data perturbation technique) that can support such a type of privacy-preserving survey. Although a similar technique has been proposed in the statistical community [102, 103, 59, 86] before, the primary efforts are focusing on computing the mean of a data set. Our method extend the technique to support various other standard statistical analysis operations, such as computing standard deviation, correlation coefficient, and linear regression line.

Problem 6.2.1. *(Privacy-Preserving Survey Problem)* To conduct a survey, an interviewer sends out questions to many interviewees; each interviewee is supposed to send answers back to the interviewer (assume the answers are quantitative answers). The interviewer, after collecting all the answers, wants to conduct certain statistical analysis, such as calculating the sum, mean, standard derivation, correlation and regression. Throughout the whole survey process, the following constraints should be satisfied:

1. The interviewer should not learn the exact answers of each interviewee.

2. No interviewee should learn the exact answers of other interviewees.

3. Interviewees are not supposed to communicate with each other.

The first two constraints guarantee the privacy of each interviewee's answer. The last constraint is necessary in the real world because it is undesirable to have the participants communicate with each other during a survey because of the scalability problem and the anonymity issue: in the anonymous survey situation, the participants' identities are not supposed to be revealed. Figure 6.2 describes such a survey process.

The Protocol

Observation 6.2.1. If every interviewee generates a random number according to certain pre-agreed random number generation parameters, the mean of these random numbers tends to be a pre-determined number (the expected value) if the number of

interviewees is large enough. For example, if random numbers are generated uniformly in $[-m, m]$, the mean of these random numbers will be 0.

Based on this observation, each interviewee could add a random number to its actual answer, thus preventing the interviewer from knowing the actual answer; however, this does not prevent the interviewer from estimating the mean value of the answers from all interviewees.

For example, let x_i be the actual answer from the ith interviewee, for $i = 1, \ldots, n$, and r_i be the random number generated by the interviewee; let $\bar{x}$ be the mean value of x_i's and E be the expected value of r_i's. After the data collection, the interviewer will get:

$$\frac{1}{n}\sum_{i=1}^{n}(x_i + r_i) = \frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n}\sum_{i=1}^{n} r_i \approx \bar{x} + E.$$

To compute the standard deviation in addition to the mean value of x_i's, the interviewer also needs to get the value of $\sum_{i=1}^{n} x_i^2$. Using the same data perturbation technique, each interviewee can make it possible for the interviewer to compute $\sum_{i=1}^{n} x_i^2$ by sending back its x_i^2 disguised by a random number.

Similar ways could be used to compute the correlation coefficient and linear regression line if the answer consist of two numbers. For example, let (x_i, y_i) be the actual answer for the ith interviewee, according to the equation for computing the correlation coefficient, the interviewer can compute r if it knows $\sum_{i=1}^{n} x_i^2$, $\sum_{i=1}^{n} y_i^2$, $\sum_{i=1}^{n} x_i \cdot y_i$, $\bar{x}$ and $\bar{y}$, all of which can be obtained using the data perturbation tech-

nique without disclosing the actual values of x_i and y_i. Moreover, knowing these numbers also allows the interviewer to compute the linear regression line.

In what follows, we assume there are two questions asked by the interviewer, and each interviewee sends back a tuple (x, y) to the interviewer, where x and y are two numbers. The protocol works for a one-question situation by ignoring y; it can be straightforwardly extended to an n-question scenario as well.

Protocol 6.2.1. *(Privacy-Preserving Survey Protocol)*

1. The interviewer sends the questions and parameters for the random number generation to the interviewees (suppose the expected value of the random number is E).

2. For the interviewee i, if the exact answers to the questions is (x_i, y_i), it generates five random numbers $r_{i,1}$, $r_{i,2}$, $r_{i,3}$, $r_{i,4}$ and $r_{i,5}$ according to the parameters from the interviewer; then the interviewee sends $x_i + r_{i,1}$, $x_i^2 + r_{i,2}$, $y_i + r_{i,3}$, $y_i^2 + r_{i,4}$, and $x_i \cdot y_i + r_{i,5}$ to the interviewer.

3. The interviewer can conduct the following statistical analysis:

   (a) sum of x: $s_x = \sum_{i=1}^{n} x_i \approx \sum_{i=1}^{n}(x_i + r_{i,1}) - nE$.

   (b) sum of y: $s_y = \sum_{i=1}^{n} y_i \approx \sum_{i=1}^{n}(y_i + r_{i,3}) - nE$.

   (c) mean: $\bar{x} \approx \frac{s_x}{n}$ and $\bar{y} \approx \frac{s_y}{n}$.

   (d) sum of x^2: $s_{x^2} = \sum_{i=1}^{n} x_i^2 \approx \sum_{i=1}^{n}(x_i^2 + r_{i,2}) - nE$.

   (e) sum of y^2: $s_{y^2} = \sum_{i=1}^{n} y_i^2 \approx \sum_{i=1}^{n}(y_i^2 + r_{i,4}) - nE$.

(f) sum of xy: $s_{xy} = \sum_{i=1}^{n} x_i \cdot y_i \approx \sum_{i=1}^{n} (x_i \cdot y_i + r_{i,5}) - nE$.

(g) standard deviation: $\sigma \approx \sqrt{\frac{s_{x^2} - n\bar{x}}{n-1}}$.

(h) correlation coefficient: $r \approx \frac{s_{xy} - n * \bar{x}\bar{y}}{\sqrt{(s_{x^2} - n\bar{x}^2)(s_{y^2} - n\bar{y}^2)}}$.

(i) regression line: $y = bx + (\bar{y} - b\bar{x})$, where $b \approx \frac{s_{xy} - \frac{s_x s_y}{n}}{s_{x^2} - \frac{s_x^2}{n}}$.

6.3 Chapter Summary and Future Work

In this chapter, we have studied the problem of how to conduct the statistical analysis in a cooperative environment where none of the cooperating parties wants to disclose its private data to the other party. Our preliminary work has shown that these problems, the secure two-party statistical analysis problem and the privacy-preserving survey problem could be solved in a way more efficient than the general circuit evaluation approach.

Apart from those basic statistical analysis computations studied in this chapter, many other types of statistical analysis are also used in practice. A future direction would be to study more complicated statistical analysis computations, such as nonlinear regression, variance analysis and so on. Furthermore, we could also study, under the same secure two-party context, various data analysis computations other than the statistical analysis. Data mining is an interesting and more complicated data analysis computation that is worth of study.

7. SECURE REMOTE DATABASE QUERY WITH APPROXIMATE MATCHING

In this chapter, we study the following problem:

Secure Database Query (SDQ) Problem: Alice has a string q, and Bob has a database of strings $T = \{t_1, \ldots, t_N\}$; Alice wants to know whether there exists a string t_i in Bob's database that "matches" q. The "match" could be an exact match or an approximate (closest) match. The problem is how to design a protocol that accomplishes this task without revealing to Bob Alice's secret query q or the response to that query.

Because of the practical importance and the lack of studies on approximate pattern matching problems under the SDQ context, our work particularly focuses on approximate pattern matching.

Unlike exact pattern matching that produces "yes" and "no" answers, approximate pattern matching measures the difference between the two targets, and produces a *score* to indicate how different the two targets are. The metrics used to measure the difference usually are heuristic and are application-dependent. For example, in image template matching [58, 64], $\sum_{i=1}^{n}(a_i - b_i)^2$ and $\sum_{i=1}^{n}|a_i - b_i|$ are often used to measure the difference between two sequences a and b. In DNA sequence matching [60], *edit distance* [6, 34] makes more sense than the above measurements; *edit dis-*

tance measures the cost of transforming one given sequence to another given sequence, and its special case, *longest common subsequence* is used to measure how similar two sequences are.

Solving approximate pattern matching problems within the SDQ framework is quite a nontrivial task. Consider the $\sum_{i=1}^{n} |a_i - b_i|$ metric as an example. The known PIR (Private Information Retrieval) techniques [26, 24, 62, 37, 73, 22, 52, 51] can be used by Alice to efficiently access each individual b_i without revealing to Bob anything about which b_i Alice accessed, but doing this for each individual b_i and then calculating $\sum_{i=1}^{n} |a_i - b_i|$ violates the requirement that Alice should know the total score $\sum_{i=1}^{n} |a_i - b_i|$ *without knowing anything other than that score*, i.e., without learning anything about the individual b_i values. Using a general secure multi-party computation protocol typically does not lead to an efficient solution. The goals of our research, and the results presented in this chapter, are finding efficient ways to do such approximate pattern matchings without disclosing private information.

The practical motivations of remote database access do not all point to the model we described in the above SDQ formulation. For example, in some situations, Bob's database contains his own private data; but in some other situations, Bob's database contains Alice's suitably disguised private data. Based on these variants of the problems, we have investigated three SDQ models, and defined a class of SDQ problems for each model according to the metrics we use for approximate pattern matching. Of course the difficulties of the problems are not the same for the different metrics, and in this dissertation we have solved a subset of those problems. A summary of our

results is listed below (the results are stated more precisely in Section 7.2, and the models are defined in Section 7.1 – in the meantime see Figure 7.1 in that section for a summary of each model).

- For the Private Information Matching Model, we have a solution to approximate pattern matching based on the $\sum_{i=1}^{n}(a_i - b_i)^2$ metric with $O(n * N)$ communication cost, where n is the length of each string and N is the number of strings in the database.

- For the Private Information Matching Model, we also have a solution to the approximate pattern matching based on the $\sum_{i=1}^{n}|a_i - b_i|$ metric using a Monte Carlo technique; the solution gives an estimated result, and it has $O(n * W * N)$ communication cost, where W is a parameter that affects the accuracy of the estimate.

- For the Private Information Matching Model, if we assume that the alphabet is known to the involved parties and its size is finite, we have a solution to approximate pattern matching based on general $\sum_{i=1}^{n} f(a_i, b_i)$ metrics, hence the solutions for the special cases of $\sum_{i=1}^{n}|a_i - b_i|$, $\sum_{i=1}^{n}(a_i - b_i)^2$, and $\sum_{i=1}^{n}\delta(a_i, b_i)$ (where $\delta(x, y)$ is 1 if $x = y$ and 0 otherwise). These solutions have $O(m * n * N)$ communication cost, where m is the number of the symbols in the alphabet. In many cases, m is small. For instance, m is four in DNA databases.

- For the Secure Storage Outsourcing Model, we have a practical solution to approximate pattern matching based on the $\sum_{i=1}^{n}(a_i - b_i)^2$ metric. The solution is practical because its $O(n)$ communication cost does not depend on N.

- For the Secure Storage Outsourcing and Computation Model, we also have a practical solution to approximate pattern matching based on the $\sum_{i=1}^{n}(a_i - b_i)^2$ metric. This solution is practical because of its communication cost is $O(n^2)$.

Motivation

Why do we care about the privacy of a database query? In the example used earlier in this section, if a match is found in the database, Bob immediately knows that Alice has such a disease; even worse, after receiving Alice's DNA sequence, Bob can derive additional information about Alice, such as other health problems that Alice might have. If Bob is not trustworthy, Bob could disclose the information about Alice to other parties, and Alice might have difficulty getting employment, insurance, credit, etc. But even if Alice trusts Bob, and Bob has no intention of disclosing Alice's private information, Bob himself might prefer that Alice's query be kept private, out of liability concerns: If Bob knows Alice's DNA information, and that information is accidentally disclosed (perhaps by a disgruntled employee of Bob's, or after a system break-in), Bob might face an expensive lawsuit from Alice. From this perspective, a trusted Bob will actually prefer not to know either Alice's query or its response.

With the growth of the Internet, more and more e-commerce transactions like the above will take place. There are already DNA pattern databases, diseases databases,

patent databases, and in the future we may see many more commercial databases and the related database access services, such as fingerprint databases, signature databases, medical record databases, and many more. Privacy will be a major issue, and assuming the trustworthiness of the service providers, as is done today, is risky; therefore protocols that can support remote access operations while protecting the client's privacy are of growing importance.

One of the fundamental operations behind the queries described in the examples above is pattern matching. Therefore, the basic problem that we face is how to conduct pattern matching operations at the server side while the server has no knowledge of the client's actual query (or the response to it). In some database access situations, exact pattern matching is used, such as query by name, query by social security number, etc. However, in many other situations, exact pattern matching is unrealistic. For instance, in fingerprint matching, even if two fingerprints come from the same finger, they are unlikely to be exactly the same because there is some information loss in the process of deriving an electronic form (usually a complex data structure of features) from a raw fingerprint image. Similarly in voice, face, and DNA matching; in these and many other situations, exact matching is not expected and some form of approximate pattern matching is more useful.

Why Not Use An Anonymous Communication Protocol

Anonymous communication protocols [89, 101] were designed to achieve somewhat related goals, so why not use them? Anonymity techniques help to hide the identity of the information sender, rather than the information being sent. For example, when

people browse the web, they can use anonymous communication techniques to keep their identities secret, but the web query usually is not secret because the web server has to know the query to send a reply back. In situations where the identity of the information sender needs to be protected, anonymous communication protocols are appropriate. However, there are situations where anonymous communication protocols cannot replace secure multi-party computation protocols. First, certain types of information intrinsically reveal the identity of someone related to the information (e.g., social security number). Second in some situations, it is the information itself that needs to be protected, not the identity of the information sender. For instance, if Alice has an invention, she has to search if such an invention is new before she files for a patent. When conducting the query, Alice may want to keep the query private (perhaps to avoid part of her idea being stolen by people who have access to her query); she does not care whether her identity is revealed. Third in certain situations, one has to be a registered member to use the database access service; this makes hiding a user's identity difficult because the user has to register and login first, which might already disclose her identity.

Furthermore, most of the known practical anonymous protocols, such as Crowds [89], Onion routing [101] and `anonymizer.com`, use one or several *trusted* third parties. In our secure multi-party computation protocols, we do not use a trusted third party; when a third party is used, we generally assume that the third party is not trusted, and should learn nothing about either Alice's query, or Bob's data, or the response to the query.

Therefore anonymity does not totally solve our problems, and cannot replace secure multi-party computation. Rather, by combining anonymity techniques with secure multi-party computation techniques, one can achieve better overall privacy more efficiently.

Private Information Retrieval

Among various multi-party computation problems, the Private Information Retrieval (PIR) problem has been widely studied; it is also the problem most related to what we present in this chapter (although here we use none of the elegant techniques for PIR that are found in the literature, for reasons we explained earlier in this chapter). The PIR problem consists of devising a protocol involving a user and a database server, each having a secret input. The database's secret input is called the *data string*, an n-bit string $B = b_1 b_2 \ldots b_n$. The user's secret input is an integer i between 1 and n. The protocol should enable the user to learn b_i in a communication-efficient way and at the same time hide i from the database. The trivial solution has an $O(n)$ communication complexity. Much work has been done for reducing this communication complexity [26, 24, 62, 37, 73, 22, 52, 51].

Chor, Gilboa and Naor point out that a major drawback of all known PIR schemes is the assumption that the user knows the *physical address* of the sought item [25], whereas in the current database query scenario the user typically holds a keyword and the database internally converts this keyword into a physical address. To solve this problem, they propose a scheme to privately access data by keywords [25]. The

difference between the problem studied in their paper and the problems in this chapter is that we extend the problem to cover approximate pattern matching.

Song, Wagner and Perrig propose a scheme to conduct searches on encrypted data [100]. In that framework, Alice has a database, and she has to store the database in a server controlled by Bob; how could Alice query her database without letting Bob know the contents of the database or the query? Here we primarily focus on extending the problem to also cover approximate pattern matching.

7.1 Framework

7.1.1 Models

Remote database access has many variants. In some e-commerce models, Bob hosts his own private database; but in some other models, Bob hosts Alice's (encrypted/disguised) database while supporting queries from Alice and other customers, in which case Bob should know neither the database nor the queries.

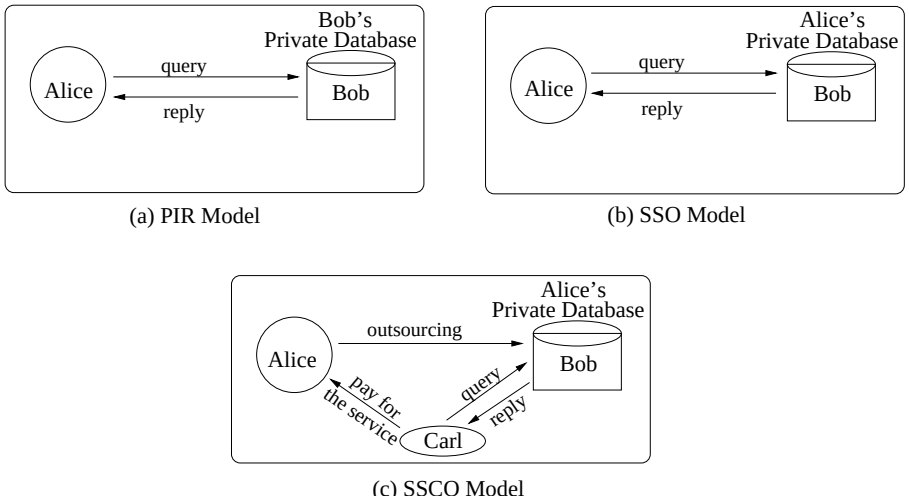

Figure 7.1. Secure Remote Database Query Models

From the various ways that remote database access is conducted, we distinguish three different e-commerce models, all of which require customers' privacy:

- PIM: Private Information Matching Model (Figure 7.1.a)
- SSO: Secure Storage Outsourcing Model (Figure 7.1.b).
- SSCO: Secure Storage and Computing Outsourcing Model (Figure 7.1.c).

For the sake of convenience, we will use $Match()$ to represent the pattern matching function, which includes both exact pattern matching and approximate pattern matching.

Problem 7.1.1. *(Private Information Matching Problem (PIM))*

Alice has a string x, and Bob has a database of strings $T = \{t_1, \ldots, t_N\}$; Alice wants to know the result of $Match(x, T)$. Because of the privacy concern, Alice does not want Bob to know the query x or the response to the query; Bob does not want Alice to know any string in the database except for what can be derived from the reply. Furthermore, Bob wants to make money from providing such a service, therefore Alice should not be able to conduct the querying by herself; in other words, every time Alice wants to perform such a query, she has to contact Bob, otherwise she cannot get the correct answer.

Problem 7.1.2. *(Secure Storage Outsourcing Problem (SSO))*

Alice has a database of strings $T = \{t_1, \ldots, t_N\}$, but she does not have enough storage for the large database, so she outsources her database (suitably disguised– more on this later) to Bob, who provides enough storage for Alice. Furthermore, from

time to time, Alice needs to query her database and retrieves the information that matches her query, i.e., Alice wants to know $Match(x, T)$ for her query x. As usual, Alice wants to keep the contents of both the database and the query secret from Bob.

Problem 7.1.3. *(Secure Storage and Computing Outsourcing Problem (SSCO))*

The SSCO problem is an extension of the SSO problem. Whereas only Alice queries her database in the SSO problem, in the SSCO model the database will also be queried by other clients of Alice. More specifically, in the SSCO model, Alice outsources her database to Bob, and she wants the database to be available to anyone who is willing to pay her for the database access service. When a client accesses the database, neither Alice nor Bob should know the contents of the query. Moreover, Alice wants to charge the clients for each query they have submitted, so the client should not be able to get the correct query result if Alice is not aware of the query's existence.

As Bob can pretend to be a client, the solutions of the SSCO problem should be secure even if Bob can collude against Alice with any client. However, the SSO problem does not have such a concern because the only client is Alice herself.

7.1.2 Notation

For each model, there is a family of problems. We will use the following notations to represents each specific problem:

- M/Exact: Exact Pattern Matching problem in model M.
- M/Approx: Approximate Pattern Matching problem in model M.

- M/Approx/f: use $\sum_{k=1}^{n} f(a_k, b_k)$ metric to measure the distance between two strings, where f is a general function.
- M/Approx/δ: use $\sum_{k=1}^{n} \delta(a_k, b_k)$ metric to measure the distance between two strings, where δ is the Kronecker symbol: $\delta(x, y) = 0$ if and only if $x = y$ and 1 otherwise.
- M/Approx/Abs: use $\sum_{k=1}^{n} |a_k - b_k|$ metric to measure the distance between two strings.
- M/Approx/Squ: use $\sum_{k=1}^{n} (a_k - b_k)^2$ metric to measure the distance between two strings.
- M/Approx/Edit:
  - M/Approx/Edit/String: use the string editing criterion [34] to measure the distance between two strings.
  - M/Approx/Edit/Tree: use the tree editing criterion to measure the distance between two trees.

The M/Exact problem has been studied extensively in certain models, such as PIM and SSO, but the M/Approx problem has not. Our results deal mostly with the M/Approx problem.

7.2 Protocols

7.2.1 PIM/Approx

Except for the research on the general secure multi-party computation problem, this specific problem has not been studied in the literature. Unless otherwise specified,

we assume the alphabet used in the following solution to be predefined and its size to be finite. This assumption is quite reasonable in many situations; for instance, DNA sequences use a fixed alphabet of four symbols. Under this assumption, we can solve the PIM/Approx/f problem. However, because the way to calculate *edit distance* cannot be represented in the form $\sum_{k=1}^{n} f(a_k, b_k)$, the PIM/Approx/Edit problem is not a special case of the PIM/Approx/f problem.

In some other situations, the above finite alphabet assumption does not apply. For instance, fingerprint, image and voice patterns use real numbers instead of characters from a known finite alphabet. The above-mentioned solution for the PIM/Approx/f problem cannot be used anymore, however by exploiting the mathematical property of $\sum_{i=1}^{n}(a_i - b_i)^2$, we have come up with a solution for the PIM/Approx/Squ problem for infinite alphabet after introducing an *untrusted* third party who does not know the inputs from either of the two parties and learns nothing about them (or about the query, or the answer to it). We also have a solution to the PIM/Approx/Abs problem using a Monte Carlo technique. All of these are given below.

PIM/Approx/Squ Protocol

Suppose that Bob has a database $T = \{t_1, ..., t_N\}$, and assume the length of each string t_i is n; Alice wants to know the $t_i \in T$ that most closely matches a query $x = x_1...x_n$ based on the PIM/Approx/Squ metric. The requirement is that Bob should not know x or the result, and Alice should not be able to learn more information than the reply from Bob.

We propose a protocol to compute the matching score using an untrusted third party, Ursula. Our assumption here is that Ursula will not conspire with either Alice or Bob. However, the third party is not fully trusted: Ursula should not be able to deduce either x or T, or the final matching score s. This protocol works for both finite and infinite alphabet.

Let $\vec{x} = (-2x_1, ..., -2x_n, 1)$; for each $t_i = y_{i,1}...y_{i,n}$, let $\vec{z_i} = (y_{i,1}, ..., y_{i,n}, \sum_{k=1}^{n} y_{i,k}^2)$, Observe that:

$$\sum_{k=1}^{n}(x_k - y_{i,k})^2 = \vec{x} \cdot \vec{z_i} + \sum_{k=1}^{n} x_k^2.$$

Since $\sum_{k=1}^{n} x_k^2$ is a constant, we can use $\vec{x} \cdot \vec{z_i}$ instead of $\sum_{k=1}^{n}(x_k - y_{i,k})^2$ to find the closest match. After we get the closest match, Alice can calculate the actual score by adding $\sum_{k=1}^{n} x_k^2$.

Protocol

1. Alice and Bob jointly generate two random numbers r and r'.

2. For each $t_i \in T$, repeat the next five sub-steps, in which $t_i = y_{i,1}...y_{i,n}$, $\vec{x} = (-2x_1, ..., -2x_n, 1)$.

   (a) Bob constructs $\vec{z_i} = (y_{i,1}, ..., y_{i,n}, \sum_{k=1}^{n} y_{i,k}^2)$.

   (b) Alice and Bob jointly generate two random vectors $\vec{R}$, $\vec{R'}$ (of size $n+1$).

   (c) Alice sends $\vec{w_1} = \vec{x} + \vec{R}$ and $s_1 = \vec{x} \cdot \vec{R'} + r$ to Ursula.

(d) Bob sends $\vec{w}_2 = \vec{z}_i + \vec{R}'$ and $s_2 = \vec{R} \cdot (\vec{z}_i + \vec{R}') + r'$ to Ursula.

(e) Ursula computes $v_i = \vec{w}_1 \cdot \vec{w}_2 - s_1 - s_2$ and gets the resulting $v_i = \vec{x} \cdot \vec{z}_i - (r + r')$.

3. Ursula computes $score' = \min_{i=1}^{N} v_i$, and sends the resulting $score'$ to Alice.

4. Alice computes $score = score' + \sum_{k=1}^{n} x_k^2 + (r + r')$, which is the closest match between x and any $t_i \in T$.

The random vectors $\vec{R}$ and $\vec{R}'$ are used to disguise Alice's and Bob's data; the random numbers r and r' are used to disguise the query results and the intermediate results. The communication cost is $O(n * N)$.

PIM/Approx/Abs Protocol

First, we will present a Monte Carlo technique for Alice and Bob to calculate $|x_k - y_k|$ (x_k is Alice's secret input and y_k is Bob's), and then use it as a building block to compute $\sum_{k=1}^{n} |x_k - y_k|$. The protocol involves an untrusted third party, Ursula, who learns nothing. The protocol works for both finite and infinite alphabets. Assume that $0 < x_k \leq U$ and $0 < y_k \leq U$ for some number U. The protocol for $|x_k - y_k|$ is as follows (where W is a parameter that affects the accuracy of the estimate, and $counter = 0$ initially):

1. Alice generates a random number R_k, and then generates a sequence of $W - R_k$ random numbers, each uniformly over $(0..U]$.

2. Alice randomly replaces half of these $W - R_k$ numbers with their negative values.

3. Alice "splices" R_k zeroes into random positions of the above sequence of $W - R_k$ numbers, resulting in a new sequence S of W numbers.

4. Alice then sends S to Bob.

5. For each number s from S, if $s = 0$, Alice sends 1 to Ursula; if $s > 0$ then Alice sends 1 to Ursula if $|s| \geq x_k$ and sends 0 otherwise; if $s < 0$ then Alice sends 0 to Ursula if $|s| \geq x_k$ and sends 1 otherwise.

6. For each number s from S, if $s = 0$, Bob sends 0 to Ursula; if $s > 0$ then Bob sends 1 to Ursula if $|s| \geq y_k$ and sends 0 otherwise; if $s < 0$ then Bob sends 0 to Ursula if $|s| \geq y_k$ and sends 1 otherwise.

7. Ursula increases *counter* by 1 if the values she receives from Alice and Bob are different.

8. Ursula computes $score = counter * \frac{U}{W}$, which is an unbiased estimate of $|x_k - y_k| + R_k * \frac{U}{W}$.

Because of R_k, Ursula does not know the actual distance between x_k and y_k, and because of the negative numbers among those W random numbers, Ursula cannot figure out whether $x_k > y_k$ or $x_k < y_k$.

Now, let us see how to use the above protocol to compute $\sum_{k=1}^{n} |x_k - y_{i,k}|$, where $x = x_1 ... x_n$ and $t_i = y_{i,1} ... y_{i,n}$:

1. Alice generates a random number R.

2. For each $t_i \in T$, suppose $t_i = y_{i,1}...y_{i,n}$ and repeat the next three sub-steps:

   (a) $counter = 0$.

   (b) For each $k = 1, ..., n$, Alice, Bob and Ursula use the above protocol to compute $|x_k - y_{i,k}|$. The random numbers $R_{i,1}, ..., R_{i,n}$ used in the above protocol are generated by Alice, such that $\sum_{k=1}^{n} R_{i,k} = R$.

   (c) Ursula computes $score_i = counter * \frac{U}{W}$, which is an unbiased estimate of $\sum_{k=1}^{n} |x_k - y_{i,k}| + \sum_{k=1}^{n} R_{i,k} * \frac{U}{W} = \sum_{k=1}^{n} |x_k - y_{i,k}| + R * \frac{U}{W}$.

3. Ursula computes $score' = \min_{i=1}^{N} score_i$, and sends $score'$ to Alice.

4. Alice computes $score = score' - R * \frac{U}{W}$ and gets the closest match between x and any $t_i \in T$.

The communication complexity is $O(n * W * N)$.

PIM/Approx/f protocol

If the alphabet is predefined and its size is finite, we can solve a general problem– computing $f(x_k, y_k)$. However, we cannot directly use this protocol n times to compute $\sum_{k=1}^{n} f(x_k, y_k)$ because that would reveal each individual $f(x_k, y_k)$ result. We will present the protocol for computing $f(x_k, y_k)$ here, and then in the following subsection, we will discuss how to use it as a building block to compute $\sum_{k=1}^{n} f(x_k, y_k)$ without revealing any individual $f(x_k, y_k)$.

Suppose Alice has an input x_k; Bob has an input y_k; Alice wants to know the result of $f(x_k, y_k)$ without revealing x_k and the result to Bob, and Bob does not want

to reveal his y_k to Alice. After presenting a solution to this problem, we later use it as a building block to construct solutions to other problems.

f-function Protocol

1. Bob computes $f(\alpha_i, y_k)$ for each $\alpha_i \in X$, where X is the finite (known) alphabet. Let m be the size of X.

2. Alice uses the 1-out-of-m Oblivious Transfer Protocol to get $f(X, y_k)$.

PIM/Approx/f Protocol

Now, let us see how to securely compute $\min_{i=1}^{N}(\sum_{k=1}^{n} f(x_k, y_{i,k}))$. As we discussed above, we cannot run the above f-function protocol n times to get $\sum_{k=1}^{n} f(x_k, y_{i,k})$. In the following protocol, we will use a disguise technique to hide each individual result of $f(x_k, y_{i,k})$.

For each $t_i = y_{i,1}...y_{i,n}$, and for each $k = 1, ..., n$, let $f_{i,k}(x_k, y_{i,k}) = f(x_k, y_{i,k}) + R_{i,k}$, where $R_{i,k}$ is a random number, the following protocol shows how A and B calculate $\min_{i=1}^{N} \sum_{k=1}^{n} f(x_k, y_{i,k})$.

1. Bob generates a random number R then sends R to Alice.

2. For each $t_i = y_{i,1}, ..., y_{i,n}$, repeat the next five sub-steps:

   (a) Bob constructs $f_{i,k}(x_k, y_{i,k}) = f(x_k, y_{i,k}) + R_{i,k}$ for $k = 1, ..., n$, where $R_{i,1}, ..., R_{i,n}$ are n random numbers.

(b) Alice and Bob use the f-function protocol to compute $f_{i,k}(x_k,\ y_{i,k})$, for each $k = 1, ..., n$.

(c) Alice sends $\sum_{k=1}^{n} f_{i,k}(x_k, y_{i,k})$ to Ursula.

(d) Bob sends $\sum_{k=1}^{n} R_{i,k} - R$ to Ursula.

(e) Ursula computes $score_i = \sum_{k=1}^{n} f_{i,k}(x_k, y_{i,k}) - (\sum_{k=1}^{n} R_{i,k} - R) = \sum_{k=1}^{n} f(x_k, y_{i,k}) + R$.

3. Ursula computes $score' = \min_{i=1}^{N} score_i$, and sends $score'$ to Alice.

4. Alice compute $score = score' - R$, thus getting the actual distance between x and the closest t_i in the database T.

Although Alice knows each individual $f_{i,k}(x_k, y_{i,k})$, she does not know the actual value of $f(x_k, y_{i,k})$ because of $R_{i,k}$. Similarly, because of R, Ursula does not know the actual score of the closest match. The communication cost of the protocol is $O(m * n * N)$, where m is the size of the alphabet, n is the length of each pattern, and N is the size of the database. In many cases, m is quite small. For instance, m is four in DNA databases.

Because $|x_k - y_k|$, $(x_k - y_k)^2$ and $\delta(x_k, y_k)$ functions are special cases of $f(x_k, y_k)$, PIM/Approx/(Abs, Squ, δ) problems can all be solved using the above protocol.

7.2.2 SSO/Approx

In this model, Bob is a service provider who provides storage and database query services to Alice. According to Alice's privacy requirement, Bob should know nothing

about the database that he stores for Alice, nor should he know the query. So Bob has to conduct a disguised database query based on the encrypted or disguised data of Alice.

The requirement that Bob should not know the query result, as in the PIM problem, is no longer needed in the SSO problem. The reason is that Bob does not know the contents of the database, he does not even know what the database is for, so that knowing whether Alice's query is in the database does not disclose any secret information to Bob.

Intuitively, it can look like that the SSO/Approx problem might be more difficult than the PIM/Approx problem because in the latter Bob at least knows the contents of the database whereas in the former he knows nothing about the database. But knowing the contents of the database has a disadvantage, in that Bob cannot know an intermediate result because he knows one of the inputs (the database); if he also knew an intermediate result, he might be able to figure out the other input (the query) of the computation. However, in the SSO/Approx problem, Bob knows nothing about the database, so it is safe for him to know intermediate results without exposing the secret query.

Whether Bob can know intermediate results is a critical issue for reducing the communication complexity. If he knew intermediate results to some extent, he could conduct the comparison operation to find the minimal or maximal score; otherwise, he has to turn to Alice to find the minimal or maximal score, which results in high communication cost in the PIM problem.

The SSO/Approx problem is similar to the secure outsourcing of scientific computations problems studied by Atallah and Rice [11]. The difference is that in secure outsourcing problems, the inputs are provided by Alice every time a computation is conducted at Bob's side; therefore, Alice can encrypt/disguise the inputs differently in different rounds of the computation. However, in the SSO problem, one of the inputs (the database) is encrypted/disguised only once, and this same input is used in all rounds of computations; this makes the problem more difficult.

So far, we have a solution only for SSO/Approx/Squ problem. The solution works for both infinite and finite alphabets.

SSO/Approx/Squ Protocol

Suppose that Alice wants to outsource her database $T = \{t_1, ..., t_N\}$ to Bob, and wants to know if query string $x = x_1...x_n$ matches any pattern t_i in the database T.

The straightforward solution would be to let Bob send the whole database back to Alice, and let Alice conduct the query by herself. Although this solution satisfies the privacy requirement, much better communication complexity can be achieved. Another intuitive question would be whether Bob can conduct the matching independently after Alice sends him the relevant information about the query. If the answer is true, Bob should be able to find the item t_i that has the closest match to the query x. In another words, if $t_i = y_1...y_n$ and $score_i = \sum_{k=1}^{n}(x_k - y_k)^2$, then Bob should be able to find the minimum value of $score_i$. However, because of the privacy requirement, Bob is not allowed to know the actual query x, nor is he allowed to know

the content of the database, so how does he compute the distance $score_i$ between x and each of the element t_i in the database?

The idea behind our solution is based on the fact that $\vec{x} \cdot \vec{z}^T = (\vec{x}Q^{-1}) \cdot (Q\vec{z}^T)$, where Q is an invertible matrix. Alice can store $Q\vec{z}^T$ instead of $\vec{z}^T$ at Bob's site, and keeps Q secret from Bob. She will send $\vec{x}Q^{-1}$ to Bob each time she wants to send a query x; therefore Bob can compute $\vec{x} \cdot \vec{z}^T$ without even knowing $\vec{x}$ and $\vec{z}$. If we can use $\vec{x} \cdot \vec{z}^T$ to represent the $\sum_{k=1}^{n}(x_k - y_k)^2$, we can make it possible for Bob to conduct the approximate pattern matching.

For each $t_i = y_{i,1}...y_{i,n}$ in the database T, let $\vec{t_i} = (\sum_{k=1}^{n} y_{i,k}^2 + R - R_i, y_{i,1}, ..., y_{i,n}, 1, R_i)$, and let $\vec{x} = (1, -2x_1, ..., -2x_n, R_A, 1)$, where R, R_A and R_i are random numbers. We will have $\vec{x} \cdot \vec{t_i}^T = \sum_{k=1}^{n} y_{i,k}^2 - 2\sum_{k=1}^{n} x_k y_{i,k} + R + R_A$, and therefore $score_i = \sum_{k=1}^{n}(x_k - y_{i,k})^2 = \vec{x} \cdot \vec{t_i}^T + (\sum_{k=1}^{n} x_k^2 - R - R_A)$. Since $(\sum_{k=1}^{n} x_k^2 - R - R_A)$ is a constant, it does not affect the final result if we only want to find the t_i that produces the minimum $score_i$. Therefore, Bob can use $\vec{x} \cdot \vec{t_i}^T$ to compute the closest match.

Before outsourcing the database to Bob, Alice randomly chooses a secret $(n+3) \times (n+3)$ invertible matrix Q, and computes $\vec{z_i} = Q\vec{t_i}^T$, then sends $T' = \{\vec{z}_1, ..., \vec{z}_N\}$ to Bob.

Protocol

1. For any query string $x = x_1...x_n$, Alice generates a random number R_A, and constructs a vector $\vec{x} = (1, -2x_1, ..., -2x_n, R_A, 1)$, then sends $\vec{x}Q^{-1}$ to Bob.

2. Bob computes $score'_i = \vec{x} \cdot \vec{z}_i^T$, for $i = 1, ..., N$.

3. Bob computes $\min_{i=1}^{N} score'_i$, and gets the corresponding i.

4. Bob returns $\vec{z}_i$ to Alice.

5. Alice computes $Q^{-1}\vec{z}_i$ and gets t_i, which is the closest match of her query.

Because Alice and Bob are involved in only one round of communication, the communication cost is $O(n)$.

Notice that we have introduced random numbers R, R_A, R_i for $i = 1, ..., N$. The purpose of R is to prevent Bob from knowing the actual distance between x and the items in the database; the purpose of R_A is to prevent Bob from knowing the relationship between two different queries; the purpose of R_i is to prevent Bob from knowing the relationship among items in the database. Without R_i, two similar items in the database T would still be similar to each other in the disguised database T'; adding a different random number to each different item will make this similarity disappear.

7.2.3 SSCO/Approx

This model poses more challenges than the SSO model because Bob could now collude against Alice with a client, or he can even become a client. Therefore, one of the threats would be for Bob to compromise the privacy of the database by conducting a number of queries and deriving the way the database is encrypted or disguised. A secure protocol should resist this type of active attack. We have a solution for the SSCO/Approx/Squ problem that works for both infinite and finite alphabets.

SSCO/Approx/Squ Protocol

One of the differences between the SSCO/Approx problem and the SSO/Approx problem is who sends the query. In the SSO/Approx/Squ protocol, Alice transforms the query x to a vector $\vec{x}Q^{-1}$, and sends the vector to Bob; in the SSCO/Approx/Squ protocol, the client Carl will send the query. Because Carl does not know Q, he cannot construct $\vec{x}Q^{-1}$ by himself. If Carl could get the result of $\vec{x}Q^{-1}$ securely, namely without disclosing $\vec{x}$ to Alice and without knowing Q of course, we would have a solution. Because $Q^{-1} = (\vec{q}_1^T, ..., \vec{q}_m^T)$, computing $\vec{x}Q^{-1}$ securely is basically a task of computing $\vec{x} \cdot \vec{q}_k^T$ for $k = 1 \ldots m$, which can be solved using the same technique as that used in solving PIM/Approx/Squ problem.

Therefore, by modifying step 2 of the SSO/Approx/Squ protocol slightly, and also by using a form of "$R_\alpha * (score + R_A)$", instead of the form of "$score + R_A$" as is used in SSO/Approx/Squ protocol, we obtain a SSCO/Approx/Squ protocol as the following:

Let $T = \{t_1, ..., t_N\}$ be the database Alice wants to outsource to Bob, and assume the length of each string t_i is n. Alice generates N random numbers $R_1, ..., R_N$. For each $t_i = y_{i,1}, ..., y_{i,n}$, let $\vec{t_i} = (\sum_{k=1}^{n} y_{i,k}^2 + R - R_i, y_{i,1}, ..., y_{i,n}, 1, 1, R_i)$; let $\vec{z_i} = Q\vec{t_i}^T$, where Q is a randomly generated $(n+4) \times (n+4)$ matrix.

In what follows, we assume that Alice outsourced the database $T' = \{\vec{z}_1, ..., \vec{z}_N\}$ to Bob.

Protocol

1. Whenever a client Carl wants to conduct a search on query $x = x_1...x_n$, he generates a random number R_C.

2. Alice generates random numbers R_A and R_α.

3. Carl and Alice jointly compute $\vec{q} = R_\alpha \vec{x} Q^{-1}$, where $\vec{x} = (1, -2x_1, ..., -2x_n, R_C, R_A, 1)$. The computation does not reveal Alice's secret Q, R_A or R_α to Carl, nor does it reveal Carl's private query x or R_C to Alice.

4. Carl then sends the vector $\vec{q}$ to Bob.

5. Bob computes $score_i = \vec{q} \cdot \vec{z}_i^T = R_\alpha(\sum_{k=1}^{n} y_{i,k}^2 - 2\sum_{k=1}^{n} x_k y_{i,k} + R_C + R_A)$.

6. Bob returns to Alice $score' = \min_{i=1}^{N} score_i$.

7. Alice computes $score'' = \frac{score'}{R_\alpha} - R_A = \sum_{k=1}^{n} y_{i,k}^2 - 2\sum_{k=1}^{n} x_k y_{i,k} + R_C$ and sends it to Carl.

8. Carl computes $score = score'' + \sum_{k=1}^{n} x_k^2 - R_C$, which is the answer he seeks.

Because of R_C, Alice cannot figure out the actual score for this query, and because of R_A and R_α, Carl cannot figure out the actual score between his query and other items in the database (except for the matched one), even if Carl could collude with Bob. The communication cost of the protocol is $O(n^2)$, most of which is contributed by the computation of $R_\alpha \vec{x} Q^{-1}$ in step 3.

7.3 Chapter Summary and Future Work

We have developed three models for secure remote database access, and presented a class of problems and solutions for these models. For some problems, such as SSO/Approx/Squ and SSCO/Approx/Squ problems, our solutions are practical, and they only need $O(n)$ and $O(n^2)$ communication cost, respectively; while for PIM/Approx problems, our results are still at the theoretical stage because of their high communication cost. Improving the communication cost for those solutions is one avenue for future work: We suspect that, whenever there is a dependence on N, that dependence could be made sub-linear (perhaps logarithmic) by combining our methods with the known powerful higher dimensional indexing techniques [92, 4, 8, 68, 71, 61, 18]. However, combining those schemes with our protocols will not be a trivial task, and the increase in the constant factors hiding behind the "big-oh" notation may well negate the benefits of the asymptotic sub-linearity in N; for example, in a tree search for processing the query, Bob has to be prevented from knowing what nodes of his tree are visited when processing the query (otherwise he gets information about the query), which requires using a PIR-like protocol at each node down the tree. But even that is not enough: Alice herself must be prevented from learning anything about Bob's data other than the answer to her query, but in most of the tree-based schemes in the literature the comparison at a node of the search tree gives information about the data that is associated with that node (these schemes were designed for an environment where the searcher is the owner, and may require substantial modification before they are used in our context).

Another avenue for future work is the pattern matching of branching structures: the pattern matching problems that we have discussed only involve patterns of simple linear structure; in many applications, patterns have a branching structure, such as a tree or a DAG. The M/Approx/Edit/Tree problem in our model is one of the examples. Developing a secure protocol to deal with this type of query is a challenging problem.

Finally, avoiding the use of a third party in the protocols that use such an Ursula is an interesting problem.

8. OTHER SECURE TWO-PARTY COMPUTATION PROBLEMS

Apart from problems discussed in the previous chapters, secure multi-party computation problems exist in many other computation domains as well. Many of the problems are yet to be identified. For this purpose we propose a general framework, which facilitates the uncovering of new specific secure multi-party computation problems in a variety of computation domains.

8.1 Framework

This framework enables us to systematically transform a normal computation problem to a secure multi-party computation problem. We start by describing two different models of computation (without the privacy requirements), and then show how to transform them to the models enhanced with privacy requirements. The privacy-enhanced model will be referred as the *Secure Multi-party Computation (SMC)* model.

According to the number of distinguished inputs, we classify computations into two different models: the multi-input computation model and the single-input computation model. The multi-input computation model usually has two distinguishable inputs. For instance, client/server computation is a multi-input computation model. The single-input computation model usually has one input or one set of inputs. For

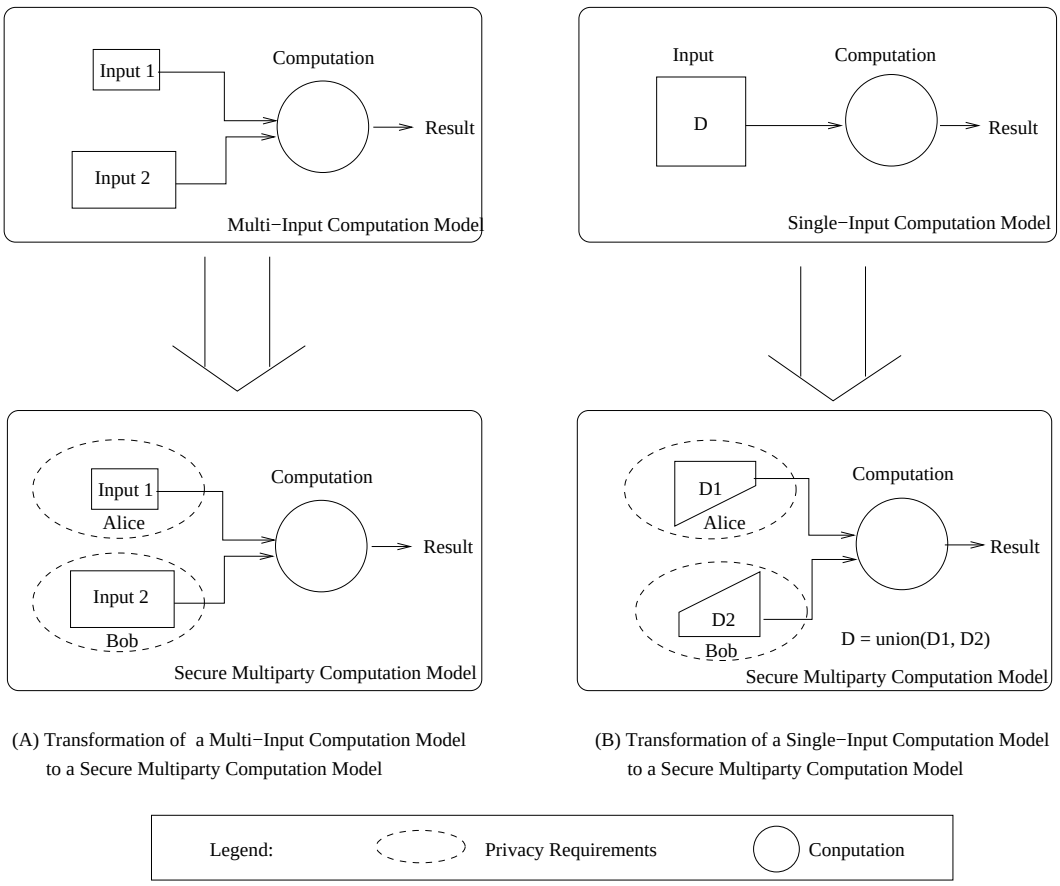

Figure 8.1. Models

example, in data mining [23] and statistical analysis, all the inputs usually come from one data set although the inputs consist of multiple data items.

Next we want to transform both models to the Secure Multi-party Computation model, in which the input from each participating party is considered as private. In certain specific cases, the computation results could also be private.

For the multi-input computation model, its transformation to the corresponding SMC model is straightforward because the model naturally has at least two inputs. Therefore, if we treat each input as coming from a different party, the new problem

now becomes "how to conduct the same computation while maintaining the privacy of each party's input." Figure 8.1(a) demonstrates such a transformation.

For the single-input computation model, because it only has one input, we cannot use the same transformation as we used for the multi-input computation model; we have to somehow transform the model to a multi-input computation model. Let us call this computation C, and assume the single input is a set D of data items. If we can divide D into two disjoint data set D_1 and D_2, we will have a multiple-input computation model. There are many ways to divide D into two data sets, and each way could lead to a different SMC problem. We are focusing on two types of transformations: homogeneous transformation and heterogeneous transformation.

In the homogeneous transformation, D's data items are divided into two sets, but each single data item maintains its all features. For example, if D is a database of student records, the homogeneous transformation will put a subset of the records into one data set, and the rest of the the records into another data set; however, each student's record is not cut into two parts: the two generated data sets maintain the same set of features. Figure 8.1(b) demonstrates such a transformation.

In the heterogeneous transformation, each single data item is cut into two parts, with each part going to a separate data set. Taking the same example used above, if each student record contains a student's academic record and medical record, the heterogeneous transformation could put all students' academic records into one data set, and all students' medical records into another data set. Figure 8.1(c) demonstrates such a transformation.

After the above transformation, the new problem now becomes "how to conduct the computation C on the union of D_1 and D_2, where D_1 belongs to one party and D_2 belongs to another party, and neither of these two parties wants to disclose his or her private data set to other."

8.2 Other Secure Two-Party Computation Problems

Using this framework, we have uncovered many interesting secure multi-party computation problems, in addition to those problems solved in this dissertation. A few examples of the problems are described in the following.

1. *Selection problem* (select median, select the kth smallest element): Alice has a private data set d_1, and Bob has a private data set d_2; they want to find the median (or the kth smallest element) among the data in $d_1 \cup d_2$.

2. *Sorting problem*: Alice has a private data set d_1, and Bob has a private data set d_2; they want to sort the elements in the union of these two data sets, such that each element in these two data sets is marked by a number representing the order of this element.

3. *Shortest path problem*: Alice and Bob each has a private graph represented by $g1$ and $g2$, respectively, and the links between these two graphs are known to both of them. Given any two points (they could be in a same graph, or in different graphs), how could Alice and Bob jointly compute the shortest distance (or path) between these two points. One of the applications of this problem is

network traffic routing between two private network service providers if they do not want to disclose too much information about their own private network.

4. *Polynomial Interpolation*: Alice has n_1 private pairs (x_i, y_i), for $i = 0, \ldots, n_1$, Bob has n_2 private pairs (x_j, y_j), for $j = n_1 + 1, \ldots, n$. Suppose $x_0, \ldots, x_n$ are distinct data points. How can Alice and Bob jointly find the polynomial $p(x)$ of degree n that interpolates the data set $\{(x_0, y_0), \ldots, (x_n, y_n)\}$, i.e. $p(x_k) = y_k$ for all $k = 0, \ldots, n$?

5. *Data Mining*: Alice has a private structured database D_1, and Bob has another private structured database D_2; How could Alice and Bob conduct any of the following data mining operations based on $D_1 \cup D_2$ without disclosing the content of one party's database to the other party?

   - Data Classification
   - Data Clustering
   - Mining Association Rules
   - Data Generalization, Summarization and Characterization

   Some of the above problems have already been under studied [66, 75, 5].

6. *Machine Learning*: Alice has a private data set D_1, and Bob has another private data set D_2. How could both parties jointly conduct machine learning without disclosing private data to the other party?

9. CONCLUSIONS AND FUTURE WORK

In this dissertation, we have developed solutions to a variety of secure two-party computation problems. We summarize what we have learned about secure two-party computation and give some thoughts on extensions to this research.

9.1 Summary of Main Results

- Data Perturbation: We have demonstrated how to use the data perturbation technique (namely hiding x by adding a random number to x) and various other techniques such as oblivious transfer, homomorphic encryption, to solve specific secure two-party computation problems.

- Scalar Product: We have developed three protocols to conduct secure two-party scalar product computation, a useful building block for solving many other secure two-party computation problems.

- Scientific Computation: We have developed protocols to solve three secure two-party scientific computation problems, including the secure two-party linear system of equations problem, the secure two-party least square problem, and the secure two-party linear programming problem.

- Geometric Computation: We have developed protocols to solve three secure two-party computational geometry problems, including the secure two-party point

location problem, the secure two-party intersection problem, and the secure two-party closest pair problem.

- Statistical Analysis: We have developed protocols to solve basic secure two-party statistical analysis problems, including computing mean value, standard deviation, correlation coefficient, and linear regression.

- Database Query: We have studied the problem of conducting database query (based on approximate matching) without disclosing private information. The problem is studied under three different models, for each of which we have developed protocols for the corresponding private database query problems.

9.2 Summary of Findings, Experience, and Challenges

9.2.1 Findings and Experience

In our studies, we have investigated various ways to reduce communication complexity over the general solutions; we summarize our findings and experience here:

Data Perturbation Technique

The data perturbation technique is an effective way of hiding data. Although this technique is not as secure as a good encryption scheme, it has the advantage of supporting various computations, such as addition, multiplication and comparison. Most encryption schemes, on the other hand, are good at hiding the data, but once the data is encrypted, conducting computations becomes impossible (with the exception of a homomorphic encryption cryptosystem).

Domain Specific Knowledge

Unlike the general solution, our protocols exploit domain specific knowledge to solve specific secure two-party computation problems. For instance, to compute a scalar product $f(x, y)$, we effectively used the fact of $f(x, y) = f(\pi(x), \pi(y))$, which obviously does not hold for an arbitrary function; to solve the secure two-party linear system of equations problem $(M_1 + M_2)x = b_1 + b_2$, we take advantage of the fact that the solution to the new linear system of equations problem $P(M_1 + M_2)QQ^{-1}x = P(b_1 + b_2)$ is equivalent to the original problem. Such domain specific knowledge allows us to disguise the inputs effectively.

Emulation of the Computation

The circuit evaluation approach uses a circuit to emulate the computation; therefore the size of the circuit as well as the communication complexity depend on the complexity of the computation. Some of our solutions take the similar approach–emulation–but at a level higher than the gate level, thus gaining performance improvement. However, we observed that if we can avoid emulation at all, we might be able to gain even more performance improvement. Our solutions to the scientific computation problems demonstrate this approach, which is generalized in the following:

Assume we want to compute $f(x, y)$ (party A has x and party B has y). Our approach transforms this problem to a problem of computing $f'(x', y')$ such that (1)

one party (e.g. party B) learns f', x', y', thus can compute $f'(x', y')$ without involving too much communication. (2) $f(x, y)$ could be derived from $f'(x', y')$.

To achieve such a transformation, it is desirable to find a function f' and two "special" functions g_1 and g_2, such that $f(x, y) = f'(g_1(x), g_2(y))$. The special functions g_1 and g_2 are generated by party A and known only to A. Both functions must satisfy the following requirements: (1) knowing the result of $g_i(\alpha)$ without knowing g_i, one should not be able to find the value of α; (2) knowing both α and the result of $g_i(\alpha)$, one should not be able to find the function of g_i. Therefore, B's knowing the results of $g_1(x)$ and $g_2(y)$ does not give him the advantage of finding g_1, g_2 and the value of x.

After knowing f' and the results of $g_1(x)$ and $g_2(y)$, B can conduct the computation of $f'(g_1(x), g_2(y))$ without intensively interacting with A.

This approach looks appealing because usually the actual computation of $g_1(x)$, $g_2(y)$, and $f'(g_1(x), g_2(y))$ involves much less interaction between A and B than the circuit evaluation approach. In contrast to the circuit evaluation approach, whose communication cost depends on the complexity of expressing f as a circuit, this approach achieves a significant improvement to the communication complexity for some computations. For example, for the widely used solution–Simplex Method–to the secure two-party linear programming problem, the computation complexity of $f(x, y)$ is not polynomial at all, which implies that the communication cost of the circuit evaluation protocol will not be polynomial. However in our solutions, because

we do not emulate the Simplex Method between two parties, the communication cost of our solutions is only linear in the size of the input.

Although it is easy to find functions g_1, g_2 and f' for some computation problems, such as the scientific computation problems and the scalar product problem, it is not always easy to find functions for an arbitrary computation, such as the secure two-party geometric computation problems. Therefore, we only consider the above approach as a heuristic.

Useful Building Blocks

Learning from our experience throughout this research, we have found that the following techniques and building blocks are particularly useful for solving specific secure two-party computation problems:

- 1-out-of-n Oblivious Transfer protocol: Oblivious Transfer is one of the most basic possible primitives that can break the "knowledge symmetry" between two parties. In most of our protocols, to hide the input of party A, A sends redundant data to the other party B along with the real data hidden among them, and gets the actual result back using the 1-out-of-n Oblivious Transfer protocol. B's chance of guessing the correct input is $\frac{1}{n}$, and can be reduced to n^{-m} by parallel m-wise execution of the protocol.

- Private Permutation: When a computation involves a set of data $x_1, \ldots, x_n$, and the order of x_i does not affect the result of the computation, it is helpful to use a permutation technique to preserve the privacy of the data.

- Private Comparison: Comparison is necessary in many computations. Knowing how to compare two secret numbers is necessary in solving the secure two-party computation problem.

- Hiding the intermediate results using data perturbation: In many cases, the solutions involve a series of steps, and the intermediate results need to be generated between two consecutive steps. If the intermediate results are not disguised appropriately, private information might be disclosed. We use the data perturbation technique to disguise the intermediate results in many of our solutions. For example, if the result of $f_i(x, y)$ is only an intermediate result, we cannot let one party to learn the result of it; otherwise we will unnecessarily disclose information of one party's input to the other party. Our protocols choose to perturb the intermediate result, namely instead of computing $f_i(x, y)$, we compute $f_i(x, y) + R_i$, where R_i is a random number.

- Circuit evaluation for simple circuit: As we mentioned, general solutions are not practical because they use the circuit evaluation protocol, whose communication cost depends on the complexity of expressing the functionality as a circuit. This assertion, however, does not exclude the use of the circuit evaluation protocol when the circuit is simple and small. In fact, we use the circuit evaluation protocol as a primitive in some of our solutions.

9.2.2 Challenges

During the course of this study, we encountered the following challenge a number of times. The challenge is about the conflict of privacy and efficiency.

For many problems, such as the point-inclusion problem (without the privacy concerns), we know there exist more efficient algorithms (than what we used in our protocol) using the divide-and-conquer technique; however we found it difficult to use those efficient algorithms because the divide-and-conquer technique leads to the disclosure of information about the inputs.

Let us use the point-inclusion problem to illustrate this challenge. An efficient solution to the point-inclusion problem works as follows: (1) draw a line between two vertices of the convex polygon to cut the polygon into two smaller polygons; (2) decide which side the point is on, and throw away the part of the polygon on the other side of the line; (3) recursively decide whether the point is inside the part of the polygon that is on the same side of the line. This algorithm has the complexity of $O(\log n)$.

If we directly emulate this efficient algorithm under the secure two-party framework to achieve sub-linear complexity, both parties need to know which side the point is on after they split the polygon. This gives them information to derive the location of the other party's point or polygon. We have thought of using the Oblivious Transfer technique to get around this, but the solutions are far from satisfactory.

The same problem exists for the database query problem, which, in reality, uses various kinds of index schemes to improve the efficiency of the query. Usually these

index schemes use certain kinds of data structures similar to a binary tree data structure, and the algorithms involve "walking" along the index structures. In our protocols, to emulate the "walk" along this kind of data structure, both parties need to know which branch to "walk," and thus gain extra information about the other party's input.

9.2.3 Trading Privacy for Efficiency

In the studies of secure multi-party computation problems, a common goal is to prevent any party from knowing *anything* about the the other parties' private inputs. This is an ideal situation, but the high cost of the solutions makes them difficult to be deployed in practice. Throughout this study, we found out that if the parties allow the disclosure of some partial information about their private inputs, the solutions might become much more efficient. For example, in the solutions to the survey problem 6.2.1, we did not use the expensive 1-out-of-n Oblivious Transfer protocol and some other expensive cryptography primitives. Our solution is very efficient and practical compared to the other protocols that depend on expensive cryptography primitives, although it does give the interviewer chances to figure out the range of some replies, especially those replies that contain much larger numbers than the other replies.

In practice, in certain situations, people are not so worried about disclosing certain information. For example, if the private information is a fingerprint (represented by a vector of numbers), it does not hurt privacy too much if the range of each number is disclosed, or if the sum of the numbers is disclosed, or even one of the

numbers is disclosed. This motivates us to look for solutions that reasonably leak some information about the inputs for better performance. We believe this direction is a more viable approach to the secure multi-party computation problems.

9.3 Future Work

Apart from future work described for each specific secure two-party computation problem, we consider the following promising areas for future research:

- A formal definition of privacy: The definition of privacy used in this dissertation is weaker than the formal definition in [55]; we need to provide a formal definition of privacy and a way to formally prove the security of our protocol.

- A search for different levels of security definition: For many practical situations, the level of security achieved in the secure two-party computation might be too high because it strictly requires that nothing about private information is disclosed. Although this notion of security is desirable, the cost of achieving it might be too expensive to be practical. Therefore, for practical purposes, it is useful to formally define different levels of security, and solve those secure two-party computation problems under these new notions of security. Our work is one step towards this goal. We believe for many practical situations, it could be acceptable to sacrifice certain degrees of privacy in exchange for efficiency.

- Extension to the secure multi-party computation problem: Extending our secure two-party computation protocols to secure multi-party protocols is not trivial because, in secure multi-party computations, we need to consider the gain of

a coalition (rather than of a single party) from participating in the protocol. For example, in secure multi-party computation, we have to make sure that the privacy of the inputs are not compromised if a set (or *coalition*) of parties can combine their knowledge.

- Investigation of how to take advantage of "good" data structures: As we mentioned before, using a "good" data structure could improve the performance of an algorithm in the normal case, but under the secure multi-party computation context using data structures such as a binary tree could lead to the disclosure of private information. That is why most of our protocols do not use those "good" data structures. If we can find ways to use them without degrading the level of privacy, we will be able to improve the performance of our solutions significantly.

- Investigation of how to extend our work from the semi-honest model to the malicious model. All of our protocols assume each participating party is honest and always follows the protocol. According to [55], any party protocol that is secure in the semi-honest model can be transformed into one that is secure in the malicious model. We will investigate whether we can find a more efficient way to perform such a transformation.

LIST OF REFERENCES

LIST OF REFERENCES

[1] M. Abadi, J. Feigenbaum and J. Kilian. On hiding information from an oracle. *Journal of Computer and System Sciences*, 39:21–50, 1989.

[2] M. Abadi and J. Feigenbaum. Secure circuit evaluation: a protocol based on hiding information from an oracle. *Journal of Cryptology*, 2:1–12, 1990.

[3] N.R. Adam and J.C. Wortman. Security-control methods for statistical databases. *ACM Computing Surveys*, 21(4):515–556, December, 1989.

[4] R. Agrawal, C. Faloutsos and A. Swami. Efficient similarity search in sequence databases. In *Proceeding of the Fourth International Conference on Foundations of Data Organization and Algorithms*, October, 1993. Also in Lecture Notes in Computer Science 730, Springer Verlag, 1993, pages 69–84.

[5] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD on Management of Data*, pages 439–450, Dallas, Texas, USA, May 15 - 18, 2000.

[6] A. Apostolico and Z. Galil, editors. *Pattern Matching Algorithms*. Oxford University Press, 1997.

[7] A. Arona, D. Bruschi and E. Rosti. Adding availability to log services of untrusted machines. In *Proceedings of the Fifteenth Annual Computer Security Applications Conference*, pages 199–206, Phoenix, Arizona, USA, December 6-10, 1999.

[8] S. Arya. Ph.D thesis: Nearest neighbor searching and applications. Technical Report CS-TR-3490, University of Maryland at College Park, June, 1995.

[9] S. Arya, D. Mount, N. Netanyahu, R. Silverman and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, November, 1998.

[10] Mikhail J. Atallah and Wenliang Du. Secure multi-party computational geometry. In *WADS2001: Seventh International Workshop on Algorithms and Data Structures*, Providence, Rhode Island, USA, August 8-10, 2001.

[11] M. Atallah and J. Rice. Secure outsourcing of scientific computations. Technical Report COAST TR 98-15, Department of Computer Science, Purdue University, 1998.

[12] D. Beaver. Commodity-based cryptography (extended abstract). In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, El Paso, Texas, USA, May 4-6, 1997.

[13] D. Beaver. Server-assisted cryptography. In *Proceedings of the 1998 Workshop on New Security Paradigms*, Charlottesville, Virginia, USA, September 22-26, 1998.

[14] A. Beimel, Y. Ishai, E. Kushilevitz and T. Malki. One-way functions are essential for single-server private information retrieval. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, Atlanta, Georgia, USA, May 1-4, 1999.

[15] J. Benaloh. Dense probabilistic encryption. In *Proceedings of the Workshop on Selected Areas of Cryptography*, pages 120–128, Kingston, Ontario, Canada, May, 1994.

[16] Benny Chor, Mihály Geréb-Graus, and Eyal Kushilevitz. Private computations over the integers. *SIAM Journal on Computing*, 24(2):376–386, 1995.

[17] J. Benaloh and M. Yung. Distributing the power of a government to enhance the privacy of voters. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing*, pages 52–62, Calgary, Alberta, Canada, August 11 - 13, 1986.

[18] S. Berchtold, D. A. Keim and H-P. Kriegel. The x-tree: An index structure for high-dimensional data. In *Proceedings of the Twenty-second VLDB Conference*, Mumbai (Bombay), India, 1996.

[19] M. Blum. Coin flipping by telephone: a protocol for solving impossible problems. In *IEEE Computer Conference*, pages 133–137, 1982.

[20] G. Brassard, C. Crépeau and J. Robert. All-or-nothing disclosure of secrets. In *Advances in Cryptology - Crypto86, Lecture Notes in Computer Science*, volume 234-238, 1987.

[21] C. Cachin. Efficient private bidding and auction with an oblivious third party. In *Proceedings of the Sixth ACM Conference on Computer and Communications Security*, pages 120–127, Singapore, November 1-4, 1999.

[22] C. Cachin, S. Micali and M. Stadler. Computationally private information retrieval with polylogarithmic communication. *Advances in Cryptology: EUROCRYPT 1999, Lecture Notes in Computer Science*, 1592:402–414, 1999.

[23] Ming-Syan Chen, Jiawei Han and Philip S. Yu. Data mining: an overview from a database perspective. *IEEE Transactions On Knowledge and Data Engineering*, 8:866–883, 1996.

[24] B. Chor and N. Gilboa. Computationally private information retrieval (extended abstract). In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, El Paso, Texas, USA, May 4-6, 1997.

[25] B. Chor, N. Gilboa and M. Naor. Private information retrieval by keywords. Technical Report TR CS0917, Department of Computer Science, Technion, 1997.

[26] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan. Private information retrieval. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, Milwaukee, Wisconsin, USA, October 23-25, 1995.

[27] Christopher W. Clifton and Don Marks. Security and privacy implications of data mining. In *ACM SIGMOD Workshop on Data Mining and Knowledge Discovery*, Montreal, Canada, June 2, 1996.

[28] J. (Benaloh) Cohen and M. Fisher. A robust and verifiable cryptographically secure election scheme. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 372–382, 1985.

[29] D. Coppersmith. Cheating at mental poker. In *CRYPTO 1985*, pages 104–107, 1985.

[30] A. Corral, Y. Manolopoulos, Y. Theodoridis, M. Vassilakopoulos. Closest pair queries in spatial databases. In *Proceedings of the 2000 ACM SIGMOD on Management of Data*, Dallas, Texas, USA, May 15 - 18, 2000.

[31] C. Crépeau. A secure poker protocol that minimizes the effect of player coalitions. In *CRYPTO 1985*, pages 73–86, 1985.

[32] C. Crépeau. A zero-knowledge poker protocol that achieves confidentiality of the players' strategy, or how to achieve an electronic poker face. In *CRYPTO 1986*, pages 239–247, 1986.

[33] C. Crépeau. Equivalence between two flavors of oblivious transfers. In *Advances in Cryptology – CRYPTO 1987, Lecture Notes in Computer Science*, volume 293, pages 350–354. Springer-Verlag, 1988.

[34] M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, 1994.

[35] D. Denning. *Cryptography and Data Security*, pages 157–159. Addison-Wesley publishing company, 1983.

[36] Y. Desmedt. Some recent research aspects of threshold cryptography. In *Lecture Notes in Computer Science 1396*, pages 158–173. Springer-Verlag, 1997.

[37] G. Di-Crescenzo, Y. Ishai and R. Ostrovsky. Universal service-providers for database private information retrieval. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, September 21, 1998.

[38] Wenliang Du, Mikhail J. Atallah and Florian Kerschbaum. Protocols for secure remote database access with approximate matching. Technical Report CERIAS TR 2001-15, Purdue University, 2001.

[39] Wenliang Du and Mikhail J. Atallah. Protocols for secure remote database access with approximate matching. In *Seventh ACM Conference on Computer and Communications Security (ACMCCS 2000), The First Workshop on Security and Privacy in E-Commerce*, Athens, Greece, November, 2000.

[40] Wenliang Du and Mikhail J. Atallah. Privacy-preserving cooperative scientific computations. In *Fourteenth IEEE Computer Security Foundations Workshop*, Nova Scotia, Canada, June 11-13, 2001.

[41] Wenliang Du and Mikhail J. Atallah. Secure multi-party computation problems and their applications: A review and open problems. In *New Security Paradigms Workshop*, Cloudcroft, New Mexico, USA, September 11-13, 2001.

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

[43] R. Fagin, M. Naor and P. Winkler. Comparing information without leaking it. *Communication of the ACM*, 39:77–85, 1996.

[44] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss and R. Wright. Secure multiparty computation of approximations. In *Twenty-eighth International Colloquium on Automata, Language and Programming*, 2001.

[45] S. Fortune and M. Merritt. Poker protocols. In *CRYPTO 1984*, pages 454–464, 1984.

[46] M. Franklin, Z. Galil and M. Yung. An overview of secure distributed computing. Technical Report TR CUCS-00892, Department of Computer Science, Columbia University, 1992.

[47] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In *Symposium on Principles of Distributed Computing*, page 320, 1998.

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

[49] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, June, 1998.

[50] P. Gemmell. An introduction to threshold cryptography. In *CryptoBytes*, volume 2. RSA Laboratories, 1997.

[51] Y. Gertner, S. Goldwasser and T. Malkin. A random server model for private information retrieval. In *Second International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '98)*, 1998.

[52] Y. Gertner, Y. Ishai, E. Kushilevitz and T. Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, Dallas, Texas, USA, May 24-26, 1998.

[53] A. Gionis, P. Indyk and R. Motwani. Similarity search in high dimensions via hashing. In *The VLDB Journal*, pages 518–529, 1999.

[54] S. Goldwasser. Multi-party computations: Past and present. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, Santa Barbara, California, USA, August 21-24, 1997.

[55] O. Goldreich. Secure multi-party computation (working draft). Available from http://www.wisdom.weizmann.ac.il/home/oded/public_html/foc.html, 1998.

[56] S. Goldwasser and S. Micali. Probabilistic encryption. 28:270–299, 1984.

[57] O. Goldreich, S. Micali and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, 1987.

[58] R. Gonzalezi and R. Woods. *Digital Image Processing.* Addison-Wesley, Reading, Massachusetts, USA, 1992.

[59] M. S. Goodstadt and V.Gruson. The randomized response technique: A test on drug use. *Journal of the American Statistical Association*, 70(352):814–818, December, 1975.

[60] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[61] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *ACM SIGMOD Workshop on Data Mining and Knowledge Discovery*, pages 163–174, Boston, Massachusetts, USA, 1984.

[62] Y. Ishai and E. Kushilevitz. Improved upper bounds on information-theoretic private information retrieval (extended abstract). In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, Atlanta, Georgia, USA, May 1-4, 1999.

[63] N. Koudas H. V. Jagadish and D. Srivastava. On effective multi-dimensional indexing for strings. In *Proceedings of the 2000 ACM SIGMOD on Management of Data*, Dallas, Texas, USA, May 15 - 18, 2000.

[64] A. Jain. *Fundamentals of Digital Image Processing.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1989.

[65] N. Karmarkar. New polynomial-time algorithm for linear programming. *Combinatorica*, 4(336):373–395, 1984.

[66] H. Kargupta, B. Park, D. Hershberger and E. Johnson. Collective data mining: A new perspective toward distributed data mining. *Advances in Distributed and Parallel Knowledge Discovery*, 1999.

[67] J. Kilian. Founding cryptography on oblivious transfer. In *Proceedings of Twentieth ACM Symposium on Theory of Computing*, pages 20–31, 1988.

[68] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of the Twenty-ninth ACM Symposium on Theory of Computing*, 1997.

[69] V. Klee and G. Minty. How good is the simplex algorithm. In Shisha, editor, *Inequalities, III*, pages 159–175. Academic Press, New York, New York, USA, 1972.

[70] P. Indyk, R. Motwani, P. Raghavan and S. Vempala. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the Twenty-ninth ACM Symposium on Theory of Computing*, pages 618–625, 1997.

[71] N. Beckmann, H-P. Kriegel, R. Schneider, B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *ACM SIGMOD Workshop on Data Mining and Knowledge Discovery*, pages 322–331, Atlantic City, New Jersey, USA, 1990.

[72] E. Kushilevitz, R. Ostrovsky and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

[73] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings of the Thirty-eighth Annual IEEE Computer Society Conference on Foundation of Computer Science*, Miami Beach, Florida, USA, October 20-22, 1997.

[74] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems.* Prentice-Hall, Englewood Cliffs, 1974.

[75] Y. Lindel and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology - CRYPTO 2000, Lecture Notes in Computer Science*, volume 1880, 2000.

[76] R. Lipton. How to cheat at mental poker. In *Proceedings of AMS Short Course on Cryptography*, 1981.

[77] U. Manber. A text compression scheme that allows fast searching directly in the compressed file. *ACM Transactions on Information Systems*, 15(2):124–136, April, 1997.

[78] W. McLewin. *Linear Programming and Applications: A Course Text.* Input-Output Publishing Company, 1981.

[79] R. Muth and U. Manber. Approximate multiple string search. In *Proceedings of the Seventh Annual Combinatorial Pattern Matching Symposium*, pages 75–86, Laguna Beach, California, USA, June, 1996.

[80] D. Naccache and J. Stern. A new cryptosystem based on higher residues. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 59–66, 1998.

[81] M. Naor. Bit commitment using pseudo-randomness. In *CRYPTO 1989*, pages 128–136, 1989.

[82] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation (extended abstract). In *Proceedings of the Thirty-first ACM Symposium on Theory of Computing*, pages 245–254, Atlanta, Georgia, USA, May 1-4, 1999.

[83] S. Nene and S. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions Pattern Analysis and Machine Intelligence*, pages 989–1003, 1997.

[84] T. Okamoto and S. Uchiyama. An efficient public-key cryptosystem. In *Advances in Cryptology – EUROCRYPT 1998*, pages 308–318, 1998.

[85] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *Advances in Cryptology – EUROCRYPT 1999*, pages 223–238, 1999.

[86] K. H. Pollock and Y. Bek. A comparison of three randomized response models for quantitative data. *Journal of the American Statistical Association*, 71(356):994–886, December, 1976.

[87] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[88] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report Technical Memo TR-81, Aiken Computation Laboratory, 1981.

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

[90] J. R. Rice. *Matrix Computations and Mathematical Software.* McGraw-Hill Book Company, 1981.

[91] N. Roussopoulos, S. Kelley, F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD on Management of Data*, San Jose, California, USA, 1995.

[92] Hanan Samet. Multidimensional data structures. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 18. CRC Press, 1999.

[93] T. Sander and C. Tschudin. Towards mobile cryptography. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, Oakland, California, USA, May, 1998.

[94] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc., 1996.

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

[96] A. Shamir, R. Rivest and L. Adleman. Mental poker. Technical Report MIT/LCS/TR-125, M.I.T, 1979.

[97] Dennis Shasha and Tsong-Li Wang. New techniques for best-match retrieval. *ACM Transactions on Information Systems*, 8(2):140–158, 1990.

[98] G. Simmons. Geometric shared secret and/or shared control schemes. In *CRYPTO 1990*, pages 216–241, 1990.

[99] G. Simmons. An introduction to shared secret and/or shared control schemes and their application. In *Contemporary Cryptology, The Science of Information Integrity*, pages 441–497. IEEE Press, 1992.

[100] D. Song, D. Wagner and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of 2000 IEEE Symposium on Security and Privacy*, Oakland, California, USA, May 14-17, 2000.

[101] P. F. Syverson, D. M. Goldschlag and M. G. Reed. Anonymous connections and onion routing. In *Proceedings of 1997 IEEE Symposium on Security and Privacy*, Oakland, California, USA, May 5-7, 1997.

[102] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, March, 1965.

[103] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 66(336):884–888, December, 1971.

[104] A.C. Yao. Protocols for secure computations. In *Proceedings of the Twenty-third Annual IEEE Symposium on Foundations of Computer Science*, 1982.

[105] A.C. Yao. How to generate and exchange secrets. In *Proceedings of Twenty-Seventh IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[106] M. Yung. Cryptoprotocols: subscription to a public key, the secret blocking and the multi-player mental poker game. In *CRYPTO 1984*, pages 439–453, 1984.

VITA

VITA

Wenliang Du was born in Nanchang, China. He received his Bachelor's Degree in Computer Science in July 1993 from the University of Science and Technology of China (USTC). He performed his undergraduate thesis research in the Institute of Software at the Chinese Academy of Sciences. He went to Florida International University on a Presidential Fellowship in 1994, and got his Master's Degree in Computer Science in 1996.

In August 1996, he enrolled in the Department of Computer Sciences at Purdue University, where he joined the Computer Operations, Audit and Security Technology (COAST) laboratory, which later became the Center for Education and Research in Information Assurance and Security (CERIAS). In 1998, he performed six months of his Ph.D. research at Microsoft Cooperation. He received the degree of Doctor of Philosophy in August 2001 under the direction of Professor Mikhail J. Atallah and Professor Eugene H. Spafford.

His research interests are in computer and information security, network security, e-commerce security and applied cryptography.