Scalar-product based

Secure Multi-party Computation

by

I-Cheng Wang

(Under the direction of Kang Li)

Abstract

Secure multiparty computation (SMC) enables parties to carry out distributed computing tasks, guaranteeing no additional private information is revealed, other than what can be inferred from each participant's inputs and outputs, and the correctness of their outputs. This thesis focuses on designing practical SMC protocols that are applicable to real world problems. The author shows that SMC on any polynomial evaluation can be solved with the proposed scalar-product based protocols and the help of additive sharing. Besides that, the author enables statistical analysis and privacy preserving distributed mining, e.g. Naïve Bayesian classification on horizontally partitioned data, by designing secure protocols based on scalar-product protocols. A novel secure auction protocol is proposed by exploring the trade-off between the strength of privacy protection and the protocol complexity.

INDEX WORDS: MPC, SMC, SMPC, Secure Computation, Secure Two-party Computation, Secure Multi-party Computation, Scalar Product, Privacy Preserving Statistical Analysis, Privacy Preserving Classification, Secure Auction

Scalar-product based Secure Multi-party Computation

by

I-Cheng Wang

B.A., National Taiwan University, 2006

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2012

Scalar-product based

Secure Multi-party Computation

by

I-Cheng Wang

Approved:

Major Professor: Kang Li

Committee: E. Rodney Canfield
Roberto Perdisci

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2012

Scalar-product based

Secure Multi-party Computation

I-Cheng Wang

April 30, 2012

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Privacy is invaluable while it is fragile. Once an individual's privacy is breached, there is no way that the clock can be turned back. For a person, privacy may be data concerning his income, loans, or medical history. For a company, privacy might be its profit, or customer database. Yao's millionaire problem [Yao82] discusses two millionaires, Alice and Bob, who are interested in knowing which of them is richer without revealing their actual wealth. Now, consider a first-price sealed-bid auction for some product.[1] Each bidder writes down one bid on a piece of paper in his/her hand. Without collecting those pieces of paper or revealing any bid, is it possible to find who wrote down the highest bid?

The answer is yes. Secure multiparty computation (SMC) enables parties to carry out distributed computing tasks (like finding who wrote down the highest bid), guaranteeing (to the extent possible) the privacy of the parties' inputs (like bidders' bids) and the correctness of their outputs (like who REALLY wrote down the highest bid). SMC is assumed that a protocol execution may come under attack. For example, entities, including a subset of the participating parties, may try to learn private information or cause the result of the computation to be incorrect. The essential security requirements for any secure computation protocol are privacy and correctness. Privacy means

[1] In a first-price sealed-bid auction, bidders submit one bid in a concealed fashion. The submitted bids are then compared and the person with the highest bid wins the award, and pays the amount of his bid to the seller.

that nothing is learned from the protocol other than the output; correctness ensures the output is distributed according to the prescribed functionality. In short, with SMC protocols, people can use private information to their benefit while keep personal information in secret.

In reality, however, privacy does not have to be absolute. Other values, including security and social welfare, compete with it. Therefore, compromise can be made. In the report [Com00], the data access and confidentiality tradeoff is well articulated. The value of data-intensive research is highly variable, and it is impossible to specify a universally applicable optimal tradeoff between privacy and data access. The report calls for the development of tools that, on a case-by-case basis, would increase data access without compromising data protection or, conversely, increase confidentiality without compromising data access.

The **research problem of this thesis** is: How to design practical SMC protocols that are applicable to real world problems? By practicality, it means a SMC-based protocol that can be deployed with commodity hardware and being used in applications involves hundreds of users or more, such as a multi-party secure auction.

The example that a group of people want to find who the richest one is without revealing their wealth (like a multi-party version of Yao's millionaire problem), it actually is a variant of first-price auction, where the highest bid wins some item for sale. The first-price sealed-bid auction can be emulated by an unconditionally fully private protocol. However, the round complexity of which is exponential in the bid size, and there is no more efficient protocol [BS08].

The author's strategy to enable designing practical SMC solution is by exploring the scope of privacy protection. When it comes to practicality, people concern not only privacy but also efficiency of solutions. If it is not possible to construct protocols that are efficient enough in practice and that meet the given definitions of security, then one should search for a different definition of privacy that is more relaxed [LP09].

One example of relaxing the privacy protection is to allow the possibility of exposing some partial information, such as the relative richness between two bidders even neither of them are not

the final winner. This is considered a privacy breach in the fully private protocol where only the final winner is to be known as the richest.

The author designs a novel secure auction protocol based on secure scalar-product protocols, the time complexity of which is $O(k \cdot \lg m)$ where m is the number of bids and each bid has at most k bits. Although some information will be revealed during the proposed secure auction protocol, it is relatively efficient and secure for some degree.

"Why scalar-products?" It has been shown that the two-party scalar-product protocol is one of the most important building blocks in the Secure Two-party Computation [Du01]. There are two main reasons. First, the building block of generic circuit evaluation, the oblivious transfer, can be implemented by the scalar-product protocol. Second, it would be much more efficient if functions are constructed from scalar-products instead of binary gates. In this paper, the author proposes a series of secure two-party computation protocols based on scalar product protocols. In the development of the protocols, the author used a compositional approach with the scalar product protocols serving as building blocks. The research of Shen et al. [SZW$^+$07] demonstrated that people can use scalar products as building blocks for SMC. In this paper, the author shows that secure multi-party computation on any polynomial evaluation can be solved by the proposed building blocks based on the Scalar-product protocols and the help of additive sharing.

One classic setting for privacy-preserving data mining is that the data is divided among two or more different parties, and the goal is to run data mining algorithms on confidential data that is not supposed to be revealed. Clearly, SMC protocols can solve these problems. Although there are lots of work about how to preserve individual privacy, it alone is not enough to transform all data mining researches into privacy preserving ones. Therefore, considering different distributed databases scenarios, the author studies how to apply secure multi-party computation to existing privacy preserving data mining problems.

The **contributions of this thesis** are: 1) The author shows that SMC on any polynomial evaluation can be solved with the proposed scalar-product based protocols and the help of additive shar-

ing. 2) The author enables statistical analysis, e.g. Variance on additively shared or horizontally partitioned data, and privacy preserving distributed mining, including Naïve Bayesian classification and Nearest Neighbor on horizontally partitioned data, by designing secure protocols based on scalar-product protocols. 3) A novel secure auction protocol that is more practical by exploring the trade-off between the strength of privacy protection and the protocol complexity.

The remainder of this paper is organized as follows. In Chapter 2, the author gives a short review of scalar-product related SMC works, and SMC research for privacy preserving data mining and secure auction. Notations used through this paper are introduced and a series of secure two-party computation protocols based on Scalar-product are proposed in Chapter 3. Solutions to real world problems, including polynomial evaluation for SMC, and protocols for privacy preserving data mining algorithms are shown in Chapter 4. After that, the author designs a novel secure auction protocol and shows the performance evaluation in Chapter 5. Finally, the author concludes this paper and lays out the future work in Chapter 6.

Chapter 2

Related Work

About Secure Multi-party Computation

Secure multi-party computation (SMC) is a sub field of cryptography. The goal of SMC is to enable parties to cooperatively compute a function over their inputs while keeping their inputs private.

Secure computation was first formally introduced by Yao about thirty years ago as secure two-party computation [Yao82]. He proposed the problem: "Two millionaires wish to know who is richer; however, they do not want to find out inadvertently any additional information about each other's wealth. How can they carry out such a conversation?" This problem was later called Yao's *millionaire problem*.

In 1986, Yao suggested a secure two-party function evaluation technique which can be applied to solve a general class of two-party cryptographic problems [Yao86]. Later, Goldreich et al. extended this to any multi-party function [GMW87]. The generic method is based on expressing the function $f(x_1, \cdots, x_n)$ as a circuit (containing gates and wires) and encrypting the gates for secure evaluation.

Yao's Secure Two-party Computation Protocol

The following is a high-level description of Yao's secure two-party protocol. A detailed proof of security of this protocol can be found in [LP04].

First let's see how to evaluate a circuit. Let x and y be the parties' respective inputs and let $f(\cdot,\cdot)$ be a polynomial-time deterministic function. To compute $f(x,y)$, the first step is to view the function $f(\cdot,\cdot)$ as a boolean circuit $C(\cdot,\cdot)$. Then, $C(x,y)$ is computed gate-by-gate from the input wires to the output wires. That is, the *circuit-input wires* of $C(\cdot,\cdot)$ receives the input values x and y and the *circuit-output wires* of $C(\cdot,\cdot)$ carry the desired value $C(x,y)$. Once the *gate-input wires* of a gate g have obtained values $\sigma,\tau \in \{0,1\}$, it is possible to give the *gate-output wires* of gate g the value $g(\sigma,\tau)$. Thus, computing a circuit involves allocating appropriate zero-one values to the wires of the circuit.

In a secure protocol, the only value learned by a party should be its output. Therefore, the central idea of Yao's protocol [Yao86] is to provide a method of computing a circuit to get the value of circuit-output wires while values on all other wires are kept in secret. In this method, two random values are specified for every wire to represent 0 and 1. For instance, let w be the label of some wire. Then, randomly choose two values k_w^0 and k_w^1, where k_w^σ represents the bit value $\sigma \in \{0,1\}$. Since both k_w^0 and k_w^1 are identically distributed, no party can determine if $\sigma = 0$ or 1 even obtaining the value of k_w^σ.

Let g be a gate with incoming wires w_1 and w_2 and the output wire w_3. Given two random values k_1^σ and k_2^τ, the following shows how to compute the output value $k_3^{g(\sigma,\tau)}$ ($g(\sigma,\tau) = 0$ or 1) of w_3. First, we need *Garbled-Truth-Table*s that map the random input values to random output values. In addition, we can get the output value $k_3^{g(\sigma,\tau)}$ while keep $k_3^{1-g(\sigma,\tau)}$ in secret by viewing the four possible inputs of gate g, which are $k_1^0, k_1^1, k_2^0, k_2^1$, as encryption keys. Besides that, the output values k_3^0 and k_3^1, which are also keys, are encrypted under the appropriate keys from the incoming wires.

For example, let g be an XOR gate. Then, the key k_3^1 is encrypted under the pairs of keys

associated with the input values $(1,0)$ and $(0,1)$. In contrast, the key k_3^0 is encrypted under the pairs of keys associated with $(1,1)$ and $(0,0)$. Table 2.1 and Table 2.2 show the XOR gate and the garbled XOR gate respectively.

input wire w_1	input wire w_2	output wire w_3
1	1	0
1	0	1
0	1	1
0	0	0

Table 2.1: XOR gate

input wire w_1	input wire w_2	output wire w_3	Garbled-Truth-Table
k_1^1	k_2^1	k_3^0	$E_{k_1^1}(E_{k_2^1}(k_3^0))$
k_1^1	k_2^0	k_3^1	$E_{k_1^1}(E_{k_2^0}(k_3^1))$
k_1^0	k_2^1	k_3^1	$E_{k_1^0}(E_{k_2^1}(k_3^1))$
k_1^0	k_2^0	k_3^0	$E_{k_1^0}(E_{k_2^0}(k_3^0))$

Table 2.2: Garbled XOR gate

Then, given the Garbled-Truth-Table (the fourth column of Table 2.2) along with input wire keys k_1^σ and k_2^τ corresponding to σ and τ, it is possible to decrypt and obtain the output wire key $k_3^{g(\sigma,\tau)}$ while keep other three keys in the Garbled-Truth-Table in secret.[1] There are several possible ways to ensure correctness in the decryption of a gate. For example, the explicit (and randomly permuted) indices method described in [NPS99a] and the (G, E, D) private-key encryption [LP04].

The above described how to construct a single garbled gate. A garbled circuit consists of garbled gates along with *Output-Decryption-Table*s. The Output-Decryption-Tables map the random values on circuit-output wires back to their corresponding real values. For a circuit-output wire w, the corresponding Output-Decryption-Table contains the pairs $(0, k_w^0)$ and $(1, k_w^1)$ which help to convert the output value k_w^γ of a circuit-output wire to actual output bit. (See Table 2.3.)

Therefore, given the keys associated with inputs x and y, it is possible to compute the entire

[1]Note that the values of the Garbled-Truth-Table should be randomly ordered so that the key's position does not reveal the associated input wire's actual values.

Output-Decryption-Table
$(0, k_w^0)$
$(1, k_w^1)$

Table 2.3: The Output-Decryption-Table of the circuit-output wire w

circuit gate-by-gate and obtain the keys on circuit-output wires. Then, with Output-Decryption-Tables, these keys on circuit-output wires can be "decrypted" to the actual result of $C(x, y)$.

Informally describing Yao's protocol, one of the parties is the sender, who constructs and sends a garbled circuit to the other party, henceforth the receiver. Then the receiver interacts with the sender to obtain the input-wire keys associated with the inputs x and y. (Next paragraph will show how they "interact" with each other.) Given these keys, the receiver then computes the circuit as described, obtains the output and concludes the protocol.

Let's see how the receiver obtains the keys for the circuit-input wires. The sender directly sends the values corresponding to its input to the receiver. That is, if its i^{th} input bit is 1 and the wire w_j receives this input, then the sender just hands the receiver the string k_j^1. The receiver cannot learn anything about the sender's input from these keys since all of the keys are identically distributed. However, the sender cannot hand all of the keys pertaining to the receiver's input since the receiver can learn more than $C(x, y)$. That is, given the keys for the sender's input x, doing so the receiver can compute $C(x, \hat{y})$ for every possible $\hat{y}$. In addition, the receiver does not want the sender to know which keys to send it because then the sender would learn the receiver's input. The solution is to use a 1-out-of-2 oblivious transfer protocol [EGL85, Rab81]. In the protocol, the sender has two values $\{k_w^0, k_w^1\}$ for some circuit-input wire w, and the receiver has a bit $\sigma \in \{0, 1\}$ corresponding to its appropriate input bit. The outcome of the protocol is that the receiver obtains k_w^σ but does not know $k_w^{1-\sigma}$ while the sender has no idea what the receiver's input σ is. By having the receiver obtain its keys in this way, the sender learns nothing of the receiver's input value, and the receiver obtains only a single set of keys and so can compute the circuit on only a single value,

as required. This completes the high-level description of Yao's protocol.

More specifically, assume Alice and Bob want to jointly execute a secure protocol based on Yao's circuit evaluation method [Yao86] to compute $f(x_a, x_b)$ where $x_a \in \{0,1\}^n$ and $x_b \in \{0,1\}^n$ are private inputs for Alice and Bob respectively. Let x_{a_i} and x_{b_i} be the i^{th} bit of x_a and x_b respectively, for $i = 1$ to n. The following describes Yao's protocol [MNPS04]:

1. Bob converts f to circuit C.

2. Bob constructs the garbled circuit $G(C)$:

   (a) For each wire $w_i \in C$, Bob randomly generates two n-bit strings, k_i^0 and k_i^1 representing the bit 0 and 1 of w_i respectively. Bob also assigns to each wire $w_i \in C$ a random bit p_i for permutation, and computes $v_i^0 = k_i^0||(0 \oplus p_i)$, $v_i^1 = k_i^1||(1 \oplus p_i)$. Let v_i^0 and v_i^1 denote the final results.

   (b) For each circuit-output wire $w_s \in C$, Bob generates an Output-Decryption-Table ODT_s contains two entries, $(0, k_s^0)$ and $(1, k_s^1)$. (See Table 2.4.)

ODT_s
$(0, k_s^0)$
$(1, k_s^1)$

Table 2.4: The Output-Decryption-Table for circuit-output wire w_s

   (c) For each gate $g \in C$ with input wires w_i (with a bit value σ) and w_j (with a bit value τ), the output wire w_t, and the truth table $g(\cdot,\cdot) \in \{0,1\}$:

      i. Bob constructs a Garbled-Truth-Table (GTT) GTT_g by replacing every 0 with v_t^0 and every 1 with v_t^1 in the original truth table. (See Table 2.5.)

      ii. Bob constructs the Encrypted-Garbled-Truth-Table (EGTT) $EGTT_g$ for g:

      For entry (σ, τ) in GTT_g, define $\sigma' = \sigma \oplus p_i$ and $\tau' = \tau \oplus p_j$. Encrypt each entry in GTT_g using k_i^σ, k_j^τ as encryption keys and t, σ', τ' as an initialization

σ	τ	GTT_g
1	1	$v_t^{g(1,1)}$
1	0	$v_t^{g(1,0)}$
0	1	$v_t^{g(0,1)}$
0	0	$v_t^{g(0,0)}$

Table 2.5: The garbled gate g

vector (IV): $EGTT_g[\sigma,\tau]$ = $E_{k_i^\sigma,k_j^\tau,t,\sigma',\tau'}(GTT_g[\sigma,\tau])$. The encryption is done by hashing $k_i^\sigma||t||\sigma'||\tau'$ and $k_j^\tau||t||\sigma'||\tau'$ using SHA-1, and XORing the two results to the plaintext.[2]

σ	τ	σ'	τ'	$EGTT_g$
1	1	$1\oplus p_i$	$1\oplus p_j$	$E_{k_i^1,k_j^1,t,\sigma',\tau'}(v_t^{g(1,1)})$
1	0	$1\oplus p_i$	$0\oplus p_j$	$E_{k_i^1,k_j^0,t,\sigma',\tau'}(v_t^{g(1,0)})$
0	1	$0\oplus p_i$	$1\oplus p_j$	$E_{k_i^0,k_j^1,t,\sigma',\tau'}(v_t^{g(0,1)})$
0	0	$0\oplus p_i$	$0\oplus p_j$	$E_{k_i^0,k_j^0,t,\sigma',\tau'}(v_t^{g(0,0)})$

Table 2.6: The encrypted garbled gate g

iii. Bob constructs the Permuted-Encrypted-Garbled-Truth-Table (PEGTT) $PEGTT_g$ for g using the permutation bits p_i and p_j:

- If $p_i = 1$, then the first two entries of the table are swapped with the last two entries.
- If $p_j = 1$, then the first and the third entries are swapped with the second and the fourth entries.

3. Bob sends the garbled circuit $G(C)$, including Output-Decryption-Tables and Permuted-

[2]SHA-1 is a basic symmetric cryptographic function. The encoding of the circuit (garbling) can be implemented using a pseudo random function (as is described in detail, for example, in [NPS99a]), where the output of the function is used as a pad that masks the values in the table representing a gate in the circuit. The masking values used here are SHA-1($k_i^\sigma||t||\sigma'||\tau'$), SHA-1($k_j^\tau||t||\sigma'||\tau'$) for entry (σ,τ) of the table of gate number t, whose input wires are w_i and w_j.

Garbled-Truth-Tables, to Alice.[3]

4. Let $w_1, \ldots, w_n$ be the circuit-input wires corresponding to x_a, and let $w_{n+1}, \ldots, w_{2n}$ be the circuit-input wires corresponding to x_b. Then,

   (a) Bob sends the strings $v_{n+1}^{x_{b_1}}, \ldots, v_{2n}^{x_{b_n}}$ to Alice.

   (b) For $i = 1$ to n, Alice and Bob execute a 1-out-of-2 oblivious transfer protocol [EGL85, Rab81] in which Alice is the receiver with the input x_{a_i} and Bob is the sender with the input (v_i^0, v_i^1). At last, Alice gets $v_1^{x_{a_1}}, \ldots, v_n^{x_{a_n}}$ from Bob.[4]

5. After obtaining $2n$ strings corresponding to the $2n$ input wires to C in Step 4, Alice can individually evaluate the garbled circuit $G(C)$ gate-by-gate from the input wires to the output wires. Let g be a garbled gate having an input wire w_i with value v_i, the other input wire w_j with value v_j, and the output wire w_t. In addition, let the least significant bits of v_i, v_j be σ, τ and the rest of the bits be k_i, k_j respectively. For each such gate:

   (a) Alice uses σ and τ as indices into an entry to be decrypted in $PEGTT_g$.

   (b) Alice uses k_i, k_j as decryption-keys, and $t||\sigma||\tau$ as an IV. That is, Alice sets $v_t = D_{k_i,k_j,t,\sigma,\tau}(PEGTT_g[\sigma, \tau])$. The decryption is done by hashing $k_i||t||\sigma||\tau$ and $k_j||t||\sigma||\tau$ using SHA-1, and XORing the two results to the ciphertext.

6. Alice uses Output-Decryption-Tables to translate the result of $G(C(x_a, x_b))$ to $C(x_a, x_b)$. Therefore, Alice gets the result of $f(x_a, x_b)$.[5]

7. Alice sends the result of $f(x_a, x_b)$ to Bob.

[3]The structure of C is given, and the garbled circuit G(C) is simply defined by specifying the Output-Decryption-Tables and Permuted-Garbled-Truth-Tables that belong to each gate.

[4]Note that Alice learns nothing about $v_i^{1-x_{a_i}}$, and Bob learns nothing about x_{a_i}. Moreover, these n oblivious transfer protocols can all be executed in parallel.

[5]Only the correct gate output bit strings will be deciphered, and the output bit strings of current gates will be the input bit strings to the next gates until the final circuit outputs $f(x_a, x_b)$ come out. Though Alice has the intermediate bit strings, she cannot get extra information from them because the semantics of these bit strings are exclusive to Bob.

Other Related Work

Since Yao's [Yao82] and Goldreich et al.'s [GMW87] proposals for SMC, researchers have been looking for new complete functions and have been looking at the foundations of completeness. Kilian shows that the oblivious transfer is complete [Kil88], and so are the functions with imbedded OR [Kil91]. Furthermore, it is claimed that the integer-based scalar products are more practical to real applications than binary-based oblivious transfer [Du01].

Over the past decade, more and more proposals for the secure scalar products seem to be released each year. Du and Zhan [DZ02] proposed the invertible-matrix, enabling tradeoffs between efficiency and privacy. They also proposed a commodity-based approach based on Beaver's commodity model [Bea97]. Goethals et al. [GLLM04] proposed the computationally secure scalar-product protocol, the security of which depended on the intractability of the composite residuosity problem. The potential of scalar-product-based protocols have been demonstrated in [WSH+08], among which the commodity-based approach has extraordinary performance, though a neutral third party is necessary when it comes to two-party computation.

Moreover, much effort has gone into building various applications using secure scalar products. Atallah and Du reduced geometry problems to scalar products [AD00]. Du et al. constructed secure protocols for statistical analysis [DA01b] and scientific computations [DA01a]. In addition, Bunn and Ostrovsky [BO07] offered a secure k-means clustering protocol based on scalar products using the composite-residuosity approach. Zhan et al. [ZWH+08] have recently constructed an efficient privacy-preserving collaborative recommender system based on the scalar product protocol using Beaver's commodity model.

There are also plenty of theoretical studies on scalar products. Chiang et al. [CWLH05] proposed a privacy measurement based on information theory, with which they analyzed various scalar-product approaches. They proved that the invertible-matrix approach discloses at least half the information whereas the commodity-based approach is perfectly secure. Wang et al. [WLCH06] proved that no information-theoretically secure two-party protocol exist for scalar

products. Moreover, the closure property of the commodity-based approach is preliminarily verified according to the security definition based on information theory [SZW+07].

Recently, Nielsen in his dissertation [Nie09] designed and implemented a domain-specific programming language for secure multiparty computation, the Secure Multiparty Computation Language (SMCL). He demonstrated the usefulness of SMCL by reporting on how an SMCL program contributed to the first large-scale practical application of secure multiparty computation, bidding in the sugar beet market in Denmark.

There are some researches about how to apply SMC approaches to privacy-preserving data mining. Lindell and Pinkas [LP09] list basic paradigms of SMC and discuss their relevance to the field of privacy-preserving data mining. In addition, they discuss the issue of efficiency and demonstrate the difficulties involved in constructing highly efficient protocols. Which problems can or cannot be solved by SMC are also discussed. Cuzzocrea and Bertino [CB10] proposed an efficient SMC-based privacy preserving online analytical processing (OLAP) framework for distributed collections of XML documents.

For secure auction, Franklin and Reiter proposed the first scheme [FR96]. The auction service provides protection for both the auction house and correct bidders. At the end of the auction, however, it opens all bids to determine the winning bid at the end of an auction. Naor et al. combined Yao's secure computation with oblivious transfer to achieve a secure second price auction assuming two types of auctioneers [NPS99b]. However, the communication cost between bidders is high since it proceeds bit by bit using the oblivious transfer protocol. Juels and Szydlo improved the efficiency with two types of auctioneers through verifiable proxy oblivious transfer (VPOT) [JS02], but all bids can be retrieved if both types of auctioneers collaborate with each other. Brandt and Sandholm investigated how to obtain bid privacy in sealed-bid auctions [BS08]. They pointed out the first-price sealed-bid auction can be emulated by an unconditionally fully private protocol, the round complexity of which, however, is exponential in the bid size, and there is no more efficient protocol. In addition, they proved the impossibility of fully privately emulating the second-price

sealed-bid (Vickrey) auction for more than two bidders.

Chapter 3

Building Block Designs for Secure Two-party Computation

In this chapter, the author proposes a series of build blocks for secure two-party computation based on secure scalar-product protocols. It has been shown that the two-party scalar-product protocol is one of the most important building blocks in the Secure Two-party Computation [Du01]. There are two main reasons. First, the building block of generic circuit evaluation, the oblivious transfer, can be implemented by the scalar-product protocol. Second, it would be much more efficient if functions are constructed from scalar-products instead of binary gates.

With proper composition of the building blocks, most applications, if not all, can be executed securely and collaboratively by potentially dishonest parties. Before introducing the building blocks, the author gives the following definitions.

Definition ($NoInformationLeakage$): Let x and y be two random variables. y leaks no information about x when $H(x|y) = H(x)$, where $H(\cdot)$ denotes the Shannon entropy [CT91].

Definition ($Information-theoreticallySecure$): Let π be a multiparty protocol, and let VIEW$_i^\pi$ be the view of party i during an execution of π. VIEW$_i^\pi$ includes party i's private input x_i, lo-

cal variables, internal coin tosses, received messages, and outputs. Protocol π is **information-theoretically secure** if the information about x_i is not leaked during an execution of π from any other party's perspective if

$$H(x_i|VIEW_j^{\pi}) = H(x_i|x_j), \text{for all } j \neq i.$$

Note that the field of secure multi-party (or two-party) computation focuses on how to securely compute a functionality, but does not deal with whether the functionality should be computed in the first place. By using SMC protocols, it is possible to compute any functionality without revealing anything beyond the output. How much information about the input is revealed by that output is not the concerned topic in this paper.

The construction principles are stated as follows.

1. The author uses bottom-up construction strategy to design all building blocks based on the Scalar-product protocol.

2. If a protocol of which the participants only compute individually is composed of building blocks, the protocol becomes another building block. In other words, every building block can be decomposed to Scalar-product protocols and individual computation.

3. Communication between participants is forbidden during protocol composition.

4. Since the construction is information-theoretically secure, the security strength relies on the security of the underlying Scalar-product implementation.

Any building blocks directly composed of the Scalar-product protocols are defined primitives, while others are defined as useful building blocks. Figure 3.1 shows the building block construction.

Notations used hereafter are introduced in Section 3.1. Primitive, useful building blocks, and general solutions are proposed in Section 3.2, 3.3, and 3.4 respectively. Since all the building

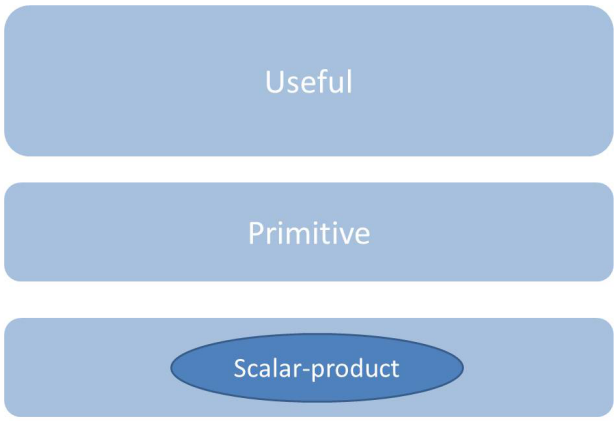

Figure 3.1: Building block construction

blocks introduced in this chapter are composed of other building blocks and based on the Scalar-product protocols, they can be decomposed to different times of Scalar-product protocols with different domain and dimension. Therefore, people can estimate the execution time of every building block. The detail complexity analysis and performance evaluation can be found in [WCH+09].

3.1 Preliminaries

For a secure m-party problem ($m \geq 2$), the notation

$$(X_1, X_2, \cdots, X_m)\{plist\} \mapsto (Y_1, Y_2, \cdots, Y_m)$$

represents that Party j has the private input X_j, for $j = 1$ to m. After the execution of some protocol, Party j gets the private output Y_j, for $j = 1$ to m. There might be a list of public variables, $plist$, which are known by all parties. For clarity purpose, subscripts in this paper indicate the data owner.[1] The domain $\mathbb{Z}_n$ denotes a ring consisting of elements $\{0, \ldots, n-1\}$,

[1] Only in Section 4.3.1 the superscripts indicate the data owner.

and the results of addition and multiplication in $\mathbb{Z}_n$ are the modular summation and the modular product. If not stated otherwise, the computations of secure proposals are over $\mathbb{Z}_n$, where n is two's power, namely, $n = 2^{k+1}, k \in \mathbb{N}$. In order to extend the domain from natural numbers to integers, elements $\{1, \ldots, \lfloor \frac{n-1}{2} \rfloor\}$ remain positive numbers, while elements $\{n-1, \ldots, n - \lfloor \frac{n-1}{2} \rfloor\}$ are interpreted as negative integers analogous to the binary system in modern computers. Therefore, the subtraction to t is equivalent to the addition to $(n - t)$.

$$x - t \pmod{n} = x + n - t \pmod{n}.$$

In this paper, there are two different concepts of the "scalar product." When it comes to lower case letters, it means all secure scalar product approaches; when it comes to capitalized words, Scalar-product, it means one of these secure scalar product approaches. The formulation for the Scalar-product protocol follows Goldreich's principle [Gol04]: The intermediate results during protocol execution are always shared among participants. In a protocol π composed of Scalar-products, current outputs can be inputs to the next Scalar-product, which are actually the intermediate results of π and should be shared. The intermediate results are always additive shared by parties, which has been proven to be perfect in ring $\mathbb{Z}_n$ [SZW$^+$07].[2] The secure Scalar-product protocol is specified as

Specification 1 (Scalar-product) *Party 1 and Party 2 want to collaboratively execute the secure protocol*

$$((x[1]_1, \ldots, x[d]_1), (x[1]_2, \ldots, x[d]_2))\{d\} \mapsto (y_1, y_2)$$

such that $y_1 + y_2 = x[1]_1 \cdot x[1]_2 + \cdots + x[d]_1 \cdot x[d]_2$ *and* $x[i]_1, x[i]_2, y_1, y_2 \in \mathbb{Z}_n$, *for* $i = 1, \ldots, d$.

Here the author merely specifies Scalar-product instead of providing a concrete approach because the focus is on building more protocols on top of Scalar-product. Similar to the software

[2] In ring $\mathbb{Z}_n$, the multiplicative sharing reveals information when either of the values of the shares is zero.

specification, as long as a new subroutine matches the interface, it can replace the old one and work perfectly. In the scalar-product based protocols, as long as a new solution matches the Specification 1, it can be used as the building block of the author's proposed protocols. A scalar-product based composition theory is proved for semi-honest adversary models [Can00], so the designed protocols preserve the entropy of the secret inputs as strong as the underlying scalar product protocol preserves the entropy of its inputs.

In passing, it should be noted that the author does not deal with the problem of combining the additively shared output variables to produce the final results, since the designed building blocks can be used to build even larger protocols. The step of combining shared variables to produce the final results does not come until after computation is completed. Therefore, it is unnecessary to combine intermediate variables.

3.2 Primitive building blocks

Four primitive building blocks are specified, which are $\mathbb{Z}_n$-to-$\mathbb{Z}_2$, $\mathbb{Z}_2$-to-$\mathbb{Z}_n$, Product, and Square.

3.2.1 $\mathbb{Z}_n$-to-$\mathbb{Z}_2$ and $\mathbb{Z}_2$-to-$\mathbb{Z}_n$

$\mathbb{Z}_n$-to-$\mathbb{Z}_2$ and $\mathbb{Z}_2$-to-$\mathbb{Z}_n$ convert to and fro between $\mathbb{Z}_n$ sharing and bitwise $\mathbb{Z}_2$ sharing. In addition to be primitive building blocks, these two protocols establish the possibility of secure computation for all functions. Although it may be inefficient, the author can always apply Yao's circuit evaluation idea after the $\mathbb{Z}_n$-to-$\mathbb{Z}_2$ protocol and followed by the $\mathbb{Z}_2$-to-$\mathbb{Z}_n$ protocol. These two protocols make the author's proposal as general as the classic circuit approaches.

Specification 2 ($\mathbb{Z}_n$-to-$\mathbb{Z}_2$) *Party 1 and Party 2 share a number in $\mathbb{Z}_n$, and they want to securely convert the $\mathbb{Z}_n$ sharing into bitwise $\mathbb{Z}_2$ sharing. More specifically, Party 1 and Party 2 want to collaboratively execute the secure protocol $(x_1, x_2) \mapsto ((y_1^0, \ldots, y_1^k), (y_2^0, \ldots, y_2^k))$ such that $(y^k y^{k-1} \cdots y^1 y^0)_2 = x_1 + x_2$, where $x_1, x_2 \in \mathbb{Z}_n$, $y_1^i, y_2^i \in \mathbb{Z}_2$, and $y^i = y_1^i + y_2^i \ (mod\ 2)$.*

To convert from $\mathbb{Z}_n$ sharing to bitwise $\mathbb{Z}_2$ sharing, the author emulates the carry ripple adder with binary Scalar-product protocol, whose $n = 2$. Let $x_1 = (x_1^k \cdots x_1^0)_2$, $x_2 = (x_2^k \cdots x_2^0)_2$, and the adder operates as long addition:

$$\begin{array}{cccccc} & c^{k+1} & c^k & \cdots & c^1 & c^0 \\ & & x_1^k & \cdots & x_1^1 & x_1^0 \\ +) & & x_2^k & \cdots & x_2^1 & x_2^0 \\ \hline & & y^k & \cdots & y^1 & y^0 \end{array}$$

where $c^0 = 0$ and $c^{i+1} = c^i x_1^i + c^i x_2^i + x_1^i x_2^i \pmod 2$ are the carry bits; $y^i = c^i + x_1^i + x_2^i \pmod 2$ is the i-th summation bit. Next, the $\mathbb{Z}_n$-to-$\mathbb{Z}_2$ protocol is as follows:

PROTOCOL $\mathbb{Z}_n$-to-$\mathbb{Z}_2$ ($n = 2^{k+1}$)

1. Party j sets $c_j^0 = 0$, and $y_j^0 = x_j^0$, for $j = 1, 2$.
2. For $i = 0, \ldots, k-1$, repeat Step 2a to Step 2b.[3]
   (a) The two parties jointly execute the binary Scalar-product protocol

$$((c_1^i, x_1^i, x_1^i), (x_2^i, c_2^i, x_2^i)) \mapsto (t_1^i, t_2^i),$$

   where

$$t_1^i + t_2^i \pmod 2 = c_1^i x_2^i + x_1^i c_2^i + x_1^i x_2^i \pmod 2.$$

   (b) For $j = 1, 2$, Party j computes

$$\begin{aligned} c_j^{i+1} &= c_j^i x_j^i + t_j^i \pmod 2 \\ y_j^{i+1} &= x^{j\,i+1} + c_j^{i+1} \pmod 2 \end{aligned}$$

[3]Since $n = 2^{k+1}$, the overflow bit c^{k+1} is discarded.

Specification 3 ($\mathbb{Z}_2$-to-$\mathbb{Z}_n\{k'\}$) *Party 1 and Party 2 share a number in $\mathbb{Z}_2$, and they want to securely convert the bitwise $\mathbb{Z}_2$ sharing into the $\mathbb{Z}_n$ sharing. More specifically, the two parties want to execute the secure protocol $((x_1^0, \ldots, x_1^{k'}), (x_2^0, \ldots, x_2^{k'})) \mapsto (y_1, y_2)$ such that $y_1 + y_2 = (x^{k'} x^{k'-1} \cdots x^1 x^0)_2$, where $n = 2^{k+1}$, $k' \leq k$, $y_1, y_2 \in \mathbb{Z}_n$, $x_1^i, x_2^i \in \mathbb{Z}_2$, and $x^i = x_1^i + x_2^i \ (mod\ 2)$, for $i = 0, 1, \cdots, k' - 1$.*

According to the above requirement, the outputs can be rewritten as the following function:

$$\begin{aligned} y_1 + y_2 =& \sum_{i=0}^{k'} x^i \cdot 2^i = \sum_{i=0}^{k'} (x_1^i + x_2^i \bmod 2) \cdot 2^i \\ =& \sum_{i=0}^{k'} (x_1^i + x_2^i - 2x_1^i x_2^i) \cdot 2^i \\ =& \sum_{i=0}^{k'} x_1^i \cdot 2^i + \sum_{i=0}^{k'} x_2^i \cdot 2^i - \sum_{i=0}^{k'} x_1^i x_2^i \cdot 2^{i+1} \end{aligned}$$

The principle of the above function rewrite is to divide the computation into two parts: individually computable part ($\sum x_1^i \cdot 2^i$ and $\sum x_2^i \cdot 2^i$) and the collaboration part ($\sum x_1^i x_2^i \cdot 2^{i+1}$).

PROTOCOL $\mathbb{Z}_2$-to-$\mathbb{Z}_n\{k'\}$ ($n = 2^{k+1}$)

1. Party 1 and Party 2 jointly run the Scalar-product protocol $((2x_1^0, \ldots, 2^{k'+1} x_1^{k'}), (x_2^0, \ldots, x_2^{k'})) \mapsto (t_1, t_2)$ such that $t_1 + t_2 = 2x_1^0 \cdot x_2^0 + \cdots + 2^{k'+1} x_1^{k'} \cdot x_2^{k'}$.
2. Party j computes $y_j = \sum_{i=0}^{k'} x_j^i \cdot 2^{k'} - t_j$, for $j = 1, 2$.

3.2.2 Product

To compute a polynomial function collaboratively, two-party addition and multiplication are essential. Since the principle of the additive sharing is adopted, the two-party addition is trivial. However, to execute a secure two-party multiplication, it is necessary to use the Scalar-product protocol.

Specification 4 (Product) *Party 1 and Party 2 additively share the multiplicand and the multiplicator. They want to securely execute the protocol* $((x_1, y_1), (x_2, y_2)) \mapsto (z_1, z_2)$ *such that* $z_1 + z_2 = (x_1 + x_2)(y_1 + y_2)$.

With a little modification, the outputs can be rewritten as

$$z_1 + z_2 = x_1y_1 + x_2y_2 + (x_1y_2 + y_1x_2).$$

After the above factoring, it is obvious that x_1y_1 and x_2y_2 are individually computable for Party 1 and Party 2 respectively. However, the other terms should be computed via the Scalar-product protocol. The following protocol describes the details.

PROTOCOL Product

1. Party 1 and Party 2 jointly execute the Scalar-product protocol $((x_1, y_1), (y_2, x_2)) \mapsto (t_1, t_2)$ such that $t_1 + t_2 = x_1y_2 + y_1x_2$.
2. Party j individually computes $z_j = t_j + x_jy_j$, for $j = 1, 2$.

3.2.3 Square

The Square protocol, which is very similar to the Product protocol and employs the same strategy, namely dividing the computation into the part which can be done individually and the rest which must be performed collaboratively. However, the Square protocol reduces the dimension of the scalar product from two to one.

Specification 5 (Square) *Party 1 and Party 2 share a number in* $\mathbb{Z}_n$*, and they want to collaboratively execute the secure protocol* $(x_1, x_2) \mapsto (y_1, y_2)$ *such that* $y_1 + y_2 = (x_1 + x_2)^2$.

The protocol details are as follows:

PROTOCOL Square

1. Party 1 and Party 2 jointly execute the Scalar-Product protocol $(x_1, x_2) \mapsto (t_1, t_2)$ such that $t_1 + t_2 \pmod{n} = x_1 \cdot x_2$.

2. Party j individually computes $y_j = x_j^2 + 2t_j$, for $j = 1, 2$.

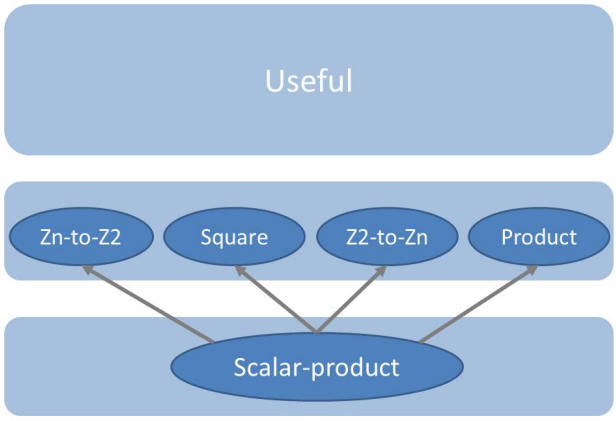

Figure 3.2: The relationship between the Scalar-product protocol and primitive building blocks

3.3 Useful building blocks

Based on primitive building blocks introduced in Section 3.2, two useful protocols are proposed here, which are Comparison and If-Then-Else.

3.3.1 Comparison

There are many variations of binary comparison: less than ($<$), greater than ($>$), less than or equal to ($\leq$), grater than or equal to ($\geq$), and equal to ($=$). However, all of them are reducible to the less than operator ($<$). In order to compare two numbers a and b, it is intuitive to compare $(a - b)$ and 0 since the intermediate results are additively shared; at the same time, it is effortless to subtract

under additive sharing. The author's proposal to compare a and b is to compute the most significant bit of $(a-b)$. According to the binary system on modern computers, if the most significant bit of $(a-b)$ is 1, $(a-b)$ is a negative number inferring that a is less than b.

Specification 6 (Comparison) *Party 1 and Party 2 share a number in $\mathbb{Z}_n$, and they want to know the sign of the number. In other words, they want to collaboratively execute the secure protocol $(x_1, x_2) \mapsto (y_1, y_2)$ such that*

$$y_1 + y_2 = \begin{cases} 1 & \textit{if } x_1 + x_2 < 0, \\ 0 & \textit{otherwise.} \end{cases}$$

The protocol details are as follows:

PROTOCOL Comparison

1. Two parties collaboratively execute the $\mathbb{Z}_n$-to-$\mathbb{Z}_2$ protocol $(x_1, x_2) \mapsto ((b_1^0, \ldots, b_1^k), (b_2^0, \ldots, b_2^k))$, such that $b^i = b_1^i + b_2^i \pmod 2$, and $(b^k \cdots b^0)_2 = x_1 + x_2$.

2. Party 1 and Party 2 collaboratively execute the $\mathbb{Z}_2$-to-$\mathbb{Z}_n\{0\}$ protocol $(b_1^k, b_2^k) \mapsto (y_1, y_2)$, such that $y_1 + y_2 = (b^k)_2$ and $b^k = b_1^k + b_2^k \pmod 2$.

From the protocol, we know that the Comparison protocol is composed of $\mathbb{Z}_n$-to-$\mathbb{Z}_2$ and $\mathbb{Z}_2$-to-$\mathbb{Z}_n\{0\}$. Table 3.3.1 lists the constituents of some building blocks introduced in this section. The author further breaks those constitutive building blocks in Table 3.3.1 into Scalar-product protocols of which Table 3.3.1 shows the domain, dimension, and times. (Note that $2^{k+1} = n$.) If each Scalar-product protocol needs $O(1)$ time of communication between Party 1 and Party 2 , then the complexity of communication times of the Comparison protocol is $O(k)$.

Protocol	Domain	Constituents	Times
$\mathbb{Z}_n$-to-$\mathbb{Z}_2$	$\mathbb{Z}_2$	Scalar-Product	k
$\mathbb{Z}_2$-to-$\mathbb{Z}_n\{k'\}$	$\mathbb{Z}_n$	Scalar-Product	1
Comparison	$\mathbb{Z}_2$	$\mathbb{Z}_n$-to-$\mathbb{Z}_2$	1
	$\mathbb{Z}_n$	$\mathbb{Z}_2$-to-$\mathbb{Z}_n\{0\}$	1

Table 3.1: Constituents of the proposed protocols

Protocol	Domain	Dimension	Times
$\mathbb{Z}_n$-to-$\mathbb{Z}_2$	$\mathbb{Z}_2$	3	k
$\mathbb{Z}_2$-to-$\mathbb{Z}_n\{k'\}$	$\mathbb{Z}_n$	$k'+1$	1
Comparison	$\mathbb{Z}_2$	3	k
	$\mathbb{Z}_n$	1	1

Table 3.2: The complexity of the proposed protocols

3.3.2 If-Then-Else

The If-Then-Else protocol is useful for functions with alternatives.

Specification 7 (If-Then-Else) *Party 1 and Party 2 additively share the predicate, IF-clause value, and the ELSE-clause value. They want to securely execute the protocol* $((b_1, x_1, y_1), (b_2, x_2, y_2)) \mapsto (z_1, z_2)$ *such that*

$$z_1 + z_2 = \begin{cases} x_1 + x_2 & \textit{if} \quad b_1 + b_2 = 1, \\ y_1 + y_2 & \textit{if} \quad b_1 + b_2 = 0. \end{cases}$$

According to the above requirement, the outputs can be rewritten as the following function:

$$\begin{aligned} z_1 + z_2 =& (b_1 + b_2)(x_1 + x_2) + (1 - b_1 - b_2)(y_1 + y_2) \\ =& (y_1 + y_2) + (b_1 + b_2)(x_1 - y_1 + x_2 - y_2) \end{aligned}$$

Again, with the strategy dividing the components into individually computable ones and those need the Scalar-Product protocols, the author proposes the following If-Then-Else protocol.

PROTOCOL If-Then-Else

1. Party j individually computes $s_j = x_j - y_j$, for $j = 1, 2$.
2. Party 1 and Party 2 collaboratively execute a Product protocol $((b_1, s_1), (b_2, s_2)) \mapsto (t_1, t_2)$ such that $t_1 + t_2 = (b_1 + b_2)(s_1 + s_2)$.
3. Party j individually computes $z_j = t_j + y_j$, for $j = 1, 2$.

If both of the alternatives are not additively shared by parties, then no secure protocol is needed. The following shows details of this point.

Specification 8 (If-Then-Else2) *Party 1 and Party 2 additively share the predicate, but the IF-clause value and the ELSE-clause value are known by these two parties, i.e. the alternatives are not additively shared by these two parties. They want to securely execute the protocol* $(b_1, b_2)\{x, y\} \mapsto (z_1, z_2)$ *such that*

$$z_1 + z_2 = \begin{cases} x & \textit{if} \quad b_1 + b_2 = 1, \\ y & \textit{if} \quad b_1 + b_2 = 0. \end{cases}$$

According to the above requirement, the outputs can be rewritten as the following function:

$$\begin{aligned} z_1 + z_2 =& (b_1 + b_2) \cdot x + (1 - b_1 - b_2) \cdot y \\ =& y + (b_1 + b_2)(x - y) \end{aligned}$$

This function actually can be computed locally, so no secure protocols are needed.

PROTOCOL If-Then-Else2

1. Party 1 and Party 2 individually compute $z_1 = y + b_1 \cdot (x - y)$ and $z_2 = b_2 \cdot (x - y)$ respectively.

3.3.3 Division/Remainder

Following the integer division in modern computers, the author uses the algorithm for long division to emulate a $(k+1)$-bit divider.

Given two integers $x, y \in \mathbb{N}$, the integer division is defined as follows:

$$\left\lfloor \frac{x}{y} \right\rfloor = q, \text{where } x = y \cdot q + r, \text{and } 0 \leq r < y.$$

During the computation of $\left\lfloor \frac{x}{y} \right\rfloor$, check iteratively whether $x \geq y \cdot 2^i$, for $i = k-1, \ldots, 0$. If it is true, $x = x - y \cdot 2^i$ and the i-th bit of q is 1; otherwise, x remains untouched and the i-th bit of q is 0.

Note that computing $y \cdot 2^i$ in $\mathbb{Z}_n$ gives unpredictable values, as a result double precision is necessary to represent $y \cdot 2^i$ correctly, for $i = k-1, \ldots, 0$. To do so, both the dividend and the divisor are converted from $\mathbb{Z}_n$ sharing to Z_{n^2} sharing, by which $y \cdot 2^i$ can always be correctly represented.

Based on the Comparison, If-Then-Else, $\mathbb{Z}_2$-to-$\mathbb{Z}_n$, and $\mathbb{Z}_n$-to-$\mathbb{Z}_2$ protocols, the author specifies the following Div/Rem protocol, which represents "Division/Remainder."

Specification 9 (Div/Rem) *Two parties share the dividend and the divisor in $\mathbb{Z}_n$, and they want to jointly execute the secure protocol $((x_1, y_1), (x_2, y_2)) \mapsto ((q_1, r_1), (q_2, r_2))$, where $x_1 + x_2 = (y_1 + y_2)(q_1 + q_2) + (r_1 + r_2)$ and $0 \leq (r_1 + r_2) < (y_1 + y_2)$.*

PROTOCOL Div/Rem

1. Party 1 and Party 2 collaboratively execute the $\mathbb{Z}_n$-to-$\mathbb{Z}_2\{k\}$ protocol followed by the $\mathbb{Z}_2$-to-$\mathbb{Z}_{n^2}\{k\}$ protocol

   $$(x_1, x_2) \mapsto ((b_1^0, \ldots, b_1^k), (b_2^0, \ldots, b_2^k)) \mapsto (X_1, X_2)$$

   such that $X_1 + X_2 \pmod{n^2} = x_1 + x_2 \pmod{n}$.

Algorithm 1 Calculate $q = \lfloor \frac{x}{y} \rfloor, r = x - y \cdot q$

$q \leftarrow 0$
for $i \leftarrow k-1, k-2, \cdots, 0$ **do**
  if $x \geq y \cdot 2^i$ **then**
    $x \leftarrow x - y \cdot 2^i$
    $bit \leftarrow 1$
  else
    $bit \leftarrow 0$
  end if
  $q \leftarrow q + bit \cdot 2^i$
end for
$r \leftarrow x$

2. Party 1 and Party 2 collaboratively execute the $\mathbb{Z}_n$-to-$\mathbb{Z}_2\{k\}$ protocol followed by the $\mathbb{Z}_2$-to-$\mathbb{Z}_{n^2}\{k\}$ protocol

$$(y_1, y_2) \mapsto ((c_1^0, \ldots, c_1^k), (c_2^0, \ldots, c_2^k)) \mapsto (Y_1, Y_2)$$

such that $Y_1 + Y_2 \pmod{n^2} = y_1 + y_2 \pmod{n}$.

3. Party j sets $X_j^k = X_j$, for $j = 1, 2$.

4. For $i = k-1, k-2, \ldots, 0$, repeat Step 4a to Step 4d.

   (a) Party j computes $t_j^i = X_j^{i+1} - Y_j \cdot 2^i \pmod{n^2}$, for $j = 1, 2$.

   (b) Party 1 and Party 2 jointly run the Comparison protocol $(t_1^i, t_2^i) \mapsto (s_1^i, s_2^i)$ such that

$$s_1^i + s_2^i \pmod{n^2} = \begin{cases} 1 & \text{if } t_1^i + t_2^i < 0, \\ 0 & \text{otherwise.} \end{cases}$$

   (c) Party j individually computes $q_j^i = 1 - s_j^i \pmod{n^2}$, for $j = 1, 2$, such that

$$q_1^i + q_2^i \pmod{n^2} = \begin{cases} 0 & \text{if } s_1^i + s_2^i = 1, \\ 1 & \text{if } s_1^i + s_2^i = 0. \end{cases}$$

(d) Party 1 and Party 2 run the If-Then-Else protocol $((s_1^i, X_1^{i+1}, t_1^i), (s_2^i, X_2^{i+1}, t_2^i)) \mapsto (X_1^i, X_2^i)$ such that

$$X_1^i + X_2^i \pmod{n^2} = \begin{cases} X_1^{i+1} + X_2^{i+1} & \text{if } s_1^i + s_2^i = 1, \\ t_1^i + t_2^i & \text{if } s_1^i + s_2^i = 0. \end{cases}$$

5. For $j = 1, 2$, Party j computes $r_j = X_j^0 \pmod{n}$, and $q_j = \sum_{i=0}^{k-1} q_j^i \cdot 2^i \pmod{n}$.

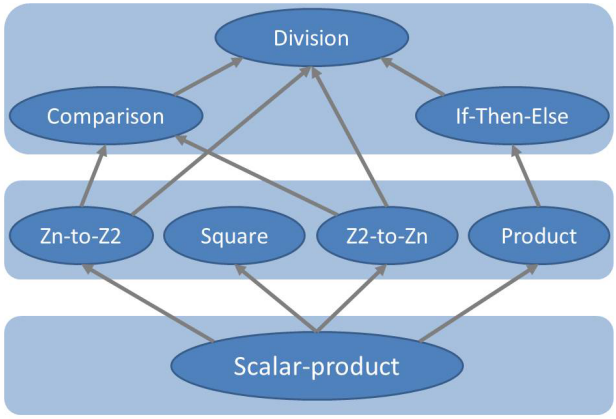

Figure 3.3: The relationship between useful building blocks and others

3.4 General Solutions

Here two protocols are designed, the Function$(x)\{s, f(\cdot)\}$ and Function$(x, y)\{s, f(\cdot)\}$ protocols, which provide general solutions to functions that have one or two additively shared parameters between two parties. For generality, assume that the output is a fixed-point number with s bits after the decimal point. Using the same strategy, these two parties can always securely compute functions with more parameters rather than one or two.

3.4.1 Function(x)$\{s, f(\cdot)\}$

Specification 10 (Function(x)) *Party 1 and Party 2 share some number x. They want to securely execute the protocol $(x_1, x_2)\{s, f(\cdot)\} \mapsto (\hat{y}_1, \hat{y}_2)$ such that $\hat{y}_1 + \hat{y}_2 = [2^s \cdot f(x_1 + x_2)]$ where s is the number of bits after the decimal point and $f(\cdot)$ is some function. (For example, $f(x) = \log_7 x$ or $f(x) = \sqrt[3]{x}$ where $x = x_1 + x_2$.)*

PROTOCOL Function(x)

1. For $i = 0, 1, \cdots, n-1$, repeat Step 1a.

   (a) Party 1 individually sets $\hat{t}[i] = [2^s f(x_1 + i)]$ while Party 2 individually sets

   $$u[i] = \begin{cases} 1 & \text{if} \quad i = x_2, \\ 0 & \text{if} \quad i \neq x_2. \end{cases}$$

2. Party 1 and Party 2 collaboratively execute the Scalar-Product protocol

   $$((\hat{t}[0], \ldots, \hat{t}[n-1]), (u[0], \ldots, u[n-1])) \mapsto (\hat{y}_1, \hat{y}_2)$$

   such that $\hat{y}_1 + \hat{y}_2 = \hat{t}[0] \cdot u[0] + \cdots + \hat{t}[n-1] \cdot u[n-1]$.

3.4.2 Function(x, y)$\{s, f(\cdot)\}$

Specification 11 (Function(x, y)) *Party 1 and Party 2 share some x and y. They want to securely execute the protocol $((x_1, y_1), (x_2, y_2))\{s, f(\cdot)\} \mapsto (\hat{z}_1, \hat{z}_2)$ such that $\hat{z}_1 + \hat{z}_2 = [2^s \cdot f(x_1 + x_2, y_1 + y_2)]$ where s is the number of bits after the decimal point and $f(\cdot)$ is some function. (For example, $f(x, y) = \log_y x$ or $f(x, y) = \sqrt[y]{x}$ where $x = x_1 + x_2$ and $y = y_1 + y_2$.)*

PROTOCOL Function(x, y)

1. For $i = 0, \cdots, n-1$, repeat from Step 1a to Step 1c.

(a) For $j = 0, \cdots, n-1$, repeat Step 1(a)i.

i. Party 1 individually sets $\hat{t}[j] = [2^s f(x_1 + i, y_1 + j)]$ while Party 2 individually sets

$$w[j] = \begin{cases} 1 & \text{if} \quad j = y_2, \\ 0 & \text{if} \quad j \neq y_2. \end{cases}$$

(b) Party 1 and Party 2 collaboratively execute the Scalar-Product protocol

$$((\hat{t}[0], \ldots, \hat{t}[n-1]), (w[0], \ldots, w[n-1])) \mapsto (\hat{u}[i]_1, \hat{u}[i]_2)$$

such that $\hat{u}[i]_1 + \hat{u}[i]_2 = \hat{t}[0] \cdot w[0] + \cdots + \hat{t}[n-1] \cdot w[n-1]$.

(More specifically, $\hat{u}[i]_1 + \hat{u}[i]_2 = \hat{t}[y_2] = [2^s f(x_1 + i, y_1 + y_2)]$.)

(c) Party 1 individually sets $\hat{v}[i] = \hat{u}[i]_1$ while Party 2 individually sets

$$q[i] = \begin{cases} 1 & \text{if} \quad i = x_2, \\ 0 & \text{if} \quad i \neq x_2. \end{cases}$$

2. Party 1 and Party 2 collaboratively execute the Scalar-Product protocol

$$((\hat{v}[0], \ldots, \hat{v}[n-1]), (q[0], \ldots, q[n-1])) \mapsto (\hat{r}_1, \hat{r}_2)$$

such that $\hat{r}_1 + \hat{r}_2 = \hat{v}[0] \cdot q[0] + \cdots + \hat{v}[n-1] \cdot q[n-1]$.

(More specifically, $\hat{r}_1 + \hat{r}_2 = \hat{v}[x_2] = \hat{u}[x_2]_1$.)

3. Party 1 individually sets $\hat{z}_1 = \hat{r}_1$, and Party 2 computes $\hat{z}_2 = \hat{u}[x_2]_2 + \hat{r}_2$.

Chapter 4

Solutions to Real World Problems

The goal of designing building blocks based on scalar-product is to offer an practically applicable way to real-world problems. In this section, the author uses building blocks proposed in Chapter 3 to solve two-party privacy preserving statistical analysis and privacy preserving classification problems, including the Variance, Naïve Bayesian Classification, and Nearest Neighbor. With these protocols, we can securely find the variance, do the Naïve Bayesian classification, and solve the Nearest Neighbor problem of the "aggregated" database between two parties.

The author first shows any secure multi-party polynomial evaluation in Section 4.1. In Section 4.2, the author shows how to do statistical analysis by designing Variance protocols on additively shared and horizontally partitioned data. Then privacy preserving classification algorithms, including Naïve Bayesian Classification and Nearest Neighbor, are discussed in Section 4.3.

4.1 Polynomial Evaluation

A polynomial is a function constructed from variables and constants using the operations of addition, subtraction, and multiplication. With the additive sharing, addition and subtraction can easily be done; with the Scalar-product protocol, multiplication can be done as well. Therefore, secure

two-party computation on any polynomial evaluation can be composed of the Scalar-product protocol and the help of additive sharing. Furthermore, the author uses the same strategy for secure multi-party computation on any polynomial evaluation.

4.1.1 Addition and Subtraction

Specification 12 (Addition1) *Party 1 , Party 2 , . . . , and Party* m *want to securely execute the protocol* $(s_1, s_2, \cdots, s_m)\{c\} \mapsto (y_1, y_2, \cdots, y_m)$ *where* $\sum_{i=1}^{m} y_i = S + c = \sum_{i=1}^{m} s_i + c$.

c is a constant. (Recall that variables in the brackets are known by all parties.) A simple solution to Specification 12 is to choose one party, say Party 1 , to add the constant.

PROTOCOL Addition1

1. Party 1 individually computes $y_1 = s_1 + c$ and Party j sets $y_j = s_j$ for $j = 2$ to m.

Specification 13 (Addition2) *Party 1 , Party 2 , . . . , and Party* m *want to securely execute the protocol* $((s_1, t_1), (s_2, t_2), \cdots, (s_m, t_m)) \mapsto (y_1, y_2, \cdots, y_m)$ *where* $\sum_{i=1}^{m} y_i = S + T = \sum_{i=1}^{m} s_i + \sum_{i=1}^{m} t_i$.

PROTOCOL Addition2

1. Party j individually computes $y_j = s_j + t_j$ for $j = 1$ to m.

Subtraction can be done by using the same strategy.

4.1.2 Multiplication

Specification 14 ((Multi-party) Product) *Party 1 , Party 2 , . . . , and Party m additively share the multiplicand* S *and the multiplicator* T*. They want to securely execute the protocol*

$$((s_1, t_1), (s_2, t_2), \cdots, (s_m, t_m)) \mapsto (y_1, y_2, \cdots, y_m)$$

where

$$\sum_{i=1}^{m} y_i = S \times T = \sum_{i=1}^{m} s_i \times \sum_{i=1}^{m} t_i.$$

First, let's see the case when $m = 3$.

$$(s_1 + s_2 + s_3) \times (t_1 + t_2 + t_3) \tag{4.1}$$

$$= (s_1t_1 + s_1t_2 + s_1t_3) + (s_2t_1 + s_2t_2 + s_2t_3) + (s_3t_1 + s_3t_2 + s_3t_3) \tag{4.2}$$

$$= (s_1t_1 + s_2t_2 + s_3t_3) + (s_1t_2 + s_2t_1) + (s_1t_3 + s_3t_1) + (s_2t_3 + s_3t_2) \tag{4.3}$$

$$= \sum_{i=1}^{3} s_it_i + [s_1, t_1] \cdot [t_2, s_2] + [s_1, t_1] \cdot [t_3, s_3] + [s_2, t_2] \cdot [t_3, s_3] \tag{4.4}$$

$$= \sum_{i=1}^{3} s_it_i + \sum_{i=1}^{2} \sum_{j=i+1}^{3} [s_i, t_i] \cdot [t_j, s_j] \tag{4.5}$$

In equation (4.5), s_it_i can be computed individually by Party i while the scalar-product $[s_i, t_i] \cdot [t_j, s_j]$ should be computed cooperatively by Party i and Party j. Use the same strategy, we can get the following equation:

$$(s_1 + s_2 + \cdots + s_m) \times (t_1 + t_2 + \cdots + t_m) = \sum_{i=1}^{m} s_it_i + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} [s_i, t_i] \cdot [t_j, s_j]$$

The details of the (Multi-party) Product protocol are as follows.

PROTOCOL (Multi-party) Product

1. For $i = 1$ to $m - 1$, repeat Step 1a
   (a) For $j = i + 1$ to m, repeat Step 1(a)i
      i. Party i and Party j jointly execute the secure two-party Scalar-product protocol $((s_i, t_i), (t_j, s_j)) \mapsto (u_{ij}, u_{ji})$ such that $u_{ij} + u_{ji} = s_it_j + s_jt_i$.
2. Party i individually computes $y_i = s_it_i + \sum_{\substack{j=1 \\ j \neq i}}^{m} u_{ij}$, for $i = 1$ to m.

For secure multi-party computation, with the additive sharing, addition and subtraction are simple (Specification 12 and Specification 13); with the (Multi-party) Product protocol (Specification 14), multiplication can be done. Therefore, secure multi-party computation on any polynomial evaluation can be composed of the (Multi-party) Product protocol and the help of additive sharing.

Here is an example.

Example Party 1 , Party 2 , . . . , and Party m want to securely execute the protocol

$$((s_1, t_1, v_1), (s_2, t_2, v_2), \cdots, (s_m, t_m, v_m))\{c, d\} \mapsto (y_1, y_2, \cdots, y_m)$$

where $\sum_{i=1}^{m} y_i = c \times S \times T \times V + d = c \times \sum_{i=1}^{m} s_i \times \sum_{i=1}^{m} t_i \times \sum_{i=1}^{m} v_i + d.$

This example can be solved by composing secure (Multi-party) Product protocols and individually computable parts. Details are as follows.

PROTOCOL Example

1. Party 1 , Party 2 , . . . , and Party m collaboratively run the secure (Multi-party) Product protocol

   $$((c \cdot s_1, t_1), (c \cdot s_2, t_2), \cdots, (c \cdot s_m, t_m)) \mapsto (u_1, u_2, \cdots, u_m)$$

   such that $\sum_{i=1}^{m} u_i = c \cdot \sum_{i=1}^{m} s_i \times \sum_{i=1}^{m} t_i.$

2. Party 1 , Party 2 , . . . , and Party m jointly execute the secure (Multi-party) Product protocol

   $$((u_1, v_1), (u_2, v_2), \cdots, (u_m, v_m)) \mapsto (w_1, w_2, \cdots, w_m)$$

   such that $\sum_{i=1}^{m} w_i = \sum_{i=1}^{m} u_i \times \sum_{i=1}^{m} v_i.$

3. Party 1 individually computes $y_1 = w_1 + d$ and Party j sets $y_j = w_j$ for $j = 2$ to m.

4.2 Statistical Analysis

Two scenarios of a shared database are considered. The $secret$ type of database indicates the data of which are additively shared between Party 1 and Party 2 ; the $private$ type one means each party privately owns their database, and a view with a union is created over both of them to provide a complete view. Figure 4.2 and Figure 4.1 show the split and the shard database architecture (also known as horizontal partitioning) relatively.

Analysis of Variance (ANOVA) is a statistical test, the simplest form of which provides a statistical test of whether or not the means of several groups are all equal. The fundamental technique of ANOVA is a partitioning of the total sum of squares (SST) into components related to the effects used in the model. For example, the following is a simplified ANOVA model with one type of treatment at different levels:

$$\text{SST} = \text{SSB} + \text{SSW},$$

where SSB is the sum of squares between (or SSTR, the sum of squares for treatments) and SSW is the sum of squares within (or SSE, the sum of squares for errors). Each of the sum of squares, when divided by its appropriate degrees of freedom, provides an estimate of the variation in the experiment. Table 4.2 is the ANOVA table. Details of ANOVA can reference statistics books.

Source	Sum of Squares (SS)	Degree of Freedom (df)	Mean Square (MS = $\frac{\text{SS}}{\text{df}}$)	F-Statistics
Between Samples (Explained)	SSB	$k-1$	MSB = $\frac{\text{SSB}}{k-1}$	F = $\frac{\text{MSB}}{\text{MSW}}$
Within Samples (Unexplained)	SSW	$n-k$	MSW = $\frac{\text{SSW}}{n-k}$	
Total	SST	$n-1$		

Table 4.1: ANOVA Table

Two secure two-party protocols, $(private)$ Variance and $(secret)$ Variance, are proposed in Section 4.2.1 and Section 4.2.2 based on two scenarios, the additively shared and the horizontally

partitioned data respectively. Both protocols involves the computation of sum of squares and division, which play an important role in ANOVA. People can replace the introduced variance formula with the formulae used in ANOVA so that the analysis of variance (to get the result of F = $\frac{\text{MSB}}{\text{MSW}}$) can be achieved.[1]

Suppose a population has d data, $x_1, x_2, \cdots, x_d$, and the variance of the population var:

$$var = E[X^2] - (E[X])^2 = \frac{d \cdot \sum_{i=1}^{d} x_i^2 - (\sum_{i=1}^{d} x_i)^2}{d^2}$$

4.2.1 Variance on horizontally partitioned data

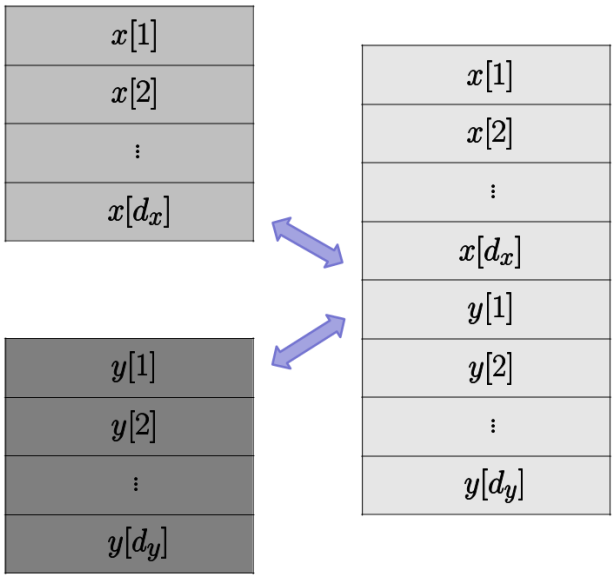

Figure 4.1: Shard database architecture (horizontal partitioning)

[1]When the data are additively shared or horizontally partitioned between two parties, e.g. Party 1 and Party 2 have k_1 and k_2 groups of data respectively where $k_1 + k_2 = k$, then these two parties can securely compute $F = \frac{\text{MSB}}{\text{MSW}}$ using the similar strategy the author used for Variance protocols. However, since the Division protocol the author proposed can apply to two-party only, we need a secure multi-party division protocol to solve problems involving multi-party division computation.

Specification 15 ((private) Variance) *Party 1 and Party 2 want to collaboratively execute the secure protocol* $((x[1], \ldots, x[d_x]), (y[1], \ldots, y[d_y])) \mapsto (q_1, q_2)$ *such that*

$$q_1 + q_2 = \lfloor \frac{(d_x + d_y)(s_1 + s_2) - (t_1 + t_2)^2}{(d_x + d_y)^2} \rfloor$$

where $s_1 = \sum_{i=1}^{d_x} x[i]^2, t_1 = \sum_{i=1}^{d_x} x[i], s_2 = \sum_{i=1}^{d_y} y[i]^2$, *and* $t_2 = \sum_{i=1}^{d_y} y[i]$.

Algorithm 2 Calculate $var =$ the variance of $(x[1], x[2], \cdots, x[d_x], y[1], y[2], \cdots, y[d_y])$

$d \leftarrow d_x + d_y$
$s \leftarrow \text{SquareSum}(x[1], x[2], \cdots, x[d_x], y[1], y[2], \cdots, y[d_y])$
$t \leftarrow \text{Sum}(x[1], x[2], \cdots, x[d_x], y[1], y[2], \cdots, y[d_y])$
$var = (d \cdot s - t^2)/d^2$

$$var = \frac{(d_x + d_y) \cdot (\sum_{i=1}^{d_x} x[i]^2 + \sum_{i=1}^{d_y} y[i]^2) - (\sum_{i=1}^{d_x} x[i] + \sum_{i=1}^{d_y} y[i])^2}{(d_x + d_y)^2}$$

PROTOCOL (private) Variance

1. Party 1 computes $s_1 = \sum_{i=1}^{d_x} x[i]^2$ and $t_1 = \sum_{i=1}^{d_x} x[i]$ while Party 2 computes $s_2 = \sum_{i=1}^{d_y} y[i]^2$ and $t_2 = \sum_{i=1}^{d_y} y[i]$.
2. Party 1 and Party 2 collaboratively execute the Product protocol $((d_x, s_1), (d_y, s_2)) \mapsto (u_1, u_2)$ such that $u_1 + u_2 = (d_x + d_y)(s_1 + s_2)$.
3. Party 1 and Party 2 collaboratively run the Square protocol $(t_1, t_2) \mapsto (v_1, v_2)$ such that $v_1 + v_2 = (t_1 + t_2)^2$.
4. Party 1 and Party 2 collaboratively execute the Square protocol $(d_x, d_y) \mapsto (w_1, w_2)$ such that $w_1 + w_2 = (d_x + d_y)^2$.
5. Party j computes $z_j = u_j - v_j$, for $j = 1, 2$.

6. Party 1 and Party 2 jointly execute the Div/Rem protocol $((z_1, w_1), (z_2, w_2)) \mapsto ((q_1, r_1), (q_2, r_2))$ such that $q_1 + q_2 = \left\lfloor \frac{z_1 + z_2}{w_1 + w_2} \right\rfloor$.

4.2.2 Variance on additively shared data

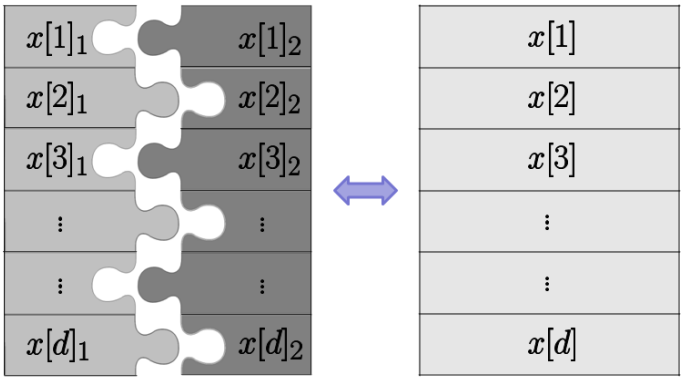

Figure 4.2: Split database architecture (secret shares)

Specification 16 ((secret) Variance) *Party 1 and Party 2 want to collaboratively execute the secure protocol* $((x[1]_1, \ldots, x[d]_1), (x[1]_2, \ldots, x[d]_2))\{d\} \mapsto (q_1, q_2)$ *such that*

$$q_1 + q_2 = \lfloor \frac{d(s_1 + s_2) - (t_1 + t_2)^2}{d^2} \rfloor$$

where $s_1 + s_2 = \sum_{i=1}^{d} (x[i]_1 + x[i]_2)^2$, *and* $t_j = \sum_{i=1}^{d} x[i]_j$, *for* $j = 1, 2$.

Algorithm 3 Calculate $var =$ the variance of $(x[1], x[2], \cdots, x[d])$

$s \leftarrow \text{SquareSum}(x[1], x[2], \cdots, x[d])$
$t \leftarrow \text{Sum}(x[1], x[2], \cdots, x[d])$
$var = (d \cdot s - t^2)/d^2$

$$
\begin{aligned}
var &= \frac{d \cdot \sum_{i=1}^{d}(\sum_{j=1}^{2} x[i]_j)^2 - (\sum_{i=1}^{d}\sum_{j=1}^{2} x[i]_j)^2}{d^2} \\
&= \frac{\sum_{i=1}^{d}(d \cdot x[i]_1^2) + \sum_{i=1}^{d}(d \cdot x[i]_2^2) + \sum_{i=1}^{d}(2d \cdot x[i]_1 \cdot x[i]_2) - (\sum_{i=1}^{d} x[i]_1 + \sum_{i=1}^{d} x[i]_2)^2}{d^2}
\end{aligned}
$$

PROTOCOL (secret) Variance

1. Party 1 and Party 2 collaboratively run the Scalar-Product protocol

$$((2x[1]_1, 2x[2]_1, \cdots, 2x[d]_1), (x[1]_2, x[2]_2, \cdots, x[d]_2)) \mapsto (u_1, u_2)$$

   such that $u_1 + u_2 = 2x[1]_1 \cdot x[1]_2 + \cdots + 2x[d]_1 \cdot x[d]_2$.

2. Party j individually computes $s_j = \sum_{i=1}^{d} x[i]_j^2 + u_j$, for $j = 1, 2$.

3. Party 1 and Party 2 collaboratively run the Square protocol $(t_1, t_2) \mapsto (v_1, v_2)$ such that $v_1 + v_2 = (t_1 + t_2)^2$.

4. Party j computes $z_j = d \cdot s_j - v_j$, for $j = 1, 2$.

5. Party 1 and Party 2 jointly execute the Div/Rem{divisor} protocol[2]

$$(z_1, z_2)\{\lceil \lg d^2 \rceil - 1\} \mapsto ((q_1, r_1), (q_2, r_2))$$

   such that $q_1 + q_2 = \left\lfloor \frac{z_1 + z_2}{d^2} \right\rfloor$.

[2]According to Specification 17 in Appendix A.1, it is necessary to compute k_y. $\because k_y \in \mathbb{Z}^+$ and $d^2 < k_y + 1, \therefore k_y = \lceil \lg d^2 \rceil - 1$

4.3 Privacy Preserving Classification

Data mining is the analysis step of the knowledge discovery in databases process. There have been two broad approaches for the concept of privacy preserving data mining [EGS03]: 1) The randomization approach focuses on individual privacy, and reveals randomized information about each record in exchange for not having to reveal the original records to anyone. 2) In the SMC approach, the goal is to build a data mining model across multiple databases without revealing the individual records in each database to the other databases. In this section, the author focuses on the latter one.

Two secure distributed classification protocols are designed. In Section 4.3.1, the naïve Bayes classifier, a highly practical Bayesian learning method, is discussed. A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong independence assumptions. Section 4.3.2 discusses the nearest neighbor algorithm (NN), classifying objects based on the closest training example in the feature space. Details of these two algorithms can reference Mitchell and Hill's book [MH97].

4.3.1 Naïve Bayesian Classification

Assume the database is horizontally partitioned ($private$ type) for Party 1 and Party 2 . Let the union database have N entries and m attributes excluding the outcome attribute, OUT. In addition, let OUT be the set of possible values in OUT, and X_i be the i-th attribute. This section focuses on solving the problem: Given an instance (not a private input) that $X_1 = x_1, \cdots, X_m = x_m$, Party 1 and Party 2 want to use Naïve Bayesian classification to estimate its most possible value of OUT. More specifically, the Bayesian approach to classifying the new instance is to assign the most probable target value, out_{MAP}, given the attribute values $\langle x_1, \cdots, x_m \rangle$ that describe the instance.

$$out_{MAP} = \underset{out_t \in OUT}{\operatorname{argmax}} P(out_t | x_1, x_2, \cdots, x_m)$$

Using Bayes theorem,

$$\begin{aligned} out_{MAP} &= \underset{out_t \in OUT}{\operatorname{argmax}} \frac{P(x_1, x_2, \cdots, x_m | out_t) \cdot P(out_t)}{P(x_1, x_2, \cdots, x_m)} \\ &= \underset{out_t \in OUT}{\operatorname{argmax}} P(x_1, x_2, \cdots, x_m | out_t) \cdot P(out_t). \end{aligned}$$

The Naïve Bayesian classifier further simplifies the assumption that the attribute values are conditionally independent given the target value. That is,

$$P(x_1, x_2, \cdots, x_m | out_t) = \prod_i P(x_i | out_t).$$

Therefore,

$$out_{NB} = \underset{out_t \in OUT}{\operatorname{argmax}} P(out_t) \cdot P(x_i | out_t),$$

where out_{NB} denotes the target value output by the naïve Bayes classifier.

Assume every attribute, including OUT, is a nominal attribute. Let N_{OUT_t} be the number of entries that OUT = out_t, where out_t is the t-th possible value of OUT.

$$OUT = \{out_1, out_2, \cdots, out_t, \cdots, out_{|OUT|}\}$$

In addition, let $res_t = P(\text{OUT} = out_t) \cdot P(X_1 = x_1, \cdots, X_m = x_m | \text{OUT} = out_t)$ and N_{ti} be the

number of entries that OUT = out_t and $X_i = x_i$. We can get the following formula:

$$\begin{aligned}
res_t = \quad & P(\text{OUT} = out_t) \cdot P(X_1 = x_1, \cdots, X_m = x_m | \text{OUT} = out_t) \\
= \quad & P(\text{OUT} = out_t) \cdot P(X_1 = x_1 | \text{OUT} = out_t) \cdot P(X_1 = x_2 | \text{OUT} = out_t) \cdots \\
& P(X_m = x_m | \text{OUT} = out_t) \\
= \quad & \frac{N_{OUT_t}}{N} \cdot \frac{N_{t1}}{N_{OUT_t}} \cdot \frac{N_{t2}}{N_{OUT_t}} \cdots \frac{N_{tm}}{N_{OUT_t}} \\
= \quad & \frac{N_{OUT_t}}{N} \cdot \frac{\prod_{i=1}^{m} N_{ti}}{(N_{OUT_t})^m} \\
= \quad & \frac{\prod_{i=1}^{m} N_{ti}}{N \cdot (N_{OUT_t})^{m-1}}
\end{aligned}$$

For $j = 1, 2$, let Party j have N^j entries, $N^j_{OUT_t}$ entries that OUT = out_t, and N^j_{ti} entries that OUT = out_t and $X_i = x_i$. Note that the superscripts in this section indicate the data owner. Since the database is horizontally partitioned, N, N_{OUT_t}, and N_{ti} would be equal to the following equations:

$$\begin{aligned}
N &= N^1 + N^2, \\
N_{OUT_t} &= N^1_{OUT_t} + N^2_{OUT_t}, \\
N_{ti} &= N^1_{ti} + N^2_{ti}.
\end{aligned}$$

Therefore,

$$res_t = \quad \frac{\prod_{i=1}^{m} N_{ti}}{N \cdot (N_{OUT_t})^{m-1}} = \frac{\prod_{i=1}^{m} (N^1_{ti} + N^2_{ti})}{(N^1 + N^2) \cdot (N^1_{OUT_t} + N^2_{OUT_t})^{m-1}}.$$

Since all of the inputs and outputs should be integers in the secure protocol, if we want the value of res_t to be accurate to the s-th decimal place, we can instead compute $res\prime_t$:

$$res\prime_t = \frac{10^s \cdot \prod_{i=1}^{m}(N_{ti}^1 + N_{ti}^2)}{(N^1 + N^2) \cdot (N_{OUT_t}^1 + N_{OUT_t}^2)^{m-1}}.$$

Apparently, $res_t = \frac{res\prime_t}{10^s}$. Using $res\prime_t$ instead of res_t does not have any difference in the algorithm when s is adequately picked. The following is the algorithm of Naïve Bayesian classification to classify a new instance.

Algorithm 4 Given an instance $\langle x_1, \cdots, x_m \rangle$, find ind such that out_{NB} = ind-th outcome value of OUT

for $t = 1, 2, \cdots, |OUT|$ **do**
  Compute N_{OUT_t} : the number of entries that OUT = out_t
  Compute N_{ti} : the number of entries that OUT = out_t and $X_i = x_i$
  $v \leftarrow N \cdot (N_{OUT_t})^{m-1}$
  $u \leftarrow \prod_{i=1}^{m} N_{ti}$
  $res_t \leftarrow \frac{u}{v}$
end for
Find ind such that res_{ind} is the maximum of the set $\{res_1, res_2, \cdots, res_{|OUT|}\}$

To decide out_{NB}, these two parties need to compute all $res\prime_t$ first, for $t = 1$ to $|OUT|$. Then, find out ind where $res\prime_{ind}$ is the maximum of all $res\prime$. Therefore, out_{NB} = ind-th outcome value of OUT.

Before showing the protocol NB (the abbreviation of Naïve Bayesian classification), the author proposes the ($secret$) IndexOfMaximum protocol in Section A.3 and the Exponential protocol in Section A.4, which will be used in the protocol NB. The former returns the index of the maximum $secret$ array element while the latter computes the exponentiation where the base is $secret$ while the exponent is $public$.

PROTOCOL NB

1. For $t = 1$ to $|OUT|$, repeat Step 1a to Step 1e.

   (a) Party j individually computes $N^j_{OUT_t}$ and N^j_{ti} for $i = 1$ to m and $j = 1, 2$.

   (b) Party 1 and Party 2 collaboratively execute $m - 1$ times of Product protocol, and Party j gets u_j for $j = 1, 2$ such that $u_1 + u_2 = 10^s \cdot \prod_{i=1}^{m}(N^1_{ti} + N^2_{ti})$.

   (c) Party 1 and Party 2 collaboratively execute the Exponential protocol $(N^1_{OUT_t}, N^2_{OUT_t})\{m-1\} \mapsto (w_1, w_2)$ such that $w_1 + w_2 = (N^1_{OUT_t} + N^2_{OUT_t})^{m-1}$.

   (d) These two parties jointly run the Product protocol $((N^1, w_1), (N^2, w_2)) \mapsto (v_1, v_2)$ such that $v_1 + v_2 = (N^1 + N^2) \cdot (w_1 + w_2) = (N^1 + N^2) \cdot (N^1_{OUT_t} + N^2_{OUT_t})^{m-1}$.

   (e) Party 1 and Party 2 collaboratively execute the Division protocol $((u_1, v_1), (u_2, v_2)) \mapsto ((q_1, r_1), (q_2, r_2))$ such that $q_1 + q_2 = \left\lfloor \frac{u_1 + u_2}{v_1 + v_2} \right\rfloor$.

   (f) Party j individually sets $x[t]_1 = q_j$, for $j = 1, 2$.

2. Party 1 and Party 2 jointly run the $(secret)$ IndexOfMaximum protocol

$$((x[1]_1, x[2]_1, \cdots, x[|OUT|]_1), (x[1]_2, x[2]_2, \cdots, x[|OUT|]_2)) \mapsto (y_1, y_2)$$

   such that $y_1 + y_2 =$ the index of the maximum of $(x[1], x[2], \cdots, x[|OUT|])$.

In protocol NB, note that $res\prime_t = v[t]_1 + v[t]_2$. For each $res\prime_t$, they need to execute m times of Product protocols, 1 Exponential protocol and 1 Division protocol. Therefore, totally they have to execute $|OUT| \cdot m$ times of Product protocols, $|OUT|$ times of Exponential protocols, $|OUT|$ times of Division protocols, and 1 $(secret)$ IndexOfMaximum protocol.

For a numeric attribute, the attribute can be assumed to have a Gaussian probability distribution, $\mathcal{N}(\mu, \sigma^2)$. Therefore, the conditional probability, $P(X_i = x_i|OUT = out_t)$, can be estimated. We can use the Division protocol to get the mean, namely μ, and the Variance protocol to get the variance, namely σ^2. The author focuses on showing how to apply the Scalar-product based secure

protocols to naïve Bayesian classification. The goal is achieved, so the author does not show details here for numeric attributes.

4.3.2 Nearest Neighbor

In pattern recognition, the nearest neighbor algorithm (NN) is a method for classifying objects based on the closest training example in the feature space. NN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification. The nearest neighbor algorithm is amongst the simplest of all machine learning algorithms: the object is simply assigned to the class of its nearest neighbor.

Assume the database is horizontally partitioned ($private$ type) for Party 1 and Party 2 .

Algorithm 5 Given an instance $node_q$, calculate $res =$ the outcome value out of $node_q$'s nearest neighbor.

1. Party j finds $node_q$'s nearest neighbor $node_j$ and the corresponding distance $dis_j = distance(node_q, node_j)$, for $j = 1, 2$.
2. $res \leftarrow (dis_1 - dis_2 < 0)?\ dis_1 : dis_2$

PROTOCOL NN

1. Given an instance $node_q$, Party j individually finds $node_q$'s nearest neighbor, $node_j$, and their distance, $dis_j = distance(node_j, node_q)$ in its database, for $j = 1, 2$.

2. Party 1 and Party 2 collaboratively execute the Comparison protocol $(dis_1, -dis_2) \mapsto (u_1, u_2)$ such that
$$u_1 + u_2 = \begin{cases} 1 & \text{if } dis_1 - dis_2 < 0, \\ 0 & \text{otherwise.} \end{cases}$$

3. Party 1 and Party 2 collaboratively execute the If-Then-Else protocol $((u_1, dis_1, 0), (u_2, 0, dis_2)) \mapsto (v_1, v_2)$ such that
$$v_1 + v_2 = \begin{cases} dis_1 & \text{if} \quad u_1 + u_2 = 1, \\ dis_2 & \text{if} \quad u_1 + u_2 = 0. \end{cases}$$

Therefore, $v_1 + v_2$ = the outcome value of $node_q$'s nearest neighbor (either dis_1 or dis_2). Only one Comparison protocol and one If-Then-Else protocol are needed. The NN protocol can be revised and applied to k nearest neighbor algorithm (kNN) as well. Here the author just wants to show the idea of how to design a privacy preserving data mining protocol based on building blocks proposed in Chapter 3.

Chapter 5

Secure Auction

In this chapter, the author introduces an efficient secure auction procedure based on secure two-party comparison protocols.

Yao's Millionaires' problem discusses two millionaires, Alice and Bob, who are interested in knowing which of them is richer without revealing their actual wealth [Yao82]. Using the Comparison protocol proposed in Section 3.3.1 can solve this problem. What if there are m millionaires in the Millionaires' problem? Using secure auction protocols which deal with first-price sealed-bid auction can solve this problem.

Brandt and Sandholm point out that first-price sealed-bid auction can be emulated by an unconditionally fully private protocol [BS08]. However, the round complexity of the protocol is exponential in the bid size, and there is no more efficient protocol. Therefore, for efficiency concern, the author design a secure auction procedure. It reveals some information of the bids, but it is simple and efficient.

The design assumptions are stated in Section 5.1. The detail procedure is proposed in Section 5.2. After that in Section 5.3, the author lists features of the designed protocol and discusses how to deal with possible saboteurs. The performance evaluation is in Section 5.4.

5.1 Design Assumptions

The designed secure auction procedure has the following assumptions.

1. The secure auction procedure is designed under the assumption of semi-honest behavior. That is, each party follows the protocol. However, after the protocol is complete, the adversary may attempt to compute additional information from the messages received during execution.

2. Every bidder knows the commodity server's ip/port.

3. Assume the networks are reliable. Any participants, including the commodity server, can successfully communicate with each other if the protocol ask them to do so.

5.2 Procedure

To implement the secure auction procedure, the author adopts the following secure scalar-product method as the secure Scalar-product protocol introduced in Section 3.1. This approach is proposed by Du and Zhan, namely, the commodity-based approach [DZ02], which is based on Beaver's commodity model [Bea97].

More specifically, Alice and Bob want to collaboratively execute the commodity-based Scalar-product protocol $(\overrightarrow{X_a}, \overrightarrow{X_b}) \mapsto (y_a, y_b)$ such that $y_a + y_b = \overrightarrow{X_a} \cdot \overrightarrow{X_b}$. The following are steps of the commodity-based scalar-product protocol.

1. Alice and Bob agree on a semi-trusted commodity server, Carol, who picks a random integer r_a uniformly from $\mathcal{D}$ and generates two d-dimensional random vectors $\overrightarrow{R_a}$ and $\overrightarrow{R_b}$ uniformly from $\mathcal{D}^d$. After Carol computes $r_b = \overrightarrow{R_a} \cdot \overrightarrow{R_b} - r_a$, she sends $(\overrightarrow{R_a}, r_a)$ and $(\overrightarrow{R_b}, r_b)$ to Alice and Bob respectively. (See Table 5.1.)

2. Alice gets $(\overrightarrow{R_a}, r_a)$ from Carol. Then, she computes $\overrightarrow{X'_a} = \overrightarrow{X_a} + \overrightarrow{R_a}$ and sends it to Bob.

3. Bob gets $(\overrightarrow{R_b}, r_b)$ from Carol and computes $\overrightarrow{X'_b} = \overrightarrow{X_b} + \overrightarrow{R_b}$. Then, he gets $\overrightarrow{X'_a}$ from Alice and selects a random number y_b uniformly from $\mathcal{D}$. After that, he computes $t = \overrightarrow{X'_a} \cdot \overrightarrow{X_b} + r_b - y_b$ and sends $(\overrightarrow{X'_b}, t)$ to Alice.

4. Alice computes $y_a = t - \overrightarrow{R_a} \cdot \overrightarrow{X'_b} + r_a$. (See Table 5.2.)

Alice	Carol (Commodity Server)	Bob
	Generate random vectors $\overrightarrow{R_a}, \overrightarrow{R_b}$	
	Generate a random number r_a	
	Compute $r_b = \overrightarrow{R_a} \cdot \overrightarrow{R_b} - r_a$	
$\overrightarrow{R_a}, r_a$ $\longleftarrow$		$\overrightarrow{R_b}, r_b$ $\longrightarrow$

Table 5.1: Step 1 of the commodity-based Scalar-product protocol

Alice		Bob
Get $(\overrightarrow{R_a}, r_a)$ from Carol		Get $(\overrightarrow{R_b}, r_b)$ from Carol
$\overrightarrow{X'_a} = \overrightarrow{X_a} + \overrightarrow{R_a}$		$\overrightarrow{X'_b} = \overrightarrow{X_b} + \overrightarrow{R_b}$
	$\overrightarrow{X'_a}$ $\longrightarrow$	
		Generate random number y_b
		$t = \overrightarrow{X'_a} \cdot \overrightarrow{X_b} + r_b - y_b$
	$t, \overrightarrow{X'_b}$ $\longleftarrow$	
$y_a = t - \overrightarrow{R_a} \cdot \overrightarrow{X'_b} + r_a$		

Table 5.2: Step 2 to Step 4 of the commodity-based Scalar-product protocol

Section B.1 shows why $y_a + y_b = \overrightarrow{X_a} \cdot \overrightarrow{X_b}$ in the commodity-based Scalar-product protocol.

The author points out that the designed secure auction needs only the secure Comparison protocol among all build blocks proposed in Chapter 3. To avoid too much communication between the commodity server and bidders, the commodity server can at once prepare all random numbers and random vectors that each secure Comparison protocol needs and then send them to Alice and Bob

together. Therefore, Alice and Bob can complete a secure Comparison protocol by communicating with the commodity server for one time each.

Under the assumptions mentioned in Section 5.1, the designed secure auction procedure is described as follows.

1. All bidders agree on an integer k. Each bidder chooses an integer as their bid, and the bidding must be less than k bits, i.e. $0 \sim 2^k - 1$.

2. Bidders submit their ip/port information to the commodity server.

3. If the commodity server receives only one ip/port before timeout, then go to Step 7. Otherwise, the commodity server makes pairs (Alice/Bob) and generates random numbers for those bidders registered in Step 2. (First-come, first-serve)

4. The bidders receive the ip/port information, random numbers, and which role to play (Alice or Bob) in the secure two-party comparison protocol from the commodity server.

5. Each bidder plays the assigned role (Alice or Bob) in Step 4 and executes the secure two-party comparison protocol with the received ip/port bidder.

6. The commodity server and the bidder with higher bidding of each secure two-party comparison protocol repeat Step 2 to Step 5.

7. The commodity server announces the ip/port bidder has the highest bidding, and that bidder reveals his/her bidding.

5.3 Features and Discussion

In this section, features of the designed secure auction procedure introduced in Section 5.2 are discussed.

- **Every bidding will not be disclosed, except the highest one.** The main goal of the designed secure auction procedure is to keep every bidder's actual bid in secret, except the final winner's. Only the result of each secure two-party comparison protocol, i.e. whose bid is higher, will be known by the two parties participating in that comparison. The actual bidding is still secret.

- **Bidders don't need to know most participants in the auction.** Indeed, each bidder will learn t other bidders' ip/port information when he gets t responses from the commodity server.[1] Note that the commodity server, however, will learn all bidders' ip/port information.

- **The commodity server must be a semi-trusted third party.** This third party should not collude with participants (bidders). All that the commodity server does is pairing the bidders, generating random numbers to them, and announcing the final winner. In real world, finding such a semi-trusted third party is much easier than finding a trusted third party.

- **If there are m bidders, the communication time between bidders to find the final winner is $O(k \cdot \lg m)$.** Recall that the communication time complexity of the secure two-party comparison protocol is $O(k)$. In addition, the expected value of secure two-party comparison protocol execution times for the final winner is $O(\lg m)$, which is the highest among all bidders. Therefore, the communication time between bidders to find the final winner is

$$O(k) \cdot O(\lg m) = O(k \cdot \lg m).$$

  Note that the communication between bidders and the commodity server is not included in the analysis.

- **Saboteurs may exist.** A saboteur mentioned here means a bidder who maliciously bids extremely high to ensure he can win the auction. However, after being announced as the

[1] Here, the response from the commodity server means the response in Step 4.

final winner, the saboteur by deliberate intension declares a relative low bidding (a fake bidding). In such situation, even though some of the participants who bade higher know the winner is a saboteur, they cannot prove the winner is lying since the bidding processes are secret.

- **To avoid saboteurs, bidders can agree on running the secure auction procedure several times.** Each round of the secure auction procedure can find the bidder with the highest bidding. If any other bidder is willing to bid higher than the announced bidding, which may be a fake one from a saboteur, another round of secure auction procedure can be done again and again until everyone else gives up. To do so, it can not only avoid possible saboteurs to some degree but let bidders who really want to win have second chances.

5.4 Performance Evaluation

As mentioned, the author adopts the secure scalar product protocol that Du and Zhan proposed, namely, the commodity-based approach [DZ02], which is based on Beaver's commodity model [Bea97]. This approach has extraordinary performance among several secure scalar product ones, though a neutral third party, the commodity server, is needed [WSH+08]. The author implemented this scalar product protocol and the secure auction program in Ruby (1.8.7 patchlevel 352). To minimize the probabilistic variation, all experimental results are the average of 100 effective executions. The specs and the operation system of the 7 servers used in the experiments are stated as follows. (Each server has the same specs and OS.)

- Dell Poweredge R210
- Quad core 2.4 GHz Xeon processors
- 4 GB Ram

- Red Hat Enterprise Linux Server release 6.2

The expected value of secure two-party Comparison protocol execution times for the final winner is $\lg m$ when there are m bidders participating in the secure auction. However, to give the impression of how efficient the designed secure auction is, Figure 5.1 shows the time cost of the secure auction when each bidder simultaneously executes and completes one time of the Comparison protocol, including getting random numbers from the commodity server. One of the 7 servers is the commodity server; the other 6 servers play the bidders' role. Note that with different k (5, 10, or 20), each of these 6 servers plays b bidders ($b = 1, 2, 3, 4,$ or 5); therefore, the total number of bidders is $6b$ (6, 12, 18, 24, or 30).

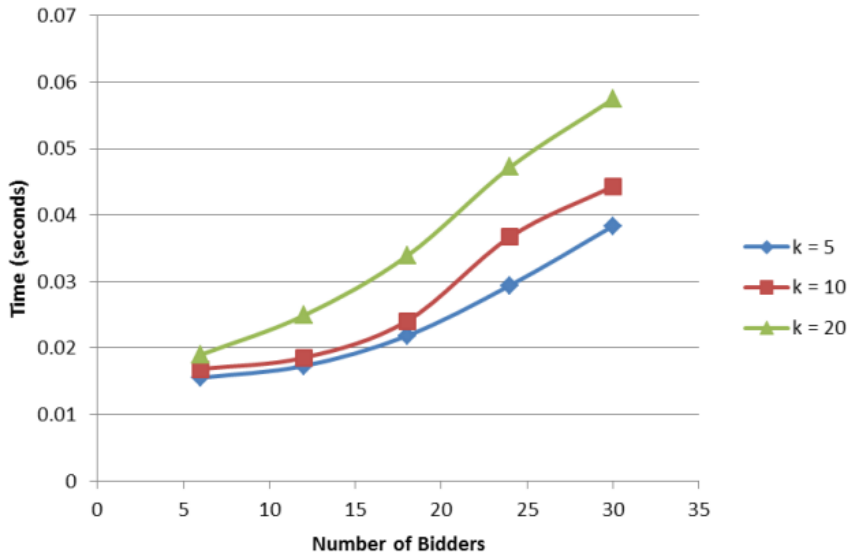

Figure 5.1: The time cost with different k and different number of bidders of the secure auction

Chapter 6

Conclusion and Future Work

Privacy is so valuable but fragile. Once an individual's privacy is breached, the clock can never be turned back. Secure multiparty computation (SMC) enables parties to carry out distributed computing tasks, guaranteeing the correctness of their outputs while no additional private information, other than what can be inferred from each participant's inputs and outputs, is revealed.

This thesis focused on designing SMC protocols based on scalar-products that are practically applicable to real world problems. The author chose to use scalar-products as the basic building block since: 1) As a general solution to SMC problems, the building block of generic circuit evaluation, the oblivious transfer, can be implemented by the scalar-product protocol. 2) It would be much more efficient if functions are constructed from scalar-products instead of binary gates [Du01].

The author emphasizes the following accomplishments: 1) With the help of additive sharing, any secure multi-party polynomial evaluation are solvable using secure protocols based on scalar-products. 2) Via the process of designing the Variance protocols on horizontally partitioned and additively shared data, the author shows complicated statistical analysis computations, e.g. analysis of variance (ANOVA), are achievable. 3) The proposed scalar-product based building blocks enables privacy preserving distributed mining, including classifying a new instance using Naïve

Bayes classifier and finding the nearest neighbor on horizontally partitioned data. 4) Sometimes people concerns not only privacy but also efficiency for SMC solutions. A novel secure auction protocol is proposed by exploring the trade-off between the strength of privacy protection and the protocol complexity.

The author did not deal with the problem combining the final results of all proposed secure protocols. The combination of the final result involves the fairness problem: The first party who receives the result might disrupt the collaboration prematurely. To adopt commitment protocols is one possible solution. However, this issue is not discussed for it is beyond the focus of this thesis. Besides that, as mentioned, all computations in the proposed protocols are done in $\mathbb{Z}_n$. On the one hand, when n is too small, the final results of protocols would be unpredictable once any inputs, intermediate results, or outputs cannot be correctly represented in $\mathbb{Z}_n$. On the other, when n is too large, it will not help the efficiency at all. Therefore, n should be carefully chosen so that protocols can be done correctly while relatively efficient.

Future work includes a compiler to automatically translate general algorithms to SMC protocols and to suggest reasonable n. Another future direction is using scalar-product based protocols to solve more complicated SMC problems. For example, problems involves distributed computation with floating-point numbers and nonlinear regression.

Appendix A

More Building Blocks

A.1 Division/Remainder with revealed divisor

Specification 17 (Div/Rem{divisor k_y}) *Two parties share the dividend in $\mathbb{Z}_n$, and they want to jointly execute the secure protocol $((x_1, y_1), (x_2, y_2))\{k_y\} \mapsto ((q_1, r_1), (q_2, r_2))$, where $x_1 + x_2 = (y_1 + y_2)(q_1 + q_2) + (r_1 + r_2)$ and $0 \leq (r_1 + r_2) < (y_1 + y_2)$ (mod n) $< 2^{k_y+1}$.*

PROTOCOL Div/Rem{divisor k_y}

1. Party j individually sets $Y_j = y_j, X_j^{k-k_y} = x_j$ where $k_y = \lfloor \log_2 y \rfloor$, for $j = 1, 2$.

2. For $i = k - k_y - 1, k - k_y - 2, \ldots, 0$, repeat Step 2a to Step 2d.

   (a) Party 1 individually computes $t_1^i = X_1^{i+1} - Y_j \cdot 2^i$ while Party 2 individually sets $t_2^i = X_2^{i+1}$.

   (b) Party 1 and Party 2 jointly run the Comparison protocol $(t_1^i, t_2^i) \mapsto (s_1^i, s_2^i)$ such that

$$s_1^i + s_2^i = \begin{cases} 1 & \text{if } t_1^i + t_2^i < 0, \\ 0 & \text{otherwise.} \end{cases}$$

(c) For $j = 1, 2$, Party j individually computes $q_j^i = 1 - s_j^i$ such that

$$q_1^i + q_2^i = \begin{cases} 0 & \text{if } s_1^i + s_2^i = 1, \\ 1 & \text{if } s_1^i + s_2^i = 0. \end{cases}$$

(d) Party 1 and Party 2 run the If-Then-Else protocol $((s_1^i, X_1^{i+1}, t_1^i), (s_2^i, X_2^{i+1}, t_2^i)) \mapsto (X_1^i, X_2^i)$ such that

$$X_1^i + X_2^i = \begin{cases} X_1^{i+1} + X_2^{i+1} & \text{if } s_1^i + s_2^i = 1, \\ t_1^i + t_2^i & \text{if } s_1^i + s_2^i = 0. \end{cases}$$

3. For $j = 1, 2$, Party j computes $r_j = X_j^0 \pmod n$, and $q_j = \sum_{i=0}^{k-1} q_j^i \cdot 2^i \pmod n$.

A.2 ($secret$) Maximum

Specification 18 (($secret$) Maximum) *Party 1 and Party 2 want to collaboratively execute the secure protocol* $((x[1]_1, x[2]_1, \cdots, x[d]_1), (x[1]_2, x[2]_2, \cdots, x[d]_2))\{d\} \mapsto (y_1, y_2)$ *such that* $y_1 + y_2 =$ *the maximum of* $(x[1], x[2], \cdots, x[d])$*, where* $x[i]_1, x[i]_2, y_1, y_2 \in \mathbb{Z}_n$*, for* $i = 1, 2, \cdots, d$.

Algorithm 6 Calculate $max =$ the maximum of $(x[1], x[2], \cdots, x[d])$.

```
max ← x[1]
for i ← 2, 3, ··· , d do
  if max < x[i] then
    max ← x[i]
  end if
end for
```

Based on Algorithm 6, the ($secret$) Maximum protocol details are as follows:

PROTOCOL ($secret$) Maximum

1. Party j individually sets $m_j^1 = x[1]_j$, for $j = 1, 2$.

2. For $i = 2, 3, \cdots, d$, repeat Step 2a to Step 2b.

(a) Party 1 and Party 2 jointly execute the Comparison protocol
$(m_1^{i-1} - x[i]_1, m_2^{i-1} - x[i]_2) \mapsto (c_1, c_2)$ such that

$$c_1 + c_2 = \begin{cases} 1 & \text{if } m_1^{i-1} + m_2^{i-1} - x[i]_1 - x[i]_2 < 0, \\ 0 & \text{otherwise.} \end{cases}$$

(b) These two parties collaboratively run the If-Then-Else protocol
$((c_1, x[i]_1, m_1^{i-1}), (c_2, x[i]_2, m_2^{i-1})) \mapsto (m_1^i, m_2^i)$ such that

$$m_1^i + m_2^i = \begin{cases} x[i]_1 + x[i]_2 & \text{if} \quad c_1 + c_2 = 1, \\ m_1^{i-1} + m_2^{i-1} & \text{if} \quad c_1 + c_2 = 0. \end{cases}$$

3. Party j individually sets $y_j = m_j^d$, for $j = 1, 2$.

A.3 ($secret$) IndexOfMaximum

($secret$) IndexOfMaximum returns the index of the maximum of the input elements. It is revised from ($secret$) Maximum in Section A.2.

Specification 19 (($secret$) IndexOfMaximum) *Party 1 and Party 2 want to collaboratively execute the secure protocol* $((x[1]_1, x[2]_1, \cdots, x[d]_1), (x[1]_2, x[2]_2, \cdots, x[d]_2))\{d\} \mapsto (y_1, y_2)$ *such that* $y_1 + y_2 =$ *the index of the maximum value of* $(x[1], x[2], \cdots, x[d])$*, where* $x[i]_1, x[i]_2, y_1, y_2 \in \mathbb{Z}_n$ *and* $x[i] = x[i]_1 + x[i]_2$*, for* $i = 1, 2, \cdots, d$.

Algorithm 7 Calculate $index$ = the index of the maximum of $(x[1], x[2], \cdots, x[d])$

$max \leftarrow x[1]$
$index \leftarrow 1$
for $i \leftarrow 2, 3, \cdots, d$ **do**
  if $max < x[i]$ **then**
    $max \leftarrow x[i]$
    $index \leftarrow i$
  end if
end for

Based on Algorithm 7, the ($secret$) IndexOfMaximum protocol details are as follows:

PROTOCOL ($secret$) IndexOfMaximum

1. Party 1 and Party 2 jointly execute the Comparison protocol

   $(x[1]_1 - x[2]_1, x[1]_2 - x[2]_2) \mapsto (c_1, c_2)$ such that

$$c_1 + c_2 = \begin{cases} 1 & \text{if} \quad x[1] - x[2] < 0, \\ 0 & \text{otherwise.} \end{cases}$$

2. These two parties collaboratively run the If-Then-Else protocol

   $((c_1, x[1]_1, x[2]_1), (c_2, x[1]_2, x[2]_2)) \mapsto (m_1^2, m_2^2)$ such that

$$m_1^2 + m_2^2 = \begin{cases} x[2]_1 + x[2]_2 & \text{if} \quad c_1 + c_2 = 1, \\ x[1]_1 + x[1]_2 & \text{if} \quad c_1 + c_2 = 0. \end{cases}$$

3. Party 1 individually computes $ind_1^2 = 1 + c_1$ and Party 2 individually sets $ind_2^2 = c_2$, for $j = 1, 2$.

4. For $i = 3, 4, \cdots, d$, repeat Step 4a to Step 4c.

   (a) Party 1 and Party 2 jointly execute the Comparison protocol

   $(m_1^{i-1} - x[i]_1, m_2^{i-1} - x[i]_2) \mapsto (c_1, c_2)$ such that

$$c_1 + c_2 = \begin{cases} 1 & \text{if} \quad m_1^{i-1} + m_2^{i-1} - x[i]_1 - x[i]_2 < 0, \\ 0 & \text{otherwise.} \end{cases}$$

(b) These two parties collaboratively run the If-Then-Else protocol
$((c_1, x[i]_1, m_1^{i-1}), (c_2, x[i]_2, m_2^{i-1})) \mapsto (m_1^i, m_2^i)$ such that

$$m_1^i + m_2^i = \begin{cases} x[i]_1 + x[i]_2 & \text{if} \quad c_1 + c_2 = 1, \\ m_1^{i-1} + m_2^{i-1} & \text{if} \quad c_1 + c_2 = 0. \end{cases}$$

(c) These two parties collaboratively run the If-Then-Else protocol
$((c_1, ind_1^{i-1}, i), (c_2, ind_2^{i-1}, 0)) \mapsto (ind_1^i, ind_2^i)$ such that

$$ind_1^i + ind_2^i = \begin{cases} ind_1^{i-1} + ind_2^{i-1} & \text{if} \quad c_1 + c_2 = 1, \\ i & \text{if} \quad c_1 + c_2 = 0. \end{cases}$$

5. Party j individually sets $y_j = ind_j^d$, for $j = 1, 2$.

Note that no If-Then-Else protocol is needed to compute ind^2 in Step 3 of protocol ($secret$) IndexOfMaximum since both alternatives (1 and 2) are $public$. But for ind^i, for $i = 3$ to d, the If-Then-Else protocol is necessary since at least one of the alternatives (ind^{i-1}) is $secret$.

A.4 Exponential

Algorithm 8 is a simple repeated squaring algorithm to compute the exponentiation [CLRS01]. Note that in this algorithm the exponent y is treated as a sequence of bits.

Specification 20 (Exponential) *Party 1 and Party 2 share the base while the exponent is public. They want to securely execute the protocol* $(x_1, x_2)\{y\} \mapsto (z_1, z_2)$ *such that* $z_1 + z_2 = (x_1 + x_2)^y$.

PROTOCOL Exponential

1. Party j individually sets $X_j = x_j$, for $j = 1, 2$.

2. Party 1 and Party 2 individually transform the public exponent y from $\mathbb{Z}_n$ into $\mathbb{Z}_2$ such that
$(b^{k_y} b^{k_y - 1} \cdots b^1 b^0)_2 = y$, where $k_y = \lfloor \log_2 y \rfloor, y \in \mathbb{Z}_n, b^i \in \mathbb{Z}_2$, for $i = 0, 1, \cdots, k_y$.

Algorithm 8 Calculate $u = x^y$

$(b^k b^{k-1} \cdots b^1 b^0)_2 \leftarrow y$
$u \leftarrow 1$
if $b^0 = 1$ **then**
  $u \leftarrow x$
end if
$v \leftarrow x$
for $i = 1, 2, \cdots, k$ **do**
  $v \leftarrow v^2$
  if $b^i = 1$ **then**
    $u \leftarrow u \cdot v$
  end if
end for

3. Party 1 sets $u_1^0 = b^0 X_1 + (1 - b^0)$, while Party 2 sets $u_2^0 = b^0 X_2$, such that

$$u_1^0 + u_2^0 \pmod{n} = \begin{cases} X_1 + X_2 & \text{if} \quad b^0 = 1 \\ 1 & \text{if} \quad b^0 = 0. \end{cases}$$

4. Party 1 and Party 2 jointly execute the Square protocol $(X_1, X_2) \mapsto (v_1^0, v_2^0)$ such that $v_1^0 + v_2^0 \pmod{n} = (X_1 + X_2)^2$.

5. For $i = 0, 1, \cdots, k_y - 2$, repeat Step 5a and Step 5b.

   (a) If $b^{i+1} = 1$, the two parties jointly execute the Product protocol $((u_1^i, v_1^i), (u_2^i, v_2^i)) \mapsto (u_1^{i+1}, u_2^{i+1})$ such that $u_1^{i+1} + u_2^{i+1} \pmod{n} = (u_1^i + u_2^i)(v_1^i + v_2^i)$. If $b^{i+1} = 0$, Party j locally sets $u_j^{i+1} = u_j^i$, for $j = 1, 2$.

   (b) The two parties jointly execute the Square protocol $(v_1^i, v_2^i) \mapsto (v_1^{i+1}, v_2^{i+1})$ such that $v_1^{i+1} + v_2^{i+1} \pmod{n} = (v_1^i + v_2^i)^2$.

6. For $i = k_y - 1$, repeat Step 5a.

7. Party j locally sets $z_j = u_j^{k_y}$, for $j = 1, 2$.

Appendix B

Proofs

B.1 The commodity-based Scalar-product protocol

In this section, the author shows why $y_a + y_b$ equals to $\overrightarrow{X_a} \cdot \overrightarrow{X_b}$ in the commodity-based Scalar-product protocol introduced in Section 5.2.

From Table 5.2, we know that $t = \overrightarrow{X'_a} \cdot \overrightarrow{X_b} + r_b - y_b$. Since $\overrightarrow{X'_a} = \overrightarrow{X_a} + \overrightarrow{R_a}$, we can get $t = \overrightarrow{X_a} \cdot \overrightarrow{X_b} + \overrightarrow{R_a} \cdot \overrightarrow{X_b} + r_b - y_b$.

$$\begin{aligned} t &= \overrightarrow{X'_a} \cdot \overrightarrow{X_b} + r_b - y_b \\ &= (\overrightarrow{X_a} + \overrightarrow{R_a}) \cdot \overrightarrow{X_b} + r_b - y_b \\ &= \overrightarrow{X_a} \cdot \overrightarrow{X_b} + \overrightarrow{R_a} \cdot \overrightarrow{X_b} + r_b - y_b \end{aligned}$$

In addition, we know $y_a = t - \overrightarrow{R_a} \cdot \overrightarrow{X'_b} + r_a$ from Table 5.2. Since $\overrightarrow{X'_a} = \overrightarrow{X_a} + \overrightarrow{R_a}$ and

$r_a + r_b - \overrightarrow{R_a} \cdot \overrightarrow{R_b} = 0$ (see Table 5.1 and Table 5.2), we can get $y_a = \overrightarrow{X_a} \cdot \overrightarrow{X_b} - y_b$.

$$
\begin{aligned}
y_a &= t - \overrightarrow{R_a} \cdot \overrightarrow{X_b'} + r_a \\
&= t - \overrightarrow{R_a} \cdot (\overrightarrow{X_b} + \overrightarrow{R_b}) + r_a \\
&= t - \overrightarrow{R_a} \cdot \overrightarrow{X_b} - \overrightarrow{R_a} \cdot \overrightarrow{R_b} + r_a \\
&= (\overrightarrow{X_a} \cdot \overrightarrow{X_b} + \overrightarrow{R_a} \cdot \overrightarrow{X_b} + r_b - y_b) - \overrightarrow{R_a} \cdot \overrightarrow{X_b} - \overrightarrow{R_a} \cdot \overrightarrow{R_b} + r_a \\
&= \overrightarrow{X_a} \cdot \overrightarrow{X_b} + (r_a + r_b - \overrightarrow{R_a} \cdot \overrightarrow{R_b}) - y_b \\
&= \overrightarrow{X_a} \cdot \overrightarrow{X_b} - y_b
\end{aligned}
$$

Therefore, $y_a + y_b = \overrightarrow{X_a} \cdot \overrightarrow{X_b}$.

Bibliography

[AD00] Mikhail J. Atallah and Wenliang Du. Secure multi-party computational geometry. *Algorithms and Data Structures, Lecture Notes in Computer Science*, 2125:165–179, 2000.

[Bea97] Donald Beaver. Commodity-based cryptography (extended abstract). In *STOC '97: Proceedings of the 29th Annual ACM Symposium on Theoryof Computing*, pages 446–455, New York, NY, USA, 1997. ACM Press.

[BO07] Paul Bunn and Rafail Ostrovsky. Secure two-party k-means clustering. In *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 486–497, New York, NY, USA, 2007. ACM.

[BS08] Felix Brandt and Tuomas Sandholm. On the existence of unconditionally privacy-preserving auction protocols. *ACM Transactions on Information and System Security (TISSEC)*, 11(2):6, 2008.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13:143–202, 2000.

[CB10] Alfredo Cuzzocrea and Elisa Bertino. A secure multiparty computation privacy preserving olap framework over distributed xml data. In *Proceedings of the 2010 ACM*

Symposium on Applied Computing, SAC '10, pages 1666–1673, New York, NY, USA, 2010. ACM.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 2 edition, 2001.

[Com00] Committee on National Statistics. Improving access to and confidentiality of research data: Report of a workshop. In Christopher Mackie and Norman Bradburn, editors, *NRC '00: National Research Council*, Washington, D.C., 2000. Commission on Behavioral and Social Sciences and Education, National Academy Press.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons Inc., 2nd edition, 1991.

[CWLH05] Yi-Ting Chiang, Da-Wei Wang, Churn-Jung Liau, and Tsan-Sheng Hsu. Secrecy of two-party secure computation. *Data and Applications Security XIX, Lecture Notes in Computer Science*, 3654:114–123, 2005.

[DA01a] Wenliang Du and Mikhail J. Atallah. Privacy-preserving cooperative scientific computations. In *CSFW '01: Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, pages 273–282, Washington, DC, USA, 2001. IEEE Computer Society.

[DA01b] Wenliang Du and Mikhail J. Atallah. Privacy-preserving cooperative statistical analysis. In *ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference*, pages 102–110, Washington, DC, USA, 2001. IEEE Computer Society.

[Du01] Wenliang Du. *A Study of Several Specific Secure Two-party Computation Problems*. PhD thesis, Purdue University, August 2001.

[DZ02] Wenliang Du and Zhijun Zhan. A practical approach to solve secure multi-party computation problems. In *NSPW '02: Proceedings of the 2002 Workshop on New Security Paradigms*, pages 127–135, New York, NY, USA, 2002. ACM Press.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[EGS03] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '03, pages 211–222, New York, NY, USA, 2003. ACM.

[FR96] Matthew K. Franklin and Michael K. Reiter. The design and implementation of a secure auction service. *Software Engineering, IEEE Transactions on*, 22(5):302–312, 1996.

[GLLM04] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. On private scalar product computation for privacy-preserving data mining. *Information Security and Cryptology – ICISC 2004*, 3506/2005:104–120, 2004.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *STOC '87: Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York, NY, USA, 1987. ACM Press.

[Gol04] Oded Goldreich. *Foundations of Cryptography, Volume II Basic Applications*. Cambridge University Press, 1st edition, 2004.

[JS02] Ari Juels and Michael Szydlo. A two-server, sealed-bid auction protocol. In *Proceedings of the 6th international conference on Financial cryptography*, pages 72–86. Springer-Verlag, 2002.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC '88: Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 20–31, New York, NY, USA, 1988. ACM.

[Kil91] Joe Kilian. A general completeness theorem for two party games. In *STOC '91: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 553–560, New York, NY, USA, 1991. ACM.

[LP04] Yehuda Lindell and Benny Pinkas. A proof of yao's protocol for secure two-party computation. In *Electronic Colloquium on Computational Complexity*, volume 11, pages 607–620. Citeseer, 2004.

[LP09] Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):59–98, 2009.

[MH97] Tom Mitchell and McGraw Hill. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 1997.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay — a secure two-party computation system. In *USENIX Security '04: Proceedings of the 13th Symposium on Security, Usenix*, pages 287–302, 2004.

[Nie09] Janus Dam Nielsen. *Languages for Secure Multiparty Computation and Towards Strongly Typed Macros*. PhD thesis, Department of Computer Science, University of Aarhus Denmark, February 2009.

[NPS99a] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *EC '99: Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, New York, NY, USA, 1999. ACM Press.

[NPS99b] Moni Naor, Benny Pinkas, and Reuben Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

[Rab81] Michael O. Rabin. How to exchange secrets by oblivious transfer. Technical report, Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[SZW+07] Chih-Hao Shen, Justin Zhan, Da-Wei Wang, Tsan-Sheng Hsu, and Churn-JungLiau. Information-theoretically secure number-product protocol. In *ICMLC '07: International Conference on Machine Learning and Cybernetics*, volume 5, pages 3006–3011, 2007.

[WCH+09] I-Cheng Wang, Kung Chen, Tsan-Sheng Hsu, Churn-Jung Liau, Chih-Hao Shen, and Da-Wei Wang. Performance evaluation and tradeoffs of essential secure multi-party computation protocols. Technical Report TR-IIS-09-005, Institute of Information Science, Academia Sinica, Taiwan, May 2009.

[WLCH06] Da-Wei Wang, Churn-Jung Liau, Yi-Ting Chiang, and Tsan-Sheng Hsu. Information theoretical analysis of two-party secret computation. *Data and Applications Security XX, Lecture Notes in Computer Science*, 4127:310–317, 2006.

[WSH+08] I-Cheng Wang, Chih-Hao Shen, Tsan-Sheng Hsu, Churn-Chung Liau, Da-Wei Wang, and Justin Zhan. Towards empirical aspects of secure scalar product. In *ISA '08: IEEE International Conference on Information Security and Assurance*, pages 573–578, April 2008.

[Yao82] Andrew C. Yao. Protocols for secure computation. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 160–164, November 1982.

[Yao86] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27rd Annual IEEE Symposium on Foundations of Computer Science*, pages 162–167, November 1986.

[ZWH+08] Justin Zhan, I-Cheng Wang, Chia-Lung Hsieh, Tsan-Sheng Hsu, Churn-Jung Liau, and Da-Wei Wang. Towards efficient privacy-preserving collaborative recommender systems. In *GrC '08: IEEE International Conference on Granular Computing*, pages 778–783, Aug. 2008.