國立交通大學

資訊科學系

碩 士 論 文

基於同代像加密法秘密計算富翁問題

An Efficient Solution to The Millionaires' Problem based on Homomorphic Encryption Schemes

研 究 生：林孝盈

指導教授：曾文貴　教授

中 華 民 國　九 十 四　年 六 月

基 於 同 代 像 加 密 法 秘 密 計 算 富 翁 問 題

學 生：林孝盈　　　　　　　　　指導教授：曾文貴 博士

國立交通大學資訊科學研究所

摘要

我們提出了一個可以秘密計算富翁問題的協定，而這個協定可以在 semi-honest 假設下被證明是安全的。我們的協定可以透過加法或者乘法的同代像加密法(homomorphic encryption schemes)來建構。在以前提出的協定中，都是以加法或者 XOR 特性的同代像加密法來建構。在計算量與傳輸量上，我們的協定與之前的協定是在同樣的漸近等級(asymptotic order)。然而因為乘法同代像加密法在實作上比加法同代像加密法來得有效率，使得我們的協定在實作上可以保有這樣的優點。

關鍵詞：秘密計算、大於問題、富翁問題、同代像加密法

An Efficient solution to the Millionaires' Problem based on Homomorphic Encryption Schemes

Student：Hsiao-Ying Lin Advisor：Dr. Wen-Guey Tzeng

Institute of Computer and Information Science
National Chiao Tung University

ABSTRACT

We proposed a two-round protocol for solving the Millionaires' Problem in the setting of semi-honest parties. Our protocol uses either multiplicative or additive homomorphic encryptions. Previously proposed protocols used additive or XOR homomorphic encryption schemes only. The computation and communication costs of our protocol are in the same asymptotic order as those of the other efficient protocols. Nevertheless, since multiplicative homomorphic encryption scheme is more efficient than an additive one practically, our construction saves computation time and communication bandwidth in practicality.

Keywords: secure computation, the greater than problem, the socialist millionaires' problem, homomorphic encryption schemes.

誌謝

首先要感謝我的指導教授曾文貴博士，在這兩年來的指導與教誨，讓我在學習與研究上獲益良多。再者要感謝我的男友明智以及我的家人，兩年來在生活鎖事上無微不致的照顧,以及精神上無比的支持。同時也感謝實驗室的康康學長，季穎學姊，以及所有成員們平時提供的諮詢服務以及數不清的歡樂時光。僅以此文獻上我誠摯的感謝。

Contents

Chapter 1

Introduction

In recent years, there are many applications in processing private data over the network, such as the private bidding. The need for privacy can be met by secure computation. Secure computation is to compute a public function with each party's private input. After computation, only the evaluation result is known and the private inputs are not exposed except those derived from the result. Secure two-party computation was first proposed [Yao82]. The two-party case was subsequently generalized to the multi-party case.

For secure two-party computation, there are two kinds of approaches. One provides the general solution to all functions that can be expressed as the circuits. The other one provides a solution to a specific function, which is often an important or a fundamental function. We illustrate the equality function and the mean function as examples. The equality function is to check whether the two inputs are equal or not. Secure computation of the equality function is also called the private matching. The mean function is to find the mean value from the union of the two private databases. It can be applied in some statistically analysis over databases that belong to different departments.

In this paper, we want to solve the greater than function. It is also called the Millionaires' problem. The Millionaires' problem is to determine who is richer between two parties such that no information about a party's amount of assets is leaked to the other party. The problem can be expressed by the greater than function. The greater than function is to determine whether one input is larger than the other or not. Secure computation of the greater than function has many applications, such as in private bidding. Consider the situation that in the internet a seller wants to sell an item at least y dollars while a buyer wants to buy it at price x. The deal can be done if and only if $x > y$, but the seller doesn't want to expose the base price before the deal is done. We call the application as the private bidding and it can be constructed by secure computation of the greater than function.

In the history of secure computation of the greater than function, Yao [Yao82] first proposed a solution to the problem. Thereafter, many other protocols with great improvement are proposed. Some protocols [BK04, IG03, Fis01] solve the problem directly by analyzing the special properties of the problem. Some others [ST04] solve it in the content of secure two-party computation in which the problem is represented as secure evaluation of the "greater than" boolean circuit with encrypted inputs. The former solutions are more efficient, while the later ones are more general.

We analyze the special properties of the greater than function. We find that the greater than function can be reduced to the intersection problem of two sets by a special coding for the private inputs. We could tackle the set intersection problem by the method in [FNP04]. Nevertheless, the protocol for the greater than function by using the set intersection protocol in [FNP04] directly is less efficient in both computation and communication. This is the same case when we use the disjointness protocol [KM05]. We also can use the equality protocol [FNP04, Lip03, BST01] to construct the set intersection protocol. However, it will need n^2 equality tests (or secure comparisons) to compute the set intersection, where n is the size of the set. We solve the greater than function by further probing the property of our coding method.

Our protocol is two-round and it can be based on either an additive or a multiplicative homomorphic encryption scheme, while most previous protocols [BK04, Fis01] are based on additive or XOR encryption schemes only.

1.1 Related Work

Secure multiparty computation (or secure function evaluation) is to compute a public function with each party's private input such that in the end only the evaluation result is known and the private inputs are not exposed except those derived from the result. Yao [Yao82] first proposed such a protocol for the greater than problem, which is an instantiation of secure computation. Nevertheless, the cost of the protocol is exponential in both time and space. Later one, Yao [Yao86] and Goldreich, etc [GMW87] used the technique of scrambled circuits to solve the general multiparty computation problem. By applying this technique to the greater than problem, the cost of the resulting protocol in computation and communication is linear. Recently, Schoenmakers and Tuyls [ST04] used threshold homomorphic encryption schemes as a tool to solve the multiparty computation problem. Applying to the concrete greater than problem, it provides a threshold greater than protocol, in which the private inputs are shared among a group of parties. The protocol takes $O(n)$ rounds.

On the other hand, protocols for solving the greater than problem directly are more efficient. These protocols usually take a constant number of rounds. Ioannidis and Grama [IG03] used 1-out-of-2 oblivious transfer scheme to construct the greater than protocol that runs n copies of the OT scheme in parallel, where n is the length of the private inputs. However, the length of the private inputs is restricted by the secure parameter of the based OT schemes. Fischlin [Fis01] used the Goldwasser-Micali encryption scheme (GM-encryption) to construct a two-round greater than protocol. The GM encryption scheme has the XOR, NOT and re-randomization properties. They modified the scheme to get an AND property, which can be performed once only. The computation cost $O(\lambda n)$ modular multiplications which is very efficient, where λ is the secure parameter. Nevertheless, the overall communication cost is $O(\lambda n \log N)$ is less efficient, where N is the modulus. In [BK04], Blake and Kolesnikov used the additive homomorphic Paillier cryptosystem to construct a two-round greater than protocol. The computation cost is $O(n \log N)$ and the communication cost is $O(n \log N)$. Our protocol keep the same communication cost as [BK04] while saving the computation cost 68%.

Chapter 2

Preliminaries and Definitions

2.1 Secure two-party computation

We first introduce some important properties of secure two-party computation.

Secure two-party computation is the two-party case of secure computation. To classify the level of privacy, we have the semi-honest case and the malicious case. Semi-honest and malicious cases are used to describe the behaviors that players can perform in the protocol. In the semi-honest setting, players can only follow the protocol and try to get some extra information from the transcripts of the protocol. In the malicious setting, players can perform any behaviors and gain extra information via those "bad" behaviors. Our protocol is secure in the semi-honest setting.

Another issue of secure two-party computation is fairness. When the protocol output the final result, does each party get the result at the same time? One party may get the result first and then abort the protocol while the other one doesn't get the result yet. This is not a fair case. The fairness is satisfied when two parties don't take too much advantages than each other. Traditionally, two parties can exchange the messages bit by bit. In every phase in the exchange process, one party get at most one more bit information than the other party.

2.2 Problem definition

The greater problem is a two-party secure computation problem of the greater than function. We define the greater than function f such that $f(x, y) = 1$ if and only if $x > y$. We say that the two parties are Alice and Bob where x is Alice's private input and y is Bob's. To formally define Alice's privacy, we need describe that the private input of Alice doesn't reveal to Bob via the exchange messages. We use a simulator to simulate a real view of Bob without the private input of Alice. If the simulator can generate an indistinguishable view from the real one, we say that Bob can not get extra information about the private input of Alice from the protocol. In the same way, we define the privacy of Bob.

A solution protocol Π to the GT problem should meet the following requirements.

1. The involved parties Alice and Bob are both polynomial-time bounded probabilistic Turing machines. We assume that Alice and Bob are semi-honest. That is, they shall follow the protocol step by step, but try to get extra information by more computation.

2. Correctness: After execution of Π, Alice returns 1 if and only if $x > y$.

3. Alice's privacy: Holdings of x or x' $(x' \neq x)$ are not computationally distinguishable by Bob. Let $View_B^{\Pi}$ be the real view of Bob when interacting with Alice with private input x. We say that Alice's privacy is guaranteed if there exists a simulator Sim_B such that for any x, $Sim_B(y)$ generates

a view indistinguishable from the view of Bob in the execution of the real protocol, that is,

$$Sim_B(y) \equiv^c VIEW_B^{\Pi}(A(x), y)$$

4. Bob's privacy: Alice cannot get extra information except those derived from x and $b = f(x, y)$. Bob's privacy is guaranteed if there exists a simulator Sim_A, such that for any y' with $f(x, y') = b$, $Sim_A(x, b)$ can generate a view indistinguishable from the view of Alice in the real execution, that is

$$Sim_A(x, f(x, y)) \equiv^c VIEW_A^{\Pi}(x, B(y'))$$

2.3 Homomorphic encryption with scalaring

We review multiplicative and additive homomorphic encryption schemes with the property of scalaring. Multiplicative homomorphic encryption schemes are usually more efficient than additive homomorphic encryption schemes,

An encryption scheme is *multiplicative homomorphic* if and only if

$$D(E(m_1) \odot E(m_2)) \cong m_1 \times m_2,$$

where $\odot$ is an operator, $E(\cdot)$ is the encryption function of the scheme, and $D(\cdot)$ is the decryption function of the scheme.

In the same way, an encryption scheme is *additive homomorphic* if and only if

$$D(E(m_1) \odot E(m_2)) \cong m_1 + m_2.$$

.

An encryption is *scalarable* if $c = E(m)$ can be mapped randomly to a ciphertext $c' = E(m^k)$ or $E(km)$ for a random k. If the encryption scheme is additive homomorphic, the scalaring process doesn't influence the encryption of 0.

$$E(0)^k = E(k \times 0) = E(0).$$

If the encryption scheme is multiplicative homomorphic, the scalaring process doesn't influence the encryption of 1.

$$E(1)^k = E(1^k) = E(1).$$

In both cases, if the message is not the specific value we mentioned, the result of the scalaring process will be a randomized value since we select k randomly. Later in our construction, we will need the property to randomize some computation result in order to protect the secret.

The ElGamal encryption scheme is a multiplicative homomorphic encryption scheme with the scalaring property. For efficiency of computation, we modify the scheme so that each decryption takes 1 modular exponentiation. This modification does not affect the security of the scheme. Let $r \in_R S$ denote that r is chosen from S uniformly and independently.

- Key generation: Let $p = 2q + 1$, where p and q are both primes. Let G_q be the subgroup QR_p and g is a generator of G_q. The public key is $h = g^{-\alpha}$, where $\alpha \in Z_q$ is the corresponding private key.

- Encryption: The encryption of message $m \in G_q$ is a pair $E(m) = (a, b) = (g^r, mh^r)$, where $r \in_R Z_q$.

- Decryption: For a ciphertext $c = (a, b)$, the message is computed from $D(c) = b \times a^\alpha = m$.

- Scalaring: We can scalarize a ciphertext $c = E(m) = (a, b)$ by computing $c' = E(m^k) = (a^k, b^k)$ for $k \in_R Z_q$. If $m = 1$, the scalaring operation does not change the content of encryption. Scalaring makes c' indistinguishable from a random pair due to the DDH assumption (below).

The ElGamal encryption scheme is multiplicative homomorphic since

$$\begin{aligned} E(m_1) \odot E(m_2) &= (g^{r_1}, m_1 h^{r_1}) \odot (g^{r_2}, m_2 h^{r_2}) \\ &= (g^{r_1+r_2}, (m_1 \times m_2) h^{r_1+r_2}) \\ &= E(m_1 \times m_2) \end{aligned}$$

The security of ElGamal scheme is based on the DDH assumption, which states that it is computationally infeasible to distinguish the following two distribution ensembles:

- $D = (g^a, g^b, g^{ab})$, where $a, b \in_R Z_q$.

- $R = (g^a, g^b, g^c)$, where $a, b, c \in_R Z_q$.

If we only need an encryption of a random number, we need not choose a random number and encrypt it. This costs an encryption time. Instead, we choose a random pair $c = (a, b) \in_R G_q^2$, which is an encryption of some random number. By this technique, we save the encryption cost, which is crucial to the efficiency of our GT protocol.

The Paillier encryption scheme is additive homomorphic, which is as follows:

- Key generation: Let $N = pq$ be the RSA-modulus and g be an integer of order αN modulo N^2 for some integer α. The public key is (N, g) and the private key is $\lambda(N) = lcm((p-1), (q-1))$.

- Encryption: The encryption of message $m \in Z_N$ is $E(m) = g^m r^N \bmod N^2$, where $r \in_R Z_N^*$.

- Decryption: For ciphertext c, the message is computed from

$$m = \frac{L(c^{\lambda(N)} \bmod N^2)}{L(g^{\lambda(N)} \bmod N^2)},$$

  where $L(u) = \frac{u-1}{N}$.

- Scalaring: For ciphertext $c = E(m)$, the scalaring is done by computing $c' = E(km) = c^k$ for $k \in Z_N^*$. If $m = 0$, the scalaring operation does not change the content of encryption.

The security of the scheme is based on the CRA (Composite Residuosity assumption, which states that it is computationally infeasible to distinguish whether an element $z \in Z_{N^2}^*$ is an n-residue or not.

The scheme is additive homomorphic since

$$\begin{aligned} E(m_1) \odot E(m_2) &= (g^{m_1} r_1^N) \cdot (g^{m_2} {r_2}^N) \\ &= g^{m_1+m_2} (r_1 r_2)^N \\ &= E(m_1 + m_2). \end{aligned}$$

2.4 0-encoding and 1-encoding

The main idea of out construction is to reduce the GT problem to the set intersection problem. We use two special encodings, 0-encoding and 1-encoding.

Let $s = s_n s_{n-1} ... s_1 \in \{0, 1\}^n$ be a binary string of length n. The *0-encoding* of s is the set S_s^0 of binary strings such that

$$S_s^0 = \{s_n s_{n-1} ... s_{i+1} 1 | s_i = 0, 1 \leq i \leq n\}$$

The *1-encoding* of s is the set S_s^1 of binary strings such that

$$S_s^1 = \{s_n s_{n-1} ... s_i | s_i = 1, 1 \leq i \leq n\}$$

Both S_s^1 and S_s^0 have at most n elements.

If we encode x into its 1-encoding S_x^1 and y into its 0-encoding S_y^0, we can see that

$$x > y \text{ if and only if } S_x^1 \text{ and } S_y^0 \text{ has a common element.}$$

We give an example. Let $x = 6 = 110_2$ and $y = 2 = 010_2$ of length 3 (we fill in the leading zeros.) We have $S_x^1 = \{1, 11\}$ and $S_y^0 = \{1, 011\}$. Since $S_x^1 \cap S_y^0 \neq \emptyset$, we have $x > y$ indeed. If $x = 2 = 010_2$ and $y = 6 = 110_2$, we have $S_x^1 = \{01\}$ and $S_y^0 = \{111\}$. Since $S_x^1 \cap S_y^0 = \emptyset$, we have $x \leq y$.

We note that the strings in S_x^1 have a prefix relation and the strings in S_y^0 also have a prefix relation when removing the last bit. Our protocol exploits this relation.

Theorem 2.4.1. *x is greater than y if and only if S_x^1 and S_y^0 have a common element.*

Proof. Let $x = x_n x_{n-1} ... x_1 \in \{0, 1\}^n$ and $y = y_n y_{n-1} ... y_1 \in \{0, 1\}^n$. For the forward direction, we can see that if $x > y$, there is a position i such that $x_i = 1$ and $y_i = 0$, and for all j, $n \geq j > i$, $x_j = y_j$. We have $x_n x_{n-1} \ldots x_i \in S_x^1$ by 1-encoding and $y_n y_{n-1} \cdots y_{i+1} 1 \in S_y^0$ by 0-encoding. Thus, S_x^1 and S_y^0 have a common element.

For the backward direction, let $t = t_n t_{n-1} ... t_i \in S_x^1 \cap S_y^0$ for some $t_i = 1$. Since $t \in S_x^1$, $x_n x_{n-1} \ldots x_i = t_n t_{n-1} \ldots t_i$, and since $t \in S_y^0$, $y_n y_{n-1} \ldots y_{i+1} \bar{y}_i = t_n t_{n-1} \ldots t_i$. We can see that $x > y$. $\square$

Chapter 3

Our Protocols against semi-honest adversaries

If Alice and Bob compare the elements in S_x^1 and S_y^0 one by one, it would need $O(n^2)$ comparisons. Nevertheless, they can only compare the corresponding strings of the same length (if both of them exist) in S_x^1 and S_y^0. This reduces the number of comparison to $O(n)$.

Let (G, E, D) be a multiplicative homomorphic encryption scheme. Alice uses a $2 \times n$-table $T[i, j]$, $i \in \{0, 1\}$, $1 \leq j \leq n$, to denote its input $x = x_n x_{n-1} \cdots x_1$ with

$$T[x_j, j] = E(1) \text{ and } T[\bar{x}_j, j] = E(r) \text{ for some random } r \in G_q.$$

Since Alice need not know r (each entry uses a distinct r), she randomly selects $(a, b) \in G_q^2$ for $E(r)$. When Bob wants to compare a string $t = t_n t_{n-1} \cdots t_i$ in S_y^0 with the corresponding string of the same length in S_x^1, he computes

$$c_t = T[t_n, n] \odot T[t_{n-1}, n-1] \cdots \odot T[t_i, i].$$

We can see that c_t is an encryption of 1 if and only if $t \in S_x^1$ except with a negligible probability of incorrectness. Furthermore, since strings in S_y^0 have some sort of prefix relations, Bob can compute all c_t's in at most $2n$ homomorphic operations, instead of n^2 homomorphic operation.

Based on the previous discussion, our GT protocol is as follows:

Protocol 1

1. Alice with private input $x = x_n x_{n-1} \cdots x_1$ does the following:

   - run G to choose a key pair (pk, sk) for (E, D).
   - prepare a $2 \times n$-table $T[i, j]$, $i \in \{0, 1\}$, $1 \leq j \leq n$, such that

   $$T[x_i, i] = E(1) \text{ and } T[\bar{x}_i, i] = E(r_i) \text{ for some random } r_i \in G_q$$

   - send T to Bob.

2. Bob with private input $y = y_n y_{n-1} \cdots y_1$ does the following:

   - for each $t = t_n t_{n-1} \cdots t_i \in S_y^0$, compute

   $$c_t = T[t_n, n] \odot T[t_{n-1}, n-1] \cdots \odot T[t_i, i].$$

   - prepare $l = n - |S_y^0|$ random encryptions $z_j = (a_j, b_j) \in G_q^2, 1 \leq j \leq l$.
   - scalarize c_t's and permutate c_t's and z_j's randomly as $c_1, c_2, \ldots, c_n$.
   - send $c_1, c_2, \ldots, c_n$ to Alice.

3. Alice decrypts $D(c_i) = m_i$, $1 \leq i \leq n$, and determine $x > y$ if and only if some $m_i = 1$.

When $x \leq y$, there is a negligible probability that our GT protocol returns a wrong answer due to randomization.

Our GT protocol can use additive homomorphic encryption. We only replace $E(1)$ by $E(0)$ in setting up the table T. In the end, Alice determines $x > y$ if and only if some $m_i = 0$.

3.1 Correctness and Security

Theorem 3.1.1. *The protocol constructed as above is a solution to the GT problem and is secure in the semi-honest setting.*

Proof. At first, we verify the correctness. If $x > y$, the error probability is

$$Pr(b = 0 \mid x > y)$$

where b is the output of the protocol. Since $x > y$, there must be an encryption of 1 in the sequence $c_1, c_2, \ldots, c_n$. Thus the output of the protocol must be 1. In other words,

$$Pr(b = 0 \mid x > y) = 0.$$

If $x \leq y$, the error probability is

$$Pr(b = 1 \mid x \leq y).$$

Since $x \leq y$, $c_1, c_2, \ldots, c_n$ must be made from random numbers. The output of the protocol is 1, it means that there exists an encryption of 1 in the sequence. Now we can conclude that the error happens in three possible situations (events).

1. A pair of random numbers forms an encryption of 1.

2. A product of those random numbers forms an encryption of 1.

3. In the scalaring phase, the random numbers become an encryption of 1 after the scalaring.

We use the union bound to bound the total error probability. So what we need to know is the probabilities of those events happen. In the protocol, we choose the random numbers from G_q^2 as a random encryption. A pair of random numbers forms an encryption of 1 with probability $1/q$. The probability of the product of random numbers becomes an encryption of 1 is equal to the probability of a random pair being an encryption of 1. The probability of a pair of random numbers forms an encryption of 1 after scalaring is $O(1/q)$. There are n elements in the sequence of the ciphertexts, and the error happens when one of the events happens in one element of the sequence. The total error probability when $x \leq y$ is

$$Pr(b = 1 \mid x \leq y) \leq n \times (1/q + c/q + 1/q) = O(n/q).$$

Since $n \ll q$, the error probability when $x \leq y$ is negligible. The total error probability of the protocol is

$$\sum_{x,y} Pr(b = 1 \mid x \leq y)Pr(x \leq y) + Pr(b = 0 \mid x > y)Pr(x > y) = O(n/q),$$

which is negligible. We can conclude our protocol satisfies the correctness since the error probability is negligible.

For Alice's privacy, we construct a simulator Sim_B which simulates the view of Bob in the real protocol. If Sim_B generates an indistinguishable view from the real one, we can say that Bob doesn't get extra information in the real protocol or we can break the encryption scheme by using the distinguisher (which can distinguish the view of Bob from the view generated by Sim_B). Sim_B takes y and the output of the protocol b as input and randomly select x' as input of Alice in the simulation. Then Sim_B generates the table $T(x')$ according to x'. The view generated by Sim_B is (y, $T_{x'}$) and the view in the real execution is (y, T_x). Now, we need to prove that T_x and T'_x are indistinguishable. We prove it by contradiction. Assume that T_x and T'_x are distinguishable by a distinguisher D_T with advantage ϵ. We can use D_T to construct an adversary to break the security of the encryption scheme with advantage ϵ/n. Due to the security of the ElGamal encryption, T_x and T'_x are indistinguishable. Thus $Sim_B(y)$ and the real view $View_B^{\Pi}(A(x), y)$ are indistinguishable.

For Bob's privacy, we construct a simulator Sim_A to simulate the view of Alice without the private input of Bob. We need the view generated by Sim_A being indistinguishable from the view of Alice in the real execution. Sim_A simulates as follows. The input of Sim_A are the comparison result $b \in \{0, 1\}$ and Alice's private input x. Sim_A uses x to construct the table T for the first step. For the second step, Sim_A generates the sequence $c_1, c_2, \ldots, c_n$ according to the result value b. If $b = 1$, Sim_A generates $n - 1$ random encryptions and one encryption of 1, then Sim_B randomly permutates them as $c_1, c_2, \ldots, c_n$. If $b = 0$, Sim_A generates n random encryptions as $c_1, c_2, \ldots, c_n$. The view generated by Sim_A is $(x, T_x, c_1, c_2, \ldots, c_n, b)$.

Since T_x is constructed by using the value x, the distribution is identical to that in the real execution. For fixed output b, the sequence of the ciphertexts are computationally indistinguishable from the sequence in the real execution due to the scalaring property. Thus, Alice cannot compute Bob's private input y except knowing its relation with x. □

3.2 Efficiency

In this analysis, the base encryption scheme is the ElGamal scheme. Let p be the modulus.

Computation Complexity. Let n be the length of the private inputs. We neglect the cost of choosing random numbers. The cost of choosing a public key pair for Alice is neglected either since this can be done in the setup stage. We don't count the cost of selecting keys in other protocols, either.

In Step 1, Alice encrypts n $1's$. In Step 2, Bob computes c_t, $t \in S_y^0$, by reusing intermediate values. This takes $(2n-3)$ multiplications of ciphertexts at most. Step 2 uses n scalaring operations at most. In Step 3, Alice decrypts n ciphertexts.

To compare fairly, we convert all operations to the number of modular multiplications. For the ElGamal scheme, each encryption takes $2\log p$ modular multiplications, each decryption takes $\log p$ modular multiplications, and each scalaring operation takes $2\log p$ modular multiplications. Overall, our GT protocol needs $5n\log p + 4n - 6$ $(= n \times 2\log p + 2 \times (2n-3) + n \times 2\log p + n \times \log p)$ modular multiplications.

Communication complexity. The size of exchanged messages between Alice and Bob is the size of T and $c_1, c_2, \ldots, c_n$, which is $6n\log p$ $(= 3n \times 2\log p)$ bits.

3.3 Extensions

We can use the collision resistant hash function to construct a simpler protocol. The protocol uses less communication bits.

The protocol is as follows:

Protocol 2

Let h be a public collision-free hash function.

1. Alice encodes x as S_x^1 and lets h_l be the hash value of the length-l string t in S_x^1 if t exists.

2. Alice encrypts h_l as c_l for existent h_l's and randomly selects $c_{l'}$ for missing $h_{l'}$, $1 \leq l' \leq n$.

3. Alice sends $c_1, c_2, \ldots, c_n$ to Bob.

4. Bob encodes y as S_y^0 and computes the hash value h_l' for the length-l string t in S_y^0 if t exists.

5. Bob computes $z_l = (a_l, b_l/h_l')$ for existent h_l' and $z_l = c_l$ for inexistent h_l', where $c_l = (a_l, b_l)$, $1 \leq l \leq n$.

6. Bob scalarizes and permutates $z_1, z_2, \ldots, z_n$ and sends them to Alice.

7. Alice decrypts $z_1, z_2, \ldots, z_l$ and outputs 1 if and only if some message is 1.

Correctness. In the protocol, we assume that h is a collision resistant function. The protocol is correct with overwhelming probability. We assume that the collision probability of the hash function h is ϵ. Let the output of the protocol be $b \in \{0, 1\}$.There are two error cases.

1. If $x > y$, the error probability is

$$Pr(b = 0|x > y).$$

   The protocol outputs 0 if the sequence $z_1, z_2, \ldots, z_n$ contains no encryption of 1. When $x > y$, the encoding set of x and the encoding set of y have a common element. After hashing, the results of the common element in two sets are also the same. Thus it is impossible that the sequence $z_1, z_2, \ldots, z_n$ contains no encryption of 1 when $x > y$. We can conclude that

$$Pr(b = 0|x > y) = 0.$$

2. If $x \leq y$, the error probability is

$$Pr(b = 1|x \leq y).$$

The protocol output 1 if the sequence $z_1, z_2, \ldots, z_n$ contains at least one encryption of 1. Under the following situations when $x \leq y$, the sequence will contain at least one encryption of 1:

- When Alice pads elements to missing items, she randomly select numbers which is an encryption of 1 in accident. This happens with probability $1/q$ for one random element being an encryption of 1.
- There exists a element in the encoding set of x and a element in the encoding set of y such that they have the same length and their hash value collide but their value are not the same. This happens with probability ϵ for a pair of elements.
- After scalaring, a ciphertext becomes an encryption of 1 while it is not originally. This happens with probability $1/q$ for a ciphertext.

We compute the error probability when $x \leq y$ by the union bound.

$$Pr(b = 1|x \leq y) \leq n \times 1/q + n \times \epsilon + n \times 1/q$$

When the collision probability of the hash function is negligible, the error probability when $x \leq y$ is negligible, too.

The total error probability can be bounded.

$$\begin{aligned} & Pr(b=1|x \leq y)Pr(x \leq y) + Pr(b=0|x>y)Pr(x>y) \\ = \;& (n \times 1/q + n \times \epsilon + n \times 1/q) \times Pr(x \leq y) + 0 \\ < \;& (n \times 1/q + n \times \epsilon + n \times 1/q) \end{aligned}$$

With ϵ is negligible, we can claim the protocol is correct with overwhelming probability.

Security. The privacy for Alice and Bob is guaranteed by the semantic secure public key encryption scheme.

For Alice's privacy, we construct a simulator Sim_B to simulate the view of Bob in the real protocol. If the simulated view is indistinguishable from the real one, we can claim that Bob can not get extra information from the transcript in the real protocol. The Bob's view in the real protocol is y, $z_1, z_2, \ldots, z_n$, b. Now we show how we construction the simulator Sim_B. With Bob's input y and the output of the protocol b, simulator randomly selects x' as Alice's input. Then Sim_B computes $z'_1, z'_2, \ldots, z'_n$ according to x'. We need the sequence $(z'_1, z'_2, \ldots, z'_n)$ being indistinguishable from $(z_1, z_2, \ldots, z_n)$. We prove it by contradiction.

If the sequence $(z'_1, z'_2, \ldots, z'_n)$ can be distinguished from $(z_1, z_2, \ldots, z_n)$, there exists a distinguisher D_Z. We can use D_Z to break the encryption scheme or find a collision of the hash function. Given $x, x', (Z_1, Z_2, \ldots, Z_n)$, we assume that D_Z can distinguish that $(Z_1, Z_2, \ldots, Z_n)$ are computed from x or x' with probability σ. By triangular theorem, there exists a position i such that z_i and z'_i can be distinguished with probability σ n where $h(x_i) \neq h(x'_i)$. Since $z_i = E(h(x_i))$ and $z'_i = E(h(x'_i))$ can be distinguished with probability σ/n, and $h(x_i) \neq h(x'_i)$. We can use the distinguisher to break the encryption scheme with probability σ/n. As above, we can see that if D_Z can distinguish with a non-negligible probability, we can break the encryption scheme with a non-negligible probability. We can conclude that $(z_1, z_2, \ldots, z_n)$ and $(z'_1, z'_2, \ldots, z'_n)$ are indistinguishable. Therefore, the requirement of Alice's privacy can be satisfied.

For Bob's privacy, we can use the same proof as in the original protocol.

Communication complexity. In the protocol, Alice sends n ciphertexts to Bob, and Bob sends n ciphertexts back to Alice. The size of a ciphertext is $2\log p$ bits. Thus, the total communication cost of the protocol is $4n\log p$ bits.

Computation comlixity. In the protocol, the computation of Bob for each value from Alice can be completed by inversion of the hash value, a multiplication and a scalaring of the ciphertext. Thus the computation cost in the protocol of Bob is $2n\log p + 2n$ modular multiplications. The computation cost of Alice is $3n\log p$ modular multiplications. The total computation cost of the protocol is $5n\log p + 2n$ modular multiplications.

Chapter 4

Malicious adversaries

Our construction in the previous chapter is secure in the semi-honest setting. In the malicious setting, each round requires additional messages to assure legality of the sent messages. The techniques are mostly based on non-interactive zero-knowledge proof of knowledge. Here we give some discussion of how to make the protocol secure in the malicious setting.

For a malicious Alice, we need to prevent Alice to put specific values in the table and get extra information. For example, Alice can put encryption of 1's in each blank of the table. After all, Alice can know that how many 0's in Bob's private input.

To make sure of legality of Alice's message, we need Alice to prove that each column in the table has exactly one encryption of 1 and the other is an encryption of a random number. For existence of the encryption of 1, we can use the technique of non-interactive zero-knowledge proof. For existence of the encryption of the random number, we let Bob to re-randomize the table and whatever Alice puts in the table (except encryption of 1), it will be randomized. Due to the special property of the encryption of 1, the randomization will not affect the correctness of the protocol. Thus, for malicious Alice, Alice need to prove that each column in the table has exactly one encryption of 1, and Bob must re-randomize the table after he receives it.

We describe how Alice prove that each column in the table has exactly one encryption of 1.

1. Each column in the table has at least one encryption of 1.

   Let $(A, B) = (g^r, m \cdot y^r)$, $(A_1, B_1) = (g^{r'}, m_1 \cdot y^{r'})$ be the values in a column of the table. If at least one of m, m_1 is 1, then $(AA_1, BB_1) = (A_2, B_2)$ will be an encryption of the same message as one of (A, B), (A_1, B_1). It means that either $\log_g A_2/A = \log_y B_2/B = r$ or $\log_g A_2/A_1 = \log_y B_2/B_1 = r'$. Alice proves it by non-interactive zero knowledge proof as follows:

   Let h be a one way hash function. W.l.o.g. we assume that $m = 1$.

   - Alice randomly selects $\beta \in Z_q$ and computes $W_1 = g^\beta, W_2 = y^\beta$.
   - Let $c = h(W_1, W_2)$ be the challenge message. Alice computes c_2 such that $m_1^{c_2} = 1 \mod p$ and lets $c_1 = c - c_2 \mod q$.
   - Let $D_1 = \beta - r'c_1 \mod q$ and $D_2 = \beta - rc_2 \mod q$. Alice sends the proof c, c_1, c_2, D_1, D_2 to Bob.

   Bob is convinced if

   - $c = c_1 + c_2$
   - $c = h(g^{D_1} \cdot (A_2/A)^{c_1}, y^{D_1} \cdot (B_2/B)^{c_1})$
   - $c = h(g^{D_2} \cdot (A_2/A_1)^{c_2}, y^{D_2} \cdot (B_2/B_1)^{c_2})$

2. Each column in the table has at most one encryption of 1.

   Let $(A, B) = (g^r, m \cdot y^r)$, $(A_1, B_1) = (g^{r'}, m_1 \cdot y^{r'})$ be the values in a column of the table. If at most one of m, m_1 is 1, then $(AA_1, BB_1) = (A_2, B_2)$ will be NOT an encryption of 1. Alice need to prove that (A_2, B_2) is not an encryption of 1.

For malicious Bob, we need to make sure of that Bob indeed obeys the protocol, or Bob can get extra information of Alice's secret. For example, if Bob selects and computes values for the first $n/2$ elements of the sequence legally, and puts random numbers for the rest. After getting the output of the protocol, Bob can know whether Alice's input is greater than his or not in the first $n/2$ bits. To protect Alice's privacy, Bob needs to prove every computation he does is legal. This is more complicated since the computation of Bob in the protocol is the major part. After all, Bob needs to prove that the sequence is made legally, too.

Chapter 5

Other protocols

For readability, we review the protocols in [BK04, Fis01].

5.1 Fischlin's GT protocol

Fischlin's GT protocol [Fis01] uses the GM-encryption scheme and a modified GM-encryption. The GM-encryption scheme is as follows:

- Key generation: Let $N = pq$ be the RSA-modulus and z be a quadratic non-residue of Z_n^* with Jacobi symbol $+1$. The public key is (N, z) and the secret key is (p, q).

- Encryption: For a bit b, the encryption is $E(b) = z^b r^2 \bmod N$, where $r \in_R Z_N^*$.

- Decryption: For a ciphertext c, its plaintext is 1 if and only if c is a quadratic non-residue. If c is a quadratic residue in Z_N, c is a quadratic residue in both Z_p^* and Z_q^*.

- xor-property: $E(b_1)E(b_2) = E(b_1 \oplus b_2)$

- not-property: $E(b) \times z = E(b \oplus 1) = E(\bar{b})$
- re-randomization: we can re-randomize a ciphertext c by multiplying an encrytion of 0.

Modified GM-encryption. To get the AND-homomorphic property, we need modify the GM encryption:

- Encryption: For encrypt a bit $b = 1$, $XE(b)$ is a sequence of quadratic residues. If $b = 0$, $XE(b)$ is a sequence of quadratic residues and non-residues. The length of the sequence is determined by a parameter λ.
- Decryption: For decrypting a ciphertext, we need check quadratic residusoity of all elements in the ciphertext.
- AND-property: For two ciphertext $XE(b_1)$ and $XE(b_2)$, their product $XE(b_1) \odot XE(b_2)$ is computed by multiplying elements pairwisely. The product of two ciphertexts is an encryption of b_1 AND b_2.

Protocol and Efficiency. The protocol in [Fis01] uses the properties of the GM- and modified-GM-encrytion schemes.

We use our notation to represent the (optimized) protocol in [Fis01] as follows:

1. Alice with private input $x = x_n x_{n-1} \cdots x_1$ does the following:
    - generate GM-instance N, z.
    - encrypt each bit of x and get $X_i = E(x_i)$ for $i = 1, \ldots, n$
    - send $N, z, X_1, \ldots, X_n$ to Bob
2. Bob with private input $y = y_n y_{n-1} \cdots y_1$ and messages $N, z, X_1, \cdots, X_n$ from Alice does the following:

- encrypt y by the extended encryption and get the result $\vec{Y_i} = (Y_{i,1}, \ldots, Y_{i,\lambda}) = XE(y_i)$, where $Y_{i,j} = E(z^{1-y_i})$ or $E(0)$ randomly.

- embed $[x_i = y_i]$=$[\neg(x_i \oplus y_i)]$ into extended encryption $\vec{E_i}$ for $i = 1, \ldots, n$. $\vec{E_i} = (E_{i,1}, \ldots, E_{i,\lambda}) = XE(\neg(x_i \oplus y_i))$, where $E_{i,j} = X_j \cdot z^{y_i} \bmod N$ or $1 \in QR_N$ randomly.

- compute extended encryptions $\vec{P_i} = \vec{P}_{i+1} \cdot \vec{E}_{i+1} \bmod N = XE(p_i)$, where $\vec{P_n} = (1, \ldots, 1)$ and $p_i = \bigwedge_{j=i+1}^{n}[x_j = y_j]$ for $i = n-1, \ldots, 1$.

- embed $\neg x_i$ into extended encryption $\vec{\bar{X_i}}$ for $i = 1, \ldots, n$. $\vec{\bar{X_i}} = (\bar{X}_{i,1}, \bar{X}_{i,\lambda}) = XE(\neg x_i)$, where $\bar{X}_{i,j} = X_i$ or $1 \in QR_N$ randomly.

- compute terms $t_i = y_i \wedge \bar{x}_i \bigwedge_{j=i+1}^{n}[x_j = y_j]$ in the extended encryption form: For $i = 1, \ldots, n$, $\vec{T_i} = \vec{Y_i} \cdot \vec{\bar{X_i}} \cdot \vec{P_i} \bmod N = XE(t_i)$.

- randomly permute $\vec{T_1}, \ldots, \vec{T_n}$ and send to Alice

3. Alice receives n sequences of λ elements from Bob and does the following:

    - If there exists a sequence of λ quadratic residues then output $y > x$, else output $y \leq x$.

For computation, the protocol needs n GM-encryptions and n modified GM-decryptions in the client side (Alice in our protocol).[1] The server side (Bob in our protocol) needs n modified GM-encryptions and $4n\lambda$ modular multiplications only.

The exchanged messages are n GM-ciphertexts and n modified GM-ciphertexts. Overall, the size is $(1 + \lambda)n \log N$ $(= n \log N + n\lambda \log N)$ bits.

[1]In [Fis01], the computation cost of the client side is neglected.

5.2 Blake and Kolesnikov's GT protocol

The GT protocol in [BK04] is based on the Paillier's encryption scheme. The additive homomorphic property is essential to their construction. Their protocol can be summarized as follows: Let $Enc(m)$ be the encryption of the message m.

1. Alice with private input $x = x_n x_{n-1} \cdots x_1$ does the following:

   (a) runs key generation phase

   (b) encrypts x bit-wise and sends $pk, Enc(x_n), \ldots, Enc(x_1)$ to Bob.

2. Bob with private input $y = y_n y_{n-1} \cdots y_1$ does the following for each $i = 1, \ldots, n$:

   (a) computes $Enc(d_i) = Enc(x_i - y_i)$

   (b) computes $Enc(f_i) = Enc(x_i - 2x_i y_i + y_i)$

   (c) computes $Enc(\gamma_i) = Enc(2\gamma_{i-1} + f_i)$ where $\gamma_0 = 0$

   (d) computes $Enc(\delta_i) = Enc(d_i + r_i(\gamma_i - 1))$ where $r_i \in_R Z_N$

   (e) randomly permutates $Enc(\delta_i)$ and sends to Alice

3. Alice obtains $Enc(\delta_i)$ from Bob, then decrypts. If there exists a value $v \in \{+1, -1\}$ and output v.

In the protocol, if $x > y$, the output value $v = +1$; if $x < y$, $v = -1$.

For computation, the receiver (Alice) needs n encryptions and n decryptions. The sender (Bob) needs n modular multiplications in the $2a$ step, n modular multiplications and n inversions in the $2b$ step, $2n$ modular multiplications in the $2c$ step, and $(2 + \log N)n$ modular multiplications in the $2d$ step. Each inversion takes 1 modular multiplications. Overall, the protocol needs $4n$ modular exponentiations $(\bmod N^2)$ and $7n$ modular multiplication $(\bmod N^2)$

The communication cost is n ciphertexts for the receiver and n ciphertexts for the sender. The overall communication cost is $4n \log N$ bits

Chapter 6

Comparison

Now, we compare our GT protocol with those in [Fis01, BK04] in computation and communication cost. We summarize the cost of operations for the protocols:

- Each GM-encryption needs 2 modular multiplications $(\bmod N)$.

- Each modified GM-encryption needs 2λ modular multiplication $(\bmod N)$ since each encryption contains λ GM-encryptions.

- Each modified GM-decryption needs λ modular multiplications $(\bmod N)$, since there λ elements in a modified GM-ciphertext and quadratic residuosity can be checked in equivalent one modular mulltiplication.

- Each Paillier's encryption requires $2 \log N$ modular multiplications (mod N^2). In [BK04], they encrypt 0 or 1 only, the encryption for $m \in \{0, 1\}$ needs $\log N$ modular multiplications $(\bmod N^2)$.

- Each Paillier's decryption requires $2 \log N$ modular multiplications (mod N^2).

	Alice's computation	Bob's computation	total computation	communication
Protocol 1	$3n \log p$	$2n \log p + 4n - 6$	$5n \log p + 4n - 6$	$6n \log p$
Protocol 2	$3n \log p$	$2n \log p + 2n$	$5n \log p + 2n$	$4n \log p$
[Fis01]	$\lambda n + 2n$	$6n\lambda$	$7n\lambda + 2n$	$(1+\lambda)n \log N$
[BK04]	$12n \log N$	$4n \log N + 28n$	$16n \log N + 28n$	$4n \log N$

*computation cost is measured in the number of modular multiplication
*communication cost is measured in bits
*Alice is called "receiver" in [BK04] and "client" in [Fis01].
*λ is set to $40 \sim 50$ in [Fis01]

Table 6.1: Comparison in computation cost and communication cost.

- Each Paillier's inversion requires one modular multiplications ($\bmod N^2$), where the inversion is done by the extended Euclidean algorithm.

- For Paillier's encryption, each modular multiplication ($\mathrm{mod} N^2$) needs 4 modular multiplication ($\mathrm{mod} N$).

Based on the above discussion, we summarize the comparison in Table 6.1. In the table, the modular multiplication for the protocols in [Fis01, BK04] is mod$N$ and ours is mod$p$. Notice that *Protocol* 2 requires a collision resistant hash function while others do not.

Bibliography

[BK04] Ian F. Blake and Vladimir Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *Proceedings of Advances in Cryptology - ASIACRYPT '04*, volume 3329 of *LNCS*, pages 515–529. Springer-Verlag, 2004.

[BST01] Fabrice Boudot, Berry Schoenmakers, and Jacques Traoré. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111(1-2):23–36, 2001.

[Fis01] Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *Proceedings of the 2001 Conference on Topics in Cryptology: The Cryptographer's Track at RSA*, volume 2020 of *LNCS*, pages 457–472. Springer-Verlag, 2001.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Proceedings of Advances in Cryptology - EUROCRYPT '04*, volume 3027 of *LNCS*, pages 1–19. Springer-Verlag, 2004.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play and mental game. In *Proceedings of the 16th Annual ACM Symposium on the*

Theory of Computing (STOC '87), pages 218–229. ACM, 1987.

[IG03] Ioannis Ioannidis and Ananth Grama. An efficient protocol for yao's millionaires' problem. In *Proceedings of the 36th Hawaii Internatinal Conference on System Sciences 2003*, 2003.

[KM05] Aggelos Kiayias and Antonina Mitrofanova. Testing disjointness of private datasets. In *Proceedings of Financial Cryptography 2005 - FC '05*, LNCS. Springer-Verlag, 2005.

[Lip03] Helger Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *Proceedings of Advances on Cryptology - ASIACRYPT '03*, volume 2894 of *LNCS*, pages 416–433. Springer-Verlag, 2003.

[ST04] Berry Schoenmakers and Pim Tuyls. Pratical two-party computation based on the conditional gate. In *Proceedings of Advances in Cryptology - ASIACRYPT '04*, volume 3329 of *LNCS*, pages 119–136. Springer-Verlag, 2004.

[Yao82] A. C. Yao. Protocols for secure computations. In *Proceedings of 23th Annual Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164. IEEE, 1982.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *Proceedings of 27th Annual Symposium on Foundations of Computer Science (FOCS '86)*, pages 162–167. IEEE, 1986.