An Efficient Solution to The Millionaires' Problem Based on Homomorphic Encryption ⋆ ⋆⋆

Hsiao-Ying Lin and Wen-Guey Tzeng
Department of Computer and Information Science
National Chiao Tung University
Hsinchu, Taiwan 30050
Email:lrain.cis92g@nctu.edu.tw,tzeng@cis.nctu.edu.tw

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

1 Introduction

Yao's Millionaires' ("greater than" or "GT") 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. Yao [Yao82] first proposed a solution for 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 multiparty 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. The GT protocol has many applications, such as in private bidding [Cac99].

In this paper, we analyze the special properties of the GT problem. We find that the GT problem 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 GT problem by using the set intersection protocol in [FNP04] directly is less efficient

⋆ Research supported in part by National Science Council grants NSC-93-2213-E-009-008 and NSC 93-2213-E-009-009
⋆⋆ This version appears in ACNS 2005.

in both computation and communication. We solve the GT problem by further probing the property of our coding method. Our protocol 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. 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 in practicality.

1.1 Related Work

Secure multiparty computation (or secure function evaluation) is to compute a public function with each party's private inputs 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 GT problem, which is an instantiation of secure multiparty 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 GT 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 GT problem, it provides a threshold GT 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 GT 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 GT 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 GT 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 is $O(\lambda n)$ modular multiplication which is very efficient, where λ is the security parameter. Nevertheless, the 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 GT protocol. The computation cost is $O(n \log N)$ and the communication cost is $O(n \log N)$.

2 Preliminaries and Definitions

The GT problem is a two-party secure computation problem of securely evaluating a predicate f such that $f(x, y) = 1$ if and only if $x > y$, where x is Alice's private input and y is Bob's private input. A solution protocol Π for 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, $Sim_B(y)$ can generate 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 $f(x, y') = b$ and $f(x, y)$, $Sim_A(x, b)$ can generate a view indistinguishable from the view of Alice in the real execution, that is
$$Sim_A(x, f(x, y)) \equiv^c VIEW_A^{\Pi}(x, B(y'))$$

2.1 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

$$E(m_1) \odot E(m_2) = E(m_1 \times m_2),$$

where $\odot$ is an operator. An encryption scheme is *additive homomorphic* if and only if

$$E(m_1) \odot E(m_2) = E(m_1 + m_2).$$

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.

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

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

2.2 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

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

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

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

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 $x \le y$.

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 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, $n \ge j > i$, $x_j = y_j$. 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 $x > y$.

3 Our Protocols

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:

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

When $x \le y$, 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 2. *The protocol constructed as above is a solution to the GT problem.*

Proof. We can verify correctness easily. The 1-encoding set of x is embedded into the table T. The 0-encoding set of y is computed by Bob directly. If there is a common element in both sets, some $c_i = E(1)$ by the previous discussion. Otherwise, no such $c_i = E(1)$ exists.

For Alice's privacy, we construct a simulator Sim_B which simulates the protocol by selecting randomly x' as input of Alice, and letting y as input of Bob. The view generated by Sim_B is $(y, T_{x'})$ and the view in the real execution is (y, T_x). 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 also 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 hash function to construct a simpler protocol. The protocol costs less communication bit.

The protocol is as follows: 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 $h_{l'}$, $1 \le l' \le n$.
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 $c_l = (a_l, b_l)$, $1 \le l \le n$.
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.

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 communication cost of the protocol is $4n\log p$ bits.

4 Other protocols

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

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 $\boldsymbol{Y_i} = (Y_{i,1}, \ldots, Y_{i,\lambda}) = XE(y_i)$, 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 $\boldsymbol{E_i}$ for $i = 1, \ldots, n$. $\boldsymbol{E_i} = (E_{i,1}, \ldots, E_{i,\lambda}) = XE(\neg(x_i \oplus y_i))$, where $E_{i,j} = X_j \cdot z^{y_i} \bmod N$ or $1 \in QR_N$ randomly.
   - compute extended encryptions $\boldsymbol{P_i} = \boldsymbol{P_{i+1}} \cdot \boldsymbol{E_{i+1}} \bmod N = XE(p_i)$, where $\boldsymbol{P_n} = (1, \ldots, 1)$ 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 $\bar{\boldsymbol{X}}_{\boldsymbol{i}}$ for $i = 1, \ldots, n$. $\bar{\boldsymbol{X}}_{\boldsymbol{i}} = (\bar{X}_{i,1}, \bar{X}_{i,\lambda}) = XE(\neg x_i)$, 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 $i = 1, \ldots, n$, $\boldsymbol{T_i} = \boldsymbol{Y_i} \cdot \bar{\boldsymbol{X}}_{\boldsymbol{i}} \cdot \boldsymbol{P_i} \bmod N = XE(t_i)$.
- randomly permute $\boldsymbol{T_1}, \ldots, \boldsymbol{T_n}$ 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.

4.1 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

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

5 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 $(\mathrm{mod} N)$.
- 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 $(\bmod N^2)$. In [BK04], they encrypt 0 or 1 only, the encryption for $m \in \{0,1\}$ needs $\log N$ modular multiplications $(\mathrm{mod} N^2)$.
- Each Paillier's decryption requires $2\log N$ modular multiplications $(\bmod N^2)$.
- 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 $(\bmod N^2)$ needs 4 modular multiplication $(\mathrm{mod} N)$.

Based on the above discussion, we summarize the comparison in Table 1. In the table, the modular multiplication for the protocols in [Fis01,BK04] is $\mathrm{mod} N$ and ours is $\mathrm{mod} p$.

	computation of Alice	computation of Bob	total computation	communication
Ours	$3n\log p$	$2n\log p + 4n - 6$	$5n\log p + 4n - 6$	$6n\log p$
Ours with hash	$3n\log p$	$2n\log p + 2n$	$5n\log p + 2n$	$4n\log p$
[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 1. Comparison in computation cost and communication cost.

6 Remarks

Our construction 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.

References

[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.

[Cac99] Christian Cachin. Efficient private bidding and auctions with an oblivious third party. In *Proceedings of the 6th ACM conference on Computer and communications security - CCS '99*, pages 120–127. ACM Press, 1999.

[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.

[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.