An Efficient Solution to Generalized Yao's Millionaires Problem*

Yuan Zhang Sheng Zhong

Abstract

We study a generalized version of Yao's Millionaires problem. Namely, n millionaires want to find out the k richest persons among them without revealing their amounts of wealth. We propose an efficient solution that takes $2n$ secure comparison operations on average to find out the correct result. Our solution protects the privacy of every participant.

1 Introduction

Yao's Millionaires problem is a well-known problem in Cryptography which was first introduced by Andrew Yao in [10]. The problem discusses how two millionaires can learn which one of them is richer without revealing their amounts of wealth to each other. In this paper, we aim to solve Yao's Millionaires problem in a generalized setting: there are n millionaires and these millionaires want to find out the k richest persons among them without revealing their own amounts of wealth ($n \geq 2, 0 < k \leq n$). We call this problem the generalized Yao's Millionaires problem.

Formally, denote by $P_1, \ldots, P_n$ the n millionaires involved, and by $w_1, \ldots, w_n$ their amounts of wealth respectively. For simplicity, assume each w_i ($i = 1, \ldots, n$) is an integer. Let $N = \{1, \ldots, n\}$ be the millionaires' index set, and R_k be any subset of N such that $|R_k| = k$ and $w_i \geq w_j$ for every $i \in R_k$, $j \in N \backslash R_k$. The generalized Yao's Millionaires problem is to design a *privacy-preserving* algorithm

*Supported by NSFC-61021062 and RPGE. Sheng Zhong is the corresponding author.
Received by the editors in February 2012.
Communicated by A. Weiermann.
2010 *Mathematics Subject Classification :* 11T71, 94A60.
Key words and phrases : Yao's Millionaires Problem; Privacy-preserving; Secure Comparison.

that takes $w_1, \ldots, w_n$ as the input and outputs R_k without disclosing any information other than R_k to any party (See section 2.1 for formal definition of privacy). To the best of our knowledge, this problem has not been studied so far.

When $(n, k) = (2, 1)$, our problem reduces to the original Yao's Millionaires problem. Since Yao' Millionaires problem can be viewed as a special case of the secure multiparty computation (MPC) problem, general solutions or frameworks [11, 7, 1] which allow any function to be securely evaluated by multiple parties can be applied to solve it. However, these general solutions require heavy computational and communication overheads, therefore efficient special-purpose solutions which allow two parties to securely compare two private numbers are also proposed [5, 2, 3, 8]. Due to the same reason, we also aim to design efficient solutions to solve our problem in this paper.

In the rest of this paper, we assume all millionaires' amounts of wealth are pairwise different(i.e. $w_i \neq w_j$ holds if $i \neq j$). In cases that millionaires may have same amounts of wealth, we can make the assumption hold by assigning each millionaire P_i the "index-based wealth" $w'_i = w_i \times n + i$ and using it as the algorithm's input. It is easy to verify that the corresponding output R_k on $\{w'_i\}$ is still a valid result for the original problem.

2 Technical Preliminaries

Before we present our algorithm, we give a brief review of the formal definition of privacy and the main techniques we are using in this paper.

2.1 The definition of privacy

In this paper, we discuss the privacy issue in the *semi-honest* model [6]. In a semi-honest model, participants of an algorithm always follow the algorithm without any deviation. However, participants are curious in the sense that they may attempt to learn more about other participants' private information.

Denote by $GYM()$ a function that takes $w_1, \ldots, w_n$ as input and outputs R_k. We give the formal definition of the privacy requirement for the solution of the generalized Yao's Millionaires problem as follows.

Definition An algorithm $\mathcal{GYM}$ for the generalized Yao's Millionaires problem is ***privacy-preserving*** if there exists a probabilistic polynomial-time simulator M_i for every $i \in N$ such that for all possible $(w_1, w_2, \ldots, w_n)$,

$$\{M_i(w_i, GYM(w_1, \ldots, w_n))\}_{(w_1,\ldots,w_n)} \stackrel{c}{\equiv} \{VIEW_i(w_1, \ldots, w_n)\}_{(w_1,\ldots,w_n)},$$

where $VIEW_i(w_1, \ldots, w_n)$ denotes the view of party i when running the $\mathcal{GYM}$ on the input $(w_1, \ldots, w_n)$, and $\stackrel{c}{\equiv}$ denotes the *computational indistinguishability* [6] of two *probability ensembles* [6].

2.2 Probabilistic public-key cryptosystem that supports rerandomization and threshold decryption

In this paper, we build our solution based on a probabilistic public-key cryptosystem that supports *rerandomization* of a ciphertext, and the *threshold decryption*.

Rerandomization operations take a ciphertext of a plain message as input, and efficiently output a new ciphertext of the same message without decrypting the input ciphertext.

A cryptosystem that supports threshold decryption generally assumes that n shares of the decryption key (in a secret sharing scheme) have been distributed to n involved parties. Decryptions can only be made when at least m (m is the security parameter which can be chosen from 1 to n) parties jointly performed a threshold decryption algorithm using their parts of the private key.

One example of such a cryptosystem can be found in [4]. In [4], Damgard, et al. adapt the original decryption scheme of the Paillier cryptosystem [9] and make it support threshold decryption. The basic idea of the adaption is to hide the private key as a coefficient of a polynomial function, and to let each party hold a different evaluation of that function. With more evaluations than the order of the function, the private key can be learned using Lagrange interpolation. The adapted cryptosystem also has the additively homomorphic property for free, since the Paillier cryptosystem is additively homomorphic (Please see [9] for detailed explanations of the homomorphic property). Based on the additively homomorphic property, the cryptosystem allows to rerandomize a ciphertext by computing a random encryption of the product of the ciphertext and the constant number 1. It has been proved that the adapted cryptosystem is semantic secure just like the Paillier cryptosystem [4] and the threshold decryption is privacy-preserving in the semi-honest model.

2.3 Secure comparison algorithm

A main tool we use to construct our algorithm is a two-party secure comparison algorithm. Denote the secure comparison algorithm $\mathcal{SC}$. $\mathcal{SC}$ takes two encrypted numbers $E(w_i)$ and $E(w_j)$ as public input and reveals nothing more than the comparison result as:

$$\mathcal{SC}(E(w_i), E(w_j)) = \begin{cases} 1 & \text{if } w_i > w_j \\ 0 & \text{if } w_i < w_j \end{cases} . \tag{1}$$

In [8], a secure comparison algorithm based on homomorphic encryption is proposed[1]. The algorithm has been proved to be privacy-preserving in the semi-honest model. In particular, the algorithm allows one party A to securely compare two numbers given only the ciphertexts of these two numbers with the help of another party B. In the end, A learns nothing more than the comparison result and B learns nothing. The algorithm can be implemented using a probabilistic

[1]The algorithm in [8] assumes that the inputs are cleartexts, not ciphertexts. It is trivial to modify it so that the inputs can be ciphertexts.

public-key cryptosystem that supports the homomorphic encryption and threshold decryption, for example the adapted Paillier cryptosystem introduced in [4].

3 Our algorithms

In this section, we first present a basic solution to the generalized Yao's Millionaires problem. After that, we give another solution that is more efficient compared with the first one. Both algorithms can be implemented using a probabilistic public-key cryptosystem that supports the homomorphic encryption, rerandomization and threshold decryption, for example the adapted Paillier cryptosystem introduced in [4].

3.1 A basic solution

Intuitively, our problem can be solved by finding out the first richest, the second richest, ..., the kth richest party. Below we give a three-stage algorithm based on this idea.

Setup: A public key pk and n shares of the corresponding private key, $pr_1, \ldots, pr_n$, are generated such that each party P_i knows pk and pr_i but not any other shares of private key. (This can be achieved using either a trusted authority or an MPC technique like [4])

Stage 1: To generate a sequence of ciphertext pairs that encodes a random permutation of all parties' amounts of wealth and indices.

step 1.1: For $i = 1, \ldots, n$, P_i computes the ciphertext pair $(e_i, f_i) = \big(E(w_i), E(i)\big)$. For $i = 2, \ldots, n$, P_i sends (e_i, f_i) to party P_1.

step 1.2: For $i = 1, \ldots, n$, P_1 rerandomizes (e_i, f_i). Denote by $\{(e'_i, f'_i)\}$ the rerandomized ciphertext pairs. Then P_1 generates a random permutation π_1 on N and sends P_2 the permuted rerandomized ciphertext pairs $\{(e'_{\pi_1(1)}, f'_{\pi_1(1)}), \ldots, (e'_{\pi_1(n)}, f'_{\pi_1(n)})\}$.

step 1.3: For $i = 2, \ldots, n-1$, P_i rerandomizes and permutes the ciphertext pairs she receives and sends them to P_{i+1}. P_n rerandomizes and permutes the ciphertext pairs she receives and sends them back to P_1 in the end. After this, P_1 has a sequence of ciphertext pairs which encodes a unknown permutation of every party's original ciphertext pair. Denote this sequence by $(e'_{\pi(1)}, f'_{\pi(1)}), \ldots, (e'_{\pi(n)}, f'_{\pi(n)})$.

Stage 2: To find the ciphertext pairs which encode the first, the second, ..., the kth richest party.

step 2.1: Let $(emax, fmax) = (e'_{\pi(1)}, f'_{\pi(1)})$. For $j = 2, \ldots, n$, P_1 runs the secure comparison algorithm on $emax$ and $e'_{\pi(j)}$ with the help of other parties. and

updates $(emax, fmax)$ as:

$$(emax, fmax) = \begin{cases} (emax, fmax) & \text{if } \mathcal{SC}(emax, e'_{\pi(j)}) = 1 \\ (e'_{\pi(j)}, f'_{\pi(j)}) & \text{if } \mathcal{SC}(emax, e'_{\pi(j)}) = 0 \end{cases}. \quad (2)$$

The final $(emax, fmax)$ encodes the first richest party's amount of wealth and index.

step 2.2: P_1 removes the ciphertext pair found in the previous step from the sequence of ciphertext pairs, then repeats step 2.1 for $k-1$ times till the other $k-1$ ciphertext pairs which encode the second,. . .,the kth richest parties' amounts of wealth and indices are found. Denote by $fmax_1, \ldots, fmax_k$ the k ciphertexts which encode the k richest parties' indices.

Stage 3: To get the indices of the k richest parties.

step 3.1: For $i = 1, \ldots, k$, P_1 rerandomizes $fmax_i$. Denote by $fmax'_i$ the new ciphertext after rerandomization.

P_1 generates a random permutation θ_1 on $\{1, \ldots, k\}$ and sends P_2 the permuted rerandomized ciphertext sequence $\{fmax'_{\theta_1(1)}, \ldots, fmax'_{\theta_1(k)}\}$.

step 3.2: For $i = 2, \ldots, n-1$, P_i rerandomizes and permutes the ciphertext sequence she receives and sends it to P_{i+1} similarly. P_n rerandomizes and permutes the ciphertext sequence she receives and sends it back to P_1 in the end. After this, P_1 has a sequence of ciphertext which encodes a random permutation of R_k. Denote this sequence by $fmax'_{\theta(1)}, \ldots, fmax'_{\theta(n)}$.

step 3.3: P_1 sends the sequence $fmax'_{\theta(1)}, \ldots, fmax'_{\theta(n)}$ to all other parties.

step 3.4: $P_2, \ldots, P_n$ help P_1 to do a threshold decryption of the ciphertext sequence. After getting the index set R_k that indicates the k richest parties, P_1 sends R_k to other parties.

Efficiency Analysis: Compared with other operations in the algorithm, the secure comparison algorithm is much more complex and time-consuming. Therefore, we can use the total number of times that the secure comparison algorithm is performed to evaluate the efficiency of our solution. In this basic solution, the total number equals to $(n-1) + \ldots + (n-k)$. Therefore the algorithm requires $O(nk)$ secure comparison operations to solve the generalized Yao's Millionaires problem. When $k \approx n/2$, the total number is quadratic in n. This would make the algorithm relatively slow especially when n is a big number.

3.2 An improved solution

To improve the performance, we propose a solution that finds the k richest parties with fewer secure comparison operations. This solution also has three stages. The only difference between the previous solution and this one is in their second stages. To save the space, we only present the second stage below.

Stage 2: *To* find the ciphertext pairs which encode the first, the second, . . . , the kth richest party.

step 2.0: Let $S_{candidate}$ be the set that contains all n rerandomized and permuted ciphertext pairs received by P_1 from stage 1, S_{output} be an empty set.

step 2.1: P_1 randomly picks a pair $(e'^{\star}, f'^{\star})$ from $S_{candidate}$ and runs the secure comparison algorithm to compare it with all other pairs in $S_{candidate}$ with the help of other parties.

Based on the comparison results, $S_{candidate}$ is divided into two parts. Let $S_{greater}$ (S_{less} resp.) be the set that contains all pairs which have greater (less resp.) wealth compared with the randomly picked pair.

If $|S_{greater}| + |S_{output}| \leq k-1$, P_1 updates S_{output} to $S_{output} \cup S_{greater} \cup \{e'^{\star}, f'^{\star}\}$, $S_{candidate}$ to S_{less}.

If $|S_{greater}| + |S_{output}| = k$, P_1 updates S_{output} to $S_{output} \cup S_{greater}$, $S_{candidate}$ to $S_{less} \cup \{e'^{\star}, f'^{\star}\}$.

If $|S_{greater}| + |S_{output}| > k$, P_1 updates $S_{candidate}$ to $S_{greater}$.

step 2.2: P_1 repeats step 2.1 till $|S_{output}| = k$. The final set S_{output} contains k ciphertext pairs which encode the wealth and indices of the k richest parties. Denote by $fmax_1, \ldots, fmax_k$ the k ciphertexts which encode these parties' indices.

Efficiency Analysis: Since the improved algorithm is a probabilistic algorithm, its efficiency is not deterministic. The algorithm stops doing secure comparison until the "pivot" ciphertext pair (the ciphertext pair of the kth richest or the $(k+1)$th richest party) is picked. In the best case, the pivot is picked in the first round which results in $n-1$ secure comparison operations. In the worst case, the pivot is picked after all other $n-2$ pairs have been picked which results in $n-1+n-2+\ldots+2 = (n^2-n-2)/2$ secure comparison operations. In the average case, every picked pair cuts approximately half of the $S_{candidate}$. This results in $n + n/2 + n/4 + \ldots \approx 2n$ secure comparison operations at most to locate the pivot pair.

4 Algorithm Analysis

In this section, we give analysis of the correctness and privacy guarantee of our algorithms.

Proposition 4.1. (Correctness) *Both algorithms output the correct R_k.*

Proof. The first algorithm performs secure comparison operations to find the first richest, second richest,..., kth richest party's pair one by one. It is straightforward to see the final output is correct.

We prove the correctness of the second algorithm in two steps.

Firstly, we prove the second algorithm will always stop and output an S_{output} of k ciphertext pairs. It is easy to see $|S_{output}|$ is a non-decreasing integer and always no greater than k. Also, $|S_{output}|$ cannot remain unchanged at a value that is smaller than k. Otherwise, $|S_{greater}| + |S_{output}|$ has to be always greater than k. However, this is impossible because $S_{greater}$ would keep decreasing till it goes to zero in this case, and causes $|S_{greater}| + |S_{output}| < 0 + k$. Therefore, the algorithm always stops. When it stops, $|S_{output}| = k$.

Secondly, we prove every ciphertext pair in S_{output} encodes one of the k richest party. This can be done by assuming S_{output} has an incorrectly inserted ciphertext pair which encodes a party who is not one of the k richest party. It directly follows that at least one party who is one of the k richest party is not in the final S_{output}, denote this party by $P_{correct}$. Denote by P_{false} the first falsely inserted party. Consider the round in which P_{false}'s ciphertext pair is inserted into S_{output}, $P_{correct}$'s ciphertext pair must have been deleted from some previous round r. In round r, $|S_{greater}| + |S_{output}| > k$ because only in this case a ciphertext pair could be excluded with all other ciphertext pairs in S_{less}. Therefore, every ciphertext pair in $S_{greater}$ should encode a party that has greater amount of wealth compared with $P_{correct}$ and thus encodes a valid k-richest party. Also, every ciphertext pair in S_{output} should also encode a valid k-richest party because the first falsely inserted party has not been inserted yet. It directly follows that $S_{greater} \cup S_{output}$ is a subset of the correct R_k i.e. $|S_{greater}| + |S_{output}| \leq k$ which is contradictive to the premise of $|S_{greater}| + |S_{output}| > k$. ∎

Privacy Guarantee: Our algorithms are privacy-preserving in the semi-honest model.

Theorem 4.2. (Privacy) *The basic solution is privacy-preserving.*

Proof. We construct M_1 as follows. On input (w_1, R_k), M_1 simulates the coin flips of P_1 as described in the algorithm. Then, M_1 simulates the received ciphertext pair from P_i ($i \neq 1$) in step 1.1 using a random encryption of a random cleartext and a random encryption of i, and simulates the sequence of permuted ciphertext pairs received from P_n in step 1.3 using $2n$ random encryptions of random cleartexts. M_1 performs cleartext comparison operations and uses the results to simulate the results of secure comparison results in P_1's view in step 2.1 and 2.2. In particular, M_1 generates $\{v_i\}$ which is a random permutation of N. For every secure comparison between $e\prime_{\pi(i)}$ and $e\prime_{\pi(j)}$, M_1 simulates the comparison result with 1 if $v_i > v_j$, otherwise M_1 simulates the comparison result with 0. In each round of the secure comparison algorithm, M_1 simulates the numbers and ciphertexts received by P_1 using the same amount of random numbers and random encryptions of random cleartexts. M_1 simulates $fmax_1, \ldots, fmax_k$ in step 2.2 and $fmax'_{\theta(1)}, \ldots, fmax'_{\theta(n)}$ in step 3.2 using random encryptions of R_k. Finally, M_1 simulates the numbers or ciphertexts received by P_1 during the threshold decryption in step 3.4 using the same amount of random numbers or random encryptions of random cleartexts.

For $i \neq 1$, we construct M_i as follows. On input (w_i, R_k), M_i simulates the coin flips of P_i as described in the algorithm. Then, M_i simulates the sequence of permuted ciphertext pairs received from P_{i-1} in step 1.2 or 1.3 using $2n$ random encryptions of random cleartexts. In step 2.1, P_i needs to help P_1 to perform the secure comparison algorithm several times. M_i simulates the numbers and ciphertexts received by P_i using the same amount of random numbers and random encryptions of random cleartexts. To simulate the ciphertext sequence received from P_{i-1} in step 3.2 and from P_1 in step 3.3, M_i uses two sequences of random encryptions on all elements in R_k. In step 3.4, P_i helps P_1 to decrypt the encryption of R_k, M_i simulates the numbers and ciphertexts received by P_i using the same amount of random numbers and random encryptions of random cleartexts.

The computational indistinguishability follows the semantic security of the cryptosystem, the randomness of the permutation operations, and the security of the secure comparison algorithm and the threshold decryption algorithm. ■

Theorem 4.3. (Privacy) *The improved solution is privacy-preserving.*

Proof. Since the improved solution differs from the basic solution only in stage 2, we can construct simulators that run exactly the same as the simulators we have constructed above in stage 1 and 3.

M_1 runs as follows in stage 2. M_1 performs cleartext comparison operations and uses the results to simulate the results of secure comparison results in P_1's view in step 2.1 and 2.2. In particular, M_1 generates a sequence $\{1, \ldots, n\}$ and repeats the similar procedure of picking a random number and using it to compare with other numbers as P_1 does in step 2.1 until the k greatest numbers are found. In each round of the secure comparison algorithm, M_1 simulates the numbers and ciphertexts received by P_1 using the same amount of random numbers and random encryptions of random cleartexts.

In stage 2, M_i ($i \neq 1$) runs as follows. In step 2.1, P_i needs to help P_1 to run the secure comparison algorithm several times. M_i simulates these secure comparison algorithms by generating a sequence $\{1, \ldots, n\}$ and repeating the similar procedure of picking a random number and using it to compare with other numbers as P_1 does in step 2.1 until the k greatest numbers are found. In each round of the secure comparison algorithm, M_i simulates the numbers and ciphertexts received by P_i using the same amount of random numbers and random encryptions of random cleartexts.

The computational indistinguishability follows the semantic security of the cryptosystem, the randomness of the permutation operations, and the security of the secure comparison algorithm and the threshold decryption algorithm. ■

References

[1] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513. ACM, 1990.

[2] I. Blake and V. Kolesnikov. Conditional encrypted mapping and comparing encrypted numbers. *Financial Cryptography and Data Security*, pages 206–220, 2006.

[3] I. Damgard, M. Geisler, and M. Kroigard. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography*, 1(1):22–31, 2008.

[4] Ivan Damgard and Mads Jurik. Efficient protocols based on probabilistic encryption using composite degree residue classes, March 17 2001.

[5] M. Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. *Topics in CryptologyCT-RSA 2001*, pages 457–471, 2001.

[6] O. Goldreich. *Foundations of cryptography: Basic applications*, volume 2. Cambridge Univ Pr, 2004.

[7] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[8] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. *Public Key Cryptography–PKC 2007*, pages 343–360, 2007.

[9] Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1999.

[10] A.C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.

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

Computer Science and Engineering Department,
State University of New York at Buffalo,
Buffalo, 14260, USA;
Email:yuanzhan@buffalo.edu

State Key Laboratory for Novel Software Technology, Nanjing University;
Department of Computer Science and Technology, Nanjing University,
Nanjing 210046, P. R. CHINA;
Email: sheng.zhong@gmail.com