CERIAS Tech Report 2001-09

A Multi-dimensional Yao's Millionaire Protocol

Mikhail Atallah, Wenliang Du
Center for Education and Research in
Information Assurance and Security
&
Department of Computer Sciences, Purdue University
West Lafayette, IN 47907

A Multi-dimensional Yao's Millionaire Protocol [*]

Mikhail J. Atallah and Wenliang Du
CERIAS and Department of Computer Sciences, Purdue University,
1315 Recitation Building, West Lafayette, IN 47907-1315

Abstract

Yao introduced the "millionaire problem", in which two parties want to determine who is richer without disclosing anything else about their wealth. This problem deals with single comparison situation; however, in many applications, one often encounters situations where one wants to make multiple comparisons in an "all-or-nothing" fashion: Alice has an n-dimensional vector $A = (a_1, \ldots, a_n)$, and Bob has another n-dimensional vector $B = (b_1, \ldots, b_n)$. Alice wants to know whether A dominates B, i.e. whether *for all* $i = 1, \ldots, n$, $a_i > b_i$. If $\exists i$ such that $a_i < b_i$, then both parties should learn nothing about the other party's information, including any partial information, such as the relationship between any a_i, b_i pair, for $i = 1, \ldots, n$. This problem cannot be solved by just using the solution to Yao's Millionaire Problem n times, once for each dimension: That would inappropriately reveal the relative ordering of individual a_i, b_i pairs in the case where the answer to the domination question is "no". We propose a novel and efficient solution to this multi-dimensional Yao's Millionaire Problem. The communication complexity of our scheme is linear in the number of bits needed to represent Alice's and bob's vectors.

1 Introduction

About two decades ago, Yao introduced the "millionaire problem" [12]: Two parties want to determine who is richer without disclosing anything else about their wealth. Several solutions have been proposed in the past to solve this problem, however, none of them is efficient, except the recent one proposed by Cachin, who has gave an elegant practical solution to this problem [1].

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

We consider the above multiple comparison problem as an extension of Yao's Millionaire Problem (a one-dimensional problem) to a multi-dimensional problem. If we consider A and B as vectors, the problem is to actually decide whether A dominates B. Therefore, in this paper, we call this problem the *private vector dominance* problem, or the *dominance* problem in short. We will use $A \succ B$ to denote that A dominates B.

[*]Portions of this work were supported by Grant EIA-9903545 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security.

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

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

The goal of this paper is to solve the above private vector dominance problem. Our solution uses Cachin's solution to Yao's Millionaire Problem as a subroutine, but it does so in a very non-obvious way, and the overall structure of our solution is novel and very different from Cachin's.

As Cachin [1] points out, the early cryptographic solutions such as Yao's millionaire problem [12] have communication complexity that is exponential in the number of bits of the numbers involved, and the later solutions using general secure multi-party computation techniques also have problems with achieving privacy efficiently. Their advantage, of course, is that they do so without using an untrusted third party. As in Cachin's work [1], here we assume an untrusted third party that can misbehave on its own (for the purpose of illegally obtaining information about Alice's or Bob's private vectors) but does not collude with Alice against Bob or vice-versa. Because we are using Cachin's scheme as a subroutine, we are also implicitly making use of the number-theoretic assumptions made in Cachin's paper [1] (such as the Φ-hiding assumption that was also used in the private information retrieval literature [7]). The communication complexity of our scheme is linear in the number of bits needed to represent Alice's and Bob's vectors.

In the rest of this section, we describe an overview of our solution followed by the review of the related work. Section 2 formally defines the private vector dominance problem, and then explains the details of the solution. In section 3, we give four particular applications of this new problem. We then conclude in section 4.

1.1 Overview of the Protocol

We give a very short and oversimplified description of private vector dominance protocol here; the complete protocol can be found in Section 2. Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ be Alice's and Bob's inputs, respectively. Because having Alice and Bob compare their vectors directly would inappropriately disclose the ordering information between individual a_i, b_i pairs, instead we ask the oblivious third party, Ursula, to compare numbers derived from Alice's and Bob's numbers in such a way that the comparison reveals no information to Ursula and yet Ursula can perform certain updates that indirectly encapsulate (for A and B) the outcome of the vector-dominance comparison. The idea is not to let Ursula know anything about A, B, or the outcome of the comparison between any a_i and b_i with which it is helping. This implies that Ursula does not find out such statistical aggregate results as how many of the a_i's are larger than the corresponding b_i's, etc.

This objective is achieved by sending Alice's disguised data to Ursula, such that Ursula does not know the actual value of any particular a_i, nor does she know which disguised entry corresponds to a_i. As we know, using protocols for Yao's Millionaire Problem [12, 1] enables two parties to compare their private inputs without disclosing the private inputs to the other party; therefore, after getting Alice's input vector, Ursula and Bob use Cachin's protocol [1] to compare the disguised elements of Alice's and Bob's input vectors one by one; we actually use a slightly modified version of Cachin's protocol, one where Ursula knows the comparison's outcome while the other party learns nothing. (*Note:* Our protocol can easily be modified so it is symmetric in the roles of Alice and Bob, as will be explained later — the symmetric version of it is just as efficient but is notationally more cumbersome and so we postpone discussing it for the sake of a simpler exposition.) Because Ursula does not know which disguised entry corresponds to a particular a_i, she knows nothing about the relationship between a_i and b_i, nor does she know the *dominance* relationship between A and B (whether the entries of one are all larger than the corresponding entries of the other). Moreover, Alice's inputs are disguised in such a way that, to Ursula, half of Alice's disguised inputs are bigger than Bob's corresponding inputs, while another half of Alice's disguised inputs are less than Bob's corresponding inputs, regardless of what the actual relationship between Alice's inputs and Bob's inputs is. Therefore Ursula gains no information about the statistical aggregate results as how many a_i's are larger than the corresponding b_i's.

So Ursula will know the results of the comparisons between disguised items of Alice's and Bob's, but she cannot give these results to Alice and Bob because they can make sense of them to inappropriately glean information: Instead, she has to let Alice and Bob know whether A dominates B, or B dominates A, or neither dominates the other, without Ursula herself knowing anything about the dominance relationship (or lack thereof). This is at the heart of the result of this paper, and is achieved by having Alice and Bob each generate a set of pairs of nonces ($=$ random numbers), after which Alice and Bob each share a different set of their own secret nonces with Ursula. Ursula then converts the comparisons' outcomes to a single number formed by the *xor* of a selected set of nonces from the two sets of nonces. It is this number that is finally sent to Alice and Bob by Ursula. The idea is to judiciously construct this number in such a way that Alice and Bob can only verify the dominance relationship between A and B, and nothing else (while Ursula learns nothing).

The above was necessarily an oversimplification of the main ideas of our protocol, and only a look at the details will reveal the subtle intricacies involved. But first, a survey of related work is given next.

1.2 Related Work

The problem we are trying to solve in this paper is actually a generalization of Yao's Millionaire Problem [12]. Yao's Millionaire Problem is to compare two private scalar numbers, in other words, it is a one dimensional version of our problem; we generalize this problem to higher dimensions, i.e., to compare two vectors. Cachin [1] gives an efficient solution to Yao's Millionaire Problem; before we came up with our solution, we fruitlessly tried directly extending Cachin's protocol to solve this problem. Naive extensions result in a communication complexity that is exponential in the number of dimensions n, but whether there is another efficient way to extend Cachin's protocol is still unknown.

The above problems are all special cases of the general secure multi-party computation problem. Generally speaking, a secure multi-party computation problem deals with computing any probabilistic function on any input, in a distributed network where each participant holds one of the inputs, ensuring independence of the inputs, correctness of the computation, and that no more information is revealed to a participant in the computation than can be computed from that participant's input and output [3]. Other examples of secure multi-party computations include: elections over the Internet, joint signatures, joint decryption, and many others [11]. The history of the multi-party computation problem is extensive since it was introduced by Yao [12] and extended by Goldreich, Micali, and Wigderson [8], and by many others.

As Goldreich states in [2] that the general secure multi-party computation problem is solvable in theory, therefore the Private Dominance Problem can also be solvable in theory; However, using the solutions derived by these general results for special cases of multi-party computation, are impractical; special solutions should be developed for this particular type of secure multi-party computation problem for efficiency reasons.

2 Problem Statement and Solution

2.1 Problem

Definition 2.1. *(Vector Dominance)* Let $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_n)$; if for all $i = 1, \dots, n$ we have $a_i > b_i$, then we say that A dominates B and denote it by $A \succ B$.

Definition 2.2. *(Two-Party Private Vector Dominance Protocol)* A two-party private vector dominance protocol is one in which the two parties determine whether one party's secret vector dominates another party's secret vector, with the help of an oblivious third party who, while it does not collude with either one of the two parties against the other, could nevertheless try to illegally acquire information about their secret data. We also assume that both of the two parties are honest-but-curious, i.e. they will follow the protocol, but like Ursula, they could nevertheless try to illegally acquire information about the other party's secret data. If we call the two parties Alice and Bob, and their secret vectors $A = (a_1, \dots, a_n)$ and (respectively) $B = (b_1, \dots, b_n)$, and we call the third party Ursula, then at the end of the protocol, the following properties must hold:

1. Alice and Bob have determined whether $A \succ B$, $B \succ A$, or neither A nor B dominates the other. However, Ursula has not made such a determination.
2. No party (including the third party) has gained any knowledge about A and B other than the one implied by the previous item. Note that this implies the following:
   (a) In the case where neither $A \succ B$ nor $B \succ A$, neither Alice nor Bob knows the relative ordering of any individual a_i, b_i pair (i.e., whether $a_i < b_i$ or not).
   (b) Ursula knows nothing about A, B, or the relationships between any elements of A, B.

2.2 Protocol

We continue to use the notation $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_n)$ to represent Alice's and (respectively) Bob's secret vectors. We also introduce an *oblivious* third party, Ursula, who will not learn any information about A, B, or the relationship between them, including partial ordering and dominance information.

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

Outline

The protocol consists of the following stages:

- Alice disguises her vector A and gets A', sends A' to Ursula. (Because of the disguise, Ursula learns nothing about Alice's secret A). Bob disguises B and gets B' but he does *not* send B' to Ursula. A' and B' each have a length m that is larger than n.
  Note: The protocol can easily be modified so it is symmetric in the roles of Alice and Bob, as will be explained later. The reason we present the details of this version (rather than the symmetric one) is that it results in a considerably less cluttered exposition both conceptually and (especially) notationally.
- Ursula and Bob use a modified version of Cachin's protocol to compare each entry of Ursula's A' to the corresponding entry of Bob's B'; in this modified version of Cachin's protocol, only Ursula know the outcome of a comparison as opposed to both knowing it in the original version. Because of this protocol, Ursula learns nothing about B', and Bob learns nothing about A' or the comparison results.

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

Step 1: Setup

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

Step 2: Inputs Disguise

In this step, Alice constructs $A' = (2a_1 + R_1, \ldots, 2a_n + R_n, (2a_1 + 1) + R_{n+1}, \ldots, (2a_n + 1) + R_{2n}, -2a_1 + R_{2n+1}, \ldots, -2a_n + R_{3n}, -(2a_1 + 1) + R_{3n+1}, \ldots, -(2a_n + 1) + R_{4n})$. Bob constructs $B' = ((2b_1+1)+R_1, \ldots, (2b_n+1)+R_n, 2b_1+R_{n+1}, \ldots, 2b_n+R_{2n}, -(2b_1+1)+R_{2n+1}, \ldots, -(2b_n+1)+R_{3n}, -2b_1+R_{3n+1}, \ldots, -2b_n+R_{4n})$. Alice and Bob then agree upon a random permutation π of $\{1, 2, \ldots, 4n\}$, and they use π to reorder the entries of A', and the entries of B'. We will next explain the rationale for constructing A' and B' in this way.

Because $a_i > b_i$ if and only if $a_i + R_i > b_i + R_i$, one might as well compare $a_i + R_i$ with $b_i + R_i$ instead of comparing a_i with b_i. Since Alice is going to send her disguised inputs to Ursula, she has to encrypt or disguise her data; adding R_i to a_i effectively hides it from Ursula. If that is the only form of disguise that was done, then Alice's A' would be $(a_1 + R_1, \ldots, a_n + R_n)$ and Bob's B' would be $(b_1 + R_1, \ldots, b_n + R_n)$. Now, although Alice and Bob could rearrange the order of the entries of such an A' and (respectively) B', Alice still cannot send this A' to Ursula and ask Ursula to run a protocol that compares its entries to those of Bob's B', because Ursula would then know for how many indices i the comparison $a_i > b_i$ was true. This is not acceptable. Furthermore, if the mean value of these random numbers is known to Ursula, Ursula can gain statistical information about a_i's. The above drawbacks are addressed by the inclusion in each of A' and B' of the n entries of the form $-a_i + R_{n+i}$ and (respectively) $-b_i + R_{n+i}$. Based on the fact that $a_i > b_i$ and $-a_i > -b_i$ cannot be true or false at the same time (if $a_i \neq b_i$), the additional n entries (involving the $-a_i$ or $-b_i$) are meant to "blind" Ursula from learning how many times $a_i > b_i$, as well as the statistical information about a_i's (the sum of a_i's and $-a_i$'s for $i = 1, ..., n$ is zero). Now, if that is the only form of disguise that was done, Alice's A' would be $(a_1 + R_1, \ldots, a_n + R_n, -a_1 + R_{n+1}, \ldots, -a_n + R_{2n})$, and Bob's B' would now be $(b_1 + R_1, \ldots, b_n + R_n, -b_1 + R_{n+1}, \ldots, -b_n + R_{2n})$. But Alice still cannot send such an A' for Ursula to use in a protocol comparing its entries to those of Bob's B', and that is because of the *equality* case, i.e., for some i having $a_i = b_i$ (and hence $a'_i = b'_i$). For example, if $a_i = b_i$ for all $i = 1, \ldots, n$, then both $a'_i > b'_i$ and $-a'_i > -b'_i$ are false; therefore if Ursula got $2n$ *false*'s, she would know that Alice and Bob have the exactly same vector. Similarly, if $a_i \neq b_i$ for all $i = 1, \ldots, n$, Ursula gets n *true* values and n *false* values; if $a_i = b_i$ for only one i, Ursula gets $n - 1$ *true* values and $n + 1$ *false* values, and so on. The above shows how Ursula would derive the number of equality cases, which is not acceptable.

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

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

We transform each a_i to $2a_i$ and $2a_i + 1$; correspondingly, we transform each b_i to $2b_i + 1$ and $2b_i$. The transformation has the following good properties: if $a_i = b_i$, then $2a_i < 2b_i + 1$ and $2a_i + 1 > 2b_i$; therefore, there will not be any equality case. However, this transformation does not affect either ">" case or "<" case: for example, if $a_i > b_i$, then both $2a_i > 2b_i + 1$ and $2a_i + 1 > 2b_i$ still hold because a_i and b_i are integers.

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

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

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

Theorem 1. *The comparison of A' and B' always generates $2n$ true values and $2n$ false values.*

Proof. Consider the possible cases:

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

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

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

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

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

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

Step 3: Nonce pairs preparation

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

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

Alice computes $\alpha_+ = q_1 \oplus \cdots \oplus q_{2n} \oplus q'_{2n+1} \oplus \cdots \oplus q'_{4n}$, and $\alpha_- = q'_1 \oplus \cdots \oplus q'_{2n} \oplus q_{2n+1} \oplus \cdots \oplus q_{4n}$. Alice then sends, as a commitment to α_+ and α_-, a one-way hash of each of them to Bob. Intuitively, α_+ represents the fact that A' dominates B', and α_- represents the fact that B' dominates A'.

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

Step 4: Evaluation

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

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

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

Step 5: Result Extraction

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

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

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

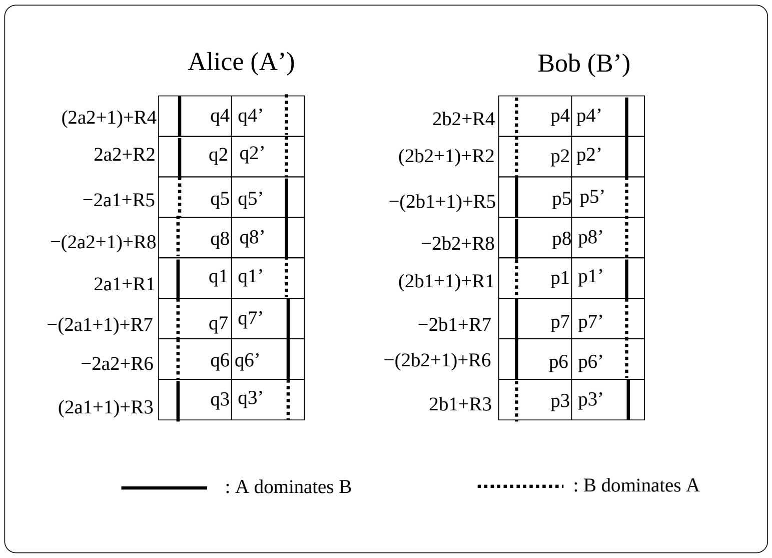

Figure 1: Example

Figure 1 is an example with $n = 2$ and $\pi(1,2,3,4,5,6,7,8) = (5,2,8,1,3,7,6,4)$. The nonces marked by the solid lines are those that will be selected by Ursula if $A \succ B$; h_1 is constructed by xor'ing all of these and only these solid-line nonces. The nonces marked by the dotted lines are those that will be selected by Ursula if $B \succ A$; h_2 is constructed by xor'ing all of these and only these dotted-line nonces. If neither A dominates B nor B dominates A, then Ursula will end up choosing some of the nonces marked by the solid lines and some of the nonces marked by the dotted lines, which causes $h \neq h_1$ and $h \neq h_2$.

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

Theorem 2. *The above protocol correctly determines whether (and which) one of A, B dominates the other.*

Proof. It clearly suffices to prove the following three sub-claims.
Sub-claim 1: If there exists $a_i = b_i$, then with overwhelming probability $h \neq \alpha_+ \oplus \beta_-$, and $h \neq \alpha_- \oplus \beta_+$.
Sub-claim 2: If for all $i = 1, \ldots, n$ we have $a_i \neq b_i$ then with overwhelming probability the case of $A \succ B$ is detected by checking whether $h = \alpha_+ \oplus \beta_-$.
Sub-claim 3: If for all $i = 1, \ldots, n$ we have $a_i \neq b_i$ then with overwhelming probability the case of $B \succ A$ is detected by checking whether $h = \alpha_- \oplus \beta_+$.
Proof of Sub-claim 1:

If there exists $a_i = b_i$, we will have $2a_i + R_i < (2b_i + 1) + R_i$ and $(2a_i + 1) + R_{n+i} > 2b_i + R_{n+i}$. Therefore, corresponding to these two comparisons, Ursula will select q_i', p_i, q_{n+i} and p_{n+i}' to compute h. However, the corresponding selection by α_+'s for these two comparisons is q_i and q_{n+i}; the corresponding choice by β_-'s for these two comparisons is p_i' and p_{n+i}'. Because with overwhelming probability $q_i' \oplus q_{n+i} \oplus p_i \oplus p_{n+i}' \neq q_i \oplus q_{n+i} \oplus p_i' \oplus p_{n+i}'$, we have $h \neq \alpha_+ \oplus \beta_-$. Although it is theoretically possible for equality to "coincidentally" occur as a chance occurrence of the xor'ing of the wrong set of nonces; choosing each nonce to be large enough easily decreases the probability of such an occurrence to almost zero (more on this later).

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

Proof of Sub-claim 2: First, we have $\alpha_+ = q_1 \oplus \cdots \oplus q_{2n} \oplus q_{2n+1}' \oplus \cdots \oplus q_{4n}'$, and $\beta_- = p_1' \oplus \cdots \oplus p_{2n}' \oplus p_{2n+1} \oplus \cdots \oplus p_{4n}$. According to the protocol, if $A \succ B$, we have $h = q_1 \oplus p_1' \oplus \cdots \oplus q_{2n} \oplus p_{2n}' \oplus q_{2n+1}' \oplus p_{2n+1} \oplus \cdots \oplus q_{4n}' \oplus p_{4n}$, therefore $h = \alpha_+ \oplus \beta_-$.

Now let us consider the other direction. Suppose $h = \alpha_+ \oplus \beta_-$, therefore $h = q_1 \oplus p_1' \oplus \cdots \oplus q_{2n} \oplus p_{2n}' \oplus q_{2n+1}' \oplus p_{2n+1} \oplus \cdots \oplus q_{4n}' \oplus p_{4n}$. Because h is constructed from $q_1, \ldots, q_{4n}$, $q_1', \ldots, q_{4n}'$, $p_1, \ldots, p_{4n}$, and $p_1', \ldots, p_{4n}'$, with overwhelming probability, h must be constructed exactly by (q_1, p_1'), $\ldots$ (q_{2n}, p_{2n}'), (q_{2n+1}', p_{2n+1}), $\ldots$ (q_{4n}', p_{4n}), which means $a_i > b_i$ and $-a_i < -b_i$ for $i = 1, \ldots, n$, indicating that A dominates B. As in Sub-claim 1, here too it is theoretically possible to fail because of a chance occurrence of the xor'ing of the wrong set of nonces: The next sub-section shows how easily the probability of this can be brought to almost zero by using suitably long nonces.
Proof of Sub-claim 3: Similar to the proof of Sub-claim 2.

□

2.3 Making the protocol symmetric

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

2.4 Communication Complexity Analysis

Reducing the communication complexity is a major concern in many secure multi-party computation protocols [5, 6, 7, 4], so it is important to discuss the communication complexity of our protocol.

As we pointed out in the proof of the Claim of the previous sub-section, the length of each nonce should be large enough to reduce the probability of coincidental equality to close to zero. We will next analyze how many bits long each nonce should be.

In the following, we will assume that the length of each nonce is L bits, and that the acceptable probability of the above-mentioned "coincidental failure" is of the form $\frac{1}{2^t}$. The possibility that a xor'ing of the wrong set of nonces "coincidentally" gives the right value (like h_1 or h_2) is $\frac{1}{2^L}$. Therefore, we need to have $\frac{1}{2^L} < \frac{1}{2^t}$, i.e., $L > t$. This indicates that L is a constant independent to the size of the inputs.
Minor implementation note: Although Alice and Bob can send directly the $8n$ nonce pairs to Ursula without violating our communication complexity claims, in practice it may suffice for each of them to choose a random seed and generate their nonce pairs using their seed. They only need to send their seeds to Ursula, who can generate the same nonce pairs as Alice's and Bob's.

According to [1], the communication complexity of Cachin's protocol is $O(\ell)$, where ℓ is the number of bits of each input number. Therefore, if all input numbers are in $[0, 2^\ell]$, the communication cost for using Cachin's protocol $O(n)$ times is $O(\ell n)$.

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

3 Applications

In many applications, one encounters situations that are of an "all-or-nothing" nature. A common category is like this: Alice wants to know if she satisfies a set of Bob's requirements; however, she is not supposed to know either how many requirements she satisfies or which requirements she satisfies, much less the requirements themselves. All she can know is *yes* or *no* answer.

In this section, we will describe some specific examples and how the multi-dimensional Yao's millionaire protocol proposed in this paper could be used in solving these problems.

Multi-commodity Private Bidding and Auctions

Bidding often involves multiple items in an "all-or-nothing" fashion: You are not interested in getting the rental car in isolation of the airline fare, cruise line, hotel room, etc. In business-to-business bidding, a manufacturer may want to deal with a single supplier that can simultaneously satisfy all of its n requirements (either because there is some coordination required in the production of the n items types, or simply to avoid the bureaucratic overhead of having to deal with multiple suppliers).

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

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

Privacy-Preserving Geometric Computations

Point-Location problem [10] is the problem of deciding if a point is in a range, usually represented by a polygon. Point-Location problem is a very common problem in geometric computations, and it has a lot of applications.

Now consider this problem: Bob has a secret polygon, and Alice has a secret point; Alice wants to know if this point is inside Bob's polygon or outside the polygon. Neither Alice nor Bob is willing to disclose any partial information about their secrets to the other party. Therefore, Alice should know only a *yes* or *no* answer; if the answer is *no*, Bob and Alice should learn nothing else.

The above problem has a potential real application: for example, country A decides to bomb a location x in some other country, so it informs its alliance countries, who are afraid that the location is within their areas of interest: for example, those countries might have secret businesses, secret military bases, or secret agents in that area. Obviously, A does not want to disclose the location information to any country if the location is not within that country's areas of interest, nor does any country want to disclose its areas of interest to the country A. How could A and its friend countries figure out whether x is within those locations? and in the case that x is not within those locations, no information should be disclosed?

By combining the geometric computation knowledge [10] and the method of secure evaluation of polynomial function [9], this problem can be reduced to the dominance problem, i.e. after the transformation, Alice gets a n-dimensional vector $A = (a_1, \ldots, a_n)$, and Bob gets a n-dimensional vector $B = (b_1, \ldots, b_n)$; the problem of knowing whether Alice's point is inside Bob's polygon is equivalent to knowing whether A dominates B. Because of space limits, we omit the reduction steps.

Privacy-Preserving Negotiation

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

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

The problem can be easily transformed to a dominance problem: to decide whether $(p_1, -p_1, \ldots, p_n, -p_n)$ dominates $(x_1, -y_1, \ldots, x_n, -y_n)$. Our multi-dimensional Yao's Millionaire protocol will solve the problem.

Ancestor-Descendent Relationship in a Tree

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

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

4 Conclusion

In this paper, we have defined a problem that generalizes the well known one-dimensional Yao's Millionaire problem to the multi-dimensional Yao's Millionaire problem. We have proposed a solution to this problem that is secure in the sense that both party either know that one dominates the other or know nothing about the other party's input; our solution is also efficient in the sense that its communication complexity is linear in the size of (= numbers of bits representation) the inputs. An interesting open problem is whether the same communication complexity can be achieved without the use of the oblivious (and untrusted) third party. This would represent an improvement on Yao's original solution [12], something which has eluded researchers so far even in the one-dimensional case (at least without the introduction of an untrusted third party).

References

[1] C. Cachin. Efficient private bidding and auctions with an oblivious third party. In *Proceedings of the 6th ACM conference on Computer and communications security*, pages 120–127, Singapore, November 1-4 1999.

[2] O. Goldreich. Secure multi-party computation (working draft). Available from http://www.wisdom.weizmann.ac.il/home/oded/public_html/foc.html, 1998.

[3] S. Goldwasser. Multi-party computations: Past and present. In *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, Santa Barbara, CA USA, August 21-24 1997.

[4] Y. Gertner, S. Goldwasser and T. Malkin. A random server model for private information retrieval. In *2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '98)*, 1998.

[5] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan. Private information retrieval. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, Milwaukee, WI USA, October 23-25 1995.

[6] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings of the 38th annual IEEE computer society conference on Foundation of Computer Science*, Miami Beach, Florida USA, October 20-22 1997.

[7] C. Cachin, S. Micali and M. Stadler. Computationally private information retrieval with polylogarithmic communication. *Advances in Cryptology: EUROCRYPT '99, Lecture Notes in Computer Science*, 1592:402–414, 1999.

[8] O. Goldreich, S. Micali and A. Wigderson. How to play any mental game. In *Proceedings of the 19th annual ACM symposium on Theory of computing*, pages 218–229, 1987.

[9] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation (extended abstract). In *Proceedings of the 31th ACM Symposium on Theory of Computing*, pages 245–254, Atanta, GA, USA, May 1-4 1999.

[10] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[11] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., 1996.

[12] A. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, 1982.