Secure Multi-Party Computational Geometry

Mikhail J. Atallah and Wenliang Du

Department of Computer Sciences and
Center for Education and Research in Information Assurance and Security
Purdue University
West Lafayette, IN 47907
{mja, duw}@cs.purdue.edu

Abstract. The general secure multi-party computation problem is when multiple parties (say, Alice and Bob) each have private data (respectively, a and b) and seek to compute some function $f(a, b)$ without revealing to each other anything unintended (i.e., anything other than what can be inferred from knowing $f(a, b)$). It is well known that, in theory, the general secure multi-party computation problem is solvable using circuit evaluation protocols. While this approach is appealing in its generality, the communication complexity of the resulting protocols depend on the size of the circuit that expresses the functionality to be computed. As Goldreich has recently pointed out [6], using the solutions derived from these general results to solve specific problems can be impractical; problem-specific solutions should be developed, for efficiency reasons. This paper is a first step in this direction for the area of computational geometry. We give simple solutions to some specific geometric problems, and in doing so we develop some building blocks that we believe will be useful in the solution of other geometric and combinatorial problems as well.

1 Introduction

The growth of the Internet opens up tremendous opportunities for cooperative computation, where the answer depends on the private inputs of separate entities. These computations could even occur between mutually untrusting entities. The problem is trivial if the context allows the conduct of these computations by a trusted entity that would know the inputs from all the participants; however if the context disallows this then the techniques of secure multi-party computation become very relevant and can provide useful solutions.

In this paper we investigate how various computational geometry problems could be solved in a cooperative environment, where two parties needs to solve a geometric problem based on their joint data, but neither wants to disclose its private data to the other party. Some of the problems we solve in this framework are:

Problem 1. (Point-Inclusion) Alice has a point z, and Bob has a polygon P. They what to determine whether z is inside P, without revealing to each other

any more than what can be inferred from that answer. In particular, neither of them allowed to learn such information about the relative position of z and P as whether z is at the northwest side of P, or whether z is close to one of the borders of P, etc.

Problem 2. (Intersection) Alice has a polygon A, and Bob has a polygon B; they both want to determine whether A and B intersect (not where the intersection occurs).

Problem 3. (Closest Pair) Alice has M points in the plane, Bob has N points in the plane. Alice and Bob want to jointly find two points among these $M + N$ points, such that their mutual distance is smallest.

Problem 4. (Convex Hulls) Alice has M points in the plane, Bob has N points in the plane. Alice and Bob want to jointly find the convex hulls for these $M+N$ points; however, neither Alice nor Bob wants to disclose any more information to the other party than what could be derived from the result.

Of course all of the above problems, as well as other computational geometry problems, are special cases of the general Secure Multi-party Computation problem [16, 9, 6]. Generally speaking, a secure multi-party computation problem deals with computing a function on any input, in a distributed network where each participant holds one of the inputs, ensuring that no more information is revealed to a participant in the computation than can be computed from that participant's input and output.

In theory, the general secure multi-party computation problem is solvable using circuit evaluation protocol [16, 9, 6]. While this approach is appealing in its generality, the communication complexity of the protocol it generates depends on the size of the circuit that expresses the functionality F to be computed, and in addition, involves large constant factors in their complexity. Therefore, as Goldreich points out in [6], using the solutions derived by these general results for special cases of multi-party computation can be impractical; special solutions should be developed for special cases for efficiency reasons. This is a good motivation for seeking special solutions to computational geometry problems, solutions that are more efficient than the general theoretical solutions.

Due to page limitations, we include detailed solutions to only two of the above problems: point-inclusion problem and intersection problem. Our work assumes that all parties are semi-honest; informally speaking, a semi-honest party is one who follows the protocol properly with the exception that it keeps a record of all its intermediate computations and might try to derive other parties' private inputs from the record. We also assume that adding a random number to an x effectively hides x. The assumption is known to be true in a finite field, in the infinite case, our protocols can be considered heuristic or approximation.

1.1 Related Work

The history of the multi-party computation problem is extensive since it was introduced by Yao [16] and extended by Goldreich, Micali, and Wigderson [9],

and by many others. These works use a similar methodology: each functionality F is represented as a Boolean circuit, and then the parties run a protocol for every gate in the circuit. While this approach is appealing in its generality and simplicity, the protocols it generates depend on the size of the circuit. This size depends on the size of the input and on the complexity of expressing F as a circuit. If the functionality F is complicated, using the circuit evaluation protocol will typically not be practical. However, if F is a simple enough functionality, using circuit a evaluation protocol can be practical.

The existing protocols listed below serve as important building blocks in our solutions. Our paper [5] contains some primitives for general scientific problems, that could be used as subroutines by some of our computations (as special cases), however the next section will give better solutions for the special cases that we need than the general ones given in [5] (more on this later).

The Circuit Evaluation Protocol In a circuit evaluation protocol, each functionality is represented by a Boolean circuit, and the construction takes this Boolean circuit and produces a protocol for evaluating it. The protocol scans the circuit from the input wires to the output wires, processing a single gate in each *basic step*. When entering each basic step, the parties hold shares of the values of the input wires, and when the step is completed they hold shares of the output wire.

1-out-of-N Oblivious Transfer Goldreich's circuit evaluation protocol uses the 1-out-of-N Oblivious Transfer, and our protocols in this paper also heavily depends on this protocol. An 1-out-of-N Oblivious Transfer protocol [7, 4] refers to a protocol where at the beginning of the protocol one party, Bob has N inputs $X_1, \ldots, X_N$ and at the end of the protocol the other party, Alice, learns one of the inputs X_i for some $1 \leq i \leq N$ of her choice, without learning anything about the other inputs and without allowing Bob to learn anything about i. An efficient 1-out-of-N Oblivious Transfer protocol was proposed in [11] by Naor and Pinkas. By combining this protocol with the scheme by Cachin, Micali and Stadler [8], the 1-out-of-N Oblivious Transfer protocol could be achieved with polylogarithmic (in n) communication complexity.

Homomorphic Encryption Schemes

We need a public-key cryptosystems with a *homomorphic* property for some of our protocols: $E_k(x) * E_k(y) = E_k(x+y)$. Many such systems exist, and examples include the systems by Benaloh [2], Naccache and Stern [10], Okamoto and Uchiyama [13], Paillier [14], to mention a few. A useful property of homomorphic encryption schemes is that an "addition" operation can be conduced based on the encrypted data without decrypting them.

Yao's Millionaire Problem

This is another protocol used as a primitive in our solutions; The purpose of the protocol is to compare two private numbers (i.e., determine which is larger). This private comparison problem was first proposed by Yao [15] and is referred as Yao's Millionaire Problem (because two millionaires wish to know who is richer, without revealing any other information about their net worth). The early cryptographic solution by Yao [15] has communication complexity that is exponential in the number of bits of the numbers involved, using an untrusted third party. Cachin proposed a solution [3] based on the Φ-hiding assumption. His protocol uses 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 either participant. The communication complexity of Cachin's scheme is $O(\ell)$, where ℓ is the number of bits of each input number.

2 New Building Blocks

In this section, we introduce two secure two-party protocols, a scalar product protocol, and a vector dominance protocol. Apart from serving as building blocks in solving the secure two-party computational geometry problems considered later in the paper, these two protocols are of independent interest and will be useful in solving other problems as well.

2.1 Scalar Product Protocol

Our paper [5] contains a matrix product protocol that could be used for scalar product (as a special case), but the scalar product protocol given below is better than using the general matrix multiplication protocol. We use $X \cdot Y$ to denote the scalar product of two vectors $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$, $X \cdot Y = \sum_{k=1}^{n} x_k y_k$. Our definition of the problem is slightly different more general: We assume that Alice has the vector X and Bob has the vector Y, and the goal of the protocol is for Alice (but not Bob) to get $X \cdot Y + v$ where v is random and known to Bob only (of course without either side revealing to the other the private data they start with). Our protocols can easily be modified to work for the version of the problem where the random v is given ahead of time as part of Bob's data (the special case $v = 0$ puts us back in the usual scalar product definition). The purpose of Bob's random v is as follows: If $X \cdot Y$ is a partial result that Alice is not supposed to know, then giving her $X \cdot Y + v$ prevents Alice from knowing the partial result (even though the scalar product has in fact been performed); later, at the end of the multiple-step protocol, the effect of v can be effectively "subtracted out" by Bob without revealing v to Alice (this should become clearer with example protocols that we later give).

Problem 5. (Scalar Product Problem) Alice has a vector $X = (x_1, \ldots, x_n)$ and Bob has a vector $Y = (y_1, \ldots, y_n)$. Alice (but not Bob) is to get the result of $u = X \cdot Y + v$ where v is a random scalar known to Bob only.

We have developed two protocols, and we will present both of them here.

Scalar Product Protocol 1 Consider the following naive solution: Alice sends p vectors to Bob, only one of which is X (the others are arbitrary). Then Bob computes the scalar products between Y and each of these p vectors. At the end Alice uses the 1-out-of-N oblivious transfer protocol to get back from Bob the product of X and Y. Because of the way oblivious transfer protocol works, Alice can decide which scalar product to get, but Bob could not learn which one Alice has chosen. There are many drawbacks to this approach: If the value of X has certain public-known properties, Bob might be able to differentiate X from the other $p-1$ vectors, but even if Bob is unable to recognize X his chances of guessing it is an unacceptably low 1 out of p.

The above drawbacks can be fixed by dividing vector X into m random vectors $V_1, \ldots, V_m$ of which it is the sum, i.e., $X = \sum_{i=1}^{m} V_i$. Alice and Bob can use the above naive method to compute $V_i \cdot Y + r_i$, where r_i is random number and $\sum_{i=1}^{n} r_i = v$ (see Figure 1). As a result of the protocol, Alice gets $V_i \cdot Y + r_i$ for $i = 1, \ldots, m$. Because of the randomness of V_i and its position, Bob could not find out which one is V_i. Certainly, there is 1 out p possibility that Bob can guess the correct V_i, but since X is the sum of m such random vectors, the chance that Bob guesses the correct X is 1 out p^m, which could be very small if we chose p^m large enough.

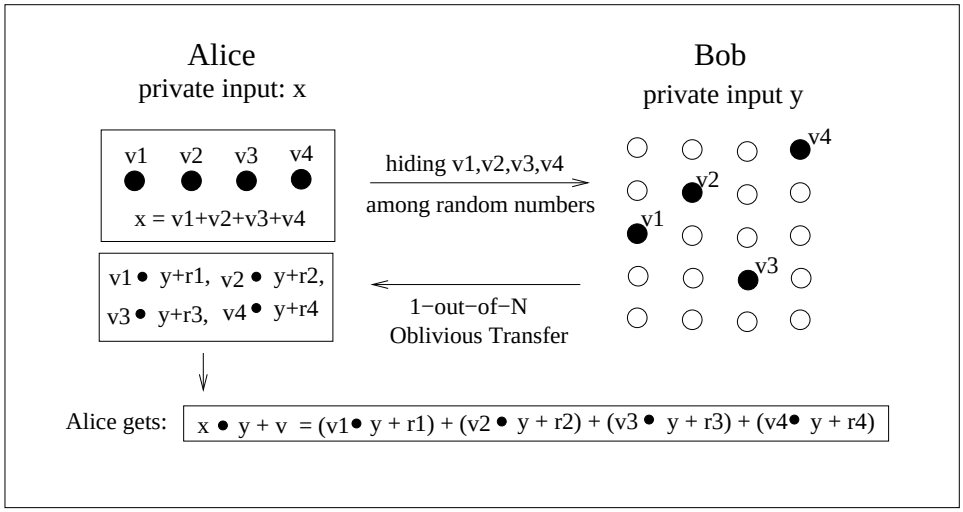

Fig. 1. Scalar Product Protocol 1

After Alice gets $V_i \cdot Y + r_i$ for $i = 1, \ldots, n$, she can compute $\sum_{i=1}^{m}(V_i \cdot Y + r_i)$ $= X \cdot Y + v$. The detailed protocol is described in the following:

Protocol 1 *(Two-Party Scalar Product Protocol 1)*
Inputs: Alice has a vector $X = (x_1, \ldots, x_n)$, and Bob has a vector $Y = (y_1, \ldots, y_n)$.

Outputs: Alice (but not Bob) gets $X \cdot Y + v$ where v is a random scalar known to Bob only.

1. Alice and Bob agree on two numbers p and m, such that p^m is so large that conducting p^m additions is computationally infeasible.
2. Alice generates m random vectors, $V_1, \ldots, V_m$, such that $X = \sum_{j=1}^{m} V_i$.
3. Bob generates m random numbers $r_1, \ldots, r_m$ such that $v = \sum_{j=1}^{m} r_j$.
4. For each $j = 1, \ldots, m$, Alice and Bob conduct the following sub-steps:
   (a) Alice generates a secret random number k, $1 \leq k \leq p$.
   (b) Alice sends $(H_1, \ldots, H_p)$ to Alice, where $H_k = V_j$, and the rest of H_i's are random vectors. Because k is a secret number known only to Alice, Bob does not know the position of V_j.
   (c) Bob computes $Z_{j,i} = H_i \cdot Y + r_j$ for $i = 1, \ldots, p$.
   (d) Using the 1-out-of-N Oblivious Transfer protocol, Alice gets $Z_j = Z_{j,k} = V_j \cdot Y + r_j$, while Bob learns nothing about k.
5. Alice computes $u = \sum_{j=1}^{m} Z_j = X \cdot Y + v$.

How is privacy achieved:

- If Bob chooses to guess, his chance of guessing the correct X is p^m.
- The purpose of r_j is to add randomness to $V_j \cdot Y$, thus preventing Alice from deriving information about Y.

Scalar Product Protocol 2 In the following discussion, we define $\pi(X)$ as another vector whose elements are random permutation of those of vector X.

We begin with two observations. First, a property of the scalar product $X \cdot Y$ is that $\pi(X) \cdot \pi(Y) = X \cdot Y$, regardless of what π is. Secondly, if Bob sends a vector $\pi(V)$ to Alice, where π and V are known only to Bob, Alice's chance of guessing the position of any single element of the vector V is 1 out of n (n is the size of the vector); Alice's chance of guessing the positions of all of the elements of the vector V is 1 out of $n!$.

A naive solution would be to let Alice get both $\pi(X)$ and $\pi(Y)$ but not π. Let us ignore for the time being the drawback that Alice gets the items of Y in permuted order, and let us worry about not revealing π to Alice: Letting Alice know $\pi(X)$ allows her to easily figure out the permutation function π from knowing both X and $\pi(X)$. In order to avoid this problem, we want to let Alice know only $\pi(X + R_b)$ instead of $\pi(X)$, where R_b is a random vector known only to Bob. Because of the randomness of $X + R_b$, to guess the correct π, Alice's chance is only 1 out of $n!$. Therefore to get the final scalar product, Bob only needs to send $\pi(Y)$ and the result of $R_b \cdot Y$ to Alice, who can compute the result of the scalar product by using

$$X \cdot Y = \pi(X + R_b) \cdot \pi(Y) - R_b \cdot Y$$

Now we turn our attention to the drawback that giving Alice $\pi(Y)$ reveals too much about Y (for example, if Alice is only interested in a single element

of the vector Y, her chance of guessing the right one is an unacceptably low 1 out of n). One way to fix this is to divide Y to m random pieces, $V_1, \ldots, V_m$, with $Y = V_1 + \ldots + V_m$; then Bob generates π random permutations $\pi_1, \ldots, \pi_m$ (one for each "piece" V_i of Y) and lets Alice know $\pi_i(V_i)$ and $\pi_i(X + R_b)$ for $i = 1, \ldots, m$. Now in order to guess the correct value of a single element of Y, Alice has to guess the correct position of V_i in each one of the m rounds; the possibility of a successful guessing becomes 1 out of n^m.

Now, let us consider the unanswered question: how could Alice get $\pi(X + R_b)$ without learning π or R_b? We do this with a technique based on a homomorphic public key system, that was used in [1] in a different context (to compute the minimum value in a vector that is the difference of Alice's private vector and Bob's private vector). Recall that an encryption scheme is *homomorphic* if $E_k(x) * E_k(y) = E_k(x+y)$. A good property of homomorphic encryption schemes is that "addition" operation can be conduced based on the encrypted data without decrypting them. Based on the homomorphic public key system, we have the following Permutation Protocol (where, for a vector $Z = (z_1, \ldots, z_n)$, we define $E(Z) = (E(z_1), \ldots, E(z_n))$, $D(Z) = (D(z_1), \ldots, D(z_n))$):

Protocol 2 *(Permutation Protocol)*
Inputs: Alice has a vector X. Bob has a permutation π and a vector R.
Output: Alice gets $\pi(X + R)$.

1. Alice generates a key pair for a homomorphic public key system and sends the public key to Bob. The corresponding encryption and decryption is denoted as $E(\cdot)$ and $D(\cdot)$.
2. Alice encrypts $X = (x_1, \ldots, x_n)$ using her public key and sends $E(X) = (E(x_1), \ldots, E(x_n))$ to Alice.
3. Bob computes $E(R)$, then computes $E(X) * E(R) = E(X + R)$; Bob then permutes $E(X + R)$ using the random permutation function π, thus getting $\pi(E(X + R))$; Bob sends the result of $\pi(E(X + R))$ to Alice.
4. Alice computes $D(\pi(E(X + R))) = \pi(D(E(X + R))) = \pi(X + R)$.

Based on Secure Two-Party Permutation Protocol, we have developed the following scalar product protocol:

Protocol 3 *(Secure Two-Party Scalar Product Protocol 2)*
Inputs: Alice has a secret vector X, Bob has a secret vector Y.
Output: Alice gets $X \cdot Y + v$ where v is a random scalar known to Bob only.

1. Bob's set up:
   (a) Bob divides Y to m random pieces, s.t. $Y = V_1 + \ldots + V_m$.
   (b) Bob generates m random vectors $R_1, \ldots, R_m$, let $v = \sum_{i=1}^{m} V_i \cdot R_i$.
   (c) Bob generates m random permutations $\pi_1, \ldots, \pi_m$.
2. For each $i = 1, ..., m$, Alice and Bob do the following:
   (a) Using Secure Two-Party Permutation Protocol, Alice gets $\pi_i(X + R_i)$ without learning either π_i or R_i.

(b) Bob sends $\pi_i(V_i)$ to Alice.
(c) Alice computes $Z_i = \pi_i(V_i) \cdot \pi_i(X + R_i) = V_i \cdot X + V_i \cdot R_i$

3. Alice computes $u = \sum_{i=1}^{m} Z_i = \sum_{i=1}^{m} V_i \cdot X + \sum_{i=1}^{m} V_i \cdot R_i = X \cdot Y + v$

How is privacy achieved:

- The purpose of R_i is to prevent Alice from learning π_i.
- The purpose of π_i is to prevent Alice from learning V_i. Although Alice learns a random permutation of the V_i, she does not learn more because of the randomness of V_i. Without π_i, Alice could learn each single value of V_i.
- If Alice chooses to guess, in order to successfully guess all of the elements in Y, her chance is $(\frac{1}{n!})^m$.
- Alice's chance of successfully guessing just one elements of Y is n^m. For example, in order to guess the kth element of Y, Alice has to guess the the corresponding elements in $\pi_i(V_i)$ for all $i = 1, \ldots, m$; for each i, the chance is $\frac{1}{n}$.
- A drawback of this protocol is that the information about $\sum_{i=1}^{n} y_i$ is disclosed because the random permutation does not help to hide this information.

Comparison of These Two Protocols The communication cost of Protocol 3 is $4m * n$, where m is a security parameter (so that $\mu' = n^m$ is large enough). The communication cost of Protocol 1 is $p * t * n$, where $p \geq 2$ and t are security parameters such that $\mu'' = p^t$ is large enough. Setting $\mu' = \mu'' = \mu$ for the sake of comparison, the communication cost of Protocol 3 is $4 \log \mu \frac{n}{\log n}$ and the communication cost of Protocol 1 is $\frac{p \log \mu}{\log p} n$. When n is large, Protocol 3 is more efficient than Protocol 1.

2.2 Secure Two-Party Vector Dominance Protocol

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

Problem 6. (Secure Two-Party Vector Dominance Problem) Alice has a vector $A = (a_1, \ldots, a_n)$ and Bob has a vector $B = (b_1, \ldots, b_n)$. Alice wants to know whether A dominates B. Note in the case where A does not dominate B, neither Alice nor Bob should learn the relative ordering of any individual a_i, b_i pair (i.e., whether $a_i < b_i$ or not).

The Protocol We first give an outline of the protocol, then discuss each step in details.

Protocol 4 *(Secure Two-Party Vector Dominance Protocol)*
Inputs: Alice has a vector $A = (a_1, \ldots, a_n)$, Bob has a vector $B = (b_1, \ldots, b_n)$.

1. Inputs Disguise: Using a disguise technique (described later), Alice gets the disguised input $A' = (a'_1, \ldots, a'_{4n})$, and Bob gets the disguised input $B' = (b'_1, \ldots, b'_{4n})$. Let V_A
$$V_A = (\overbrace{1, \ldots, 1}^{2n}, \overbrace{0, \ldots, 0}^{2n})$$
2. Private Permutation: Bob generates a random permutation π and a random vector R. Using the Permutation Protocol (Protocol 2), Alice gets $A'' = \pi(A' + R)$. Bob also computes $B'' = \pi(B' + R)$, $V'_A = \pi(V_A)$.
3. Yao's Millionaire Comparison: Alice and Bob use Yao's Millionaire protocol as subroutine to compare A''_i with B''_i, for $i = 1, \ldots, 4n$, where A''_i (resp., B''_i) is the ith element of vector A'' (resp., B''). At the end, Alice gets the result $U = \{u_1, \ldots, u_{4n}\}$, where $u_i = 1$ if $A''_i > B''_i$, otherwise $u_i = 0$.
4. Dominance Testing: Alice and Bob use a private equality-testing protocol to compares U with V'_A: If $U = V'_A$, then A dominates B; otherwise, A does not dominate the B. (Note: when we later use this protocol for the intersection protocol, this step must be skipped.)

Outputs: If the Dominance Testing step needs to be skipped, Alice outputs U and Bob outputs V'_A. Otherwise, Alice and Bob each output the dominance testing results.

Step 1: Inputs Disguise

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. The disguised inputs are the followings:

$$\begin{aligned} A' = (&2a_1, \ldots, 2a_n, (2a_1 + 1), \ldots, (2a_n + 1), \\ &- 2a_1, \ldots, -2a_n, -(2a_1 + 1), \ldots, -(2a_n + 1)) \qquad (1) \\ B' = (&(2b_1 + 1), \ldots, (2b_n + 1), 2b_1, \ldots, 2b_n, \\ &- (2b_1 + 1), \ldots, -(2b_n + 1), -2b_1, \ldots, -2b_n) \qquad (2) \end{aligned}$$

The purpose of the inputs disguise is to get the same number of $a'_i > b'_i$ situations as that of $a'_i < b'_i$ situations; therefore, nobody knows how many a_i's are larger than b_i's and vice versa. The disguise is based on the fact that if $a_i > b_i$, then $2a_i > 2b_i + 1$, $(2a_i + 1) > 2b_i$, $-2a_i < -(2b_i + 1)$, and $-(2a_i + 1) < -2b_i$, which generates two $>$'s, and two $<$'s.

Step 2: Private Permutation: This step is fully discussed in Secure Two-Party Permutation Protocol (Protocol 2).

Step 3: Yao's Millionaire Comparison

Alice now has $A'' = \pi(A' + R) = (a''_1, \ldots, a''_{4n})$, Bob has $B'' = \pi(B' + R) = (b''_1, \ldots, b''_{4n})$. They can use Yao's Millionaire Protocol to compare each a''_i with b''_i. Actually it is an one-side (asymmetric) version of it because only Alice learns the result. So at the end of this step, Alice gets $U = (u_1, \ldots, u_{4n})$, where for $i = 1, \ldots, 4n$, $u_i = 1$ if $a'_i > b'_i$, otherwise $u_i = 0$.

Step 4: Dominance Testing

Because V_A is exactly what U should be if vector A dominates B, we only need to find out whether $U = V_A$. Alice cannot just send U to Bob because it will allow Bob to find out the relationship between a_i and b_i for each $i = 1, \ldots, n$. So we need a way for Alice and Bob to determine whether Alice's U equals Bob's V_A without disclosing each person's private input to the other person.

This comparison problem is well studied, and was thoroughly discussed by Fagin, Naor, and Winkler [12]. Several methods for it were discussed in [12, 11]. For example, the following is part of the folklore:

Protocol 5 *(Equality-Testing Protocol)*
Inputs: Alice has U, Bob has V_A.
Outputs: $U = V_A$ iff $E_B(E_A(U)) = E_A(E_B(V_A))$.

1. Alice encrypts U with a commutative encryption scheme, and gets $E_A(U)$; Alice sends $E_A(U)$ to Bob.
2. Bob encrypts $E_A(U)$, and gets $E_B(E_A(U))$; Bob sends the result back to Alice.
3. Bob encrypts V_A, gets $E_B(V_A)$; Bob sends $E_B(V_A)$ to Alice.
4. Alice encrypts $E_B(V_A)$, gets $E_A(E_B(V_A))$.
5. Alice compares $E_B(E_A(U))$ with $E_A(E_B(V_A))$.

3 Secure Two-Party Geometric Computations

In the following, we want to illustrate how the building blocks we studied earlier can be put together to solve geometric problems. Many other geometric problems are amenable to such solutions; and in fact we suspect that the solutions we give below can be further improved.

3.1 Secure Two-Party Point-Inclusion Problem

We will look at how the point-location problem is solved in a straightforward way without worrying about the privacy concern. The computation cost of this straightforward solution is $O(n)$. Although we know the computation cost of the best algorithm for the point-location problem is only $O(\log n)$, we are concerned that the "binary search" nature of that solution might lead to the disclosure of partial information. Therefore, for a preliminary result, we focus on the $O(n)$ solution. The algorithm works as follows:

1. Find the leftmost vertex l and the rightmost vertex r of the polygon.
2. Decide whether the point $p = (\alpha, \beta)$ is above all the edges on the lower boundary of the polygon between l and r.
3. Decide whether the point α is below all the edges on the upper boundary of the polygon between l and r.
4. If the above two tests are both true, then the point is inside the polygon, otherwise it is outside (or on the edge) of the polygon.

If we use $f_i(x, y) = 0$ for the equation of the line boundary of the polygon, where $f_i(x, y) = 0$ for $i = 1, \ldots, m$ represent the edges on the lower part of the boundary and $f_i(x, y) = 0$ for $i = m + 1, \ldots, n$ represent the edges on the upper part of the boundary, then our goal is to decide whether $f_i(\alpha, \beta) > 0$ for all $i = 1, \ldots, m$ and $f_i(\alpha, \beta) < 0$ for all $i = m + 1, \ldots, n$.

The Protocol First, we need to find a way to compute $f_i(\alpha, \beta)$ without disclosing Alice's $p : (\alpha, \beta)$ to Bob or Bob's f_i to Alice. Moreover, no party should learn the result of $f_i(\alpha, \beta)$ for any i because that could disclose the relationship between the location and the edge. Since $f_i(\alpha, \beta)$ is a special case of scalar product, we can use Secure Two-Party Scalar Product Protocol to solve this problem. In this protocol, we will let both party share the result of $f_i(\alpha, \beta)$, namely, one party will have u_i, the other party will have v_i, and $u_i = f_i(\alpha, \beta) + v_i$; therefore nobody learns the value of $f_i(\alpha, \beta)$, but they can find out whether $f_i(\alpha, \beta) > 0$ by comparing whether $u_i > v_i$, which could be done using Yao's Millionaire Protocol [15, 3].

However, we cannot use Yao's Millionaire Protocol for each (u_i, v_i) pair individually because that would disclose the relationship between u_i and v_i, thus reveal too much information. In fact, all we want to know is whether $(u_1, \ldots, u_n)$ dominates $(v_1, \ldots, v_n)$. This problem can be solved using the *Vector Dominance Protocol* (Protocol 4)

Based on the Scalar Product Protocol and Vector Dominance Protocol, we have the following Secure Two-Party Point-Inclusion Protocol:

1. Bob generates n random numbers $v_1, \ldots, v_n$.
2. Alice and Bob use Scalar Product Protocol to compute $u_i = f_i(\alpha, \beta) + v_i$, for $i = 1, \ldots, m$ and compute $u_i = -f_i(\alpha, \beta) + v_i$ for $i = m + 1, \ldots, n$. According to the scalar product protocol, Alice will get $(u_1, \ldots, u_n)$ and Bob will get $(v_1, \ldots, v_n)$. Bob will learn nothing about u_i and (α, β); Alice will learn nothing about v_i and the function $f_i(x, y)$.
3. Alice and Bob use the *Vector Dominance Protocol* to find out whether vector $A = (u_1, \ldots, u_n)$ dominates $B = (v_1, \ldots, v_n)$. According to the Vector Dominance Protocol, if A does not dominate B, no other information is disclosed.

Claim. If $A = (u_1, \ldots, u_n)$ dominates $B = (v_1, \ldots, v_n)$, then the point $p = (\alpha, \beta)$ is inside the polygon; otherwise, the point is outside (or on the edge) of the polygon.

3.2 Secure Two-Party Intersection Problem

Two polygons intersect if (1) one polygon is inside another, or (2) at least one edge of a polygon intersects with one edge of another polygon. Since (1) can be decided using the Point-Inclusion Protocol, we only focus on (2).

We will first look at how the intersection problem could be solved in a straightforward way ($O(n^2)$) without worrying about the privacy concern. For

the same reason we decide not to use the more efficient $O(n)$ algorithm because of the concern about the partial information disclosure. The algorithm works as follows:

1. For each pair (e_i, e'_j), decide whether e_i intersects with e_j, where e_i is an edge of polygon A and e'_j is an edge of polygon B,
2. If there exists an edge $e_i \in A$ and an edge $e'_j \in B$, such that e_i intersects with e'_j, then A and B intersect.

We use $f_i(x,y),(x_i,y_i),(x'_i,y'_i)$, for $i = 1,\ldots,n_a$, to represent each edge of the polygon A, where $f_i(x,y)$ is the equation of the line containing that edge, (x_i,y_i) and (x_i,y_i) represents the two endpoints of the edge. We use $g_i(x,y)$, for $i = 1,\ldots,n_b$, to represent each edge of the polygon B.

The Protocol During the testing of whether two edges intersect with each other, obviously, nobody should learn the result of each individual test; otherwise, he knows which of his edge intersects with the other party's polygon. In our scheme, Alice and Bob conduct these n^2 testings, but nobody knows the result of each individual test, instead, they share the results of each test, namely each of them gets a seemly-random piece of the result. One has to obtain both pieces in order to know the result of each test. At the end, all these shared pieces are put together in a way that only a single result is generated, to show only whether the two polygon boundaries intersect or not.

First, let us see how to conduct such a secure two-party testing of the intersection. Assume Alice has a edge $f_1(x,y) = 0$, where $f_1(x,y) = a_1x + b_1y + c_1$, and $a_1 \geq 0$; the two endpoints of the edge are (x_1,y_1) and (x'_1,y'_1). Bob has a line $f_2(x,y) = 0$, where $f_2(x,y) = a_2x + b_2y + c_2$, $a_2 \geq 0$; the two endpoints of the edge are (x_2,y_2) and (x'_2,y'_2). According to the geometries, f_1 and f_2 intersect if and only if f_1's two endpoints (x_1,y_1), (x'_1,y'_1) are on the different sides of f_2, and f_2's two endpoints (x_2,y_2) and (x'_2,y'_2) are on the different sides of f_1. In another words, f_1 and f_2 intersect if and only if one of the following expressions is true:

- $f_1(x_2,y_2) > 0 \wedge f_1(x'_2,y'_2) < 0 \wedge f_2(x_1,y_1) > 0 \wedge f_2(x'_1,y'_1) < 0$
- $f_1(x_2,y_2) > 0 \wedge f_1(x'_2,y'_2) < 0 \wedge f_2(x_1,y_1) < 0 \wedge f_2(x'_1,y'_1) > 0$
- $f_1(x_2,y_2) < 0 \wedge f_1(x'_2,y'_2) > 0 \wedge f_2(x_1,y_1) > 0 \wedge f_2(x'_1,y'_1) < 0$
- $f_1(x_2,y_2) < 0 \wedge f_1(x'_2,y'_2) > 0 \wedge f_2(x_1,y_1) < 0 \wedge f_2(x'_1,y'_1) > 0$

We cannot let either party know the results of $f_1(x_2,y_2)$, $f_1(x'_2,y'_2)$, $f_2(x_1,y_1)$, or $f_2(x'_1,y'_1)$ (in the following discussion, we will use $f(x,y)$ to represent any of these expressions). According to the Scalar Product Protocol, we can let Alice know the result of $f(x,y) + r$, and let Bob know r, where r is a random number generated by Bob. Therefore, nobody knows the actual value of $f(x,y)$, but Alice and Bob can still figure out whether $f(x,y) > 0$ by comparing $f(x,y) + r$ with r.

Let $u_1 = f_1(x_2,y_2) + r_1$, $u'_1 = f_1(x'_2,y'_2) + r'_1$, $u_2 = f_2(x_1,y_1) + r_2$, and $u'_2 = f_2(x'_1,y'_1) + r'_2$. Alice has (u_1,u'_1,u_2,u'_2) and Bob has (r_1,r'_1,r_2,r'_2). Then f_1 and f_2 intersect if and only if one of the following expressions is true:

- $u_1 > r_1 \wedge u_1' < r_1' \wedge u_2 > r_2 \wedge u_2' < r_2'$
- $u_1 > r_1 \wedge u_1' < r_1' \wedge u_2 < r_2 \wedge u_2' > r_2'$
- $u_1 < r_1 \wedge u_1' > r_1' \wedge u_2 > r_2 \wedge u_2' < r_2'$
- $u_1 < r_1 \wedge u_1' > r_1' \wedge u_2 < r_2 \wedge u_2' > r_2'$

Our next step is to compute each of the above expressions. As before, nobody should learn the individual comparison results, just the aggregate. Let us use E to denote any one of the above expressions. Using the Vector Dominance Protocol, we can get Alice to know a random piece t, and Bob to know another random piece s, such that E is true if and only if $t = s$.

Now Alice has $4 * n^2$ numbers $(t_1, \ldots, t_{4n^2})$, Bob has $(s_1, \ldots, s_{4n^2})$. We want to know whether there exists an $i = 1, \ldots, 4n^2$, such that $t_i = s_i$. Although there are some other approaches to achieve this, we believe using the circuit evaluation protocol is efficient in this case, because the size of the circuit is small (linear in the number of the items). The security of the circuit evaluation protocol guarantees that only the final results–`yes` or `no`–will be disclosed; nobody learns any other information, such as how many t_i's equal to s_i's, and which $t_i = s_i$.

The following is the outline of the protocol:

1. Let $m = n_a * n_b$.
2. For each pair of edges, perform the following sub-protocol. Suppose the index of this edge pair is i, for $i = 1, \ldots, m$; suppose $(f, (x_1, y_1), (x_1', y_1')) \in A$ and $(g, (x_2, y_2), (x_2', y_2')) \in B$ are two edges.
   (a) Using the scalar product protocol, Alice gets $U = (u_1, u_1', u_2, u_2')$, and Bob gets $R = (r_1, r_1', r_2, r_2')$, where $u_1 = f(x_2, y_2) + r_1$, $u_1' = f(x_2', y_2') + r_1'$, $u_2 = g(x_1, y_1) + r_2$, and $u_2' = g(x_1', y_1') + r_2'$.
   (b) Using the Vector Dominance Protocol, Alice gets $t_{i,1}, t_{i,2}, t_{i,3}, t_{i,4}$, and Bob gets $s_{i,1}, s_{i,2}, s_{i,3}, s_{i,4}$
3. Alice has $(t_{1,1}, t_{1,2}, t_{1,3}, t_{1,4}, \ldots, t_{m,1}, t_{m,2}, t_{m,3}, t_{m,4})$, and Bob has $(s_{1,1}, s_{1,2}, s_{1,3}, s_{1,4}, \ldots, s_{m,1}, s_{m,2}, s_{m,3}, s_{m,4})$. Alice and Bob uses circuit evaluation method to find out whether there exists $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, 4\}$, such that $t_{i,j} = s_{i,j}$.

4 Applications

The following two scenarios describe some potential applications of the problems we have discussed in this paper.

1. Company A decided that expanding its market share in some region will be very beneficial after a costly market research; therefore A is planning to do this. However A is aware of that another competing company B is also planning to expand its market share in some region. Strategically, A and B do not want to compete against each other in the same region, so they want to know whether they have a region of overlap? Of course, they do not want to give away location information because not only does this information cost both companies a lot of money, but it can also cause significant damage

to the company if it were disclosed to other parties: for example, a larger competitor can immediately occupy the market there before A or B even starts; or some real estate company can actually raise their price during the negotiation if they know A or B is very interested in that location. Therefore, they need a way to solve the problem while maintaining the privacy of their locations.

2. A country decides to bomb a location x in some other country; however, A does not want to hurt its relationship with its friends, who might have some areas of interests in the bombing region: for example, those countries might have secret businesses, secret military bases, or secret agencies in that area. Obviously, A does not want to disclose the exact location of x to all of its friends, except the one who will definitely be hurt by this bombing; on the other hand, its friends do not want to disclose their secret areas to A either, unless they are in the target area. How could they solve this dilemma? If each secret area is represented by a secret polygon, the problem becomes how to decide whether A's secret point is within B's polygon, where B represents one of the friend countries. If the point is not within the polygon, no information should be disclosed, including the information such as whether the location is at the west of the polygon, or within certain longitude or latitude. Basically it is "all-or-nothing": if one will be bombed, it knows all; otherwise it knows nothing.

5 Conclusions and Future Work

In this paper, we have considered several secure two-party computational geometry problems and presented some preliminary work for solving such problems. For the purpose of doing so, we have also presented two useful building blocks, Secure Two-Party Scalar Product Protocol and Secure Two-Party Vector Dominance Protocol.

In the protocols for the Point-Inclusion problem and the Intersection problem, we use an inefficient algorithm to decide whether a point is side a polygon (or whether two polygon intersect) although more efficient solutions exist, because of the concern about information disclosure. In our future work, we will study how to take advantage of those efficient solutions without degrading the privacy.

References

1. Wenliang Du, Mikhail J. Atallah and Florian Kerschbaum. Protocols for secure remote database access with approximate matching. Technical report, 2001.
2. J. Benaloh. Dense probabilistic encryption. In *Proceedings of the Workshop on Selected Areas of Cryptography*, pages 120–128, Kingston, ON, May 1994.
3. 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.

4. G. Brassard, C. Crépeau and J. Robert. All-or-nothing disclosure of secrets. In *Advances in Cryptology - Crypto86, Lecture Notes in Computer Science*, volume 234-238, 1987.
5. Wenliang Du and Mikhail J. Atallah. Privacy-preserving cooperative scientific computations. In *14th IEEE Computer Security Foundations Workshop*, Nova Scotia, Canada, June 11-13 2001.
6. O. Goldreich. Secure multi-party computation (working draft). Available from http://www.wisdom.weizmann.ac.il/home/oded/public_html/ foc.html, 1998.
7. S. Even, O. Goldreich and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28:637–647, 1985.
8. 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.
9. 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.
10. D. Naccache and J. Stern. A new cryptosystem based on higher residues. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 59–66, 1998.
11. 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.
12. R. Fagin, M. Naor and P. Winkler. Comparing information without leaking it. *Communication of the ACM*, 39:77–85, 1996.
13. T. Okamoto and S. Uchiyama. An efficient public-key cryptosystem. In *Advances in Cryptology – EUROCRYPT 98*, pages 308–318, 1998.
14. P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *Advances in Cryptology – EUROCRYPT 99*, pages 223–238, 1999.
15. A.C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, 1982.
16. A.C. Yao. How to generate and exchange secrets. In *Proceedings 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.