Secure Multiparty Computation: Line Segment Intersection Evaluation

CS468 - Advanced Network Security

William Harding and Jonathan McCluskey

April 26, 2014

Abstract

The authors present a secure two-party protocol, using the Paillier cryptosystem, allowing the parties to determine whether or not their line segments intersect (within an agreed upon coordinate plane) without revealing any information about the line segments themselves. An example military application is given to demonstrate the significance of the problem. Source code, available on Github, allows for easier reproduction of this protocol.

Contents

List of Tables

List of Figures

1 Background

The field of Secure Multiparty Computation (SMC) is both narrow and broad. It is narrow in the sense that every SMC problem has the goal of revealing some information while hiding other information. It is broad in that it is a field which is applicable to innumerable problems where two or more parties wish to share or determine specific information without revealing the whole.

Shamir's Secret Sharing [6] is an early example of multiple parties using secret information to attain a common goal without revealing that information to each other. Shamir uses polynomials and the principle of function composition for hiding secrets. One of the classic problems in the field of SMC was given by Yao in [11]. Yao's toy problem consists of two wealthy individuals who want to compare the amount of money that they each have (i.e. is $a < b$?) without revealing, to one another, their actual wealth.

At the heart of at least one branch of Secure Multiparty Computation is the principle of homomorphism found in certain encryption schemes. That is, that certain encryption schemes allow some mathematical operations to be performed on their encrypted values without losing the ability to recover the resulting plaintext. One such encryption scheme is the popular RSA cryptosystem [5]. RSA is multiplicative homomorphic. Another well known cryptosystem was developed by Paillier in [4]. The Paillier cryptosystem is additive homomorphic, and, hence, has the added benefit of being able to obtain the encrypted result of a multiplication between a plaintext value and an encrypted value.

2 Problem Statement

2.1 Motivation

In [2], Du and Atallah present several open problems for Secure Multiparty Computation. One of their problems, "Privacy Preserving Geometric Computation", consist of two parties that both have a shape. The two parties wish to determine whether their shapes intersect without revealing their shapes. In our work we have sought to find a solution to at least a portion of this problem.

2.2 Problem

Given two semi-honest parties, each having a line segment (within the same coordinate plane); without revealing their line segments to each other, or any potential eavesdropper, the parties would like to know if their line segments intersect.

2.3 Significance

The application of a solution to this problem could be extended to many fields (commercial, military, etc). For a specific example imagine two armies from nations that are pseudo-allies

(i.e. not really friends, but having a common enemy). Imagine they are both planning to advance their armies, but neither is willing to reveal, to the other party, their mission plans (i.e. line segment routes of each of their respective units). Yet so, if their armies meet (i.e. intersect) en route, it could be problematic.

Using Secure Multiparty Computation could be extremely useful, given the above scenario. In our work we have developed:

- A protocol for securely determining the intersection of two line segments.
- A networked application in Java to demonstrate the protocol.

3 Proposed Solution

3.1 Geometric Principles

In order to understand a secure multiparty approach to determining line segment intersection, it would be beneficial to introduce the fundamental geometric principles that will be used. These include an intersection algorithm and, used within that algorithm, a point ordering function. The reader, if interested in further details, is directed toward [8] and [9].

First, let us define terms and assumptions. Intersection can be simply defined as two line segments that cross. We add the assumption that "crossing" includes even the endpoint of one line on the other (including the other's endpoint itself). With this definition of intersection, non-intersection is clear; namely, two lines that are not crossing, in fact not even touching each other.

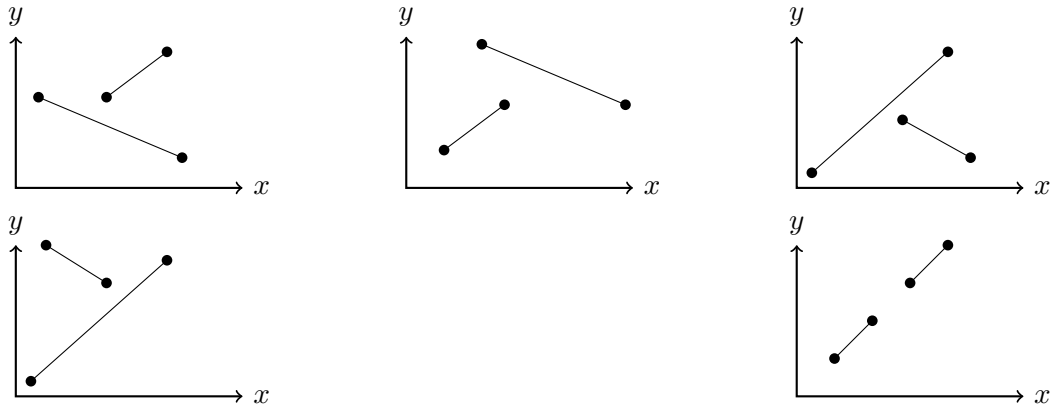

Figure 1: Examples of Non-intersection

In order to determine if two line segments intersect, we could choose to determine if they do not intersect and negate the result. Non-intersection, as shown in Figure 1, can be seen in the following way: Either the two points of one line are both on the same side of the

other line, or the two points of the other line are both on the same side of the first line (for the time being, please ignore the collinear non-intersection edge case shown in Figure 1; it will be discussed in further detail in section 4) Formally, this would look like:

$$\begin{aligned} ccw &\leftarrow ORDER.Counterclockwise \\ cw &\leftarrow ORDER.Clockwise \\ col &\leftarrow ORDER.Collinear \\ nonIntersect &\leftarrow ((Order(p_{1a}, p_{1b}, p_{2a}) = ccw) \wedge (Order(p_{1a}, p_{1b}, p_{2b}) = ccw)) \vee \\ &\quad ((Order(p_{1a}, p_{1b}, p_{2a}) = cw) \wedge (Order(p_{1a}, p_{1b}, p_{2b}) = cw)) \vee \\ &\quad ((Order(p_{2a}, p_{2b}, p_{1a}) = ccw) \wedge (Order(p_{2a}, p_{2b}, p_{1b}) = ccw)) \vee \\ &\quad ((Order(p_{2a}, p_{2b}, p_{1a}) = cw) \wedge (Order(p_{2a}, p_{2b}, p_{1b}) = cw)) \\ intersect &\leftarrow \overline{nonIntersect} \end{aligned} \tag{1}$$

Where $ORDER$ is an enumeration containing $Counterclockwise$, $Clockwise$, and $Collinear$; and where p_{lk} defines a point k on line segment l; and where $Order$ is a function defining the order of a given 3-point sequence. The $Order$ function is further explained in the following paragraphs.

In order to understand if two points are both on the same side of a line, we should first seek to understand what it means that one point is on a given side of a line. More to the point, since the line is defined by two points, we are looking to understand one point in relation to the two others (i.e. the two that define the line segment). The relationship of the three points can be understood in terms of "point order." The points, in the sequence provided, are either Counterclockwise, Clockwise, or Collinear. See Figure 2.

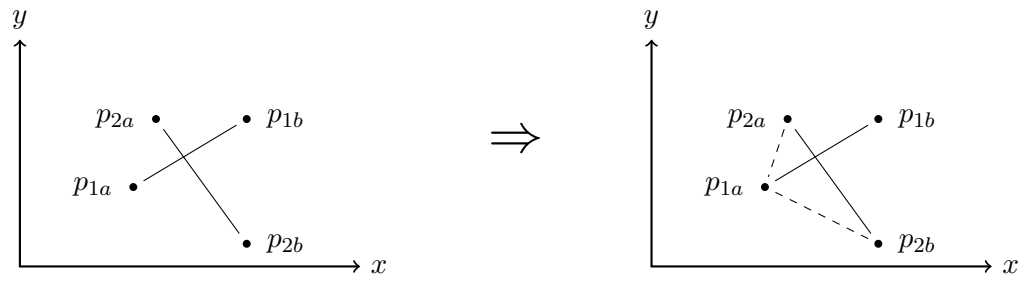

Figure 2: Example of Point Order: p_{1a} is Counterclockwise to $\overline{p_{2a}p_{2b}}$

Mathematically, point order can be understood as:

$$Order(j,k,l) = \begin{cases} ORDER.Counterclockwise & (k_y - l_y)(j_x - l_x) > (j_y - l_y)(k_x - l_x) \\ ORDER.Clockwise & (k_y - l_y)(j_x - l_x) < (j_y - l_y)(k_x - l_x) \\ ORDER.Collinear & (k_y - l_y)(j_x - l_x) = (j_y - l_y)(k_x - l_x) \end{cases} \tag{2}$$

This equation can be arranged, for reasons that will be clear later, as follows:

$$Order(j,k,l) = \begin{cases} ORDER.Counterclockwise & calcDet(j,k,l) > 0 \\ ORDER.Clockwise & calcDet(j,k,l) < 0 \\ ORDER.Collinear & calcDet(j,k,l) = 0 \end{cases} \tag{3}$$

Where

$$\begin{aligned} calcDet(j,k,l) = & \begin{vmatrix} j_x & j_y & 1 \\ k_x & k_y & 1 \\ l_x & l_y & 1 \end{vmatrix} \\ = & j_x k_y + j_y l_x + k_x l_y - j_x l_y - j_y k_x - k_y l_x \end{aligned} \tag{4}$$

3.2 Paillier Cryptosystem

As seen in equation 4 determining the intersection of two line segments requires addition, subtraction, and multiplication.

We decided that the Paillier cryptosystem [4] was a good choice because of its additive homomorphic properties as well as providing the ability to multiply an encrypted value by a plaintext value to get the encrypted result of the multiplication. Using these properties, and organizing the *Order* function as in equation 3 allows us to securely determine whether the two lines intersect. For additional details on the implementation and theory behind the Paillier cryptosystem, we refer the reader to [4] and [3]. We have used a third-party library written in Java [7] for the underlying encryption functions in our application.

Encryption and the homomorphic properties for the Paillier cryptosystem are as follows:

$$\begin{aligned} E(p) &= g^p r^n \mod n^2 \\ E(p_1) \cdot E(p_2) &= (g^{p_1} r_1^n)(g^{p_2} r_2^n) \\ &= g^{p_1+p_2}(r_1 r_2)^n \\ &= E(p_1 + p_2 \mod n) \\ E(p_1)^{p_2} &= E(p_1 p_2 \mod n) \\ E(p_1)^{-1} &= E(-p_1 \mod n) \\ E(p_1) \cdot E(p_2)^{-1} &= E(p_1 - p_2 \mod n) \end{aligned} \tag{5}$$

See source code Listing 1 for an example MATLAB implementation of generating a public/private key for the Paillier cryptosystem. Listing 1 requires the third-party Variable Precision Integer package [1]. For more specific details on implementing the Paillier cryptosystem, see [3] and [10].

Listing 1: Paillier Crytosystem Functions in MATLAB

```
1 % Paillier Crytposystem
```

```
p = vpi(7);
q = vpi(11);

% Verify p and q are prime
if gcd(p*q, (p-1)*(q-1)) == 1
    fprintf("p and q are good\n");
end;

n = p*q;
lambda = lcm(p-1, q-1);

g = 12;

% Verify g is acceptable
if gcd(L(mod(g^lambda, n^2), n), n) == 1
    fprintf("g is good\n");
end

%minv = modulo inverse
mu = minv(L(mod(g^lambda, n^2), n), n);


%% Example Encryption and Homomorphic Addition
plain_text1 = 2;
plain_text2 = 5;

% random non-zero integers in the set Z_n
r1 = 6;
r2 = 2;

% Encryption
cipher_text1 = mod(g^plain_text1 * r1^n, n^2);
cipher_text2 = mod(g^plain_text2 * r2^n, n^2);

% Homomorphic Addition
sum_of_ciphers = cipher_text1*cipher_text2;

% Decryption
decrypted_text1 = mod(L(mod(cipher_text1^lambda, n^2), n) * mu, n)
decrypted_text2 = mod(L(mod(cipher_text2^lambda, n^2), n) * mu, n)

decrypted_sum = mod(L(mod(sum_of_ciphers^lambda, n^2), n) * mu, n)
```

3.3 Protocol

Our protocol assumes two semi-honest parties. The two parties are represented by Alice (holding line segment 1) and Bob (who has line segment 2). Alice's points are represented by the subscripts 1a and 1b, likewise Bob's points are represented by the subscripts 2a and 2b. Alice and Bob each have private keys and have published their public keys to each other.

1. Alice (A) encrypts the plaintext points (p_{1a_x}, p_{1a_y}, p_{1b_x}, and p_{1b_y}) with her public key (A_{pubKey}) giving her the ciphertexts (c_{1a_x}, c_{1a_y}, c_{1b_x}, and c_{1b_y}). Alice then sends the

ciphertexts to Bob (B).

$$\begin{aligned} A \rightarrow B :& \{p_{1a_x}\}_{A_{pubKey}} \\ A \rightarrow B :& \{p_{1a_y}\}_{A_{pubKey}} \\ A \rightarrow B :& \{p_{1b_x}\}_{A_{pubKey}} \\ A \rightarrow B :& \{p_{1b_y}\}_{A_{pubKey}} \end{aligned} \tag{6}$$

2. Using equation 3 Bob calculates the orders of both of his points with each of Alice's points. The resulting values are encrypted based on the homomorphic properties of the Paillier cryptosystem. [1]

$$\begin{aligned} c_{o_{1a}} =& (p_{2b_y} * p_{2a_x}) - (p_{2a_y} * p_{2b_x}) - (c_{1a_x} * p_{2b_y}) + \\ & (c_{1a_x} * p_{2a_y}) + (c_{1a_y} * p_{2a_y}) - (c_{1a_y} * p_{2a_x}) \end{aligned} \tag{7}$$

and

$$\begin{aligned} c_{o_{1b}} =& (p_{2b_y} * p_{2a_x}) - (p_{2a_y} * p_{2b_x}) - (c_{1b_x} * p_{2b_y}) + \\ & (c_{1b_x} * p_{2a_y}) + (c_{1b_y} * p_{2a_y}) - (c_{1b_y} * p_{2a_x}) \end{aligned} \tag{8}$$

3. Bob sends $c_{o_{1a}}$ and $c_{o_{1b}}$ to Alice.

$$\begin{aligned} B \rightarrow A :& \{c_{o_{1a}}\} \\ B \rightarrow A :& \{c_{o_{1b}}\} \end{aligned} \tag{9}$$

4. Alice decrypts $c_{o_{1a}}$ and $c_{o_{1b}}$ with her private key obtaining $p_{o_{1a}}$ and $p_{o_{1b}}$.

$$\begin{aligned} p_{o_{1a}} \leftarrow& \{c_{o_{1a}}\}_{A_{privKey}} \\ p_{o_{1b}} \leftarrow& \{c_{o_{1b}}\}_{A_{privKey}} \end{aligned} \tag{10}$$

5. Alice performs the $Sign$ function on $p_{o_{1a}}$ and $p_{o_{1b}}$.

$$Sign(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases} \tag{11}$$

$$\begin{aligned} p_{s_{1a}} \leftarrow& Sign(p_{o_{1a}}) \\ p_{s_{1b}} \leftarrow& Sign(p_{o_{1b}}) \end{aligned} \tag{12}$$

[1] For simplicity, we have overloaded each of the mathematical operators that include ciphertext using the relationships found in equation 5.

6. Alice encrypts $p_{s_{1a}}$ and $p_{s_{1b}}$ with Bob's public key and sends $c_{s_{1a}}$ and $c_{s_{1b}}$ to Bob.

$$\begin{aligned} A \rightarrow B : &\{p_{s_{1a}}\}_{B_{pubKey}} \\ A \rightarrow B : &\{p_{s_{1b}}\}_{B_{pubKey}} \end{aligned} \tag{13}$$

7. Bob decrypts $c_{s_{1a}}$ and $c_{s_{1b}}$ with his private key obtaining $p_{s_{1a}}$ and $p_{s_{1b}}$.

$$\begin{aligned} p_{s_{1a}} \leftarrow &\{c_{s_{1a}}\}_{B_{privKey}} \\ p_{s_{1b}} \leftarrow &\{c_{s_{1b}}\}_{B_{privKey}} \end{aligned} \tag{14}$$

The entire protocol is run a second time with Alice and Bob having swapped roles. This results in both Alice and Bob having $p_{s_{1a}}$, $p_{s_{1b}}$, $p_{s_{2a}}$, and $p_{s_{2b}}$; where each of those plaintext values are equivalent to the results of equation 4, $calcDet(j, k, l)$. Then using equation (1) they determine whether or not their line segment intersects with the other party's line segment.

The sign function, equation 11, is critical; without it Bob will have gained a second set of meaningful equations which will allow him to discover Alice's points using simple algebra.

4 Conclusions

Using geometry coupled with the Paillier cryptosystem we have designed and developed a protocol for securely determining whether two line segments intersect. Below we will discuss the security of the protocol, computational complexity, and communications overhead.

4.1 Results

The security properties of this algorithm rely upon the security properties of the Paillier cryptosystem; whose security is fundamentally built on the decisional composite residuosity assumption.

Regarding the computational complexity, from the perspective of Alice in section 3.3, six encryptions are required and two decryptions. Additionally, eight homomorphic multiplications, four homomorphic additions and four homomorphic subtractions are required. [2]

The communications overhead for a given test of line segment intersection involves each party sending eight messages, hence sixteen total messages are required to complete one instance of the protocol. Each message is an encrypted value, that will vary in size based on the size of n, the composite of two very large prime numbers (i.e. the size of the public key). Assuming n is 1024 bits then each protocol instance will be $1,024 * 2 * 8 = 16,384$ bits, or 2MB.

[2] Per [3] the Paillier cryptosystem is a fairly efficient encryption scheme. Encryption requires 2 exponentiations mod n^2, where n is the composite of two very large prime number, and decryption requires 1 exponentiation mod n^2.

4.2 Future Work

Additional work needs to be performed to handle edge cases. At least one edge case has been identified, collinear non-intersection, that causes our protocol to return incorrect results. Fortunately this edge case will produce a false positive, which in our case is better than being told that the two lines don't intersect when they really do. We have a potential solution which would require an increase in handshaking when the order of all points is determined to be collinear.

Another useful addition would be a threshold were the two parties not only determine whether or not their lines intersect, but also whether they are within some threshold distance from one another.

Another layer of encryption could be added in the future to authenticate the sender. The sender could sign each piece of data that they send with their private key, and the receiver could decrypt the data with the sender's public key in order to verify that the data was in fact sent by the other trusted party.

The protocol could be expanded so that two parties could securely determine polygon intersection.

And lastly, the protocol could be expanded in order to securely determine the intersection of line segments over time. The two parties could identify securely the points where they will be at specific times, the lines created by these points could be used to create the lines for intersection determination.

4.3 Resources

The source code, this paper, and a video demonstration of our work are all available: http://toadredcarp.github.io/secure-multiparty-line-segment-intersection/

5 Acknowledgements

We would like to thank Dr. Sriram Chellappan for his encouragement and support throughout this effort. Also, we would like to thank our families, as they endured many hours deprived of our company while we labored away with great enthusiasm.

References

[1] John D'Errico. Variable precision integer (vpi) arithmetic. http://www.mathworks.com/matlabcentral/fileexchange/22725-variable-precision-integer-arithmetic.

[2] Wenliang Du and Mikhail J. Atallah. Secure multi-party computation problems and their applications: A review and open problems. In *Proceedings of the 2001 Workshop on New Security Paradigms*, NSPW '01, pages 13–22, New York, NY, USA, 2001. ACM.

[3] Michael O'Keeffe. The paillier cryptosystem. 2008.

[4] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *IN ADVANCES IN CRYPTOLOGY — EUROCRYPT 1999*, pages 223–238. Springer-Verlag, 1999.

[5] R.L. Rivest, A. Shamir, and L.M. Adleman. Cryptographic communications system and method, September 20 1983. US Patent 4,405,829.

[6] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[7] thep. The homomorphic encryption project (thep). https://code.google.com/p/thep/.

[8] Alper Unger. Convex hull. https://www.cise.ufl.edu/class/cot5520fa13/Convexhull.pdf.

[9] Alper Unger. Line segment intersection. http://compgeom.cs.uiuc.edu/~jeffe/teaching/373/notes/x06-sweepline.pdf.

[10] Wikipedia. Paillier cryptosystem. http://en.wikipedia.org/wiki/Paillier_cryptosystem, March 2014.

[11] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.