Private Proximity Detection for Convex Polygons[*]

Bin Mu
The Graduate Center
City University of New York
bmu@gc.cuny.edu

Spiridon Bakiras
John Jay College
City University of New York
sbakiras@jjay.cuny.edu

ABSTRACT

Proximity detection is an emerging technology in Geo-Social Networks that notifies mobile users when they are in proximity. Nevertheless, users may be unwilling to participate in such applications if they are required to disclose their exact locations to a centralized server and/or their social friends. To this end, *private* proximity detection protocols allow any two parties to test for proximity while maintaining their locations secret. In particular, a private proximity detection query returns only a boolean result to the querier and, in addition, it guarantees that no party can derive any information regarding the other party's location. However, most of the existing protocols rely on simple grid decompositions of the space and assume that two users are in proximity when they are located inside the same grid cell. In this paper, we extend the notion of private proximity detection, and propose a novel approach that allows a mobile user to define an arbitrary convex polygon on the map and test whether his friends are located therein. Our solution employs a secure two-party computation protocol and is provably secure. We implemented our method on handheld devices and illustrate its efficiency in terms of both computational and communication cost.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*spatial databases and GIS*; K.4.1 [**Computers and Society**]: Public Policy Issues—*privacy*

General Terms

Algorithms

Keywords

Proximity detection, secure computations, location privacy

[*]This work was supported by the NSF CAREER Award IIS-0845262.

1. INTRODUCTION

The emergence of Geo-Social Networks (GeoSNs), such as Foursquare[1] and Loopt[2], facilitates the development of novel applications that combine social networking features with location based services. In particular, a GeoSN enhances the traditional social networking graph with spatial information, by allowing users to "check in" at arbitrary geographic locations. Mobile users may then utilize this information to identify their social *friends* that are spatially close (e.g., in order to meet at a nearby coffee shop). This is typically called a *proximity detection* query.

A trivial way to process such queries is to store all location information at the GeoSN server, in plaintext format. Alternatively, users may choose to bypass the server and exchange their locations (in plaintext), on-demand, in a peer-to-peer manner. Clearly, both methods might reveal a lot of information about an individual's lifestyle to the GeoSN server and/or his friends. If the leaked information is more than what the user is willing to disclose, he may be discouraged from registering with the GeoSN. Therefore, to protect privacy, GeoSN queries should not disclose any additional information regarding the location of a user, besides the information that can be derived from the query result.

To this end, *private* proximity detection protocols allow any two parties to test for proximity while maintaining their locations secret. In particular, a private proximity detection query returns only a boolean result to the querier and, in addition, it guarantees that no party can derive any information regarding the other party's location. Nevertheless, the current state-of-the-art private proximity detection protocols [15, 23] rely on simple grid decompositions of the space and assume that two users are in proximity when they are located inside the same grid cell.

However, this approach may not be sufficient in certain situations. Assume, for example, that Alice is enjoying her free time in the Central Park area of Manhattan. She wants to know whether Bob is also visiting the park (the area marked with the dashed line in Figure 1), in order to pursue some outdoor activities together. Traditional proximity detection queries are only able to discover Bob within a (roughly) circular area around Alice, as shown in Figure 1. A naïve solution to overcome this limitation is to enlarge the search range, so that it encloses the entire Central Park area. Clearly, this approach is not optimal, as it may lead to a large number of false positives. An alternative method is to leverage the existing space partitioning protocols, i.e.,

[1]`https://foursquare.com/`
[2]`https://www.loopt.com/`

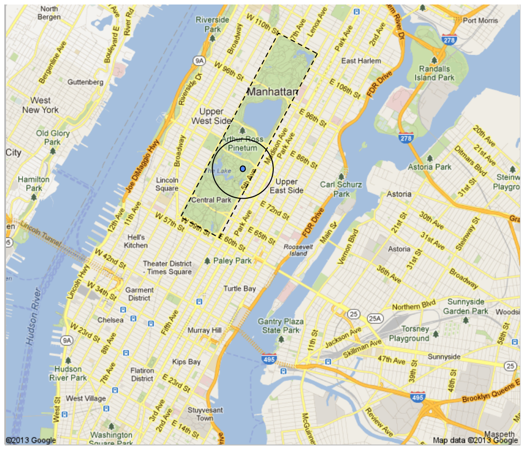


Figure 1: Motivating example

allow Alice to search for Bob in multiple grid cells, instead of just one. While this is a viable solution, it has two major shortcomings. First, to achieve an acceptable accuracy level, the underlying grid has to be very fine, thus leading to high query processing costs. Second, the query itself will reveal the size of Alice's search space, which is obviously a privacy breach.

To this end, this paper extends the notion of private proximity detection, and introduces a novel approach that allows a mobile user to define an arbitrary convex polygon on the map and test whether his friends are located therein. Returning to the example of Figure 1, Alice would simply specify the coordinates of the four rectangular edges in order to detect Bob's presence in Central Park. Our solution employs a secure two-party computation protocol that is based purely on public key homomorphic encryption operations. In addition, it offers the highest level of privacy to the involved parties, since (i) Alice only learns the query result and (ii) Bob only learns the number of edges in Alice's polygon. We implemented our method on handheld devices running iOS 6, and illustrate its efficiency in terms of both computational and communication cost.

In summary, the main contributions of our work are the following:

- We introduce a generalized proximity detection query that incorporates arbitrary convex polygons instead of fixed tessellations.
- We provide a secure and efficient solution based on public key homomorphic encryption.
- We show how to extend our basic method to handle certain instances of concave shapes.
- We implement our basic protocol on handheld devices and present real query processing times.

The rest of the paper is organized as follows. Section 2 describes the cryptographic primitives utilized in our methods and summarizes previous work on private proximity detection. Section 3 presents the formal definition of the new proximity detection query and describes the underlying threat model and security. Section 4 introduces the details of our solutions and Section 5 illustrates the results of our implementation on iOS devices. Finally, Section 6 concludes our work.

2. BACKGROUND

Section 2.1 introduces the cryptographic primitives utilized in our methods and Section 2.2 surveys the related work on private proximity detection.

2.1 Preliminaries

Homomorphic encryption. Most public key cryptosystems in the literature are partially homomorphic, i.e., they facilitate the evaluation of one algebraic operation (either addition or multiplication) directly on the ciphertext space. In our work, we utilize *additively* homomorphic encryption, which allows for the following computations. First, given the encryptions $E(m_1)$ and $E(m_2)$ of two plaintext messages m_1 and m_2, we can compute the encryption of $(m_1 + m_2)$ by multiplying the two ciphertexts:

$$E(m_1 + m_2) = E(m_1) \cdot E(m_2)$$

Second, any message m can be multiplied with a *plaintext* constant c as follows:

$$E(c \cdot m) = E(m)^c$$

In our implementation, we utilize two different additively homomorphic cryptosystems, namely the Paillier [16] and ElGamal [4] cryptosystems. The major advantage of Paillier's scheme (Figure 2) is that it can decrypt arbitrarily large plaintexts very efficiently. However, all operations are computed in modulo n^2 arithmetic, where n^2 is typically a 2048-bit number. As a result, the basic cryptographic operations, such as modular multiplication and exponentiation, are relatively expensive. On the other hand, the modulus size in ElGamal's scheme (Figure 3) is typically 1024 bits, so it is much more efficient in terms of computational cost. The limitation of the ElGamal cryptosystem is that it can only decrypt small plaintext values, because the decryption function involves the computation of a discrete logarithm, which is a very difficult problem in cryptography. Both schemes produce ciphertexts of size 256 bytes, so they are comparable in terms of communication cost.

Paillier cryptosystem

Key generation
1. Choose two large primes p and q of equal length, and compute the RSA modulus $n = pq$
2. The *public* key is n
3. The *private* key is $\varphi(n) = (p-1)(q-1)$

Encryption
1. Let m be the private message
2. Choose r uniformly at random from $\mathbb{Z}_n^*$
3. Compute ciphertext $c = (mn + 1)r^n \bmod n^2$

Decryption
1. Compute $m = \frac{(c^{\varphi(n)} \bmod n^2) - 1}{n} \cdot \varphi(n)^{-1} \bmod n$

Figure 2: The Paillier cryptosystem

Note that, both cryptosystems are semantically secure, i.e., it is infeasible to derive any information about a plaintext, given its ciphertext and the public key that was used

ElGamal cryptosystem

```
    Key generation
1.  Instantiate a cyclic group $G$ of prime order $p$, with
    generator $g$ ($G$, $g$, and $p$ are public knowledge)
2.  Choose a private key $x$, uniformly at random from $\mathbb{Z}_p^*$
3.  Publish the public key $h = g^x$

    Encryption
1.  Let $m$ be the private message
2.  Choose $r$, uniformly at random from $\mathbb{Z}_p^*$
3.  Compute ciphertext $(c_1, c_2) = (g^r, h^{r+m})$

    Decryption
1.  Compute $h^m = c_2 \cdot (c_1^x)^{-1}$
2.  Solve the discrete logarithm to retrieve $m$
```

Figure 3: The ElGamal cryptosystem

to encrypt it. The security of Paillier's scheme is based on the decisional composite residuosity assumption, while the security of ElGamal's scheme is based on the decisional Diffie-Hellman assumption.

Secure two-party computation. A secure two-party computation protocol [10] enables two parties, Alice and Bob, to jointly compute a function based on their respective inputs, without having to reveal their input to the other party. In other words, the two parties will only learn the result of the computation and nothing else. Yao's *garbled circuit* technique [21] is a generic two-party computation protocol that can evaluate securely any function f, given its Boolean circuit representation. In particular, Bob first generates an *encrypted* version of the circuit that incorporates his own input, and sends it to Alice. Then, Alice and Bob engage in a series of Oblivious Transfer (OT) [14] executions that allow Alice to retrieve securely the keys corresponding to her input bits. Finally, Alice evaluates the circuit and learns the result of the function. Even though the actual circuit evaluation can be very efficient [8], the large number of OT invocations is a performance bottleneck in terms of both computational and communication cost.

Consequently, researchers have looked into more specialized, i.e., application dependent, protocols that are typically built around homomorphic encryption. One such example is *private equality testing*, which is used extensively in previous work [15, 23]. In this protocol, Alice and Bob hold a pair of values a and b, respectively, and want to know if the two values are equal. Alice encrypts her input with her public key and sends $E(a)$ to Bob. Next, Bob utilizes the properties of homomorphic encryption to produce $E(r \cdot (a - b))$, where r is a random number that masks the result so that it is infeasible for Alice to derive any information regarding the value $(a-b)$. Finally, Alice decrypts the result and infers that $a = b$ if and only if the result is zero. The security of this protocol has been proven in [11, 15].

2.2 Related Work

Ruppel et al. [17] utilize a symmetric key cipher that encrypts locations by applying a distance-preserving transformation. A set of friends share a common key and use it to encrypt their location prior to uploading it to the server. Due to the distance-preserving property of the transformation, the server can determine whether any two friends are within a given proximity threshold. Clearly, this approach leaks some location information, as the server learns the actual distances among all users. Furthermore, if a user colludes with the server and reveals the shared key, all user locations are compromised. *Longitude* [12] is a similar approach, but the underlying transformation does not disclose the exact distances (i.e., it results in a loss of accuracy).

Most private proximity detection algorithms in the literature employ a tessellation method (typically a regular grid) to partition the space into a fixed number of cells. In this way, they reduce the proximity detection problem into an *equality testing* problem: identify whether the two parties are located inside the same or nearby cells. *FriendLocator* [19] and *VicinityLocator* [18] assume that the two parties share a secret key and use it to encrypt (with a deterministic symmetric cipher) the *ids* of certain cells nearby their location. The encrypted values are uploaded to server, who can determine (by matching the ciphertexts) whether the two users lie in the same or adjacent cells of the grid. Clearly, both schemes are vulnerable to collusions with the server, since the party that colludes can learn the approximate location of the other party.

In *C-Hide&Seek* [13], every user shares his secret key with his friends, and uses this key to encrypt his up-to-date location. When another user issues a proximity request, the server simply forwards all the encrypted locations that it currently stores. Therefore, the proximity detection is performed at the querier, which enables him to identify (with a simple brute force approach) the approximate locations of all his friends. *C-Hide&Hash* [13] assumes a similar location update procedure as *C-Hide&Seek*. However, for proximity detection, it employs a secure computation protocol between the server and the querier. Nevertheless, due to the shared keys among the users, this scheme is also vulnerable to collusions with the server.

Zhong et al. [23] propose three schemes, namely *Louis*, *Lester* and *Pierre* that are based on secure computations. The main idea in all protocols is to compute the distance between two parties using the properties of homomorphic encryption. First, *Louis* computes the actual distance, but requires a trusted third-party that only returns the result (i.e., true or false) of the proximity detection query. *Lester* does not require a third-party, but instead masks the actual distance d in a way that its computation time increases linearly with d (i.e., d is retrieved efficiently only when its value is relatively small). Finally, *Pierre* utilizes a regular grid to discretize the users' locations. It then employs a secure two-party computation protocol to determine whether the users lie within the same or adjacent cells in the grid. Narayanan et al. [15] partition the space with three overlapping tessellations, in order to improve the accuracy of the proximity detection. When two parties want to test for proximity, they employ a secure computation protocol (similar to *Pierre*) to identify whether they are located in the same cell of at least one tessellation.

Among all the aforementioned protocols, *Pierre* [23] and Narayanan et al. [15] provide the strongest privacy guarantees, i.e., the querier only learns the proximity result, while all remaining parties learn nothing. However, as mentioned in Section 1, these schemes are not directly applicable to our problem setting, because they cannot handle arbitrary polygonal shapes.

More similar to our work are protocols for the *secure point inclusion* problem. This problem was first introduced by Atallah and Du [1] as part of a collection of protocols for secure multiparty computational geometry. The two-party protocol for the secure point inclusion problem leverages a number of basic sub-protocols (such as scalar product and vector dominance) that are also proposed in [1]. Nevertheless, these protocols are computationally expensive and lack formal security proofs. Thomas [20] solves the secure point inclusion problem for star-shaped polygons, while Yun et al. [22] address convex polygons. However, these methods are not secure in our problem setting, because the proximity result is computed by the party that owns the fixed point, and has to be transmitted to the querier in an additional round. This property enables collusions among the participating entities that may leak location information regarding the querier's polygon. On the other hand, our method avoids such collusions, because the proximity result is computed at the querier.

3. PROBLEM DEFINITION

Alice (the querier) holds a convex polygon P consisting of N vertices $p_0, p_1, \dots p_{N-1}$. The vertices are labeled in a counterclockwise order, as shown in Figure 4. The coordinates of vertex p_i are denoted as (p_i^x, p_i^y). Bob holds a single point $q = (q^x, q^y)$ that represents his current location. Alice wants to know whether Bob is located inside, or on the boundary of, P. The privacy guarantees provided by our protocol are the following:

- Alice learns only the result of the proximity detection query (a boolean value). Bob's exact location remains secret.
- Bob does not learn the query result. Furthermore, the only information he can derive about Alice's polygon is the number of edges N. The location, shape, and size of P remain secret.

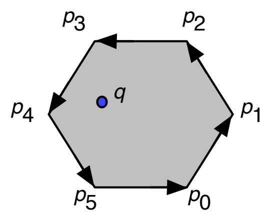


Figure 4: Proximity detection example

We assume that both parties can be the adversaries in this protocol. Alice's goal is to pinpoint Bob in an area smaller than the one that can be inferred from the outcome of the protocol. On the other hand, Bob wants to deduce any additional information regarding Alice's polygon, besides the number of edges. Finally, we assume that both parties run in polynomial time and are "semi-honest," i.e., they will follow the protocol correctly, but will try to gain any advantage by analyzing the information exchanged during the protocol execution.

Note that Bob may have his own privacy requirement, namely that he does not want to be found within an area smaller than a certain threshold. Unfortunately, this requirement cannot be enforced in our current protocol. One direction that we plan to investigate in the future is to use cryptographic commitment schemes [7] that will allow Alice to commit to her input prior to the protocol execution. Then, if the proximity result is true, Alice will be forced to reveal her polygon to Bob, in order for him to verify that his privacy was not violated. Furthermore, it is possible for Alice to locate Bob with a brute-force attack, i.e., she can initiate a sequence of proximity detection queries that cover the entire space where Bob might be located. A straightforward solution here is for Bob to decline successive queries that are not sufficiently apart in time.

4. PRIVATE PROXIMITY DETECTION

Section 4.1 presents a geometric, i.e., insecure, algorithm to our problem and Section 4.2 introduces a secure two-party computation protocol that implements this algorithm. Section 4.3 describes a secure comparison protocol that we utilize in our method and Section 4.4 outlines a security sketch of our protocol. Finally, Section 4.5 describes a solution that can handle certain instances of concave polygons.

4.1 Geometric Solution

Our proximity detection query can be easily solved by performing N point orientation computations [3]. Specifically, given an ordered triple of points $\langle p_i, q, p_{i+1} \rangle$ in the plane, we say that they have (Figure 5)

- *Positive* orientation if their angle is a counterclockwise turn
- *Negative* orientation if their angle is a clockwise turn
- *Zero* orientation if they are collinear

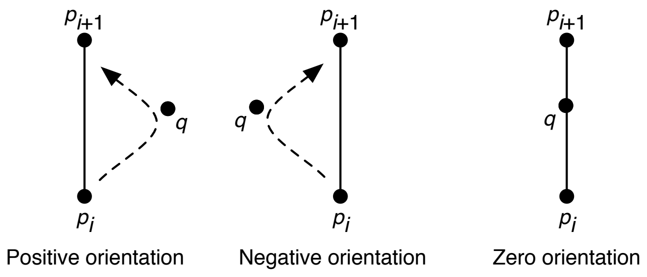


Figure 5: Point orientation

Given the coordinates of the three points, the orientation is computed by the sign of the following determinant:

$$\theta_i = \begin{vmatrix} 1 & p_i^x & p_i^y \\ 1 & q^x & q^y \\ 1 & p_{i+1}^x & p_{i+1}^y \end{vmatrix} = q^x(p_{i+1}^y - p_i^y) + q^y(p_i^x - p_{i+1}^x) + (p_i^y p_{i+1}^x - p_i^x p_{i+1}^y) \quad (1)$$

Therefore, the following algorithm computes the correct proximity result:

1. For $i \in \{0, 1, \dots N-1\}$ and $j = (i+1) \bmod N$, compute the orientation θ_i of point q with respect to the line segment $\overline{p_i p_j}$. The vertices are visited in a counterclockwise order, as shown in Figure 4.

2. If $\theta_i \leq 0 \; \forall i \in \{0, 1, \ldots N-1\}$, return *true*; otherwise, return *false*.

The correctness of this algorithm (for convex polygons) follows from the point orientation property. That is, the orientation result determines the half-plane where point q is located if you infinitely extend the line segment $\overline{p_i p_j}$ in both directions. In other words, q lies in the intersection of the N half-planes. When all point orientations are negative, this area is equal to the convex polygon.

4.2 Secure Protocol

We assume that, prior to the protocol execution, Alice has published her public keys for the Paillier and ElGamal cryptosystems. In what follows, we use $E_P(\cdot)$ to denote encryption under Paillier's scheme and $E_G(\cdot)$ to denote encryption under ElGamal's scheme. The protocol consists of three major steps. First, Bob computes the encryptions of all θ_i's under Alice's public key. Next, Alice and Bob engage in a series of secure comparison protocols that allow Bob to compute the encryptions of the signs for all θ_i's. Finally, Bob merges these results into a single message that he sends to Alice. The detailed protocol is shown in Figure 6.

PPD_Convex

Input: Alice has polygon P with N vertices $p_0, p_1, \ldots, p_{N-1}$
Bob has point q
Output: **true** if q in P, **false** otherwise

1. For $i \in \{0, 1, \ldots N-1\}$ and $j = (i+1) \bmod N$, Alice sends to Bob $E_P(p_j^y - p_i^y)$, $E_P(p_i^x - p_j^x)$, and $E_P(p_i^y p_j^x - p_i^x p_j^y)$
2. For $i \in \{0, 1, \ldots N-1\}$ and $j = (i+1) \bmod N$, Bob computes $E_P(\theta_i) = E_P(p_j^y - p_i^y)^{q^x} \cdot E_P(p_i^x - p_j^x)^{q^y} \cdot E_P(p_i^y p_j^x - p_i^x p_j^y) \cdot E_P(-1)$
3. Alice and Bob engage in a series of secure comparison protocols and Bob computes, $\forall i \in \{0, 1, \ldots, N-1\}$, $E_P(\sigma_i)$, where $\sigma_i > 0$ if $\theta_i \geq 0$ and $\sigma_i = 0$ if $\theta_i < 0$
4. Bob chooses r uniformly at random from $\mathbb{Z}_n^*$, where n is the RSA modulus of Alice's public key
5. Bob computes the masked result $E_P(r \cdot \sigma) = [\prod_{i=0}^{N-1} E_P(\sigma_i)]^r$ and sends it to Alice
6. Alice decrypts $E_P(r \cdot \sigma)$ with her private key and, if $r \cdot \sigma = 0$, she returns **true**; otherwise, she returns **false**

Figure 6: The private proximity detection protocol

Initially, Alice uses her Paillier public key to encrypt (for each edge) her own input, as dictated by Equation (1). She then sends a total of $3N$ ciphertexts to Bob (Step 1). Bob cannot decrypt any of these ciphertexts because he does not have Alice's public key. However, he is able to incorporate his own input, using the properties of additively homomorphic encryption (Step 2). Note that, the final result in Step 2 is not the encryption of θ_i, as it is shown in Equation (1). The last term adds the value -1 to the result, in order to produce the encryption of $(\theta_i - 1)$. The reason behind this approach is to enforce the points that lie on a polygon edge to produce a negative orientation result, instead of zero.

Next, Alice and Bob employ a secure comparison protocol (which is introduced in the next section) to compute an encrypted representation of the sign of each θ_i (Step 3). Specifically, the protocol allows Bob to compute the encryption of σ_i, where $\sigma_i = 0$ if and only if $\theta_i < 0$ (otherwise it has a fixed value $t > 0$). During the protocol execution, no party can derive any information regarding the actual value or the sign of θ_i. Bob then combines all orientation results into one ciphertext, i.e., $E_P(\sigma) = E_P(\sigma_0 + \sigma_1 + \ldots + \sigma_{N-1})$. However, he cannot send this value to Alice because, if $\sigma > 0$, Alice can figure out the number of edges that produced a positive orientation and, thus, she can eliminate some areas from the search space. Therefore, Bob multiplicatively masks the aggregate result (Steps 3–5), by computing $E_P(r \cdot \sigma)$. Finally, Bob sends the masked result to Alice who decrypts it with her private key (Step 6). If the decrypted value is zero, q is located inside (or on the boundary of) P. Otherwise, it is impossible for Alice to determine how many point orientations were positive.

4.3 Secure Comparison Protocol

In Step 3 of the PPD_Convex protocol, Bob holds the encryptions of all point orientation results and wants to compute the encryptions of their corresponding signs. For this task, we borrow a secure comparison protocol (Figure 7) that is introduced by Erkin et al. [5] as part of their privacy preserving face recognition protocol.

Sec_Comp

Input: Bob has $E_P(\theta_i)$ under Alice's public key
ℓ is the max bit-size of θ_i
Output: Bob computes $E_P(\sigma_i)$ where $\sigma_i > 0$ if $\theta_i \geq 0$ and $\sigma_i = 0$ if $\theta_i < 0$

1. Bob computes $E_P(s) = E_P(\theta_i + 2^\ell) = E_P(\theta_i) \cdot E_P(2^\ell)$
2. Bob generates a uniformly random $(k + \ell + 1)$-bit number r (where $k = 100$), computes $E_P(s + r) = E_P(s) \cdot E_P(r)$, and sends it to Alice
3. Alice decrypts the message, computes $a = (s + r) \bmod 2^\ell$, and sends $E_P(a)$ to Bob
4. For $i \in \{0, 1, \ldots, \ell-1\}$, Alice sends to Bob $E_G(a_i)$, i.e., the ElGamal encryption of the i-th bit of a_i
5. Bob computes $b = r \bmod 2^\ell$
6. For $i \in \{1, 2, \ldots, \ell-1\}$, Bob sets $E_G(w_i)$ equal to $E_G(a_i)$ if $b_i = 0$, or $E_G(1) \cdot E_G(a_i)^{-1}$ if $b_i = 1$ (the i-th bit of b)
7. Bob chooses $d \in \{1, -1\}$ and, for $i \in \{0, 1, \ldots, \ell-1\}$, he computes $E_G(c_i) = E_G(a_i) \cdot E_G(b_i)^{-1} \cdot E_G(d) \cdot [\prod_{j=i+1}^{\ell-1} E_G(w_j)]^3$
8. For $i \in \{0, 1, \ldots, \ell-1\}$, Bob chooses v_i uniformly at random from $\mathbb{Z}_p^*$, computes $E_G(v_i \cdot c_i) = E_G(c_i)^{v_i}$, and sends a permuted version of the results to Alice
9. Alice decrypts all messages and, if one of them is zero, she sets $\delta = 1$; otherwise $\delta = 0$. She sends $E_P(\delta)$ to Bob
10. If $d = -1$, Bob sets $E_P(\delta) = E(1) \cdot E_P(\delta)^{-1}$
11. Bob computes $E_P(\sigma_i) = E_P(s) \cdot [E_P(a) \cdot E_P(b)^{-1} \cdot E_P(\delta)^{2^\ell}]^{-1}$

Figure 7: The secure comparison protocol

Suppose we use ℓ-bit numbers to represent the orientation. In Step 1, Bob computes the encryption of $s = (\theta_i + 2^\ell)$, which is an $(\ell + 1)$-bit number whose most significant bit (MSB) determines the sign of θ_i: if it is 1, $\theta_i \geq 0$; otherwise, $\theta_i < 0$. The value of the MSB can be inferred from the result of $[s - (s \bmod 2^\ell)]$, which is 0 if MSB $= 0$ and 2^ℓ if MSB $= 1$. Therefore, Alice and Bob engage in a series of steps to securely compute $(s \bmod 2^\ell)$. Initially (Step 2), Bob generates a uniformly random number r in order to additively mask s, i.e., he sends the encryption of $(s + r)$ to Alice. Alice decrypts the message, reduces it modulo 2^ℓ, and sends the encryption of $a = [(s + r) \bmod 2^\ell]$ back to Bob (Step 3).

Bob now needs to subtract $b = (r \bmod 2^\ell)$ from a, in order to compute $(s \bmod 2^\ell)$. However, this is not sufficient, as the subtraction may cause the result to underflow when $a < b$. Instead, the correct approach is to securely compute the outcome of the above comparison ($\delta = 1$ if true, $\delta = 0$ if false) and then compute the desired result:

$$(s \bmod 2^\ell) = a - b + \delta \cdot 2^\ell$$

Consequently, we are left with an instance of Yao's millionaire problem, i.e., we need to determine whether a (held by Alice) is smaller than b (held by Bob). Both inputs are ℓ-bit numbers, so we use index $i \in \{0, 1, \ldots, \ell - 1\}$ to represent the individual bits. First (Step 4), Alice sends to Bob the encryptions of all her bits a_i. Note that, at this point, Alice switches to the more computationally efficient ElGamal scheme. Bob then chooses a random value d (either 1 or -1) and computes the following encryptions (Steps 6–7), where $w_j = a_j \oplus b_j$:

$$c_i = a_i - b_i + d + 3 \sum_{j=i+1}^{\ell-1} w_j$$

Suppose that $d = 1$. If $a \geq b$, then all c_i's will be non-zero. On the other hand, if $a < b$, then exactly one c_i will be zero (at the most significant bit position where the corresponding bits differ). If $d = -1$ the situation is identical, except that the zero value occurs when $a \geq b$. Next, Bob multiplicatively masks the individual c_i's by raising them to a random power v_i. He also permutes the encryptions and sends them back to Alice (Step 8).

Alice decrypts all the c_i's and checks whether one of them is zero. If this is the case, she sets $\delta = 1$; otherwise she sets $\delta = 0$. Note that, Alice does not know the value of d that Bob has selected, so she cannot determine whether an underflow has occurred. Finally, she switches back to Paillier's cryptosystem and sends the encryption of δ to Bob (Step 9). If $d = -1$, Bob adjusts the value of δ (Step 10) and eventually computes (Step 11) the encryption of

$$\sigma_i = s - (a - b + \delta \cdot 2^\ell)$$

where $\sigma_i = 0$, if $\theta_i < 0$ and $\sigma_i = 2^\ell$, if $\theta_i \geq 0$.

4.4 Security

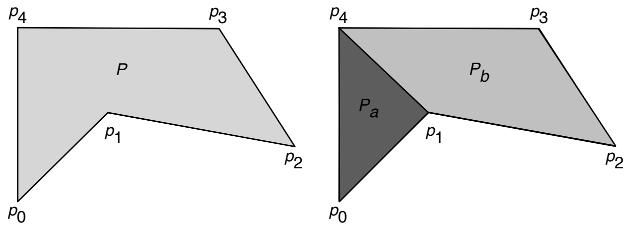
We will prove the security of the PPD_Convex protocol (which also includes the Sec_Comp protocol) for semi-honest adversaries, following the simulation paradigm [10]. In particular, we need to show that, for each party, we can simulate the distribution of messages that the party receives, given only the party's input and output in this protocol. This is true because, if we can simulate each party's view from only their respective input and output, the messages themselves reveal no additional information.

First, Alice's input consists of N vertices and her output is $r \cdot \sigma$. In Step 2 of Sec_Comp, Alice receives the encryption of a uniformly random number from Bob. The simulator knows Alice's public key, so it can simply generate the encryption of a random $(k + \ell + 1)$-bit number. Furthermore, in Step 8 of Sec_Comp, Alice receives ℓ numbers that are either all random or there is a single one with value zero. The simulator knows how the protocol works, so it can either generate ℓ random encryptions, or $(\ell - 1)$ random encryptions plus an encryption of zero. Finally, in Step 5 of PPD_Convex, Alice receives the encrypted result from Bob. The simulator knows Alice's output and can, thus, produce the corresponding ciphertext (an encryption of either zero or a random number).

In Bob's case, the input is a point q and there is no output. In Step 2 of PPD_Convex, Bob receives $3N$ encryptions from Alice. Here, the simulator can simply generate $3N$ encryptions of zero. Given the assumption that the underlying encryption scheme is semantically secure, Bob cannot distinguish these ciphertexts from the ones that are produced by Alice's real input. Similarly, Bob receives a number of encryptions in Steps 3, 4, and 9 of Sec_Comp. This is also simulated by multiple encryptions of zero.

4.5 Handling Concave Polygons

In certain cases, the querier may need to define a concave polygon, in order to better approximate the underlying proximity region. The PPD_Convex protocol of Figure 6 is not applicable in this scenario, as it may produce some false negatives. Nevertheless, using well-known algorithms for the optimal convex decomposition problem [9] in computational geometry, we can decompose any concave shape into the minimum number of convex polygons. An example is shown in Figure 8, where P is decomposed into convex polygons P_a and P_b.



(a) Concave polygon (b) Convex decomposition

Figure 8: Optimal convex decomposition

A straightforward algorithm to evaluate private proximity detection queries would then be to invoke PPD_Convex multiple times, and have Bob return all results (permuted) back to Alice. Alice would then decrypt the results and infer that Bob lies within P if and only if there is a zero value among the plaintexts. Furthermore, the permutation prevents Alice from identifying the exact polygon where Bob is located. While this approach would work in some cases, it is not secure if Bob lies on an edge that is shared between two convex polygons. In the example of Figure 8, if Alice decrypts two zero values she can be certain that Bob lies somewhere along the line segment $\overline{p_1p_4}$.

This privacy breach is due to the fact that Alice receives the actual proximity result for each convex polygon. Therefore, if Bob were able to multiply the underlying plaintexts and return a single result to Alice, the protocol would be secure. Fully homomorphic encryption [6] can accomplish that, because it allows both addition and multiplication operations on the underlying plaintexts. Nevertheless, fully homomorphic encryption is extremely expensive in terms of computational and communication cost and, thus, it is not practical for a real implementation. What we propose instead, is a solution based on the BGN cryptosystem [2] that is additively homomorphic, but allows a single multiplication between two plaintexts. This property enables us to develop a secure protocol for concave shapes that are decomposable into exactly two convex polygons.

The resulting protocol is illustrated in Figure 9, where $E_B(\cdot)$ denotes encryption with the BGN cryptosystem (Alice is again the owner of the corresponding private key). The key idea is for Bob to obtain the encryptions of σ_a and σ_b under the BGN cryptosystem (Steps 1–5), in order to multiply them and compute the encryption of the result $(\sigma_a \cdot \sigma_b)$. An important remark here concerns the utilization of both Paillier and BGN cryptosystems. One could argue that Steps 2–5 can be avoided if we replace Paillier's scheme in the PPD_Convex algorithm with BGN. The reason is that, similar to ElGamal's scheme, BGN's decryption function is based on discrete logarithms, so it cannot decrypt arbitrarily large plaintexts. Furthermore, the BGN cryptosystem is based on elliptic curve groups of composite order and is, thus, significantly more expensive than Paillier's scheme. Note that we did not implement the PPD_Concave protocol on our handheld devices, but this is something we plan to do in the future.

PPD_Concave

Input: Alice has two polygons P_a and P_b
Bob has point q
Output: **true** if q in $P_a \cup P_b$, **false** otherwise

1. Alice and Bob invoke PPD_Convex twice (for P_a and P_b), and Bob computes the results $E_P(\sigma_a)$ and $E_P(\sigma_b)$
2. Bob chooses r_a, r_b uniformly at random from $\mathbb{Z}_m$, where m is a large (e.g., 200-bit) number
3. Bob computes the masked results $E_P(r_a + \sigma_a)$ and $E_P(r_b + \sigma_b)$, and sends them to Alice
4. Alice decrypts the ciphertexts, re-encrypts them with BGN, and sends $E_B(r_a + \sigma_a)$, $E_B(r_b + \sigma_b)$ to Bob
5. Bob uses the additive property of BGN to compute $E_B(\sigma_a)$ and $E_B(\sigma_b)$
6. Bob uses the multiplicative property of BGN to compute $E_B(\sigma) = E_B(\sigma_a \cdot \sigma_b)$
7. Bob chooses random r and uses the additive property of BGN to compute $E_B(r \cdot \sigma)$, which he sends to Alice
8. Alice decrypts $E_B(r \cdot \sigma)$ with her private key and, if $r \cdot \sigma = 0$, she returns **true**; otherwise, she returns **false**

Figure 9: The private proximity detection protocol for concave polygons

5. IMPLEMENTATION RESULTS

In this section, we present our results from an actual implementation of the PPD_Convex protocol on iOS 6 devices. The implementation of the cryptographic primitives is written in C, and leverages the GMP[3] multiple precision arithmetic library and the OpenSSL[4] cryptographic library. In particular, we cross compiled both libraries for the ARM architecture and incorporated them in our app. We deployed the app on two devices (an iPhone 5 and a 3rd generation iPad) and connected the devices over a WiFi network using a secure SSL connection.

Before running the actual protocol, we created a benchmark program to test the performance of the two homomorphic encryption schemes. Specifically, we deployed the benchmark app on the iPhone 5 device and measured the cost of the basic cryptographic operations. The results are shown in Table 1. The two modular exponentiation entries correspond to the size of the exponent, which is the deciding factor for the cost of this operation. Small exponents are involved when a party multiplies its plaintext input into an existing ciphertext, e.g., as Bob does in Step 2 of protocol PPD_Convex. Large exponents are normally necessary during a multiplicative masking operation, such as the one in Step 5 of PPD_Convex. The advantage of ElGamal's scheme is very clear in this table (as explained in Section 2.1), and justifies the usage of two different cryptosystems in the same protocol.

Table 1: Cost of cryptographic primitives

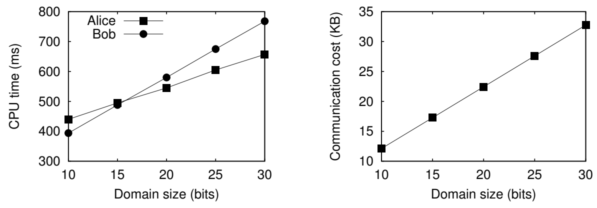
Paillier cryptosystem	
Encryption	40.7 ms
Decryption	40.6 ms
Modular exponentiation (small exponent)	1.3 ms
Modular exponentiation (large exponent)	39.9 ms
Modular multiplication	0.9 μs
ElGamal cryptosystem	
Encryption	3.6 ms
Decryption	1.8 ms
Modular exponentiation (small exponent)	0.7 ms
Modular exponentiation (large exponent)	3.6 ms
Modular multiplication	0.7 μs

Next, we investigate the impact of the domain size (i.e., the number of bits required to store one coordinate) on the performance of a single point orientation computation. Figure 10 illustrates the CPU time required at the two parties, as well as the overall communication cost. Both costs grow linearly with the domain size, because the bit-size ℓ of the orientation result increases. This, in turn, increases the cost of the Sec_Comp protocol that has a linear complexity with respect to ℓ. Nevertheless, the CPU time is affected less than the communication cost, due to the dominant effect of the Paillier operations that are not influenced by the domain size. Also, Bob's CPU time increases more sharply than Alice's, because Bob needs to perform a lot of public key operations in Steps 6 and 7 of the Sec_Comp protocol.

Figure 11 shows the CPU time and the communication cost as a function of the number of edges N in the proximity region (for a domain size of 20 bits). Clearly, both costs scale

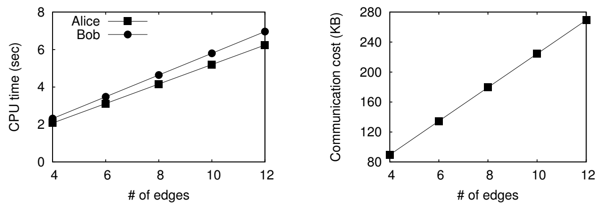
[3]`http://gmplib.org/`
[4]`http://www.openssl.org/`

linearly with N, since the proximity detection query involves exactly N point orientation computations. We expect that, in a real application, a rectangular region would probably be the most common query type. In this case, the query could be answered in around 5 sec and incur a communication cost of 90 KB. We believe that this is an acceptable cost for privacy preserving query processing on handheld devices.



(a) CPU time (b) Communication cost

Figure 10: Cost vs. domain size (for a single edge)



(a) CPU time (b) Communication cost

Figure 11: Cost vs. number of polygon edges

6. CONCLUSIONS

Traditional private proximity detection protocols are very restrictive in the definition of the proximity region. In particular, they typically constrain users to select (at most) a few cells from a fixed grid decomposition of the space. To this end, this paper extends the notion of private proximity detection, by allowing users to define regions of arbitrary convex shapes. We propose a novel solution based on a secure two-party computation protocol that is provably secure. By slightly modifying our basic protocol, we show that it is also possible to handle certain instances of concave polygons. Finally, we implement our basic method on handheld devices and illustrate its applicability in a real-life application.

7. REFERENCES

[1] M. J. Atallah and W. Du. Secure multi-party computational geometry. In *WADS*, pages 165–179, 2001.

[2] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, pages 325–341, 2005.

[3] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 3rd edition, 2008.

[4] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[5] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *PETS*, pages 235–253, 2009.

[6] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[7] O. Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools.* Cambridge University Press, 2001.

[8] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.

[9] J. M. Keil. Decomposing a polygon into simpler components. *SIAM Journal of Computing*, 14(4):799–817, 1985.

[10] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):59–98, 2009.

[11] H. Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *ASIACRYPT*, pages 416–433, 2003.

[12] S. Mascetti, C. Bettini, and D. Freni. Longitude: Centralized privacy-preserving computation of users' proximity. In *Secure Data Management (SDM)*, pages 142–157, 2009.

[13] S. Mascetti, D. Freni, C. Bettini, X. S. Wang, and S. Jajodia. Privacy in geo-social networks: Proximity notification with untrusted service providers and curious buddies. *VLDB Journal*, 20(4):541–566, 2011.

[14] M. Naor and B. Pinkas. Computationally secure oblivious transfer. *Journal of Cryptology*, 18(1):1–35, 2005.

[15] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *NDSS*, 2011.

[16] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[17] P. Ruppel, G. Treu, A. Küpper, and C. Linnhoff-Popien. Anonymous user tracking for location-based community services. In *LoCA*, pages 116–133, 2006.

[18] L. Siksnys, J. R. Thomsen, S. Saltenis, and M. L. Yiu. Private and flexible proximity detection in mobile social networks. In *MDM*, pages 75–84, 2010.

[19] L. Siksnys, J. R. Thomsen, S. Saltenis, M. L. Yiu, and O. Andersen. A location privacy aware friend locator. In *SSTD*, pages 405–410, 2009.

[20] T. Thomas. Secure two-party protocols for point inclusion problem. *International Journal of Network Security*, 9(1):1–7, 2009.

[21] A. C.-C. Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167, 1986.

[22] Y. Ye, L. Huang, W. Yang, and Y. Zhu. Efficient protocols for point-convex hull inclusion decision problems. *Journal of Networks*, 5(5):559–567, 2010.

[23] G. Zhong, I. Goldberg, and U. Hengartner. Louis, Lester and Pierre: Three protocols for location privacy. In *PETS*, pages 62–76, 2007.