Near-Pri: Private, Proximity Based Location Sharing

Ed Novak, Qun Li
{*ejnovak, liqun*}@*cs.wm.edu*
Department of Computer Science
College of William and Mary

***Abstract*—As the ubiquity of smartphones increases we see an increase in the popularity of location based services. Specifically, online social networks provide services such as alerting the user of friend co-location, and finding a user's k nearest neighbors. Location information is sensitive, which makes privacy a strong concern for location based systems like these. We have built one such service that allows two parties to share location information privately and securely. Our system allows every user to maintain and enforce their own policy. When one party, (Alice), queries the location of another party, (Bob), our system uses homomorphic encryption to test if Alice is within Bob's policy. If she is, Bob's location is shared with Alice only. If she is not, no user location information is shared with anyone. Due to the importance and sensitivity of location information, and the easily deployable design of our system, we offer a useful, practical, and important system to users. Our main contribution is a flexible, practical protocol for private proximity testing, a useful and efficient technique for representing location values, and a working implementation of the system we design in this paper. It is implemented as an Android application with the Facebook online social network used for communication between users.**

I. Introduction

Online social networks (OSNs) have been growing rapidly in size and popularity since their inception around 2004. The most popular online social network, Facebook, now has approximately 1.1 billion registered, active users and has an Alexa rating of two, [1], [2]. As online social networking, aided by smart-phone catalysts, becomes more ubiquitous, we are seeing an equal increase in the use of user location information in a social context. The dominant OSNs all have location elements and sites like Foursquare, Jiepang, and FullCircle are referred to as location-based online social networks (LB-OSNs), due to their focus on location. These providers can offer useful services based on user location such as cab hailing, directions, nearby friend alerts, fair rendezvous points between friends [3], and simple location sharing.

As these online social networking and location based services grow in popularity, important questions about privacy are raised. Location privacy is nuanced and valuable because it can easily be used to derive other information about a person. For example, it is trivial, given a location trace, to determine the home of the user (the address they are at most frequently between 12:00am and 7:00am), or their place of work (similarly) [4]. Inferring information from a location trace can reveal other, sensitive information when more sophisticated analysis is employed [5], [6].

Using this information, adversaries can perform a variety of attacks. These range from mild inconveniences, such as dropping by for lunch unexpectedly and targeted advertisements, to a variety of much more serious attacks. An adversary can use location information to more easily steal a user's identity, determine the answer to common security questions, and even stalk the user. Insurance companies can use location information as leverage to raise premiums. It can even be used by governments to enforce strict, totalitarian-style laws.

Currently, it is common for location based services to require the user to upload their location information in plain text. They use this location information to perform some computation, which is vital to their service. This means, however, that service providers have unfettered access to user location data. Furthermore, these providers are weakly motivated to protect this information from the world at large, as has been demonstrated by the recent NSA privacy revelations [7]. Users should be wary of providing their location information to any third parties. Because of this situation, systems that offer location based services that protect users' location information are very valuable.

Our paper aims to offer one such location based service to users in a convenient, privacy preserving, and secure way. Specifically, we build a system which allows a user, (Alice), to query the location of her friend, (Bob), in an online social network. Our system, without exposing location information to either party, or the OSN provider, allows Bob to test if Alice is within X meters of him (private proximity detection). We refer to this X as Bob's *policy* or P_b. If Alice and Bob are not within P_b meters of one another, both parties learn only this fact. If they are within P_b then Bob's location is revealed to Alice. Our system allows for every user to maintain their own policy. By requiring that users are near Bob, we protect Bob's location information from his friends, the service providers, and any other, possibly malicious, third parties. When the GPS coordinates are transmitted from Bob to Alice, asymmetric encryption is used to hide the values.

In this paper we make several contributions. We present a protocol that allows one party to determine, privately and securely, if a second party is within a certain distance and we use our protocol to enforce distance based access control in location queries. Our protocol is flexible in that each user is able to maintain their own policy and can even assign different policies for different queriers. To the best of the authors' knowledge, this is the first protocol of its kind. Our system does not rely on any dedicated, trusted third party server to

978-1-4799-3360-0/14/$31.00

perform any computation. We overcome the challenge of name resolution, for Alice and Bob to send messages to one another, by utilizing the Facebook messaging service and building our system in a distributed way.

We present a novel and elegant method of discretizing GPS coordinates for use in our system. We utilize a binary tree data structure and we support granularity as small as 10m. This presents the challenge that the tree is very large, as each leaf represents a 10m section of the Earth. We overcome this challenge by building only the necessary segment of the tree, and building from the leaf level up. By doing this, we are able to significantly decrease the running time of our system, decreasing the amount of data sent between Alice and Bob, and reducing computation.

We offer a working implementation, which is the first and only, to the author's knowledge, that enforces proximity based content access control (through private proximity detection). It does not require modification of the infrastructure and does not leak information to any third parties, despite its use of Facebook as a message channel. Queries are processed in tens of seconds and the system scales well as the user's policy increases.

In order to overcome the challenge of making our system practical and useful for users we perform an extensive evaluation, which shows query times, data usage, and scalability analysis. Our system can process a single query in roughly 30 seconds. We also consider the case where Alice wants to be notified when any of her hundreds of Facebook friends become near. We make key improvements, including pre-processing, to make sure our system is scalable in this way.

II. Related Work

Several papers propose methods for making location information private that can be applied to any location based service ([8]–[11]). These papers obfuscate location and temporal information, usually based on the k-anonymous measure. Although all of these systems intend for location based services to still function effectively, they all gradually erode service quality. As the user's location becomes more and more vague, location based services become less and less useful. For example, if a user's location data has been obfuscated to the granularity of 400m, a cab hailing service becomes useless. There is even literature that offers a framework for choosing the best location privacy system for the task at hand, [12], [13]. However, even after choosing the best system for preserving privacy, the location based service may not function correctly or adequately. The idea of a one-size-fits-all solution to location privacy is, at the least, controversial, [14].

Obfuscation, even k-anonymous obfuscation, does not provide adequate privacy when applied to location information, because attackers can still perform trajectory attacks. Furthermore, k-anonymity does not protect users when they are in a densely populated area. While it is true the attacker cannot distinguish the user from the many other users near her, the attacker is actually given higher accuracy results.

In order to achieve location privacy while still maintaining the proper operation of location based services, we must take each service on a case by case basis. Privacy and security must be built into the design. This is the goal of [15], [16], and [3]. Each of these papers attempt to design one specific location-based service in a privacy preserving way. Mobishare, [15], allows users to share their (authenticated) location values with one another, without allowing any third parties to link user (OSN profile) with location. However, it requires significant changes to the infrastructure including cellular towers. Our protocol avoids any changes to the network infrastructure.

There are several papers that focus on a service similar to ours [17]–[21]. All of these papers offer the theory behind private proximity testing. Our system closely relies on Nergiz's et. al., work [22] but makes a simple improvement which allows our protocol run in a much lower time. The main contribution of our work is a pragmatic, working implementation of the theory involved in these works. We provide a thorough evaluation, including end-to-end measurements to ascertain the system speed. The technical challenges here are the numerous engineering problems and security concerns. Additionally, we show an extension of our system that allows users to be alerted as soon as their friends are nearby efficiently.

Other encryption systems, such as order preserving encryption, may be used to build a system similar to the one described in this work [23], [24]. We leave this avenue open as possible future work.

III. Problem Description

Our system allows users to share location information privately and securely. When one user, Alice, requests the location of another user, Bob, the location is disclosed if Alice is within a certain range of Bob, P_b. Every user in the system defines their own policy, P_b. Determining if Alice is within P_b, without revealing the location of either user to the other, is known as private proximity detection.

Each user has a current *location*; a 2-tuple of that user's GPS coordinates $< lat, lon >$. Where $lat \in \mathbb{R}$ in the range $[-90, 90]$, and $lon \in \mathbb{R}$ in the range $[-180, 180]$. Each user also maintains a policy, P_b, which is a factor of ten meters in the range $[30m, 20000km]$, ($20000km \approx \frac{1}{2}$ the Earth's circumference). The goal is to determine if Alice's *location* is in the circle centered at Bob's location, with radius P_b. Our protocol allows Bob to test if Alice is near him, without revealing his *location* to Alice and without Alice revealing her *location* to him.

In our implementation, we separate the Earth's surface based on a grid formed by longitude and latitude lines. A small spherical cap of this grid is depicted in Fig. 1. We approximate the circle, centered at Bob, with a subset of these grid squares. This allows us to significantly simplify the private proximity detection protocol. Each square's edges are five meters from the center in the cardinal directions. Due to the curvature of the Earth, they are slightly distorted. The corners of the squares introduce a small margin of error, because they are slightly larger than the would-be circle. However, this

error is negligible, because as P_b grows larger, the squares approximate the shape of a circle with better accuracy, and the error is minimized.

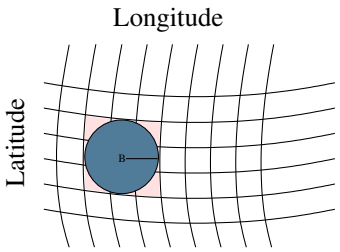

Fig. 1: A small error is introduced due to the difference between the theoretical circle, centered at Bob, and the squares used as an approximation. Because each square is only approximately ten square meters, this error area is miniscule.

A. Assumptions

We assume that Alice and Bob are curious but not malicious. They will follow the protocol correctly but once the protocol has ended they can perform any computation they want on the information (encrypted or otherwise) acquired.

We assume that users cannot forge their GPS coordinates. While it is possible for users to forge their location using apps or developer tools, it is detectible. Either the user has modified the developer option "Allow Mock Locations", or the user has root access on the device [25], which is very difficult to be made undetectable. Although we did not implement these checks in our application, adding them would be trivial.

IV. System Design

Our system is purely distributed, existing only as client software running on Android smartphones. Our client software connects to Facebook Chat, which is depicted in Fig. 2 to send messages. By doing this, we do not have to do name/address resolution, and, because Facebook stores messages when users are not online, our system can tolerate users being unavailable. Despite the fact that our system sends messages over the Internet, and directly through the OSN, user location information is never leaked.

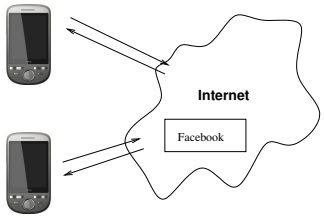

Fig. 2: Phones communicate by sending messages, over the Internet, in the Facebook chat API to one another.

A. Homomorphic Cryptography

Homomorphic encryption takes a foundational role in our system. We use Paillier encryption, in which the following equations hold true.

$$D[(E(m_1) * E(m_2))\%n^2] = (m_1 + m_2)\%n \quad (1)$$

$$D[(E(m_1)^{m_2})\%n^2] = (m_1 * m_2)\%n \quad (2)$$

$$D[(E(m_2)^{m_1})\%n^2] = (m_1 * m_2)\%n \quad (3)$$

In Eqs. 1, 2, and 3, E() is Paillier encryption, D[] is Paillier decryption, n is the product of two large primes, p and q, and m_i is an integer. This properties are very useful. Computation can be done with encrypted values that translates to operations in the plain text domain. This allows parties to do blind computation on encrypted values. For more information on Paillier key generation, please refer to citation [26].

B. Main Idea

In order to better understand the protocol, we first introduce a simple example. Alice wants to learn Bob's location, but Bob only wants to share his location if Alice is within P_b of Bob. Bob wants to determine if Alice is within P_b of Bob without revealing his location to her, or learning her location. In order to achieve this, we begin with two sets of location values α and β, which represent latitude and longitude respectively.

The elements in β each represent approximately $0.00008°$ of longitude area on the Earth, which is about ten meters at the equator. At other latitude cross sections, $0.00008° < 10m$. $0.00004°$ is used for α, because latitude degrees cover more area. These values are not precise; in our actual implementation we use values with higher precision. All the discrete longitude location values (roughly four million) in the range $[-180, 180]$ are numbered uniquely from $\mathbb{N}$ and together form the set β. The same is done with all the discrete latitude location values in the range $[-90, 90]$ which forms the set α. This numbering is used as *location values*.

In order to determine if Alice is near Bob, Bob builds two subsets, one from α and the other from β. Bob's subsets contain all the elements in the range $[L_b - P_b, L_b + P_b]$ where L_b is Bob's location (longitude or latitude). Alice builds two singleton sets, each containing the location value corresponding to her latitude or longitude coordinate respectively.

If Alice's longitude and latitude singleton sets intersect with Bob's corresponding sets, we know that Alice must be within Bob's policy, P_b. Determining if one of Alice's singleton sets intersects with Bob's set is trivial if Alice and Bob can share their values with one another. However, our goal is to hide their locations at this point. In order to do this, Bob generates several, degree one, polynomials. Each polynomial is rooted at one value from his set $[L_b - P_b, L_b + P_b]$. This can be seen in Eq. (4) where each C_i is equal to a corresponding location value.

$$(X - C_1), \quad (X - C_2), \quad \ldots \quad (X - C_n) \quad (4)$$

Bob sends the encryption of the negated coefficients ($E(-C_i) \quad \forall i \in [1, n]$) from these polynomials to Alice. Referring to Eq. (1) we can see that Alice can evaluate the polynomials at her singleton set element's value, L_a.

$$D[(E(L_a) * E(-C_i))\%n^2] = (L_a - C_i)\%n \quad (5)$$

Therefore, Alice computes $E(L_a) * E(-C_i)\%n^2$, and the resulting value can be decrypted by Bob only. If the decrypted

result is 0, Bob knows that Alice's value $L_a = C_i$ and must have been in Bob's set. This process is repeated once for longitude and then again for latitude.

At this point, the software alerts the user. If Alice is within Bob's policy in **both** longitude and latitude, the client software notifies Bob that his location has been shared and his GPS coordinates are sent to Alice. When this message arrives at Alice, the GPS coordinates are presented to her in the application. From this point she can open a map centered at these coordinates.

C. An Improvement

If Bob uses every node in the range $[L_b - P_b, L_b + P_b]$, the protocol will not scale well as Bob's policy increases. Bob will generate hundreds or thousands of polynomials, as shown in Table III. To solve this problem, we use a binary tree to reduce the size of Bob's set, which is inspired by authors A.E. Nergiz et. al., [22]. For the following we perform everything identically twice, once for longitude and again for latitude. We begin by taking the set of longitude (latitude) location values, β (α), and use these as the leaf nodes in the binary tree.

The leaves have already been uniquely numbered from $\mathbb{N}$. The values used for the tree nodes above the leaf level are chosen carefully so that every node in the entire tree is numbered uniquely. For a given, non-leaf node, $value = leftChild + 4003017$. If the node does not have a left child, then $value = rightChild + (4003017 - 2^{h-1})$ where h is the current height in the tree. 4,003,017 is used because that is the largest longitude leaf node value. 4,003,003 is used in the latitude tree. This guarantees that any node at level h will be greater than any node at level $h-1$ and that all nodes will be uniquely numbered.

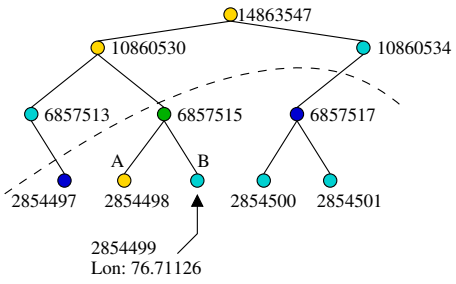

Fig. 3: Example of a longitude tree segment. This tree was generated by Bob at 37.2710, 76.71126, and $P_b = 10$ meters. Bob's wall set = {2854497, 6857515, 6857517}. Alice's path set = {2854499, 6857515, 10860530, 14863547, ...}. Node 6857515 can be generated from the left child or the right child. $6857515 = 2854498 + 4003017 = 2854499 + (4003017 - 2^{1-1})$. The leaf node at longitude 76.71126 generates $2854499 \approx \frac{76.71126+180}{0.0000899321}$.

When building the parent of a particular node, we must know the orientation of the branch connecting these two nodes. The correct branch directions are encoded in the reverse binary representation of the nodes' values. We use 0 to represent a rightward branch and 1 to represent a leftward branch. This allows us to build only a segment of the tree from the leaf level up.

D. Utilizing The Binary Tree

Once this tree is built, Bob and Alice select subsets of the nodes of their tree segment. We refer to Bob's set as the **wall set**. The *span* of a node is the set of leaf nodes which can be reached as children of that node, (i.e., can be reached by following depth first search from that node). Bob's wall set is the set of nodes, highest in the tree, with spans that collectively cover all of, and only, the elements in Bob's range, $[L_b - P_b, L_b + P_b]$. Bob uses the values from the wall set to build his polynomials. It might seem obvious that the root node alone would be an adequate (singleton) wall set. However, the root node often spans a larger range of leaves then P_b allows. In Fig. 3, Bob's wall set = {2854497, 6857515, 6857517}.

We refer to Alice's set as the **path set**. The path set is simply the set of nodes starting at Alice's node and traversing to its parent and its parent's parent iteratively. In Fig. 3 Alice is located at node 2854498 and her path set = {2854498, 6857515, 10860530, 14863547, ...}. Alice's path set actually extends further than is depicted in Fig. 3. The rest of her path set is omitted for brevity.

The intuition behind these two sets is that Bob's set forms a wall, (the dashed line in Fig. 3). Alice's path set will necessarily break through this wall if Alice is within P_b of Bob. In Fig. 3 Alice's and Bob's sets intersect at node 6857515. The value at which the two sets intersect is an element of Bob's **wall set** and therefore, a root of one of Bob's polynomials.

E. Putting It All Together

Alice first sends a location query to Bob. Bob builds his tree segment, based on his policy and location. He then finds his wall set and builds n polynomials, each rooted at a value in his wall set. He sends the encryption of the negated value of each coefficient to Alice. Along with this cipher-text, Bob sends the public key he used to encrypt his values, and the height, in the tree segment, of each coefficient. Alice, upon receiving these values, can reconstruct and evaluate each polynomial. She uses the height information to evaluate each polynomial only once, using her path set node at the corresponding height. Without this modification, the method proposed in [22], is too slow. Alice sends the polynomial evaluations, (encrypted), back to Bob, who can decrypt them. Bob decrypts these values and, if any value decrypts to zero, Bob knows that his wall set and Alice's path set intersect at this point, and that she is within his policy. This is repeated twice for latitude and longitude

When the policy testing is completed, Bob notifies Alice, Alice sends her public key, which Bob uses to encrypt his longitude and latitude coordinates. Bob sends these ciphers to Alice, who decrypts them, and the query is completed.

V. Implementation Details

A. Messages

In any distributed system, it is assumed that nodes, cell phones in our case, can send messages between one another. Our message channel is implemented using the Facebook Chat protocol, which provides simple user-oriented name resolution.

The Facebook Chat protocol is built on top of the Extensible Messaging and Presence Protocol (XMPP), which features, among many other things, guaranteed delivery of messages. Users start the application, set a username and password, and are able to log in to Facebook Chat using our application directly. Once the user is logged in, they can be queried by the other users using our system and make queries themselves. Because Facebook stores messages sent to users that are not currently online, our system tolerates downtime (although processing these buffered messages is not currently implemented).

The Facebook XMPP Chat protocol enforces a limit of 1,000 ASCII characters per message, which we discovered empirically. In order to circumvent this limit, we are forced to split messages into 1,000 character *packets*. In order to reduce the number of packets sent (and therefore reduce user wait time), we trans-code the messages into base 32 encoding.

When packets arrive, if the user is currently logged into Facebook chat using our application, the application will parse the packet. If the packet begins with a special symbol (e.g., "@@" in our implementation), we know that this packet pertains to the protocol, and should be processed. Packets beginning with the special symbol are collected until a packet is found that ends with the special symbol, which denotes the last packet of the message.

Packet collisions, in the traditional sense, are not an issue when using XMPP, as messages retain the correct sending order on arrival. However, if one user receives two, simultaneous queries from another user, the recipient will not be able to distinguish which messages pertain to which query. In order to avoid these logical session collisions, we issue session numbers, in the range $[1, 10000)$, to the messages.

When Bob sends messages to Alice they are in the form

$$< @@T : sess : E(C_1), h_1, ..., E(C_n), h_n, p_k > \quad (6)$$

Here, $@@T$ is the packet type T preceded by our implementation's special character, and $sess$ is the session number. Each $E(C_i)$ is the i-th encrypted coefficient, and each h_i is the corresponding height. p_k is Bob's public key, which Alice uses to encrypt her path set values to perform the homomorphic computation of the polynomials.

The type number T allows the receiver to determine what stage of the protocol this packet is intended for. For example, a type 2 packet contains the encrypted coefficients from Bob's longitude polynomials. Their are approximately nine packet types that handle different segments of the protocol, which are omitted for brevity.

When Alice sends messages to Bob they are in the form

$$< @@T : sess : w_1, w_2, ..., w_k > \quad (7)$$

In Eq. 7, each w_i is the i-th polynomial evaluation which are transmitted back in random order to preserve Alice's privacy. Use use Facebook chat to send messages because it brings with it several features including guaranteed message delivery, guaranteed message order between two users, it requires the two users to be friends on the social network to send messages, and it buffers messages when users are not available. However, we can also use a third party server to speed up message communication, and avoid splitting our messages into packets manually. We evaluate the speed gains of using such a server in our evaluation, Section VII-C.

B. Key Management

Our system uses two sets of Paillier, public/private key pairs. The first set is used for private proximity detection. Bob generates a key pair which he uses to encrypt the coefficients of his polynomial. He sends these coefficients, along with the matching public key, to Alice. The second set is used to send the actual GPS coordinates from Bob to Alice. Alice generates a key pair and sends the public key to Bob, which he uses to send his location back to her.

These keys are never explicitly saved to disk. Although time can be saved not generating new keys for each query, this time is negligible and the security risk of re-using keys is large. It is important to note that the user never has to do any manual key management.

VI. Security Analysis

The goal of our system is to provide a private and secure service to users. This section analyzes the security of the system from the different perspectives of the parties involved.

The protocol has one potential issue, because the proximity detection is handled in two rounds. When the first round has finished, Bob can decrypt the values and, if any one of them decrypts to 0, Bob knows that Alice is within P_b in longitude only. At this point information has been leaked to Bob, who should only learn location information about Alice if she is near him in **both** longitude **and** latitude. He can fool Alice by continuing the protocol regardless of the result so she cannot infer anything about Bob's location.

Fortunately, the information exposed to Bob is limited. He knows that Alice is within P_b of Bob in longitude only. As a result, he can guess she is in a $2 * P_b$ wide vertical band, centered on him, of longitude values all the way around the Earth. This disclosure is relatively mild. There are still millions of latitude values to guess. Due to the weak disclosure of this issue, we felt it unnecessary to modify the system. We have implemented a trivial amount of protection in that the client software does not reveal any information to the user until the protocol has finished. Only at this point can it be confirmed that Alice is within P_b in both longitude **and** latitude.

We can solve this problem completely if we represent all the permutations of all possible $< lat, lon >$ pairs. This is substantially different then the current system, which creates two sets of polynomials, one rooted at longitude values and the other rooted at latitude values. Due to the complexity, the significant difference from our current system, and the relative insignificance of the information leaked, we did not attempt this approach.

Social Networking Site Provider Messages are passed in-band (i.e., over the social networking service). We assume that the social network provider can see and read all messages. However, information is not leaked to the provider, because

our system maintains that the sensitive contents of all messages are encrypted, and new key pairs are generated for each query. The social network provider (or any message provider) may attempt to perform IP geolocation. However, IP geolocation is easily circumvented using IP address obfuscation techniques such as TOR or a VPN.

In the worst case, the social network provider can sabotage the system by scrambling messages or refusing to send encrypted messages. However, determining which messages are related to the protocol, and which are simply users conversing, comes down to identifying which messages are cipher-text and which are plain-text. This is non-trivial on a large scale and yields little reward. Therefore, it is not practical for the social network provider to do this.

Alice Alice receives, from Bob, the coefficients for a polynomial rooted at Bob's wall set values. However, they are encrypted and, therefore, incoherent to Alice. If these coefficients were in plain text Alice could easily determine the roots.

Alice may cache the encrypted coefficients sent to her in queries (from Bob). She might then recognize the encrypted coefficients sent to her in future queries. This would only occur if Bob was at the same location, with the same policy, and using the same key. Fortunately, in our system we generate a new key for every query. By doing this we significantly decrease the likelihood of one user recognizing another re-using the encrypted coefficients from a previous query.

Bob also reveals height information about each coefficient to Alice. Alice may be able to deduce Bob's policy from this height information, (larger heights indicate a larger policy), but this disclosure is minimal and does not indicate to Alice anything about Bob's location.

Bob Bob receives, from Alice, her evaluations of the polynomial and he has the polynomial (which he generated). Because this function is simple and well-behaved he can determine the input necessary to generate the given output. This allows Bob to determine Alice's location. In order to stop this attack we implement a random number r which Alice uses after she has generated her w_i values. $\forall w_i$ we modify $w_i = (w_i)^r \% n^2$. If $w_i = 0$ it remains 0. However, other values are hidden from Bob who does not know the value of r.

VII. Performance Evaluation

In order to evaluate the performance of this system, we ran several experiments which are detailed in the following subsections. The first subsection details the speed of homomorphic encryption on smartphones. In the second subsection, we examine the savings from utilizing a binary tree. In the third subsection, we analyze the running time of our system in order to determine the practicality for end users.

For all of these experiments, we used one or both of two Android OS based smartphones connected to the Internet using WiFi. WiFi is always used in our experiments because of the excessive cost of mobile phone 3g or 4g data plans. The first phone *p1* is an LG-C800 with a 1Ghz processor and 512Mb of RAM running Android 2.3. The second phone *p2* is a Samsung Nexus S with a 1Ghz Cortex-A8 processor and 512MB of RAM. Our Paillier cryptography implementation was borrowed from the University of Maryland Baltimore County [27].

A. Homomorphic Encryption

In the first experiments we wanted to determine if the weak ARM processors on typical Android cell phones are powerful enough to perform Paillier cryptographic operations in a timely fashion. A small Android application was written and used, which generates a Paillier key, encrypts a user selected value, and decrypts the value. We ran this program five times, with a random value as input, and summarized the results in Table I. We chose 1024 bit encryption because it is a good balance between security and speed. To get a better estimate of the protocol's running time, we examine lower-level, cryptographic functions, shown in Table II. Here one key is generated and the random value is encrypted and decrypted once.

Bits	Run 1	Run 2	Run 3	Run 4	Run 5	Avg.
256	123	75	82	74	82	115.3
512	208	139	374	134	177	257.3
1024	1536	1366	1042	933	1158	1176.5
2048	6773	8105	9453	8007	3710	6349.3
4096	91785	44144	31311	26047	81108	46415.2

TABLE I: Time in milliseconds of generating a Paillier public, private key pair, encrypting a value with the public key, and decrypting that value with the private key.

Run	Key Gen	Encryption	Decryption	Total
1	769	80	147	1006
2	1601	81	147	1830
3	624	80	146	852
4	756	80	151	989
5	1891	80	146	2118
Avg.	1128.2	80.2	147.4	

TABLE II: Time in milliseconds of generating a Paillier public, private key pair, encrypting a value with the public key, and decrypting that value with the private key on an ARM 1Ghz processor

During our protocol, there are only a few dozen encryptions / decryptions performed, each one costing approximately 100ms and there are only two key generations, each costing approximately one second. This shows that 1024 bit encryption is appropriate for our system.

B. Binary Tree Efficiency

In the second experiment, we wanted to determine the savings we achieved by using a binary tree data structure in order to encode the discretized location values. We ran the protocol six times, on *p2*, while varying P_b. If the tree structure affords us a large saving, the protocol will transmit less information and there will be fewer cryptographic operations.

In Table III, we can see that, as the policy increases, the number of leaves increase. This is expected, $leaves = ((policy/10) * 2) + 1$. Bob's wall set grows at approximately O(log(n)) because of the nature of binary trees.

Paillier encryption generates cipher-text that is twice the bit length of the key [28]. We use 1024 bit encryption so we expect cipher-text will be 2048 bits long. Messages are transcoded, for transmission, in a base 32 numbering system. As

Policy	30	100	300	3000	30000	160000
Leaves	7	21	61	601	6001	32001
Wall Size	3	6	6	10	12	12
Message Length	1638	2867	2867	4505	5324	5324
Packets	2	3	3	5	6	6

TABLE III: Analysis of wall size as a function of Bob's policy. Also included,is the approximate message length that can be expected for the given policy.

discussed in Section 5, messages are then broken up into 1000 character packets. Therefore, it is reasonable to see only a few packets generated from thousands of characters.

C. System Running Time

1) End-To-End Running Time: To get an idea of what users will experience when using the application, we ran ten location queries with different policies, between *p1*, Bob, and *p2*, Alice. In this experiment Alice and Bob are at the same location (37.2701, -76.7119) which was selected randomly. The results of these experiments are summarized in Fig. 4.

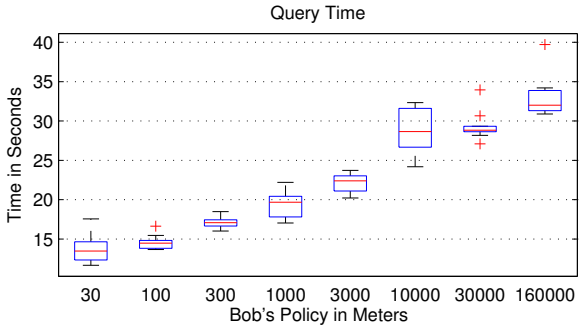

Fig. 4: Box plot of the total running time of the system. Our system takes on the order of tens of seconds to run depending on the policy.

We can see that the protocol runs in a reasonable time. Even a very large policy of 160km only takes $[30-35]$ seconds to complete. More reasonable queries take tens of seconds and very small policies require under 15 seconds. Because the user is alerted via an Android notification when a query has completed and query progress is indicated to the user while waiting, this is a reasonable waiting time [29].

2) Message Transmission Speed: To evaluate the data transmission speed of our system, we had one phone, *p1*, send a random string of characters, to a second phone, *p2*. *p2* immediately responds back with the same characters. *p1* records the time it takes to send and receive the entire message, (round trip time). *p1* also records the time between receiving the first and last packet from *p2*. This shows the overhead incurred by the packet size limitation. We performed the experiment five times for each message length and plotted the results in Fig. 5.

Unsurprisingly, messages of 500 or 1000 characters take a short time to transmit; median of 1.6 seconds and 1.8 seconds respectively. As the message length grows larger, the round trip time and packet receiving time increases. In the bottom of Fig. 5, we can see plateaus in each pair of sizes (e.g., 2.5k and 3k, 3.5k and 4k, etc). This is due to the 1000 character limit. A 3000 character message and a 2500 character message are both

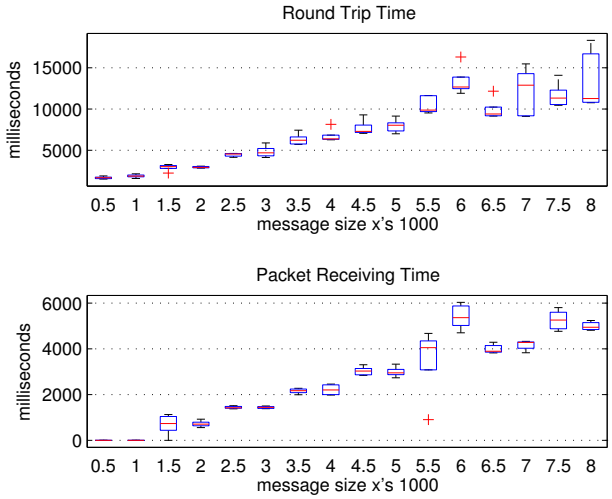

Fig. 5: Top: Round trip time, in ms, of sending a random message. Bottom: packet receiving time, in ms, between receiving the first and last packet. For each size, a random message was generated and sent five times (round trip).

broken up into 3 packets. When the message sizes are very large, > 5000 characters, the volatility increases greatly. We can attribute this to the fact that the Facebook message system is not designed to send such high volumes of data. Humans typically converse at a few dozen characters per second.

3) Message Size: To evaluate how much data is sent during a typical query, we ran sets of ten queries varying Bob's policy, similar to our experiment in Section VII-C1, on *p1* and *p2*. During each query we recorded the total bytes transmitted and received by Bob, (*p2*). It is safe to assume that the traffic on Alice is accurately reflected at Bob, because Bob and Alice are the only two parties communicating. The results of this experiment are summarized in Fig 6.

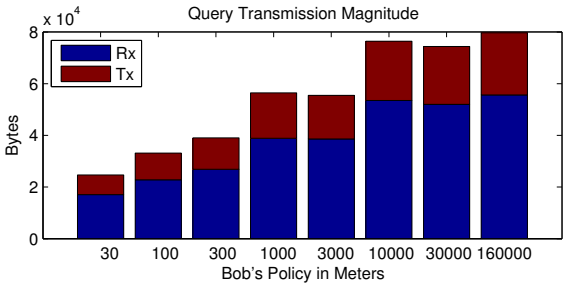

Fig. 6: Transmission size of a query as measured by Bob.

Because the Android API only provides a measure of total bytes transmitted or received, our results include some small background data transmissions from other miscellaneous applications and services (e.g., email sync, automatic application updates). Also, the size of the cipher text transmitted varies slightly. Different keys will generate different cipher text values, requiring more or less characters to represent, given the same input. To capture these variations, the standard deviation of each set of ten queries is presented in Table IV.

Our system takes only tens of kilobytes to perform a query,

Policy	Rx Standard Deviation	Tx Standard Deviation
30	2441.8	54.4
100	1257.7	79.4
300	1283.7	113.9
1000	1338.2	223.6
3000	1776.4	133.8
10000	3186.4	122.7
30000	1877.0	458.9
160000	2905.2	404.8

TABLE IV: Standard deviation of the transmission size.

even for very large policies. The standard deviation, which is a small fraction of the bytes transmitted, demonstrates that our protocol is stable in data transmission size.

Combining all the results from section VII, we conclude that the largest bottleneck in our system is message transmission, which takes up the vast majority of the query time. As evidenced by the message size, and the experiments in Sections VII-C2 and VII-C3, this is because the Facebook Chat protocol is slow, not because our system sends large amounts of data. Although all of our experiments utilize the mobile phones' WiFi connections, our system is practical for mobile 3g or 4g data plans. Message transmission speed is limited by the speed of Facebook Chat, only roughly 8k-bits per second. We can easily increase this speed by using Facebook Chat to initiate a query, allowing Alice and Bob to exchange references. We then offload all message sending to a third party server. Here the bottleneck for data transmission would be 3g speeds which are at least 384k-bits per second [30].

D. Scalability Analysis

Our system is concerned with one user querying the location of another. However, users may want to learn their friends' location as soon as they are nearby (within P_b). We face many challenges when trying to scale our system up for many users in use cases like these. Each query takes approximately 40 seconds and 80 KBytes in the worst case, which means Alice can send queries at a maximum rate of 90/hr. and 7.03MB/hr. In the most naive approach, where Alice queries one user at a time repeatedly, it will take over three hours in the worst case for her to be alerted when a friend is nearby if she has 300 Facebook friends.

To scale our system we can make several key improvements. Firstly, we note that Alice can query many of her friends in parallel. That is, she can send messages using Facebook chat to query all of her friends simultaneously. In this way, Alice doesn't wait for each query to finish before starting the next one. If she does this, several other factors become the bottleneck. Alice will need to to generate one key for each friend, and a second key if the query is successful to send the location coordinates. To save time, we can pre-compute many keys and store them locally. Keys are only 1024 bits long, so a table of 10,000 pre-computed keys will only take up approximately 1MB of local storage and access to them will be instant. In our protocol, in the worst case, Bob will have a policy of 160km, which means he will generate a wall set of twelve nodes, or 5k characters. Bob will send this to Alice. Using our third party server for communication, and opening many connections, Alice can accept all of this data in parallel from each of her Facebook friends. However, her 3g bandwidth will limit the transfer to roughly 384kbit/s, the minimum speed set forth by the ITU [30]. So sending these encrypted coefficients will take 31 seconds with 300 friends. Once Alice has received these coefficients, she reconstructs and evaluates the polynomials which require multiplications of all the values. The time to perform these multiplications is negligible for the BigInteger java library with relatively small values. Once the values are calculated, they're sent back to Bob who decrypts to see if any of the results are zero. This incurs another 31 seconds of transfer time by opening multiple connections. Because each Bob is doing the decryption independently, it's effectively in parallel and only adds roughly one second to the running time. The data transfers, multiplications, and decryptions are all done twice. Once for latitude and again for longitude. Therefore, it will take between two and three minutes to complete the protocol for 300 friends.

Once this is completed, Bob knows if Alice was nearby and he can use her public key (sent to him in the final transmission) to transmit his actual location values. In the worst case, all of Alice's friends are nearby, and Alice must receive another set of 300 messages. Fortunately, the final location messages are small, only taking up about 1k characters. So this transfer only takes about one second.

E. Mobility Analysis

Our system captures the use case where Alice wants to learn Bob's location one time, when they are not moving. When Alice and Bob are moving, they have some speed relative to one another. When P_b is very small, but the relative speed is very large, the required query rate becomes too high. In these situations, Alice and Bob may move toward one another, spend some brief moment nearby, and move far apart again before the query has finished. In this case, Alice and Bob will not be alerted by the system (false negative). We simulated these scenarios by analyzing many combinations of relative speed (in the range $[1, 200]$km) and policy (in the range $[100, 3000]$meters). For the vast majority of scenarios, (there are many policies greater than 3000m), the required query rate is below this threshold. However, at very small policies, the query rate required is far too fast. We summarize some significant relative speeds and query rates in Table V. Fortunately, these are reasonable policy sizes given the corresponding speed the users are traveling.

Activity	Relative Speed	Minimum Policy
Driving (Highway)	200km/hr	2220m
Driving (City)	90km/hr	1000m
Biking	50km/hr	550m
Walking	10km/hr	110m

TABLE V: Minimum policy size required to allow maximum query rate (90/hr) at the given relative speed. These points are all at the intersection of the plane at Z=90.

VIII. Conclusion

In this paper we present a working system with which users can privately and securely share their location information with one another. The system enforces location information access

control through private proximity detection by utilizing Paillier homomorphic encryption. We offer an implementation of this system on the Android platform and Facebook online social network. It is our hope that this system can be used by people in order to enjoy the convenience of proximity detection and location sharing without worrying about the implications of strangers, or online social network providers, gaining access to their location data. Additionally, our system can be deployed as a layer in another location based service or system to protect user privacy while maintaining the proper operation of the service. The source code is freely accessible online for users to install or make improvements on [31].

REFERENCES

[1] Wikipedia, "List of online social networks," http://en.wikipedia.org/wiki/List_of_social_networking_websites, September 2012.

[2] C. Johnson, "Facebook hits one billion users, not counting fake accounts," http://arstechnica.com/business/2012/10/facebook-hits-one-billion-users-not-counting-fake-accounts, October 2012.

[3] I. Bilogrevic, M. Jadliwala, K. Kalkan, J.-P. Hubaux, and I. Aad, "Privacy in mobile computing for location-sharing-based services," in *Proceedings of the 11th international conference on Privacy enhancing technologies*, ser. PETS'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 77–96. [Online]. Available: http://dl.acm.org/citation.cfm?id=2032162.2032167

[4] "Saga android and ios application," http://www.getsaga.com/, July 2013.

[5] J. Cranshaw, E. Toch, J. Hong, A. Kittur, and N. Sadeh, "Bridging the gap between physical location and online social networks," in *Proceedings of the 12th ACM international conference on Ubiquitous computing*, ser. Ubicomp '10. New York, NY, USA: ACM, 2010, pp. 119–128. [Online]. Available: http://doi.acm.org/10.1145/1864349.1864380

[6] T. Pontes, M. Vasconcelos, J. Almeida, P. Kumaraguru, and V. Almeida, "We know where you live: privacy characterization of foursquare behavior," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, ser. UbiComp '12. New York, NY, USA: ACM, 2012, pp. 898–905. [Online]. Available: http://doi.acm.org/10.1145/2370216.2370419

[7] G. Greenwald and W. MacAskill, "Edward snowden: the whistleblower behind the nsa surveillance revelations," June 2013.

[8] F. Rahman, M. E. Hoque, F. A. Kawsar, and S. I. Ahamed, "Preserve your privacy with pco: A privacy sensitive architecture for context obfuscation for pervasive e-community based applications," in *Proceedings of the 2010 IEEE Second International Conference on Social Computing*, ser. SOCIALCOM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 41–48. [Online]. Available: http://dx.doi.org/10.1109/SocialCom.2010.16

[9] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new casper: query processing for location services without compromising privacy," in *Proceedings of the 32nd international conference on Very large data bases*, ser. VLDB '06. VLDB Endowment, 2006, pp. 763–774. [Online]. Available: http://dl.acm.org/citation.cfm?id=1182635.1164193

[10] C.-Y. Chow and M. F. Mokbel, "Enabling private continuous queries for revealed user locations," in *Proceedings of the 10th international conference on Advances in spatial and temporal databases*, ser. SSTD'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 258–273. [Online]. Available: http://dl.acm.org/citation.cfm?id=1784462.1784477

[11] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *Proceedings of the 1st international conference on Mobile systems, applications and services*, ser. MobiSys '03. New York, NY, USA: ACM, 2003, pp. 31–42. [Online]. Available: http://doi.acm.org/10.1145/1066116.1189037

[12] R. Shokri, G. Theodorakopoulos, C. Troncoso, J.-P. Hubaux, and J.-Y. Le Boudec, "Protecting Location Privacy: Optimal Strategy against Localization Attacks," in *19th ACM Conference on Computer and Communications Security*, 2012.

[13] R. Shokri, G. Theodorakopoulos, J.-Y. Le Boudec, and J.-P. Hubaux, "Quantifying location privacy," in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, ser. SP '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 247–262.

[14] M. Srivatsa and M. Hicks, "Deanonymizing mobility traces: Using social network as a side-channel," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, Oct. 2012.

[15] W. Wei, F. Xu, and Q. Li, "Mobishare: Flexible privacy-preserving location sharing in mobile online social networks." in *INFOCOM*, A. G. Greenberg and K. Sohraby, Eds. IEEE, 2012, pp. 2616–2620. [Online]. Available: http://dblp.uni-trier.de/db/conf/infocom/infocom2012.html#WeiXL12

[16] S. Papadopoulos, S. Bakiras, and D. Papadias, "Nearest neighbor search with strong location privacy," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 619–629, Sep. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1920841.1920920

[17] L. Zhang and X.-Y. Li, "Message in a sealed bottle: Privacy preserving friending in social networks," *CoRR*, vol. abs/1207.7199, 2012.

[18] R. Tonicelli, B. M. David, and V. de Morais Alves, "Universally composable private proximity testing," in *Proceedings of the 5th international conference on Provable security*, ser. ProvSec'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 222–239. [Online]. Available: http://dl.acm.org/citation.cfm?id=2042756.2042776

[19] Z. Lin, D. F. Kune, and N. Hopper, "Efficient private proximity testing with gsm location sketches," in *Financial Cryptography*, 2012, pp. 73–88.

[20] L. Siksnys, J. R. Thomsen, S. Saltenis, and M. L. Yiu, "Private and flexible proximity detection in mobile social networks," in *Proceedings of the 2010 Eleventh International Conference on Mobile Data Management*, ser. MDM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 75–84. [Online]. Available: http://dx.doi.org/10.1109/MDM.2010.43

[21] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh, "Location privacy via private proximity testing," in *NDSS*, IEEE. The Internet Society, 2011.

[22] A. Nergiz, M. Nergiz, T. Pedersen, and C. Clifton, "Practical and secure integer comparison and interval check," in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, aug. 2010, pp. 791 –799.

[23] G. Ozsoyoglu, D. A. Singer, and S. S. Chung, "Anti-tamper databases: Querying encrypted databases," in *In Proc. of the 17th Annual IFIP WG 11.3 Working Conference on Database and Applications Security, Estes Park*, 2003, pp. 4–6.

[24] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '04. New York, NY, USA: ACM, 2004, pp. 563–574. [Online]. Available: http://doi.acm.org/10.1145/1007568.1007632

[25] D. Doonan, "Android gps forensics," http://dandoonan.blogspot.co.uk/2013/03/mock-locations.html, March 2013.

[26] Wikipedia, "Paillier encryption," http://en.wikipedia.org/wiki/Paillier_encryption, September 2012.

[27] B. C. University of Maryland, "Umbc implementation of paillier encryption in java," http://www.csee.umbc.edu/~kunliu1/research/Paillier.html, September 2012.

[28] M. Prabhakaran, "Homomorphic encryption, lecture 15," courses.engr.illinois.edu/cs598man/fa2011/slides/ac-f11-lect15.pdf, p. 94, Fall 2011.

[29] J. Nielsen, "Response times: The 3 important limits," http://www.nngroup.com/articles/response-times-3-important-limits/, January 1993.

[30] "wikipedia", "Wikipedia 3g article," http://en.wikipedia.org/wiki/3G, January 2014.

[31] E. Novak, "Near-pri online code repository," https://github.com/deadmund/Nearby, November 2012.

[32] A. Narayanan, V. Toubiana, S. Barocas, H. Nissenbaum, and D. Boneh, "A critical look at decentralized personal data architectures," *CoRR*, vol. abs/1202.4503, 2012.

[33] W. Wei, F. Xu, C. C. Tan, and Q. Li, "Sybildefender: Defend against sybil attacks in large social networks," in *IEEE Infocom*, Orlando, FL, March 2012.