Location Privacy via Private Proximity Testing

Arvind Narayanan Narendran Thiagarajan Mugdha Lakhani
Michael Hamburg Dan Boneh*

Stanford University

Abstract

We study privacy-preserving tests for proximity: Alice can test if she is close to Bob without either party revealing any other information about their location. We describe several secure protocols that support private proximity testing at various levels of granularity. We study the use of "location tags" generated from the physical environment in order to strengthen the security of proximity testing. We implemented our system on the Android platform and report on its effectiveness. Our system uses a social network (Facebook) to manage user public keys.

1 Introduction

Location-aware devices, mainly smartphones, are ubiquitous. While the number of location-based services has mushroomed, adoption remains stagnant [52]. Privacy concerns are a major factor holding up the growth of this market [38, 40]. Users take location privacy seriously because of the possibility of physical harm [42]. For an analysis of the prevalence of location data on the Internet and the dangers posed by it, see [14].

Current location-based services require the user to constantly transmit their location to a server (or at least whenever services are required). Users have accepted being tracked by the carriers, and there is some regulatory oversight over this; however, they are far more reluctant to share location information with third party services whose reputation is unknown. It would therefore be of great benefit to design practical location-based services in a way that minimizes the information transmitted to the service provider.

We consider the problem of *proximity testing with privacy*, which is an important and burgeoning type of location-based service in social networking[1]. Private proximity testing enables a pair of friends to be notified when they are within a threshold distance of each other, but otherwise reveal no information about their locations to anyone.

Location-based social networking is broader than just proximity detection. Users may want location based gaming, activity streams filtered by location, etc. We seek to construct the fundamental cryptographic primitives on top of which such functionality can be built in a privacy-preserving way. While proximity detection by itself covers a broad range of use cases (we give several examples below), our solutions are also useful for the techniques that can be used to construct other types of privacy-preserving functionality.

Our contributions:

1. We put forth a set of desiderata for privacy-preserving proximity testing. This is tricky because there are several desirable security and efficiency properties that conflict with each other. We aim for rigorous cryptographic security definitions. In Section 6 we explain why this is necessary and why more ad-hoc definitions are inadequate.

2. We reduce private proximity testing to the underlying cryptographic problem of private equality testing (PET). We consider this problem in two different settings: with or without the involvement of the server. The server-mediated setting is more efficient, but is not secure against the collusion of the server with the user's friends. While PET has been considered before, we design efficient protocols in both settings that use less bandwidth than older designs.

3. We show how to use *location tags* to enhance the security of private proximity testing. A location tag is an ephemeral, unpredictable nonce associated with a location and can be derived from various electromagnetic signals available in the environment, such as WiFi and Bluetooth. It can be thought of as a shared pool of entropy between all users at a given location at a given time. The location tags we use are a variant of the concept introduced by Qiu et al. [37, 36].

4. We describe a prototype that we have built for the Android platform. One component of our system establishes shared keys between pairs of users and we take a novel approach to this problem. We designed and implemented a key agreement system using a social network as a key-distribution mechanism, instead of

*Supported by NSF, MURI, and the Packard Foundation.
[1] Foursquare and Facebook Places are two prominent examples.

traditional PKI. Our system binds the user's public key to the user's social network account (the current implementation uses Facebook). We argue that using a social network is a simpler and potentially more user-friendly alternative to traditional PKI for managing user public keys.

Motivation. Let us consider several applications of proximity testing, keeping in mind that different applications require different proximity granularity.

- Alice and Bob are friends, and are serendipitously notified that they are shopping in the same mall. They meet and have a pleasant time together. Alternatively, Alice and Bob first meet online, but later decide to meet in person at a coffee shop. Alice arrives first and is notified when Bob arrives.
- Alice would like to get dinner with her friend Bob who travels a lot. Using privacy-preserving proximity testing, Alice can check if Bob is town before calling him. Note that for this application the proximity granularity is a wide geographic area.
- Bob, a student lands at his college airport and wants to check if anyone from his college is currently at the airport and can give him a ride to campus.
- Alice is a manager who wants to automatically record who is present at her daily meetings. However, her employees do not want their location tracked. Privacy-preserving proximity testing over this well organized group allows satisfying both requirements.

Using our system, existing location-based social networks such as Loopt and Google Latitude could offer a basic or 'lite' version with limited functionality but respecting privacy. This may spur adoption by privacy-conscious users, and make the product more useful overall due to the positive network externality (i.e., even the users who don't care about privacy benefit from having more friends in the system).

As discussed above, proximity detection is particularly useful for group meetings, either ad-hoc groups or well organized groups such as project teams. More esoteric applications may include unit detection in ground combat. Privacy ensures that in case of capture, the proximity test does not reveal the location of all combat units.

2 Model

All communication in our system takes place over the Internet, i.e., we do not use direct physical channels between nearby devices such as Bluetooth. All location-based services today operate this way. We argue in Section 6 that a peer-to-peer communication model does not support the functionality we want.

Our system requires the existence of a social network, i.e., a graph that captures trust relationships between users. Our protocols allow detection of proximity between any two users connected by an edge and we assume the existence of shared secret keys between connected users (more details on this are in Section 5.1).

The reason we only allow proximity testing between adjacent nodes in a social network is that proximity detection between strangers is a useful functionality, but is impossible to do efficiently and privately in a client-server model. The reason is that either the server will need to learn some information about users' locations, or it will need to treat every pair of users identically, resulting in overall bandwidth requirements quadratic in the number of users, unless limited to pairs of friends. As we discuss in Section 6, revealing even a minimal amount of information about users' locations (e.g., the single-bit outcome of proximity testing between pairs of users) to the server results in an unacceptable privacy leak when aggregated over time and users.

The ideal functionality of our model is in Section 2.5. When location tags are available, the model is somewhat different, and the ideal functionality is in Section 4. But first let us discuss some of the desiderata that will motivate our model.

2.1 Desiderata: Functionality

Asymmetry. Proximity testing is asymmetric: one party will learn if the other party is nearby whereas the other party learns nothing. This is necessary because asymmetric edges are common in social networks — Alice may be willing to let Bob test proximity to her, but not vice versa. Of course, if an edge is symmetric, then it can be treated as a pair of directed edges, and we can execute the protocol twice in either direction.

One important side-benefit of asymmetry is the some of our protocols can be executed in an *asynchronous* manner. This has the potential to greatly decrease the communication cost. We explain this in more detail in Section 2.4.

Proximity threshold. The distance threshold for proximity detection should not be globally fixed but instead configurable by each user. This is because a larger threshold is neither strictly worse nor strictly better than a smaller one, either from the security or the functionality perspective. With a larger threshold, the user is easier to locate but in case of a match their location is revealed less accurately.

Ideal functionality. The obvious way to define the "ideal functionality" is as follows: whenever Alice and Bob are within a distance δ (defined by Alice) of each other, Bob outputs 1, otherwise he outputs 0. However, there is a problem with this definition: even when δ is large, Bob can determine Alice's exact location by moving around and applying "triangulation" (assuming Alice is stationary).[2]

Since the natural ideal functionality is flawed, we must necessarily "quantize" the space of locations. We describe our quantization technique in Section 2.5.

2.2 Desiderata: Security

The adversary might be law enforcement coercing the service provider into revealing the user's location, or it might be someone colluding with the service provider. It could also be one of the user's friends — either because the friend's account has been compromised, or the attacker set up a fake profile, or simply because friendship on social networks does not imply a high level of trust. Stalking is an example of the threat from this type of adversary.

Broadly, these break down into untrusted server and untrusted friend(s). Our overall goal is to reveal as little as possible to each party while enabling proximity testing. We now discuss each threat in more detail.

Honest-but-curious friend. The space of possible locations has low entropy, and is therefore vulnerable to a brute-force or dictionary attack. Suppose that the threshold distance for proximity detection is 10 meters; this results in a search space of roughly 10^{10}, which is less than 34 bits. Therefore, the only meaningful way to define privacy is in a manner analogous to *semantic security* or *indistinguishability* in cryptography; roughly, the protocol should reveal no information other than the output of the computation.

Malicious friend. If the attacker has some background information about the user, the search space becomes smaller. In the extreme case, the attacker might only be interested in answering a binary question, e.g., whether the user is at home or at work, in which case the search space is only 1 bit.

This shows another weakness of the ideal functionality: even a protocol that reveals nothing but the output of the computation is vulnerable to an online guessing attack by a malicious friend (who is willing to lie about his own location). Resisting such an attack is not critical: as we explain in the next subsection, protocols in our system are naturally rate-limited. Nevertheless, it is a useful desideratum. In Section 4 we explain how to resist online attacks using location tags.

Server. In some of our protocols, the server is treated as just a router of messages between users. These protocols are secure against a malicious server due to the existence of shared secret keys between friends, which are used to establish an encrypted and authenticated channel between the parties. The server can of course refuse to deliver messages, but can do no other harm.

In our most efficient protocols the server acts as a participant. Here, we require that the server be able to learn no information about users' locations, even if it may be able to cause the users to compute an incorrect answer.

Collusion. Malicious friends may collude with each other; the security requirements should ideally still hold. As for the server, in protocols where it acts a router security should hold even if it colludes with any of the user's friends. On the other hand, when the server is a participant in the protocol, we do not expect to have privacy against the collusion of the server with a friend (indeed, it is this very relaxation that allows efficiency gains in this model).

2.3 Desiderata: Efficiency

Mobile devices are resource-constrained, so we would like to construct solutions that minimize computation and bandwidth. Since we have an eye toward implementation, we go beyond asymptotic analysis and also focus on optimizing the constant factors involved.

Computation. While any two-party functionality can be securely computed by expressing it as a special case of Yao garbled-circuit evaluation [49], this can be too inefficient for use in practice. In general, we seek to minimize the number of modular exponentiations and other expensive operations. Protocols that avoid the use of large groups altogether are particularly attractive.

Bandwidth is perhaps the most constrained resource, and we would like to minimize the bandwidth required per edge. It is an intriguing question whether we can use amortization to design a protocol whose bandwidth requirement is asymptotically smaller than the number of edges. It does not appear to be possible to do so unless we compromise on the security requirements.

Number of connections. Proximity testing needs to be carried out between each pair of friends; however, executing an instance of the protocol independently for each friend would involve opening multiple connections to the server (one for each edge in the system) and quickly becomes infeasible.

Instead, in our system, each user (say Alice) sends a single message to the server that encapsulates her messages from all the instances of the protocol — one for each friend. The server de-multiplexes these messages and then re-multiplexes them by recipient. In this manner, the number of connections grows asymptotically as the number of nodes in the system.

[2] Loopt has reported that users often carry out this attack on their system.

There are added subtleties in the case of synchronous protocols; details are in Section 2.4.

2.4 Synchronous vs. asynchronous execution

A proximity testing protocol with a *synchronous* communication pattern requires both parties to be online at the same time. The role of the server is merely message passing.

In the synchronous pattern, the execution of the protocol needs to happen at globally fixed time intervals (say once every 5 minutes). We call these intervals *epochs*. Each protocol round is executed synchronously by all users participating in the system, so that the multiplexing of messages described in Section 2.3 can happen.

On the other hand, the communication pattern in some protocols is *asynchronous*, which means that the two parties do not have to be online at the same time. In other words, it can be seen as a two party protocol between the "sender" and the server, followed by a two-party protocol between the "receiver" and the server.

In this setting, the server needs to be involved in the computation rather than being simply used for message passing. The reason is that if the server is only used for message passing, then it is equivalent to a two-party non-interactive protocol. This makes the message essentially a hash of the location, which allows a dictionary attack on the location.

The asynchronous setting allows a privacy-efficiency tradeoff due to the fact that the sender and receiver can each execute their half of the protocol with the server at arbitrary times. Specifically, a user might configure her device to participate in the protocol in the role of sender only when her location changes. Similarly, she might configure her device to participate in the role of receiver only when she explicitly checks the proximity testing application.

The detriment to privacy comes from the fact that the server learns when the user is moving. This is of course a much less severe leak than leaking the location. Nevertheless, the server might be able to tell, for example, when a user went home for the night. It is important to note that each user can control their participation schedule and change it at any time.

2.5 Reducing proximity testing to equality testing.

Let G be a grid or a tessellation of the plane,[3] and let L_u be the location of user u expressed as the center of the grid cell that the user is located in. For any two users u and v, the equality testing protocol $\Pi_{u,v}$ must satisfy the following:

- if $L_u = L_v$ then u learns L_v

[3]more accurately, the surface of the Earth. In Section 5 we describe how we apply a grid to a curved surface.

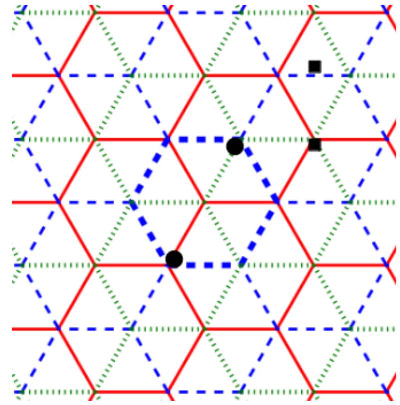

Figure 1. Overlapping grid system. The squares represent minimally separated users who are not part of the same cell in any grid. The round dots represent maximally separated users are in the same cell in one of the grids (the bold one).

- if $L_u \neq L_v$ then u learns nothing about L_v except that $L_u \neq L_v$

By running any such protocol Π on three hexagonal grids $\mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_3$ which are mutually offset in the manner shown in Figure 1, we obtain a proximity testing protocol Π' such that

- if $||X_u - X_v|| \leq \delta$ then u learns at least one of $L_{v,1}, L_{v,2}$ and $L_{v,3}$.
- if $||X_u - X_v|| \geq \gamma\delta$ then u learns nothing about X_v except that $||X_u - X_v|| > \delta$

where $L_{u,i}$ is the quantized location of user u in grid i, δ is half the height of each hexagonal cell, $\gamma\delta$ is the diagonal of the hexagons and X_u is the exact location of a user u. Π' is the ideal functionality for proximity testing.

Intuitively, what this means is that (i) proximity testing only reveals quantized locations, (ii) it is guaranteed to do so when the parties are sufficiently close, and (iii) guaranteed not to do so when they are sufficiently far apart.

The values δ and $\gamma\delta$ are respectively the closest two users can be and not be detected as nearby, and the farthest they can be and still be detected as nearby. Note that δ is $\sqrt{3}/2$ times the side of the hexagons and γ is $4/\sqrt{3}$ (dimensionless).

The value of $\gamma\delta$ is visually clear from Figure 1. To derive the value of δ, we argue as follows (Figure 2): consider two users not detected as nearby. If one of them (Alice) lies in the triangle marked by the dot, the other (Bob) cannot lie in the region marked X, and therefore the two locations are separated by at least one of six pairs of parallel lines (one such pair is shown in the diagram), which are a distance δ apart.

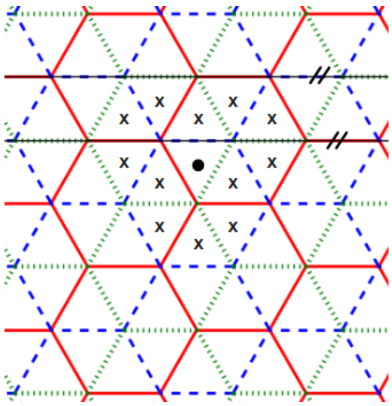

Figure 2. Neighborhood of a point

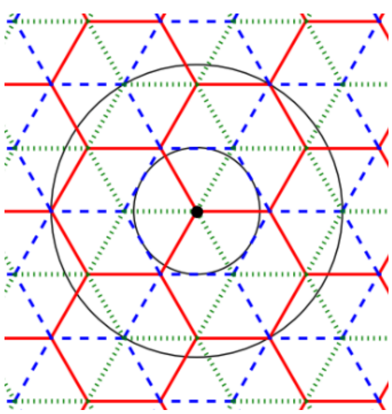

Figure 3. Circles with radii δ and $\gamma\delta$

In practice Alice will detect Bob as nearby when he is in one of the triangles marked X, and will learn nothing about Bob's location when he is not in this region. There is no region of unpredictable behavior, and there is no compromise of security.

The reason there is a gap between δ and $\gamma\delta$ is because the X region is not circular and not necessarily centered around Alice's location. This is clear when we look at Figure 3: for a user located as at the center — considering that she can be in any one of the six triangles incident at that point by moving an infinitesimal distance — the region that is guaranteed to be in her neighborhood is a hexagon, and the largest circle that can be inscribed in this hexagon has radius δ. Similarly, the region guaranteed not to be in her neighborhood is the complement of a bigger hexagon, and the smallest circle that can be exscribed in this hexagon has radius $\gamma\delta$.

Our choice of grid is the simplest one that works, because at least three overlapping grids are required (with only two grids, consider any point at which they intersect, then two people arbitrarily near this point but on opposite sides will not be detected as nearby).

The ideal functionality generally behaves as expected when invoked repeatedly over time, *i.e.*, no security is lost is lost by running multiple instances. The one exception is when a user is moving and crosses a grid boundary: a friend who is nearby will be able to observe this, and under assumptions about speed of movement, etc., may be able to infer the user's proximity to an edge, or more rarely, a vertex. It does not appear possible to eliminate this vulnerability.

3 Private Equality Testing

The private equality testing problem was studied in a number of papers [12, 32, 6, 26]. Here we describe two concrete protocols that are especially well suited for our purposes. They solve the following problem:

Input: Alice has value a representing her location. Bob has value b representing his location.

Output: Alice learns if $a = b$ and nothing else. Bob learns nothing.

We call this problem *asymmetric* equality testing because Alice learns the answer but Bob does not. This sort of asymmetry is often needed in social networks. Our first protocol is computationally more expensive than the second, but provides stronger security guarantees.

3.1 Protocol 1: Synchronous private equality testing

In this protocol the server is used only to forward messages between the two parties, and does not perform any computation. It is based on a mechanism of Lipmaa [26]; our contribution is a adaption for the asymmetric setting with an efficiency improvement.

The protocol has the following characteristics: (1) it is synchronous, *i.e.*, both parties need to be online (2) each party performs either 2 or 3 exponentiations, (3) there are two rounds, namely Alice sends a message to Bob (through the server) and Bob responds to Alice, (4) communication is about 40 bytes per edge per time interval using elliptic curves of size 160 bits (additional end-to-end encryption introduces a negligible overhead). It is secure against arbitrary collusion assuming the hardness of the standard Decision Diffie-Hellman problem. The protocol proceeds as follows.

Global setup: Let G be a cyclic group of prime order p and g a generator of G. We will assume that the Decision Diffie-Hellman problem is hard in G. All clients in the system are pre-configured with the same G and g. In what follows we will use $\mathbb{Z}_p$ to denote the set $\{0, \ldots, p-1\}$.

Client setup: When the system is first installed on the client it chooses a random x in $\mathbb{Z}_p$ and computes $h \leftarrow g^x$. Thus, Alice has x and h; h will be used as her ElGamal pubic key. We assume Bob already has Alice's public key h (more on this in Section 5.1).

Round 1: Alice computes an ElGamal encryption of her location a encoded as h^a and sends the resulting ciphertext to Bob (through the server). More precisely, Alice chooses a random r in $\mathbb{Z}_p$, computes

$$C_{\text{a}} \leftarrow (g^r,\ h^{a+r})$$

and sends C_{a} to Bob.

Round 2: Bob chooses a random non-zero s in $\mathbb{Z}_p$ and uses his own location b and the ciphertext $C_{\text{a}} = (g_1, g_2)$ from Alice to construct a fresh ElGamal encryption of the message $s \cdot (a-b)$. More precisely, Bob chooses a random t in $\mathbb{Z}_p$ and computes

$$C_{\text{b}} \leftarrow \left(g_1^s\, g^t,\ \ g_2^s\, h^{(t-sb)} \right)$$

Observe that setting $w := sr + t$ we have

$$C_{\text{b}} = \left(g^{sr+t},\ h^{s(a-b)+sr+t} \right) = \left(g^w,\ h^{s(a-b)+w} \right)$$

so that C_{b} is a fresh encryption of $s(a-b)$ under the public key h. Bob sends C_{b} to Alice through the server.

Obtain answer: Alice now has

$$C_{\text{b}} = (u_1, u_2) = \left(g^w,\ h^{s(a-b)+w} \right)$$

and her secret key x. She decrypts C_{b}, namely computes $m \leftarrow u_2/u_1^x = h^{s(a-b)}$. If $m = 1$ she concludes that $a = b$ and if not she concludes that $a \neq b$.

Security. The protocol above is an optimization of the generic mechanism of Lipmaa who provides a proof of security [26, Theorem 4]. We briefly sketch the argument: Alice's privacy is assured under the DDH assumption since the only thing she sends to Bob is an ElGamal encryption of her location. Bob's privacy is assured unconditionally since all Alice learns is $s(a-b)$ which is either 0 if $a = b$ or random non-zero in $\mathbb{Z}_p$ if $a \neq b$. When $a \neq b$ this reveals no other information about b.

With this protocol, it is possible for a malicious Bob to convince Alice that he is nearby even without knowing her location: he can simply choose a random s and send (g^s, h^s). While this is not a privacy violation (Bob cannot learn Alice's location or vice-versa), it is still important to keep in mind.

Performance. Since computing a product of exponents such as $g_1^s g^t$ is not much more expensive than computing a single exponent (see [30, p. 619]) we count these as a single exponentiation. Overall, Alice performs three exponentiations and Bob does two. Two of Alice's exponentiations use a fixed base which can be sped up considerably using pre-computations. The total traffic amounts to two ElGamal ciphertexts which is about 320 bytes using 160-bit elliptic curves.

3.2 Protocol 2: Fast asynchronous private equality test with an oblivious server

Our second private equality test, shown in Figure 4, is novel and requires far less communication and computation, but is only secure assuming the server does not collude with either party. The server learns nothing at the end of the protocol. The reason for the performance improvements is that this protocol uses three parties (Alice, Bob, and server) and is therefore able to rely on information theoretic methods such as secret sharing.

We describe the protocol as it would be used in our system, namely at time t Alice is in location a_t and Bob is in location b_t. At certain times t Alice wants to test if $a_t = b_t$, but should learn nothing else about b_t. The server and Bob should learn nothing.

Global setup: Let p be a prime so that all possible locations are in the range $[1, p]$. If location data is 32 bits then one can set $p := 2^{32} + 15$. All clients in the system are pre-configured with the same p. As before we let $\mathbb{Z}_p$ denote the set $\{0, \ldots, p-1\}$.

Client setup: When Alice and Bob declare themselves as friends they setup a shared secret key which we denote as k_{ab}. We explain key setup in more detail in Section 5.1. Both clients also maintain a counter ctr which is initially set to 0 (there is a separate counter for each pair of clients). When Alice and Bob sign up for the service we assume that

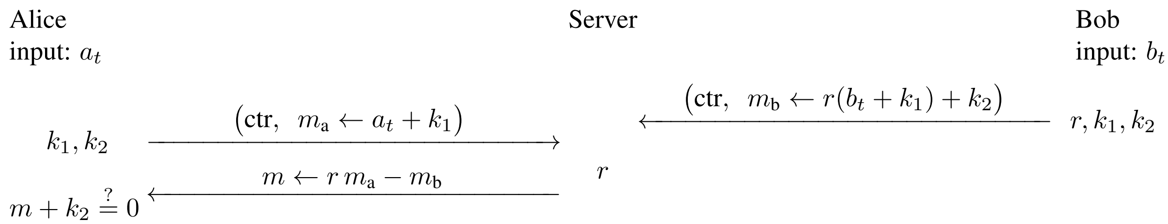

Figure 4. Asynchronous equality test with an oblivious server

they generate a secret key with the server denoted k_a and k_b respectively.

The keys $k_\text{ab}, k_\text{a}, k_\text{b}$ will be used as keys to a Pseudo Random Function (PRF) denoted by $F(k, x)$, where k is the PRF key and x is the point at which the function is evaluated. (Our implementation uses AES as the PRF.) All communication between Alice and the server and Bob and the server is encrypted with an authenticated encryption scheme.

Step 1 (Bob's message): Bob increments ctr by one and computes

$$(k_1, k_2) \leftarrow F(k_\text{ab}, \text{ctr}) \quad \text{and} \quad r \leftarrow F(k_\text{b}, \text{ctr})\ .$$

Bob parses the result so that k_1, k_2 are elements in $\mathbb{Z}_p$ and r is an element in $\mathbb{Z}_p \setminus \{0\}$ (i.e. $r \neq 0$). Bob computes $m_\text{b} \leftarrow r(b_t + k_1) + k_2$ and sends m_b and ctr to the server.

Step 2 (Alice queries server): This step consists of two rounds: first Alice queries the server to obtain the latest value of ctr from Bob. If the value received is not fresh, *i.e.*, Alice used it before, she aborts.

Alice then computes $(k_1, k_2) \leftarrow F(k_\text{ab}, \text{ctr})$. Alice parses the result so that k_1, k_2 are elements in $\mathbb{Z}_p$ and sends $m_\text{a} \leftarrow a_t + k_1$ and ctr to the server.

Step 3 (Server responds to Alice): The server finds a message from Bob that has the same counter value ctr as the message from Alice. It computes $r \leftarrow F(k_\text{b}, \text{ctr})$ and parses the result as an element in $\mathbb{Z}_p \setminus \{0\}$. It sends to Alice the message

$$\begin{aligned} m &\leftarrow r\, m_\text{a} - m_\text{b} \\ &= r(a_t + k_1) - r(b_t + k_1) - k_2 = r(a_t - b_t) - k_2 \end{aligned}$$

Alice computes $m + k_2 = r(a_t - b_t)$. If the result is 0 then she knows $a_t = b_t$ and otherwise not.

Security. We show that the protocol is secure (i.e. no party learns more than it should) as long as no two parties collude. First, since F is a secure Pseudo Random Function, the outputs k_1, k_2, and r at every iteration are indistinguishable from truly random and independent values in $\mathbb{Z}_p$ with $r \neq 0$.

Now, observe that Bob learns nothing from the protocol since he receives no messages. The server sees m_a and m_b and both are independent of the users' inputs a_t and b_t. To see why, recall that m_a is blinded by k_1 and m_b is blinded by k_2. Since k_1 and k_2 are independent random values in $\mathbb{Z}_p$ so are m_a and m_b. Therefore, the server's view in this protocol can be easily simulated by two random elements in $\mathbb{Z}_p$.

Finally, Alice learns $m = r(a_t - b_t) - k_2$ for some a_t of Alice's choice. Consider the case $a_t \neq b_t$. Since r is uniform in $\mathbb{Z}_p \setminus \{0\}$ and unknown to Alice this m is uniformly distributed in $\mathbb{Z}_p \setminus \{k_2\}$ in Alice's view. Therefore, when $a_t \neq b_t$, Alice's view of the server's response can be simulated by one random element in $\mathbb{Z}_p \setminus \{k_2\}$.

Note that if Alice colludes with the server they easily learn Bob's location. Similarly, if Bob colludes with the server they easily learn Alice's location.

However, unlike in Protocol 1, Bob cannot trick Alice into thinking he is nearby without correctly guessing her location; likewise, the server cannot trick Alice into thinking that Bob is nearby without guessing the difference of their locations.

Performance. The entire protocol requires a small number of additions and one multiplication mod p plus two evaluations of AES. The total message traffic is 12 bytes per edge assuming p is 32 bits. Message encryption adds a few bytes to every message.

Comparing the two protocols. We presented two private equality testing protocols. Which protocol to use depends on the level of trust in the server and the clients' computing power. The first protocol makes no use of a server, but is computationally expensive and requires the two users to communicate synchronously. The second protocol is asynchronous (Bob need not be online when Alice queries the server) which fits better with our application.

In spite of the asynchronicity, due to the need for a counter, each message from Bob can be used in one protocol execution by Alice, which means that we cannot use the optimization where Bob only sends messages when he is moving. In the appendix we present an asynchronous variant of Protocol 1 using an oblivious server (synchrony is not inherent in Protocol 1). The protocol has only one round of communication between Alice and the Server (Protocol 2 has two such rounds). This variant is identical to Protocol 1 in terms of computational efficiency (per invocation). The security properties are similar to protocol 2: a collusion between Alice and the server reveals Bob's location.

All the protocols are asymmetric since only Alice learns the answer. They can be made symmetric (if needed) by executing the protocol twice, once in each direction.

4 Location tags

Location tags were first studied by Qiu et al [36]. For our purposes, a *location tag* is a secret associated with a point in space and time. It is a collection of *location features* derived from (mostly electromagnetic) signals present in the physical environment.

Location tagging is a procedure to extract the tag from a point in space-time, together with a comparison or matching function. The matching function can be based on Hamming distance, set distance, etc. The two key properties are:

Reproducibility. If two measurements at the same place and time yield tags τ and τ', then τ and τ' match with high probability. Note that they need not be equal as strings.

Unpredictability. An adversary not at a specific place or time should be unable to produce a tag that matches the tag measured at that location at that time.

To fix intuition, a single location feature can be thought of as having around 10 bits of entropy, whereas the predictability of a location tag is much lower, say 2^{-64}.

Location tags provide a different model for proximity testing. The main advantage is that since the location tags of the two parties need to match, spoofing the location is no longer possible, which stops online brute force attacks.

The main disadvantage is that users no longer have control over the granularity of proximity: the notion of neighborhood is now entirely dependent on the type of location tag considered. For example, with WiFi packets, the neighborhood is defined by the range of the wireless network in question. To further exemplify, we found that three contiguous buildings at our university belonging to the CS and EE departments comprise one "virtual LAN," and form one neighborhood in our system.

Some types of location tags we consider below are more time-varying than others. Note that the location tags studied in the original paper [36] are required to be time invariant. The authors take extra steps to make the tag stable over time. In our settings, the time-varying nature of location tags is necessary for security (otherwise an attacker can record location tags in advance).

Protocol	Device 1	Device 2	Common
ARP	1088	1071	832
BROWSER	262	286	255
DHCP	249	237	208
MDNS	600	551	541
NBNS	1134	1190	1117
All	3333	3335	2953

Table 1. Packet counts

4.1 Constructing location tags

Now we discuss several possible ways to extract location tags. We have experimentally studied the first method and show it to be well suited as a source of location tags. The remaining methods depend on additional features on the phone and we point to papers that study their viability as location tags.

WiFi: broadcast packets. The contents of the WiFi traffic offer a rich potential for extracting location tags. Broadcast packets comprise a variety of different protocols. The source and destination IP addresses, sequence numbers of packets, and precise timing information all offer varying degrees of entropy.

We performed an experiment on the Stanford university campus to test the rate and quality of location features that can be extracted. To do so, we ignored the traffic that is not restricted to the local network. For example, TCP packets likely originate from external networks, and so an attacker might be able to predict or control the contents of some of those packets, so we ignore them. Of the protocols carrying purely local traffic, we restricted ourselves to the top 5 protocols by number of packets: 'ARP', 'BROWSER',[4] 'DHCP', 'MDNS', and 'NBNS'. We further limited ourselves to broadcast packets; capturing all packets in "promiscuous mode" would yield many more packets if this mode is supported by the device.

We logged the packets thus filtered from two different devices over a period of 223 seconds and compared them. We received around 3,300 packets on each, for a rate of around 15.0 packets/sec. Table 1 summarizes the degree of similarity between the two logs.

As we can see, around 90% of the packets are common to both devices. Based on the volume of traffic, we can derive meaningful location tags within 2-3 seconds.

According to a simple histogram estimator, the entropies of the various fields are 6.9 ± 0.1 bits for the source address, 1.9 ± 0.1 bits for the destination address, 2.07 ± 0.01 bits for the protocol identifier and 7.6 ± 0.1 bits for the content of the packet. These measurements were computed from 4 samples of 1,000 packets each. The numbers are likely

[4] This refers to Microsoft SMB server and not to web browsing activity.

underestimates since the histogram estimator has negative bias; on the other hand, the total entropy is less than the sum due to correlations between the attributes. It appears reasonable to conclude that each location feature has an entropy of at least 10 bits under traffic conditions similar to those in our experiment.

One problem with using WiFi packets for location tags is that both users need to use the same wireless network. One way to get around this, if the device has multiple interfaces, to execute the protocol multiple times, once for each network. Heuristics such as picking the network with the highest signal strength or the network with the alphabetically lowest MAC address among the networks with signal strength greater than a threshold might be sufficient in practice.

GPS. The Global Positioning System works by timing signals sent by a number of orbital satellites. GPS consists of several signals, including "$P(Y)$" (encrypted precision code) and "M-code" (military). Unlike the civilian code, military codes are designed to be unpredictable without a secret key. Recent work by Lo et al. [27] shows that the composition of $P(Y)$ or M signals from four or more satellites forms a secure location tag that is temporally and spatially unpredictable by design, as needed for our application. These tags would have a configurable granularity and are therefore ideal for our settings.

Since these codes run at a higher frequency than the civilian code (to provide better accuracy), current commercial GPS receivers are too slow to measure them. Lo et al. argue that the next generation Broadcom GPS receivers will have the ability to measure at these higher rates making GPS-based location tags practical for mobile devices.

GSM. In cellular networks, various (GSM) cell towers are in range at any one time. Each tower has space- and time-varying characteristics such as signal strength. This seems like a promising location tag and we leave it for future work to confirm its reproducibility and unpredictability.

Bluetooth. Like WiFi IDs, Bluetooth IDs are unique to each device. They have the advantage over WiFi IDs of almost always being associated with mobile rather than fixed devices, making Bluetooth-based location tags more time-variable. One concern with Bluetooth is that the range may be too small to be very useful. We therefore believe that Bluetooth is not a good source for location tags.

Audio. Audio might be useful in certain limited circumstances to extract location features. For example, music playing in the background in a coffee shop or a talk in a conference room. Acoustic fingerprinting is a well-known technique to extract features from audio in a way that is robust to transformation of the audio such as compression and adding noise. It is used as a way to identify an audio signal from a database of tracks (e.g., in the mobile application Shazam). Similar feature extraction techniques can provide part of a location tag.

Atmospheric gases. The cell-phone-as-sensor project has experimented with CO, NOx and temperature sensors on cell phones and hopes this will become standard on smartphones so that it can be used for real-time pollution mapping and other applications [20, 34]. If these sensors ever become mainstream, they are another potential source of location tags.

4.2 Proximity testing using location tags

One way to formulate the underlying cryptographic problem of private proximity testing using location tags is *private set intersection*. Alice and Bob have sets A and B respectively and wish to privately compute $|A \cap B|$. A and B represent location tags; the players conclude that they are nearby if the size of the intersection exceeds a threshold t.

Private set intersection is a well studied problem and an elegant solution was given by Freedman, Nissim and Pinkas [13]. Other approaches to private set intersection are given in [18, 21, 22, 19]. The protocol of Freedman, Nissim and Pinkas makes use of a homomorphic encryption scheme E that anyone can encrypt and Alice can decrypt. At the end of the following protocol, Alice learns $|A \cap B|$ and Bob learns nothing:

- Alice interpolates a polynomial p whose set of zeroes is the set A.
- Alice sends $E(p)$ to Bob, i.e., the sequence of encrypted coefficients
- Bob evaluates $E(p(b))$ for each $b \in B$.
- For each $b \in B$, Bob picks a random r and computes $E(rp(b))$
- Bob sends to Alice a permutation of the encryptions computed in the previous step
- Alice decrypts each value she receives; she outputs the number of nonzero decryptions as $|A \cap B|$.

Note that there are homomorphic encryption schemes where it is possible to multiply inside encryption by a constant using $O(1)$ modular exponentiations. To see that this protocol works, observe that $rp(b)$ is zero if $b \in A$ and a random nonzero value otherwise.

This protocol has two disadvantages in our setting. First, it requires $\theta(|A| \cdot |B|)$ modular exponentiations ($E(p(b))$ can be evaluated using Horner's rule using $O(|A|)$ modular exponentiations, and there are $|B|$ such encryptions to compute). Second, it is only secure against semi-honest players. There is a version that handles malicious players, but it is significantly less efficient. More recent protocols can be more efficient [18, 21, 22, 19].

More importantly, revealing the size of the intersection can be a privacy breach. This is of course a weakness of the problem formulation rather than the protocol. It allows Alice to carry out an *online brute force attack* by incorrectly reporting her input set A. This can be a problem when the individual location features have low entropy.

To avoid this weakness, we formulate the problem differently. Specifically, we consider the following relaxation of *private threshold set intersection*: Alice and Bob have sets A and B respectively of n elements each and wish to determine if $|A \cap B| \geq t$. If the condition does not hold, then the parties should learn nothing except the fact that $|A \cap B| < t$. But if it does hold, then no privacy is required: the values of A and B can be leaked. Except for the relaxation of the privacy requirement when $|A \cap B| > t$, the formulation is similar to private threshold set intersection. Note that in our settings the quantities n and t are globally fixed since if they were allowed to depend on the location, that might itself be a privacy leak. Freedman, Lindell and Pinkas [13] provide a solution that always keeps A and B private (even when $|A \cap B| \geq t$) using generic Yao circuit evaluation which is too slow for our purposes.

We construct an efficient protocol for relaxed private threshold set intersection. Security depends on the randomness of the participants' input distribution (i.e. the randomness of location tags). More precisely, let X denote the domain of location tags. We need the following properties:

- If Alice and Bob are "nearby," we assume that A and B are sampled as follows: for some $t' \geq t$, $A = C \cup A'$ and $B = C \cup B'$ where $C \in_R X^{t'}, A \in_R X^{n-t'}, B' \in_R X^{n-t'}$. This means that their location tags are random subject to the constraint that at least t of them match.
- Alice and Bob are "apart," A and B are sampled as before for some $t' < 2t - n$.

Observe that there is a gap (of $n - t$) between the two conditions above. The protocol makes no guarantees on correctness or security when neither the "nearby" nor the "apart" condition holds. This is to be expected, and is analogous to the approximation factor in the grid-based protocols. If we imagine the predictability of location tags increasing gradually from 0 to 100% as one approaches the target location, then it translates to a gap between the maximum distance at which we can guarantee detection and the minimum distance at which we can guarantee privacy.

We now describe a protocol for the asymmetric version of relaxed private threshold set intersection. Here it is Bob who obtains the answer while Alice learns nothing.

Protocol 3.

- Alice encodes her input as a set P of points $\{(p_1, x_1), (p_2, x_2) \ldots (p_n, x_n)\}$ where $p_i \in \mathbb{F}$ and $x_i \in \mathbb{F}$. Similarly Bob encodes his input as a set $Q = \{(q_1, y_1), (q_2, y_2) \ldots (q_n, y_n)\}$.
- Alice constructs a polynomial p of degree $n-1$ defined by the points P. Alice picks a random set of points R on p such that $R \cap P = \{\}$ and $|R| = 2(n - t)$.
- Alice sends R to Bob.
- Bob attempts to find a polynomial p' of degree $n - 1$ that passes through at least $2n-t$ of the points in $Q \cup R$. If he is successful, he outputs 1 otherwise he outputs 0.

Correctness. We state a generalized form of the Berlekamp Massey decoding algorithm from [24]:

Theorem 1 (Berlekamp-Massey decoding) *There is an algorithm BM such that, given k pairs (x_i, y_i) over a field $\mathbb{F}$ and a degree parameter d*

- *if there exists a polynomial p that passes through at least $\frac{k+d}{2}$ of the points, BM outputs p*
- *otherwise BM outputs $\perp$*

The proof of correctness now continues.

Case 1. When Alice and Bob are nearby, there are at least $t + 2(n - t) = 2n - t$ points on the polynomial. Substituting $k = n + 2(n - t) = 3n - 2t$ and $d = n - 1$ in Theorem 1, we find that the condition is satisfied, and therefore Bob succeeds in finding a polynomial.

Case 2. When Alice and Bob are far apart, we start with the observation that $|A' \cap B'| < n - t$ w.h.p. (specifically, with probability at least $1 - (\frac{n^2}{N})^{n-t}$). This is a matter of bounding probabilities and we omit the proof. By construction this implies $|A \cap B| < t$. This means that there are fewer than $2n - t$ points on the polynomial p, and by Theorem 1, Bob will fail to find an appropriate polynomial.

This completes the proof of correctness. ■

Security. We show that when Alice and Bob are far apart, Alice has information theoretic security. We argue as follows: let us give Bob some auxiliary information: specifically, which of his location features are common with Alice. In spite of this extra information, Bob has $2(n-t)+t' < n$ points from a polynomial of degree $n - 1$, and therefore his view of the protocol is statistically indistinguishable from random.

Using the location itself. As presented, this protocol does not use the location co-ordinates. This might be desirable in practice due to the poor coverage of GPS in indoor environments (where location tags are abundant), or to enable location features on devices such as laptops that do not have GPS capabilities. Alternately, the protocol could be modified to treat the x and y location co-ordinates as two of the location features.

Collusion resistance. This protocol is not secure when multiple instances are run with the two sets of inputs being dependent. Therefore, only location tags that are completely time-varying (i.e., independent in each epoch) will work. Among the types of tags that we have empirically studied, that boils down to WiFi broadcast packets. It also means that the protocol needs to be run synchronously, even though there is only a single message sent.

To achieve collusion resistance, each of Alice's executions of the protocol with her different friends must use different location tags. Given the abundance of broadcast packets in our experiment[5], this is feasible. We can use a hash function to ensure that in each instance a random subset of location features are used. This would work as follows:

The protocol instance between players i and j uses only location features f for which $H(i, j, \eta, f) = 0$ where η is the epoch and H is a hash function modeled as a random oracle with range $\{1, \ldots k\}$, k being (say) 20. This way, if Bob and Carol collude against Alice, Alice is still safe because on average only $\frac{1}{20}$ of the features used in the two protocol instances are common.

However this protocol is not cryptographically composable; therefore if sufficiently many (specifically, $\Omega(k)$) friends collude then security will be lost.

5 Implementation

We have built a prototype implementation on the Android platform. The proximity detector is a "background activity" that alerts the user when one or more friends are nearby. Protocols 1 and 2 have been implemented. In this section we describe some of the implementation details.

On the client we use Bouncy Castle, the standard cryptography toolkit for Android. Unfortunately, Bouncy Castle does not yet support Elliptic Curve Cryptography, so we implemented Protocol 1 over Z_p^* with a length of 1024 bits. We implemented our own server in Python. The server acts as a router in protocol 1; in either protocol, there is no big integer arithmetic on the server.

Grid. We allow the grid size to be user-configurable with pre-defined defaults of $10m$, $100m$ and $1000m$. There is no conflict is the grid sizes of a pair of users don't match. The protocols are asymmetric, and in each instance of the protocol the "receiver" quantizes his location according to the "sender's" grid size. Since the sender gets no information at the end of the protocol, the receiver can do so without worrying about security.

Since the Earth is round and the tessellation in discussed Section 2.5 assumes a plane world, we divide the Earth into strips (corresponding to 1° latitude each). The curvature of the Earth within a single strip is small enough to ignore. This allows us to keep the size of the grid cells approximately the same everywhere, but there is a small error probability at the boundary between two of the strips (because the cells on either side of the strip don't line up).

[5] If there are 300s in an epoch, the number of usable location features is around 4,500.

Synchronization. Recall that protocol 1 needs to run every few minutes (epochs) in a synchronized manner. The epoch duration in our implementation is fixed globally at 5 minutes.

All communication is through HTTP. We chose to eschew a "push" infrastructure such as Comet and instead adopted the following solution. For each round, each device first performs computations and sends the results to the server; then it waits for a fixed time interval (30s) from the start of the round for the other devices to finish their send steps. Then the device downloads the data from the server that it is supposed to receive in that round.

5.1 SocialKeys: key exchange over social networks

Conventional PKI using certificate authorities is too heavyweight for the needs of most users. A well known usability evaluation of PGP in 1999 concluded that it is nearly unusable for the majority [48]. While the software has improved since then, the underlying architectural and design issues remain.

SocialKeys embraces the idea that public keys must be associated with the digital identities that people already own and use, i.e., their social network accounts, rather than requiring the creation of new identities for cryptographic purposes. While this is not a new idea, we go one step further by enabling such an association for existing social networks, even though none of them support such a capability. We achieve this not by relying on a third-party service, but rather by repurposing social network features in unintended ways.

By offloading the establishment of trust between users to the social network, SocialKeys obviates the need for a "key manager". As a consequence, it is almost completely transparent to the user.

Currently we have implemented key distribution over Facebook as well as an Android "service" that exposes an API that any SocialKeys-aware applications can use. Support for other popular social networks as well as extensions for Firefox (to enable secure web mail, including Facebook messages) and Pidgin (to enable secure instant messaging) are potential future extensions. We are exploring the possibility of tighter integration with a couple of social networks. That said, the SocialKeys architecture is interoperate with OpenID or any other social network.

While SocialKeys aims to be more user friendly than a traditional PKI, it is probably not as secure. The two main weaknesses are trust in the identities represented by social networking profiles and the difficulty of key revocation.

Verifying the identity of a social profile is not foolproof [46] and this remains a topic of research [45]. However, in the context of a social network-based application that already relies on the trustworthiness of friends' profiles, we do not lose security by leveraging the same social network for key distribution.

Architecture of client application

- On the user's first interaction with SocialKeys, the client application generates a public/private key pair. The user is asked for a password; in addition to this password, they are also presented with three random dictionary words to memorize. Each dictionary word has around 16 bits of entropy, and therefore we can get a 80-bit security level starting with a reasonably strong password with around 32 bits of entropy. The password is generated according to PKCS 5 (RFC 2898) [25], with an iteration count of 1000.

  To set up SocialKeys on a new device, the user reenters the password and dictionary words from their first setup. This ensures that the same key pair is reproduced.
- The public key can be uploaded to the social network in one of two ways: 1. directly, encoded as a URL; 2 by pointing to a key server. Currently only the first method is implemented. An example of such a URL is given below:

  ```
  https://socialkeys.org/pubkey?alg=DH
  &keylen=1024&p=oakley&g=2&key=LlI+lKCAIE...
  ```

  The latter approach is more complex to implement but has some advantages in that it is more extensible and avoids URL length limits. We decided that it was overkill for our purposes.
- Most social networks allow the user to specify one or more "websites" in her profile. We make use of this feature to store the key.
- The client application lives as an Android background process and exposes an API that allows it to receive the identity of any Facebook user (or any supported identity). On receiving an identity the client downloads the public key of that identity and computes and returns a shared secret key.
- Once an identity has been seen by SocialKeys it is periodically polled and the key is kept up to date. Currently this is the only revocation mechanism. The Facebook API allows retrieving the public keys of all of one's friends with a single query, so it ends up being quite efficient in practice.

5.2 Evaluation

Performance. We calculated the CPU time required to run Protocol 1 and Protocol 2 with 100 (mutual) friends, i.e., the protocol was executed in both directions. Protocol 1 took 46.2 ± 1.4 seconds and Protocol 2 took 3.0 ± 0.3 seconds. The device we used was a Motorola Droid, with a ARM Cortex-A8 500MHz processor.

Note that the epoch for Protocol 1 is 5 minutes, so we can easily handle several hundred friends (although the battery life may be a concern; we have not tested this aspect.) Also, we have not implemented any of the optimizations described in Section 3.1 for computing exponentiations. Furthermore, as mentioned earlier Elliptic Curve Cryptography libraries are not yet available on the Android platform. Once we are able to switch to ECC, we expect to be able to see a significant speedup. For Protocol 2, the CPU load is far lower.

6 Discussion

Proximity testing in the peer-to-peer model. The previous discussion describes proximity testing using a client-server model, but a peer-to-peer model might also yield interesting results. For example, each node could simply broadcast its identity to all neighboring nodes. If the broadcast is in the clear then this scheme allows proximity testing between complete strangers, but provides no privacy whatsoever. With a suitable encryption scheme, however, it can provide private proximity testing between friends only. Because this approach is passive on the receiving end, it would use less bandwidth than a server-based approach.

However, there are some notable difficulties compared to a server-based approach. First, it is difficult to control the granularity of proximity testing; granularity is simply equal to the broadcast radius on the channel used and it is not possible to implement different granularities for different friends. More importantly, such a peer-to-peer mechanism has a significant security drawback. Imagine an attacker that records and uploads all the broadcasts from a particular area. Alice can use this data to see if any of her friends passed through that area (and when), even though she was not there herself.

Thus, a peer-to-peer broadcast might serve as a nice complement to the client-server protocols we have developed — users might be comfortable broadcasting in some situations but not in others. Due to the challenges with this design we do not pursue it here.

Leaking the result of proximity testing to the server. In the protocol of Section 3.2, the server remains oblivious to the result of the proximity test. It is tempting to design protocols in which the server does learn the result, because it can be made even more efficient.

However, the outcome of pairwise proximity testing can be much more revealing than is at first apparent, especially when combined with *auxiliary information*. Suppose the server has auxiliary information about the work locations of most of the users. Then it can match the proximity-testing adjacency graph with the adjacency graph of known work locations to essentially learn the locations of all the users. Highly robust algorithms for matching/inferring such graphs are known [33, 11].

7 Related Work

Location privacy. A non-technical discussion of the issues around location privacy is provided by Blumberg and Eckersley [5].

The work closest to ours is perhaps by Zhong, Goldberg and Hengartner who study three protocols for privacy-preserving proximity detection in the two party setting [51]. The main difference is that they require the users to learn the mutual distance, which necessitates computationally expensive cryptography. Their work builds on Atallah and Du's study of secure multi-party computational geometry [2].

A privacy-aware friend locator was studied in [47]. Unfortunately the technique used therein has many flaws including the fact that the server always learns the result of proximity testing.

There is a large body of work on using anonymization for location privacy. The work of Gruteser and Grunwald [17] kicked off a long line of papers in this area. Another seminal work is by Beresford and Stajano who introduced "mix zones" [4]. For examples of more recent work, see [23, 28, 44].

This approach has several potential limitations including the highly identifying nature of location information [16] and the limited location resolution resulting from the obfuscation or quantization needed for anonymization. At any rate, since the application we have in mind is fundamentally social in nature, pseudonymous identities are not suitable.

Ghinita et al. show how to use Private Information Retrieval in Location-based services such as search [15].

Location tags were introduced by Qiu et al. in [37, 36]. There were significant differences from our use of location tags: they studied signals such as Loran which are not available on consumer devices, and they require location tags to be stable with time.

There is a body of work on location-based encryption. Some of it assumes the existence of tamper-proof ("anti-spoof") hardware [43]. Other work such as [41] is more rigorous and involves securely verifying the location of the receiver based on timing or strength of electromagnetic signals.

Many papers have studied security and privacy issues in vehicular communications. Ad-hoc networks of proximate vehicles have been studied in [50, 39]. Another line of work aims to mitigate the privacy issues in tracking and monitoring of vehicles; see for example [35].

PET and Private set intersection. Fagin, Naor and Winkler discuss numerous protocols for PET with a focus on protocols that can be executed by humans [12].

Freedman, Nissim and Pinkas described a private set intersection protocol based on homomorphic encryption [13]. For the special case where each party's input is a singleton set, this yields a protocol with $O(1)$ modular exponentiations and communication of a constant number of field elements, assuming Paillier encryption is used. Many other papers have studied private set intersection: [10, 18, 9, 7, 21, 22, 8, 19]. Juels and Sudan construct "fuzzy vaults" which is a related primitive [24]. Our protocol 3 has similarities to their construction.

Cryptography and social networking. The intersection of cryptography and social networking does not appear to be well-studied in the academic literature. The one major work we are aware of in this area is Persona [3], which is a system that gives users fine-grained control of privacy policies using attribute-based encryption.

FaceTrust is a system for users to prove their identity and attributes in a peer-to-peer manner [45], reminiscent of the Web of Trust paradigm [1]. This kind of system can be complementary to SocialKeys because user trust in their friends' online identities is important in the absence of a certificate infrastructure.

A Facebook application called My Public Key allows users to associate a public key with their Facebook account [29]. Of course, the interoperability is limited to Facebook users who also install this application.

There is a draft specification for OpenID key discovery [31]. It appears not to be final and there is no actual support from any vendors yet.

8 Conclusion

Location privacy is an important and growing concern in the real world today. Even problems that appear simple such as privacy-aware proximity testing can prove to be surprisingly tricky. In contrast to previous work, we studied this problem on a firm theoretical footing and presented a variety of cryptographic protocols motivated by and optimized for practical constraints. While we have built a prototype implementation, it remains to be seen if any vendors of location-based services will deploy cryptographic systems in the market.

Several of the techniques we came up with may be of independent interest. Asymmetric private equality testing is a versatile primitive, and in Sections 3.1 and 3.2 we provided new protocols and improvements over previous work. Finally, SocialKeys, our approach to managing user public keys via a social network, is likely to have many applications beyond location privacy.

9 Acknowledgments

Di Qiu deserves special thanks for discussions about location tags, and Elie Bursztein for pointing out problems with protocols where the server learns the result of proximity testing. Frank Wang and Kina Winoto provided invaluable help with the implementation. We are also grateful to thank Andrew Bortz, Aaron Johnson, Ann Kilzer, Ananth Raghunathan, Nitesh Saxena and the anonymous reviewers for comments on a draft, and Eric Lam, David Molnar, Hart Montgomery, Ram Ravichandran, Elaine Shi and Vinod Vaikuntanathan for useful discussions.

References

[1] A. Abdul-Rahman and S. Halles. A distributed trust model. In *New security paradigms*, pages 48–60. ACM, 1997.

[2] M. J. Atallah and W. Du. Secure multi-party computational geometry. In *International Workshop on Algorithms and Data Structures*, pages 165–179. Springer-Verlag, 2001.

[3] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: an online social network with user-defined privacy. *SIGCOMM Comput. Commun. Rev.*, 39(4):135–146, 2009.

[4] A. R. Beresford and F. Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 2(1):46–55, 2003.

[5] A. J. Blumberg and P. Eckersley. On locational privacy, and how to avoid losing it forever. `http://www.eff.org/wp/locational-privacy`.

[6] F. Boudot, B. Schoenmakers, and J. Traore. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111:23–36, 2001.

[7] J. Camenisch and G. Zaverucha. Private intersection of certified sets. In *Proc. of Financial Crypto 2009*, volume 5628 of *LNCS*, pages 108–127, 2009.

[8] E. D. Cristofaro and G. Tsudik. Practical set intersection protocols with linear complexity. In *Proc. of Financial Crypto 2010*, volume 6052 of *LNCS*, pages 143–159, 2010.

[9] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient robust private set intersection. In *Applied Cryptography and Network Security (ACNS)*, pages 125–142, 2009.

[10] L. K. Dawn and D. Song. Privacy-preserving set operations. In *Proc. of Crypto*, pages 241–257. Springer, 2005.

[11] N. Eagle, A. S. Pentland, and D. Lazer. Inferring friendship network structure by using mobile phone data. *Proceedings of the National Academy of Sciences*, 106(36):15274–15278, August 2009.

[12] R. Fagin, M. Naor, and P. Winkler. Comparing information without leaking it. *Comm. of the ACM*, 39:77–85, 1996.

[13] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proc. of Eurocrypt'04*, pages 1–19. Springer-Verlag, 2004.

[14] G. Friedland and R. Sommer. Cybercasing the joint: On the privacy implications of geotagging. In *Proceedings of the Fifth USENIX Workshop on Hot Topics in Security (HotSec 10)*, 2010.

[15] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private queries in location based services: anonymizers are not necessary. In *SIGMOD Conference*, pages 121–132, 2008.

[16] P. Golle and K. Partridge. On the anonymity of home, work location pairs. In *Proceedings of the 7th International Conference on Pervasive Computing*, pages 390–397. Springer, 2009.

[17] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *MobiSys '03*, pages 31–42. ACM, 2003.

[18] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of cryptography (TCC)*, pages 155–175, 2008.

[19] C. Hazay and K. Nissim. Efficient set operations in the presence of malicious adversaries. In *Proc. of public key crypto (PKC)*, volume 6056 of *LNCS*, pages 312–331, 2010.

[20] R. Honicky, E. A. Brewer, E. Paulos, and R. White. N-smarts: networked suite of mobile atmospheric real-time sensors. In *NSDR '08*, pages 25–30. ACM, 2008.

[21] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *Proc. of Theory of Cryptography*, pages 577–594, 2009.

[22] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *Proc. of Security and Cryptography for Networks (SCN'10)*, volume 6280 of *LNCS*, pages 418–435, 2010.

[23] T. Jiang, H. J. Wang, and Y.-C. Hu. Preserving location privacy in wireless lans. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 246–257, New York, NY, USA, 2007. ACM.

[24] A. Juels and M. Sudan. A fuzzy vault scheme. *Designs Codes and Cryptography*, 38(2):237–257, 2006.

[25] B. Kaliski. PKCS 5: Password-based cryptography specification. `http://www.apps.ietf.org/rfc/rfc2898.html`.

[26] H. Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *Proc. of Asiacrypt*, pages 416–433, 2003.

[27] S. Lo, D. D. Loenzo, P. Enge, and P. Bradley. Signal authentication: A secure civil gnss for today. *Inside GNSS Magazine*, 4(5), October 2009. `www.insidegnss.com/node/1633`.

[28] J. Manweiler, R. Scudellari, Z. Cancio, and L. P. Cox. We saw each other on the subway: secure, anonymous proximity-based missed connections. In *HotMobile '09: Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, pages 1–6, New York, NY, USA, 2009. ACM.

[29] R. McGeehan. My public key (facebook application). `http://www.facebook.com/apps/application.php?id=7923770364`.

[30] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC, 1996.

[31] G. Monroe and C. Howells. Openid service key discovery 1.0 - draft 01. `http://openid.net/specs/openid-service-key-discovery-1_0-01.html`, 2006.

[32] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proc. of symposium on the theory of computing (STOC)*, pages 245–254, 1999.

[33] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *IEEE security and privacy*, pages 173–187, 2009.

[34] D. Pescovitz. Cell phone as sensor. `http://coe.berkeley.edu/labnotes/0805/honicky.html`.

[35] R. A. Popa, H. Balakrishnan, and A. Blumberg. VPriv: Protecting Privacy in Location-Based Vehicular Services. In *18th USENIX Security Symposium*, Montreal, Canada, August 2009.

[36] D. Qiu, D. Boneh, S. Lo, and P. Enge. Robust location tag generation from noisy location data for security applications. In *The Institute of Navigation International Technical Meeting*, 2009.

[37] D. Qiu, S. Lo, P. Enge, D. Boneh, and B. Peterson. Geoencryption using loran. In *The Institute of Navigation International Technical Meeting*, 2007.

[38] R. Ravichandran, M. Benisch, P. G. Kelley, and N. M. Sadeh. Capturing social networking privacy preferences: Can default policies help alleviate tradeoffs between expressiveness and user burden? In *PETS*, pages 1–18, 2009.

[39] M. Raya and J.-P. Hubaux. The security of vehicular ad hoc networks. In *SASN '05: Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, pages 11–21. ACM Press, 2005.

[40] P. F. Riley. The tolls of privacy: An underestimated roadblock for electronic toll collection usage. *Computer Law and Security Report*, 24(6):521 – 528, 2008.

[41] N. Sastry, U. Shankar, and D. Wagner. Secure verification of location claims. In *WiSe '03: Proceedings of the 2nd ACM workshop on Wireless security*, pages 1–10. ACM, 2003.

[42] J. Scheck. Stalkers Exploit Cellphone GPS, 2010. Wall Street Journal.

[43] L. Scott and D. E. Denning. A location based encryption technique and some of its applications. In *Institute of Navigation National Technical Meeting*, pages 734–740, 2003.

[44] P. Shankar, V. Ganapathy, and L. Iftode. Privately querying location-based services with sybilquery. In *UbiComp'09: Proceedings of the 11th International Conference on Ubiquitous Computing*, pages 31–40, Orlando, Florida, September/October 2009. ACM Press, New York, NY, USA. http://doi.acm.org/10.1145/1620545.1620550.

[45] M. Sirivianos, K. Kim, and X. Yang. FaceTrust: Assessing the Credibility of Online Personas via Social Networks. In *Usenix HotSec*, 2009.

[46] R. Thomas. Getting in Bed with Robin Sage, 2010. Black Hat.

[47] L. Šikšnys, J. R. Thomsen, S. Šaltenis, M. L. Yiu, and O. Andersen. A location privacy aware friend locator. In *SSTD '09*, pages 405–410. Springer-Verlag, 2009.

[48] A. Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *8th USENIX Security Symposium*, 1999.

[49] A. C.-C. Yao. How to generate and exchange secrets. In *Proc. of foundations of computer science (FOCS)*, pages 162–167, 1986.

[50] M. E. Zarki, S. Mehrotra, G. Tsudik, and N. Venkatasubramanian. Security issues in a future vehicular network. In *In European Wireless*, pages 270–274, 2002.

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

[52] K. Zickuhr and A. Smith. 4% of online Americans use location-based services, 2010. Pew Research Center Survey.

A An asynchronous variant of protocol 1 using an oblivious server

In Section 3.1 we presented a synchronous private equality test using an additively homomorphic encryption scheme such as ElGamal. The protocol requires both Alice and Bob to be online at the same time. Here we present an asynchronous variant of the protocol using a trusted server. The server learns nothing at the end of the protocol, assuming the server does not collude with Alice.

Let (G, E, D) be a public-key additively homomorphic encryption scheme over a group of size p. We assume Alice has the secret key sk and Bob and the server have the public key pk. Using the notation of Section 3.1 the protocol works as follows:

- Bob's message: at any time Bob sends $E(\mathsf{pk}, b)$ to the server,
- Alice queries the server: At any time Alice sends $E(\mathsf{pk}, a)$ to the server. The server generates a random $r \in \mathbb{Z}_p$ and sends $C \leftarrow E\big(\mathsf{pk}, r(a-b)\big)$ back to Alice. The server constructs C using the additive property of encryption as in Section 3.1.
- Alice obtains answer: Alice decrypts C using sk and obtains $r(a-b)$ from which she learns if $a = b$, but nothing else.

Security follows as in Protocol 1 as long as Alice does not collude with the server. The advantage of this protocol over Protocol 2 is that there is only one round of communication between Alice and the server, since there is no need for a counter. The disadvantage over Protocol 2 is the increased computational and bandwidth requirement.