Secure Friend Discovery in Mobile Social Networks

Wei Dong Vacha Dave Lili Qiu Yin Zhang
The University of Texas at Austin
{wdong86,vacha,lili,yzhang}@cs.utexas.edu

Abstract– Mobile social networks extend social networks in the cyberspace into the real world by allowing mobile users to discover and interact with existing and potential friends who happen to be in their physical vicinity. Despite their promise to enable many exciting applications, serious security and privacy concerns have hindered wide adoption of these networks. To address these concerns, in this paper we develop novel techniques and protocols to compute social proximity between two users to discover potential friends, which is an essential task for mobile social networks. We make three major contributions. First, we identify a range of potential attacks against friend discovery by analyzing real traces. Second, we develop a novel solution for secure proximity estimation, which allows users to identify potential friends by computing social proximity in a privacy-preserving manner. A distinctive feature of our solution is that it provides both privacy and verifiability, which are frequently at odds in secure multi-party computation. Third, we demonstrate the feasibility and effectiveness of our approaches using real implementation on smartphones and show it is efficient in terms of both computation time and power consumption.

I. Introduction

Motivation. Online social networks, such as Facebook and Myspace, have experienced an explosive growth recently. Mobile social networks, which bring social networking to mobile phones, represent a natural next step and have already generated a lot of excitement. For example, cell phone manufacturers and cellular service providers have developed their own social networks (*e.g.*, Nokia, Virgin Mobile) and provided software support for mobile social networks (*e.g.*, Motorola). Compared to (Internet-based) online social networks, mobile social networks offer several distinctive advantages: (i) much larger potential user base, with 4 billion mobile phone subscribers [33] compared to only 300 million broadband subscribers [7], (ii) the built-in localization capability of mobile phones which enables rich and new location-based services, and (iii) applicable to a wider range of application scenarios because unlike online social networks, they do not necessarily require access to a computer or the Internet.

An essential capability offered by mobile social networks is to allow mobile users to discover and interact with friends who happen to be in their physical vicinity. Suppose you are waiting for your flight in an airport and your mobile phone discovers your friend's friend is in the next aisle and you can talk with face-to-face. Or you visit a new place and your mobile phone finds someone in your vicinity shares similar attributes as you so that you can interact with.

While mobile social networks hold great promise for enabling many exciting new applications, they also create serious privacy and security concerns. In particular, people are often reluctant to reveal their presence and personal profile to an arbitrary person in their vicinity. It is also unwise to blindly trust information received from an arbitrary person. While similar issues also exist in online social networks, these concerns become more serious because mobile social networks blur the boundary between the cyberspace and the physical world. Moreover, the broadcast nature of wireless medium also makes it easy for a malicious user to spoof and inject traffic into the mobile social networks. It is thus imperative to address these privacy and security concerns before mobile social networks can receive wide adoption.

One way to address the privacy and security issues is to take advantage of a trusted central server, which collects information from individual users, computes and disseminates the proximity results on demand. Server-based solution is not suitable for mobile social networks for the following reasons. First, users in a mobile social network may not have direct access to a computer or the Internet. While cellular data service increases in popularity, the number of data service subscribers is still very limited due to its high cost [23], [36]. As a result, a server-based solution cannot work in such an environment unless dedicated servers are deployed locally to support mobile social networks, which is prohibitively expensive if not infeasible. A distributed solution is more appealing because it obviates the need of always having access to a server. Second, as people are increasingly concerned about their location privacy and personal data, they may not want to reveal their current location or other personal information even to a trusted server. Wills *et al.* [18], [24] studied 13 mobile online social networks, such as Facebook, Friendster, Hi5, LinkedIn, Myspace, Twitter, and found all of them leaked some private information to tracking sites and several of them passed users' location information to a third party. Third, the server can easily become a bottleneck, a single point of failure, and the target for denial-of-service attack. These limitations of server-based approach motivate us to move away from a centralized solution and design distributed secure protocols for friend discovery in mobile social networks.

Approach and contributions. We consider how to identify potential friends in one's physical vicinity, which is essential to mobile social network. We assume that users only have occasional connectivity with a trusted server on the Internet (*e.g.*, once a day) and immediate communication occurs locally within a mobile social network and does not need access to the server; moreover the server does not know users' locations.

This is an important yet challenging problem because it involves joint computation between two parties that do not trust each other. The standard approach for predicting friendship is based on the notion of *proximity measure*, which quantifies the closeness or similarity between nodes in a social network. Many proximity measures have been proposed, including the number of common neighbors, the number of common attributes, cosine similarity, and path-ensemble-based measures. As shown in Section II, they can all be cast as a dot product operation on two vectors. Therefore we consider proximity computation as a dot product operation without loss of generality. To identify potential friends, two users can exchange their social coordinates (*i.e.*, a user's attributes) and compute the proximity between them; if their proximity exceeds a threshold, they try to make friends with each other.

However, directly exchanging social coordinates and computing their dot product opens door to a variety of attacks. In particular, we identify two serious attacks: (i) fingerprinting an individual user (based on either her social coordinate or her proximity with another known social coordinate), and (ii) falsifying proximity (*e.g.*, an attacker forges a social coordinate close to a target user's social coordinate to trick the user to make friend with the attacker). Defending against these attacks is particularly challenging because we want to simultaneously achieve two conflicting goals: ensuring verifiability (so that a malicious user cannot forge his/her social coordinate or forge the outcome of proximity computation), yet preserving privacy (*i.e.*, divulge no private information if the true proximity between two users is below the desired threshold).

To address the challenge, we develop novel techniques and protocols for computing proximity in a privacy-preserving, verifiable, and efficient manner. Specifically, we first develop a proximity pre-filtering protocol for determining whether the proximity between two users exceeds a given threshold. The protocol ensures that the initiator can only learn the comparison result between the estimated proximity and the threshold. The protocol does not involve any expensive cryptographic computation and is thus highly efficient. However in this protocol a malicious user can forge the comparison result. To defend against such attacks, we then develop two secure dot product protocols: one is based on homomorphic cryptography and the other leverages both homomorphic cryptography and obfuscation for higher efficiency. To the best of our knowledge, they are the first secure dot product protocols that are both privacy-preserving and verifiable.

This paper makes three major contributions: (i) identify a range of security attacks against friend discovery in mobile social networks and use real traces to analyze their impact, (ii) develop secure proximity computation protocols to identify potential friends, and (iii) demonstrate the effectiveness of our approach using both analysis and real implementation on smartphone. A key component of our solution is the first secure dot product protocol that is both privacy-preserving and verifiable. It has applications beyond mobile social networks because dot product is a fundamental primitive in secure multi-party computation and privacy-preserving data mining.

II. Coordinate-based Proximity Estimation

In this section, we first introduce the notion of proximity measure and describe how one can compute social proximity based on social coordinate. Our key assumption is that users are already part of a common (online or physical) social network. We can then identify potential friends by checking whether two users are sufficiently close in this social network.

Proximity measure in social networks. A *social network* [35] is a social structure modeled as a graph, where nodes represent people and edges represent relationships between them (*e.g.*, friendship). A central concept in social networks is *proximity measure*, which quantifies the closeness or similarity between nodes in a social network. Proximity measure serves as the basis for many social network applications (*e.g.*, [5], [11], [14], [37]). As a result, a variety of proximity measures have been proposed. The simplest proximity measures include the number of common neighbors or the number of common attributes between the two users. More sophisticated proximity measures involve infinite sums over the ensemble of all paths between two nodes in the social networks (*e.g.*, Katz measure [16], rooted PageRank [19], [20], and escape probability [31]). Compared to simple proximity measures, path-ensemble based proximity measures capture more information about the underlying social structure and have been shown to be more effective in social networks [19], [20], [31]. In particular, the Katz measure is shown to be particularly effective in predicting new links in social networks [19], [20], [30]. It is defined as

$$\mathsf{Katz}[x,y] = \sum_{\ell=1}^{\infty} \beta_{\text{Katz}}^{\ell} \cdot |\mathsf{paths}_{x,y}^{\langle\ell\rangle}| \tag{1}$$

where $\mathsf{paths}_{x,y}^{\langle\ell\rangle}$ is the set of length-ℓ paths from x to y in a social network, and $\beta_{\text{Katz}} < 1$ is a damping factor. We focus on computing Katz measure in our evaluation, but our protocols are general and can be applied to many other proximity measures including those simple proximity measures, since proximity computation can be considered as a dot product operation (also known as the inner product) without loss of generality, as shown below.

Estimating proximity from social coordinates. [30] developed a technique called *proximity embedding* for efficient and accurate computation of path-ensemble based proximity measure (*e.g.*, Katz measure, rooted PageRank, escape probability) in large social networks with millions of nodes. It approximates the entire $m \times m$ proximity matrix P, where $P(i,j)$ denotes the proximity between users i and j, as the product of two rank-r factor matrices U and V, where m can be millions but r is much smaller (*e.g.*, $r = 30$): $P_{m\times m} \approx U_{m\times r} \cdot V_{m\times r}^T$ While proximity embedding was originally designed for centralized social network analysis [30], once the decomposition $P \approx UV^T$ becomes available, we can immediately use U and V to enable efficient, distributed computation of social proximity. Specifically, each node i is associated with a pair of vectors $U[i,*]$ and $V[i,*]$, which we term as i's *social coordinates* since they represent a user's position in the social network. The proximity from node i to node j can then be approximated as the dot product of their social coordinates $U[i,*]$ and $V[j,*]$, which can be efficiently computed in $O(r)$ time. Similarly, the proximity from node j to node i is simply the dot product of $U[j,*]$ and $V[i,*]$.

In addition, dot product can be used to compute many other proximity measures. For example, given a global list of attributes, each user is assigned a vector with 0s or 1s, where 0 means that the user does not have the attribute and 1 otherwise. The dot product of two vectors essentially captures the number of matches between two users and can also be used to predict new friendship. When the list gets too long, a bloom filter can be used to summarize the list in a compact fashion. One can then estimate the similarity between two sets based on the dot product of the corresponding two bloom filters. As another example, consider two feature vectors $\mathbf{u}$ and $\mathbf{v}$. If we normalize them to have unit length, the dot product $\mathbf{u}\circ\mathbf{v}$ gives the cosine similarity between the two feature vectors.

A main component of our solution for secure proximity computation is the first secure dot product protocol that is both privacy-preserving and verifiable. Since secure dot product is a fundamental primitive in privacy-preserving data mining and secure multi-party computation, our technique can have many applications beyond mobile social networks.

Problem formulation. The social coordinates for individual users can be precomputed by a trusted central server in the social network that users belong to. For example, one can imagine that existing online social networking site, such as Facebook and Myspace, to provide such a service. Our goal then is to determine whether the dot product of two users' social coordinates exceeds a given threshold. Only when the dot product is large enough, will the two users start further communication. We want such computation to be efficient, privacy-preserving, and verifiable: (i) no user can forge social coordinates or the result of dot product without getting caught, and (ii) if the dot product between two users is below the threshold, they learn only this fact and nothing more.

III. Attack Models

Adversaries are curious about other users' personal information and location information and will try their best to extract information from every message. They may also lie or even collude in order to get more information. Below we identify a range of potential attacks against coordinate-based social proximity computation and quantify their effectiveness using analysis of real traces.

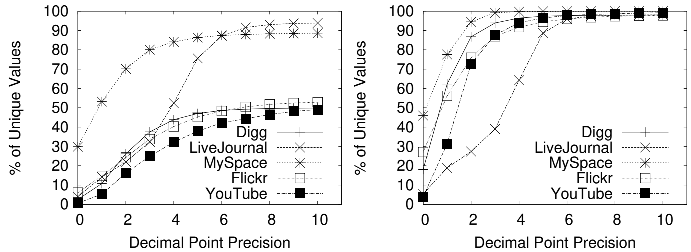
Compromising location privacy. A number of attacks can be launched to breach a user's location privacy based on her social coordinate or her social proximity to a known social coordinate. We identify four such attacks below.

• *Fingerprinting users based on social coordinates.* A naïve way to support proximity computation is to let every user publish her social coordinate for computing dot product. However, if the social coordinates of a user are relatively unique, it becomes easy to identify a user based on them. To assess the potency of such an attack, we use five popular social networks: Digg, Flickr, LiveJournal, Myspace and YouTube, as shown in Table I. Figure 1(a) plots the percentage of unique social coordinates in a given social network as a function of the number of digits used to represent one dimension in each coordinate. The curve is based on the first snapshot but the results from the second snapshot are similar. It is evident from the figure that with just 4 decimal-point precision, which is required for accurately computing proximity metric between any user pair, 35%-80% of the users become uniquely identifiable by their social coordinates. Even with 1 decimal point precision, there are still 5%-50% unique social coordinates. Further inspection suggests that most non-unique social coordinates belong to socially inactive users with few friends. For example, all users with zero friends have the same all-zero social coordinate. To illustrate this effect, Figure 1(b) plots the percentage of unique social coordinates for users with at least five friends. We see a dramatic increase in the fraction of unique social coordinates. Therefore, social coordinate based fingerprinting is more potent against socially active users.

Network	Date	# Connected nodes	Links
Digg	9/15/2008	535,071	4,432,726
	10/25/2008	567,771	4,813,668
Flickr	3/01/2007	1,932,735	26,702,209
	4/15/2007	2,172,692	30,393,940
LiveJournal	11/13/2008	1,769,493	61,488,262
	12/05/2008	1,769,543	61,921,736
Myspace	12/11/2008	2,128,945	89,138,628
	1/11/2009	2,137,773	90,629,452
YouTube	4/30/2007	2,012,280	9,762,825
	6/15/2007	2,532,050	13,017,064

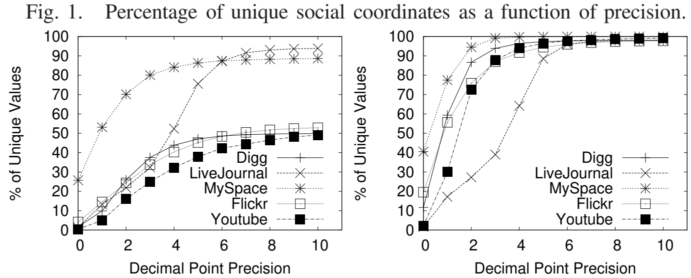
TABLE I
DATASET SUMMARY.

• *Fingerprinting users based on social proximity.* Even if the social coordinates are encrypted, a smart adversary can try



(a) all users (b) users with ≥ 5 friends

Fig. 1. Percentage of unique social coordinates as a function of precision.



(a) all users (b) users with ≥ 5 friends

Fig. 2. Percentage of unique dot products as a function of precision.

to infer the social coordinates based on the proximity metric. This is demonstrated by the following analysis. Figure 2 shows the uniqueness of the dot product value between a users' social coordinate and a given random coordinate that an attacker would use. As we can see, with 1 decimal point precision, 5-55% of all users are uniquely identifiable; as the precision increases to 4 decimal points, around 30-80% of the users are uniquely identifiable. Moreover, the fraction of unique social proximity is much higher among users with at least five friends. So social proximity based fingerprinting is a more serious attack against socially active users.

• *Inferring social coordinates from proximity measure.* Apart from acting as an identifier, knowing exact proximity to a given user may even allow an adversary to reconstruct a user's social coordinates. Specifically, since the proximity between two users is a dot product between their coordinates, each proximity value gives one linear constraint on the social coordinates of the other party. An adversary just needs d linearly independent constraints to reconstruct a d-dimension social coordinate. This can be achieved by having an adversary generate d (fake) linearly independent coordinates and then compute the proximity with an intended victim, using each of the d different coordinates or having d colluding adversaries.

• *Binary search on social proximity.* The above attacks suggest that we should protect privacy of both social coordinates and proximity measure. In order to facilitate the decision of whether to make friends with a user, ideally we only want to reveal 1 bit of information, *i.e.*, whether the dot product is above or below a threshold (instead of its exact value). However, even this 1 bit of information could be potentially exploited by a patient adversary to launch a binary search attack that adaptively adjusts the threshold to quickly narrow down the value range for the social proximity between the victim and a social coordinate chosen by the adversary.

Tracking users. We further observe that social coordinates are relatively stable over time. Figure 3 plots the cumulative distribution function (CDF) of the cosine similarity, which measures the angle between a user's social coordinates across the two snapshots indicated in Table I for all the social networks. As shown in Figure 3, for most networks, a large fraction of users' coordinates have high cosine similarity across the two snapshots. Around 50-60% people in Digg and Flickr have social coordinates' cosine similarity above 0.9, and around 90% of users in Myspace have cosine similarity over 0.9. Such high stability in social coordinates indicates that an adversary could potentially link social coordinate announcements at different times, locations, and networks to the same user and enable continuous user tracking.

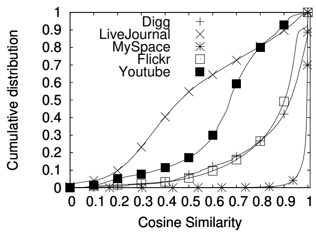


Fig. 3. CDF of cosine similarity of social coordinates in two snapshots.

Other attacks. Other attacks include forgery and DoS. Since users rely on the social coordinates to make friendship decisions, it would be very damaging if an adversary is able to lie about her social coordinate to trick others into believing that the adversary is socially close. Also existing private dot product computation protocols are not verifiable (*e.g.*, [15], [34]), making it possible to lie about the computation result. Finally, an adversary may send many fake social coordinates for proximity computation and overwhelm the victim.

IV. Our Approach

A. Design Goals

Based on the above attacks, our design goals are as follows:

1. **Preserving the privacy of social coordinates.** Due to the uniqueness of social coordinates, we should encrypt social coordinates. Since each dot product reveals one linear constraint about the social coordinates, d independent constraints would reveal the whole coordinate. Therefore we should limit the number of linear constraints revealed.
2. **Preserving the privacy of social proximity.** Due to uniqueness of proximity values and the possibility of reconstructing social coordinates based on proximity values, ideally a scheme should just reveal whether the proximity value is above a threshold used for making friendship decisions, while protecting against binary search attacks.
3. **Preventing user tracking.** Even when a user's social coordinate does not change, her social coordinate announcement should look different to prevent tracking.
4. **Providing authentication and verification.** Users should not be able to forge their social coordinates. Any user should be able to authenticate another user's identity and social coordinate.
5. **Efficient filtering.** Since a user is interested in quickly finding potential friends and the number of potential friends is usually much smaller than the total number of users in the network, we should efficiently filter out social coordinates with low proximity values.

B. Overview

To achieve the above design goals, we develop a novel secure proximity computation protocol, which consists of three major components: (i) authentication without long-term linkability (Section IV-C), (ii) efficient and privacy-preserving proximity pre-filtering (Section IV-D), and (iii) private and verifiable proximity computation (Section IV-E).

Specifically, in (i) users use virtual ID to announce their presence and social coordinates to remove long-term linkability and prevent user tracking, and use digital signatures for authentication (*i.e.*, design goals 3 and 4). Then the proximity prefiltering in (ii) allows users to quickly filter out users that are unlikely to become friends (*i.e.*, design goals 1, 2, 5). However it does not prevent an adversary from forging social coordinates or the final proximity result. So we further develop a technique for private and verifiable proximity computation based on homomorphic cryptography in (iii) to check the validity of social coordinates and proximity result (*i.e.*, design goals 1 and 4). The step (iii) is only invoked when proximity pre-filtering determines that the proximity exceeds the threshold because (1) homomorphic cryptography is computationally expensive and should be called upon for verification only when necessary, and (2) this step reveals the exact proximity value. It is unacceptable to reveal the exact proximity value to an arbitrary user, which can easily compromise location privacy (see Section III). However, we believe that it is acceptable to reveal the exact proximity value to those users that are sufficiently close socially and thus likely to become friends. A lying adversary can get caught by this step and reported.

C. Authentication Without Long-Term Linkability

When two strangers encounter each other, proximity computation needs to be performed on their social coordinates though they may not trust each other. To support authentication in such an untrusted environment, we require a trusted server, which can be the same server that computes social coordinates. Note that the server is only used for bootstrapping trust. *It does not need to be contacted in a mobile social network.*

To protect Alice's identity and prevent long-term linkability, the trusted server assigns Alice a separate *virtual ID*, which is the only ID sent during communication. The virtual ID is essentially a temporary private/public key pair, which is valid for only a specified time period. Alice needs to get her new virtual ID by contacting the trusted server when it is reachable. Every time Alice contacts the server, the server can use standard digital signature techniques to authenticate Alice (as in Internet server) before issuing her new virtual ID.

After authenticating Alice, the server sends Alice her new virtual ID, expiration time, encrypted social coordinates, and the server's digital signature. Others can use the server's signature and Alice's own signature to authenticate Alice and her encrypted social coordinates to prevent Alice from faking invalid IDs and coordinates. The encrypted social coordinates are used for computation in a verifiable secure dot product protocol introduced in Section IV-E. In this way, we cannot link messages sent before and after changing virtual IDs, which prevents long-term linkability. To reduce the frequency of contacting the server, the server may issue Alice multiple virtual IDs at a time, each with a different valid period.

D. Proximity Pre-filtering

The goal of proximity pre-filtering is to quickly identify potential friends from all the users in a mobile social network. Our requirement is to efficiently compute whether the proximity between two social coordinates is below or above a threshold without revealing the actual social coordinates or the exact proximity. The threshold depends on a user's interest in making new friends. In this step, we do not verify the correctness of the coordinates or proximity values, which will be handled by the private and verifiable proximity computation in Section IV-E.

There are several existing protocols that compute dot product while preserving the privacy of the vectors (*e.g.*, [15], [34]), however they reveal the final value of the dot product to one of the parties and do not satisfy our requirements (*i.e.*, preserve the privacy of the proximity value and only reveal whether it is above or below the threshold). We first review one of the existing secure dot product proposed [15], which we use as the basis for our protocol. Then we present our modifications to achieve our design goal.

Existing secure dot product protocol [15]. Suppose Alice has a d-dimensional vector $\mathbf{v}$ and Bob has a d-dimensional vector $\mathbf{u}$. s is a security parameter that controls the amount of random information we use to hide the social coordinate.

1. Alice initiates the dot product computation by asking Bob to start the computation. Bob constructs a $s\times(d+1)$ matrix $\mathbf{X}$. Its i-th row $\mathbf{x_i}$ is defined as

$$\mathbf{x_i} = \begin{cases} \langle u_1, u_2, \ldots, u_d, 1\rangle, & i = r, \\ \mathbf{k_i}, & \forall i \neq r, \end{cases} \tag{2}$$

   where r is a randomly chosen row, which contains the social coordinate, and the other row vectors $\mathbf{k_i}$ ($i \neq r$) are all randomly generated.
   Bob also creates an $s \times s$ random matrix $\mathbf{Q}$, a random $(d+1)$-dimensional vector $\mathbf{f}$, and three random numbers r_1, r_2, r_3. He computes the following five terms: (i) $b = \sum_{i=1}^{s} Q_{ir}$, (ii) $\mathbf{c} = \sum_{i=1, i\neq r}^{s} (\mathbf{x_i} \cdot \sum_{j=1}^{s} Q_{ji})$, (iii) $\mathbf{Q} \cdot \mathbf{X}$, (iv) $\mathbf{c}' = \mathbf{c} + r_1 \cdot r_2 \cdot \mathbf{f}$, and (v) $\mathbf{g} = r_1 \cdot r_3 \cdot \mathbf{f}$. Bob then sends $\mathbf{Q} \cdot \mathbf{X}, \mathbf{c}', \mathbf{g}$ to Alice.
2. Alice chooses a random number α and creates a vector $\mathbf{v}' = \langle v_1, v_2, ...v_d, \alpha\rangle$. Alice then computes the following four terms: (i) $\mathbf{y} = \mathbf{Q} \cdot \mathbf{X} \cdot \mathbf{v}'$, (ii) $z = \sum_{i=1}^{s} y_i$, (iii) $a = z - \mathbf{c}' \circ \mathbf{v}'$, and (iv) $h = \mathbf{g} \circ \mathbf{v}'$. Here $\mathbf{v}_1 \circ \mathbf{v}_2$ represents the dot product (*i.e.*, inner product) of two given vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. Alice then sends a and h to Bob.
3. Bob computes $\beta = \frac{a + h\cdot(r2/r3)}{b}$ and sends it to Alice.
4. Alice computes $\beta - \alpha$, which is the desired dot product as shown in [15] (*i.e.*, $\beta - \alpha = \mathbf{v} \circ \mathbf{u}$).

Proximity pre-filtering. Our goal is to only reveal whether the dot product is above a threshold to Alice and Bob, and prevents both of them from learning the exact dot product. Since in the last step of the above protocol the dot product is given by $\beta - \alpha$, a simple way to hide the true dot product is to let Alice choose a threshold T and send $T + \alpha$ to Bob. If β is larger than $T + \alpha$, Bob replies YES; otherwise he replies NO. The problem with this approach is that Bob can guess a reasonable T in this system, and with $T + \alpha$ and β he can estimate $\beta - \alpha$, which is the dot product.

To address this issue, we propose a proximity pre-filtering protocol by making the following modifications to the original secure dot product protocol. At the beginning of the protocol, instead of using her real vector, Alice uses a scaled version of her vector $\rho \cdot \mathbf{v}$, where ρ is a random number chosen by Alice. At the end of the protocol, along with a and h, Alice also picks another random number ρ' such that $0 \leq \rho' < \rho$, computes $\gamma = \alpha + \rho \cdot T + \rho'$ and sends γ to Bob. Bob compares γ with β, and replies YES only when $\beta > \gamma$, and otherwise replies No. All the other operations remain the same as [15].

Public key: (g, n), Private key: σ
Encryption:
Plaintext $m < n$
Select random $r < n$
Ciphertext $c = E(m, r) = g^{m+nr} mod\ n^2$
Decryption:
Ciphertext $c < n^2$
Plaintext $m = D(c) = \frac{L(c^\sigma mod\ n^2)}{L(g^\sigma mod\ n^2)} mod\ n$, where $L(u) = \frac{u-1}{n}$

Fig. 4. The fast variant of Paillier's Cryptosystem

Correctness. Since Alice scales her vector by ρ, we have $\beta - \alpha = \rho \cdot p$, where p is the dot product. $\beta - \gamma = \beta - \alpha - \rho \cdot T - \rho' = \rho \cdot (p - T) - \rho'$. Given p and T are both integers, we have that $\beta > \gamma$ implies $p > T$ and $\beta < \gamma$ implies $p \leq T$.

Setting the threshold. Allowing users to set an arbitrary threshold T is undesirable because an inappropriate threshold can complicate liar detection. We assume a system suggests minimum threshold for all users. Users can set a higher threshold to be more selective but are prohibited from setting lower thresholds than the suggested value. Therefore whenever the outcome of proximity prefiltering is YES the proximity value must be larger than this suggested minimum threshold, which can be used as ground truth for liar detection.

E. Private and Verifiable Proximity Computation

Proximity pre-filtering alone is insufficient because adversaries can falsify their social coordinates or the final proximity result. In this section, we develop a solution to address these issues. It achieves both privacy and verifiability, which are two conflicting goals in secure multi-party computation. We first introduce an existing homomorphic protocol for computing a dot product. It preserves privacy but does not provide authentication or verification. Then we describe our approach to provide both privacy and verification.

Homomorphic encryption. Homomorphic encryption is an effective solution to privacy-preserving computation. It allows certain algebraic operations on the plaintext to be performed using (possibly different) algebraic operations directly on the ciphertext. There are several homomorphic cryptosystems. We use the fast variant of Paillier's cryptosystem [25], shown in Figure 4. Its semantic security relies on the premise that the Decisional Partial Discrete Logarithm Problem [25] is hard. This cryptosystem has the following two useful properties:

1. *Additive Homomorphic Property*: Multiplication of the ciphertexts results in addition of the plaintexts:

$$\begin{aligned} \mathsf{E}(m_1, r_1)\,\mathsf{E}(m_2, r_2) \quad \text{mod } n^2 &= \quad \mathsf{E}(m_1 + m_2, r_1 + r_2) \\ \mathsf{E}(m_1, r_1)^{m_2} \quad \text{mod } n^2 &= \quad \mathsf{E}(m_1 \cdot m_2, r_1 \cdot m_2) \end{aligned}$$

2. *Self-Blinding Property*: Any ciphertext can be changed to another without affecting the plaintext:

$$\mathsf{D}(\mathsf{E}(m, r_1)) = \mathsf{D}(\mathsf{E}(m, r_1 + r_2))$$

For our protocol, we expect the trusted server to generate the required keys and send them to the user in addition to the virtual ID information specified in Section IV-C.

Homomorphic dot product protocol. A secure dot product protocol has been proposed based on homomorphic encryption [12]. It achieves the design goal that Alice gets the dot product, but neither Alice nor Bob learns any other useful information about other party's vector, under certain security assumptions. To summarize, let H_A^+ and H_B^+ denote Alice and Bob's homomorphic public keys, respectively. Suppose Alice's vector is $\mathbf{v} = \langle v_1, v_2, \ldots, v_d\rangle$ and Bob's vector is $\mathbf{u} = \langle u_1, u_2, \ldots, u_d\rangle$. The protocol works as follows:

1. For each dimension of vector $\mathbf{v}$, Alice generates a random number r_i and sends $\mathsf{E}_{H_A^+}(v_i, r_i)$ to Bob.
2. Bob computes $w = \prod_{i=1}^{d} \mathsf{E}_{H_A^+}(v_i, r_i)^{u_i}$ (we will refer to this operation as multiplication throughout our paper), generates a random number r', and sends $w' = w \cdot \mathsf{E}_{H_A^+}(0, r')$ back to Alice.
3. Alice then decrypts w' to get the dot product.

This protocol is provably privacy-preserving. However, it assumes an *honest-but-curious* adversary model and provides neither authentication nor verification. Malicious participants can lie about their encrypted vectors and the resulting dot product of the two vectors, since Bob knows Alice's public key and can encrypt an arbitrary value to send back to Alice instead of the real dot product. In the rest of the paper we refer to this protocol as *Protocol 0*.

Designing a verifiable secure dot product protocol. Authentication and verification are essential to guard against malicious users who falsify the social coordinates and proximity metric. Section IV-C describes authentication, and below we discuss how to make the protocol verifiable.

Note that for both parties to obtain the dot product, both Alice and Bob run two separate instances of protocol in parallel. Then, a naïve verification approach for Bob may be to first decrypt the result sent by Alice using his private key and encrypt it using Alice's public key and compare it with w that he computed before for consistency. However, since encryption in Paillier's cryptosystem involves a random number, the same plaintext may generate different ciphertext, causing this approach to fail.

Suppose Alice picks $\mathbf{r_1} = \langle r_{11}, r_{12}, \ldots, r_{1d}\rangle$ as her random numbers and Bob picks $\mathbf{r_2} = \langle r_{21}, r_{22}, \ldots, r_{2d}\rangle$ as his random numbers. Let r_1' and r_2' be the numbers used for self-blinding by Alice and Bob, respectively. Alice's encrypted vector is $\mathsf{E}_{H_A^+}(\mathbf{v}, \mathbf{r_1}) = \langle \mathsf{E}_{H_A^+}(v_1, r_{11}), \mathsf{E}_{H_A^+}(v_2, r_{12}), \ldots, \mathsf{E}_{H_A^+}(v_d, r_{1d})\rangle$, and Bob's encrypted vector $\mathsf{E}_{H_B^+}(\mathbf{u}, \mathbf{r_2})$ is similar.

In order to solve the verification problem, we observe that when the fast variant of Paillier's cryptosystem is used, the value computed by Alice (before self-blinding) is $\mathsf{E}_{H_B^+}(\mathbf{v} \circ \mathbf{u}, \mathbf{r_2} \circ \mathbf{v})$. This is because

$$\prod_{i=1}^{d} \mathsf{E}_{H_B^+}(u_i, r_{2i})^{v_i} = \prod_{i=1}^{d} \mathsf{E}_{H_B^+}(u_i \cdot v_i, r_{2i} \cdot v_i) = \mathsf{E}_{H_B^+}(\mathbf{v} \circ \mathbf{u}, \mathbf{r_2} \circ \mathbf{v})$$

Assume Alice decrypts the result received from Bob and gets $result1$, without knowing $\mathbf{r_2} \circ \mathbf{v}$, Alice cannot generate $\mathsf{E}_{H_B^+}(result1, \mathbf{r_2} \circ \mathbf{v})$ to test the consistency with $\mathsf{E}_{H_B^+}(\mathbf{v} \circ \mathbf{u}, \mathbf{r_2} \circ \mathbf{v})$ computed by herself.

To overcome such difficulty, below we design two protocols that support verification.

Verifiable secure dot product protocol 1: Here, we apply Protocol 0 to compute $\mathbf{r_2} \circ \mathbf{v}$ in addition to $\mathbf{v} \circ \mathbf{u}$ so that the dot product value obtained can be verified.

1. As in Protocol 0, Alice and Bob start by exchanging their encrypted vectors $\mathsf{E}_{H_A^+}(\mathbf{v}, \mathbf{r_1})$ and $\mathsf{E}_{H_B^+}(\mathbf{u}, \mathbf{r_2})$.
2. Alice computes $\mathsf{E}_{H_B^+}(\mathbf{v} \circ \mathbf{u}, \mathbf{r_2} \circ \mathbf{v})$ and $\mathsf{E}_{H_B^+}(\mathbf{r_1} \circ \mathbf{u}, \mathbf{r_1} \circ \mathbf{r_2})$ and send them to Bob after self-blinding. Bob computes $\mathsf{E}_{H_A^+}(\mathbf{v} \circ \mathbf{u}, \mathbf{r_1} \circ \mathbf{u})$ and $E_{H_A^+}(\mathbf{r_2} \circ \mathbf{v}, \mathbf{r_1} \circ \mathbf{r_2})$ and also send them after self-blinding.
3. Alice decrypts and gets two numbers $result1$ and $result2$, which are supposed to be $\mathbf{v} \circ \mathbf{u}$ and $\mathbf{r_2} \circ \mathbf{v}$. Alice computes $\mathsf{E}_{H_B^+}(result1, result2)$ and compares with $\mathsf{E}_{H_B^+}(\mathbf{v} \circ \mathbf{u}, \mathbf{r_2} \circ \mathbf{v})$. If they are consistent, the dot product $result1$ is correct. Bob decrypts and verifies in the same way.

To ensure that Alice can properly decrypt $\mathbf{v} \circ \mathbf{u}$ and $\mathbf{r_2} \circ \mathbf{v}$ in step 3 of the protocol, we require that $\mathbf{v} \circ \mathbf{u} < n_A$ and $\mathbf{r_2} \circ \mathbf{v} < n_A$, where $(g_A, n_A) = H_A^+$ is Alice's homomorphic public key given in Figure 4. This can be achieved by limiting the number of bits of each element in $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{r_2}$. In our current implementation, n_A has 1024 bits, each element in $\mathbf{u}$ and $\mathbf{v}$ is a 32-bit integer, and the number of dimensions d is a 16-bit integer. As a result, each element in $\mathbf{r_2}$ can have as many as $1024 - 16 - 32 = 976$ bits without causing $\mathbf{r_2} \circ \mathbf{v}$ to exceed n_A. Such a large number of bits is sufficient to defend against brute-force attacks that enumerate all possible ciphertexts for a given plaintext.

Compared to the original Protocol 0, our new protocol has slightly more communication overhead and more than twice computation overhead since it computes two dot products using the Protocol 0 and has a verification phase, which is essentially an encryption. Also note that the second dot product incurs more computation overhead because the elements of the random vectors can be much larger than elements of $\mathbf{v}$ and $\mathbf{u}$. Therefore, we call the multiplication operation with random vector as *BigMul*. Next we develop a light-weight protocol with much less computation overhead, but slightly more communication overhead.

Verifiable secure dot product protocol 2: The idea is to only compute $\mathbf{v} \circ \mathbf{u}$ using Protocol 0 and compute $\mathbf{r_2} \circ \mathbf{v}$ using the secure dot product protocol [15], which is much cheaper than the homomorphic protocol.

1. Alice and Bob exchange their encrypted vectors $\mathsf{E}_{H_A^+}(\mathbf{v}, \mathbf{r_1})$ and $\mathsf{E}_{H_B^+}(\mathbf{u}, \mathbf{r_2})$ as before.
2. Alice computes $\mathsf{E}_{H_B^+}(\mathbf{v} \circ \mathbf{u}, \mathbf{r_2} \circ \mathbf{v})$ and sends it to Bob after self-blinding. Bob computes $\mathsf{E}_{H_A^+}(\mathbf{v} \circ \mathbf{u}, \mathbf{r_1} \circ \mathbf{u})$ and does the same thing.
3. Alice uses the secure dot product protocol [15] to compute $\mathbf{v} \circ \mathbf{r_2}$ with Bob. We denote the result she gets from this step as $result2$. Similarly, Bob uses secure dot product protocol to compute $\mathbf{u} \circ \mathbf{r_1}$.
4. Alice decrypts the received message from step 2 and get $result1$. With $result1$ and $result2$, verification is the same as Protocol 1. The same holds for Bob.

Compared to Protocol 0, the additional computation cost is close to just one more encryption for verification, since the additional secure dot product computation is much cheaper than the encryption. In terms of communication, the secure dot product protocol requires at least four messages, which slightly increases communication cost. Since the computation cost dominates, the small increase in communication is worthwhile.

Homomorphic encryption with negative numbers: Negative numbers in the social coordinates do not affect the correctness of the protocols since the homomorphic properties still hold. A negative dot product can cause ambiguity on the results because the decrypted result x may imply that the original result may be x, $x - n$, or $x - 2 \cdot n$, etc. To handle this issue, we leverage the fact that n is a 1024-bit number, which is huge, whereas all proximity values comparatively small. Therefore, whenever the proximity is too large (*i.e.*, $> n/2$), this indicates that the real value is negative and the actual result is $result - n$.

F. Security Analysis

1) Proximity Pre-filtering: Here we analyze the information flow between Alice and Bob. The first several steps of our protocol are the same as the secure dot product protocol, except that we scale the coordinate. As shown in [15], from Bob to Alice, $\mathbf{Q} \cdot \mathbf{X}$, $\mathbf{c}'$ and $\mathbf{g}$ do not leak any information. From Alice to Bob, h reveals one equation about Alice's coordinate $\mathbf{v}$. This is possible because we have two relationships $h = \mathbf{g} \circ \mathbf{v}'$ and $\beta = p + \alpha$; canceling out α from both relationships give us one constraint on $\mathbf{v}$. Now we consider the additional information flow introduced in our protocol. The final result from Bob to Alice contains 1 bit information, which is the outcome of the protocol and required. From Alice to Bob, γ reveals $\rho \cdot (p - T) - \rho'$. So essentially we hide $p - T$ using two random numbers ρ and ρ'. [17] used the same hiding technique in Yao's millionaire's protocol, which is shown to be secure when the range of ρ is big enough and one can increase the difficulty of guessing $p - T$ by randomly using many possible distributions for choosing random numbers. It will reveal $p = T$ only when $\rho \cdot (p - T) - \rho' = 0$, which occurs with an extremely low probability.

Next we examine if binary search is still possible. There are two ways to perform binary search: (i) an adversary computes proximity measure with the victim multiple times, each time using a different threshold, or (ii) an adversary colludes with others and fakes the same social coordinate and compute proximity measure between the fake coordinate with the victim's coordinate multiple times, each time using a different threshold. The first attack can be easily prevented by permitting only one proximity computation between any two users. To defend the second attack, we note that in binary search about half of the trials will have YES result. Whenever the result is YES, we run our verifiable secure dot product protocol, which will detect the liar. So binary search is still possible, but each trial will be caught with 50% probability and the success probability of binary search exponentially decreases with the number of trials involved in binary search.

The pre-filtering protocol does not have authentication or verification on the vectors used by participants. It is easy for either side to manipulate the final result. Alice can manipulate γ and make the result NO even when the real result is YES. This means that Alice decides not to make friend with Bob despite the high proximity measure, which should be permitted as Alice should have freedom to choose friends even when the proximity measure exceeds Bob's threshold. Similarly, Bob is allowed to reply NO when the actual result is YES. But anyone that makes the result YES while it is actually NO is considered cheating and can be detected by our verifiable secure dot product protocol.

2) Private and Verifiable Proximity Computation:

Theorem 1 (Privacy of Coordinates in Protocol 1): Assuming the fast variant of Paillier's cryptosystem is semantically secure, Alice and Bob only get the dot product and no more useful information about each other's coordinate.

Proof: Assuming the cryptosystem is semantically secure, Protocol 0 reveals no useful information other than the result [12]. In Protocol 1, each participant gets two dot product by running two instances of Protocol 0. Thus each of them gets no more useful information than two dot products. The extra dot product used for verification, *i.e.*, $\mathbf{r_2} \circ \mathbf{v}$, does not reveal more information about $\mathbf{u}$ because $\mathbf{r_2}$ is hard to guess. In fact, guessing $\mathbf{r_2}$ based on $\mathbf{r_2} \circ \mathbf{v}$ is just as hard as guessing $\mathbf{u}$ based on $\mathbf{v} \circ \mathbf{u}$, which is proven to be hard [12]. ■

Theorem 2 (Privacy of Coordinates in Protocol 2): Assuming the fast variant of Paillier's cryptosystem is semantically secure, Alice and Bob get the dot product and one more linear constraint about the other party's coordinate.

Proof: The only difference between Protocol 1 and Protocol 2 is that in Protocol 2 $result2$ is computed using secure dot product protocol [15], which reveals one more constraint about the other user's coordinate. ■

Theorem 3 (Verification Guarantee): If Alice's verification is successful in Protocol 1 or Protocol 2, Alice gets the real dot product.

Proof: The only case when verification is successful but Alice gets a wrong number is Bob found two numbers $result1 < min(n_A, n_B)$ and $result2 < min(n_A, n_B)$ such that $result1 \neq \mathbf{v} \circ \mathbf{u}$, $\mathsf{E}_{H_B^+}(result1, result2) = \mathsf{E}_{H_B^+}(\mathbf{v} \circ \mathbf{u}, \mathbf{r_2} \circ \mathbf{v})$. This is impossible because decryption result is unique in Paillier's cryptosystem. ■

V. Evaluation

We implement our protocols on HP IPAQ 910, which has Marvell PXA270 416 MHz Processor, 128 MB RAM, Windows Mobile 6.1 Professional operating system, 802.11 b/g WiFi card, and .NET Compact Framework. Since the cryptosystem involves very large numbers, which is not supported by the Compact Framework, we use a public BigInteger Library for C# [6]. The communication uses 802.11b ad hoc mode. We also implemented all computations in all the protocols on Motorola Droid. It has 550 MHz Arm Cortex A8 processor, 230 MB RAM, and uses Java and the Android development toolkit. We use a publicly available implementation [26] of Paillier's Cryptosystem in Java as a basis for our Fast Variant implementation. Since the Android Platform has not yet supported wireless ad-hoc mode, we quantify the communication cost using 802.11b infrastructure mode. For comparison, we also evaluate our protocols on a laptop PC with Windows Vista system, P8600 processor, and 2GB memory. Our implementation uses Java. We perform power measurement on the Droid using PowerTutor [29]. All the implementations use 30-dimension social coordinates. Unless otherwise specified, the performance numbers are based on computing a proximity metric between a pair of nodes.

A. Proximity Pre-filtering

The efficiency of pre-filtering depends on the security parameter s, which controls the amount of random information we add. We use $s = 2$ in our evaluation as in [15].

This protocol altogether involves 4 messages: (i) a control message to initiate proximity pre-filtering, (ii) a reply containing a matrix and two vectors, which has altogether $(s+2) \cdot (d+1)$ numbers, (iii) a response containing 3 numbers, and (iv) a final answer containing 1-bit: either YES or NO to the question whether the proximity exceeds the threshold. The message size depends on the selected random numbers.

Table II shows the mean, maximum, minimum and average execution time for prefiltering over 100 runs on PC, IPAQ and

Device	Mean	Max	Min	Std	Median
PC	0.44	2.15	0.29	0.30	0.32
IPAQ	172	633	44	78	154
Droid	44	112	22	23	23

TABLE II
EXECUTION TIME OF PROXIMITY PREFILTERING (MS)

PC					
Operation	Mean	Max	Min	Std	Median
Authentication	0.48	1.31	0.37	0.12	0.46
Encryption	77.79	96.07	76.77	2.23	77.20
Decryption	14.11	17	13	0.5667	14
Multiplication	81.92	124	78	5.25	81
BigMul	2253.8	2581	2223	52.02	2238
Self-blinding	83	120	81	4.73	82
Verification	82.43	93	80	2.27	82
IPAQ					
Operation	Mean	Max	Min	Std	Median
Authentication	27.02	61	24	7.7485	25
Decryption	2193.2	2620	2176	48.3319	2178
Multiplication	4055.2	6033	2312	1134.7	4146
BigMul	382026	390237	375463	7523.6	380378
Self-blinding	12615	13353	12280	196.74	12606
Verification	12807	14270	12452	240.10	12776
Droid					
Operation	Mean	Max	Min	Std	Median
Authentication	2.54	54.4	2.23	3.01	2.32
Decryption	27.01	63	24	8.6416	24
Multiplication	160.41	554	150	44.5594	153
BigMul	3780.7	4250	3733	71.48	3760.5
Self-blinding	144.69	596	136	47.6366	137
Verification	144.18	192	136	14.1581	138

TABLE III
BREAKDOWN OF COMPUTATION TIME FOR VERIFIABLE SECURE DOT PRODUCT PROTOCOLS ON PC, IPAQ, AND DROID (MS)

Droid. The average time is 0.17 second on the IPAQ and 0.044 second on the Droid. Both are fast enough for practical use.

B. Private and Verifiable Proximity Computation

We implement both versions of our verifiable secure dot product protocols. Our implementation is based on the fast variant of the Paillier's cryptosystem, where n has 1024 bits and σ has 160 bits according to [25]. The random numbers used have 900 bits, big enough to prevent brute force attacks.

Table III shows the breakdown of the computation time on PC, IPAQ, and Droid. Protocol 1 involves one authentication, one multiplication, one BigMul, two self-blindings, two decryptions, and one verification. Protocol 2 includes one authentication, one multiplication, one self-blinding, one decryption, one secure dot product protocol [15], and one verification. We do not show the performance of secure dot product protocol because it has almost the same performance as the proximity pre-filtering shown in Table II. As we would expect, the PC is faster than Droid, which is faster than IPAQ. The average computation time for protocol 1 is 2.61 seconds on PC, 4.4 seconds on Droid, and 6 minutes on IPAQ. The time for protocol 2 is 0.276 seconds on PC, 0.523 seconds on Droid, and 31.869 seconds on IPAQ. The difference between computation time on IPAQ and Droid comes from two factors: (i) the IPAQ is older and has a slower processor, and (ii) more importantly, the built-in Java BigInteger implementation is much more efficient than C# implementation [6] for the IPAQ. Microsoft is planning to introduce built-in BigInteger class in their new .NET framework and we expect the running time on IPAQ can reduce significantly with a more efficient library. The server pre-computes vector encryption on behalf of a mobile host. The time to perform encryption for a single dimension on PC is 77.79 ms, as shown in Table III.

Protocol	Power Consumption (mJ)
Announcement - precompute	0.4773
Announcement - identify	2.907
Proximity Prefiltering	43.388
Protocol 1	2277.9
Protocol 2	286.3608

TABLE IV
CPU POWER CONSUMPTION ON DROID

Table IV shows CPU power consumption for running protocol 1 and 2 on Droid. To put the number into perspective, a fully charged Droid has 18,648,000 mJ. So the computation involved in the two protocols consume 0.0122% and 0.0015% total power, respectively. These numbers indicate that both protocols are practical in terms of power, as well.

In terms of operation, BigMul is the most expensive since it involves d 900-bit random numbers. Self-blinding cost also depends on the random number used (900 bits). In our evaluation, we do not use negative numbers. While negative numbers affect the performance of multiplication (since raising a negative power requires an inverse operation), their influence can be eliminated by letting the server pre-compute the inverse of each dimension in the encrypted vector and giving them to users. This increases the communication cost between users without leaking more information.

In terms of message sizes, for protocol 1, request message contains the homomorphic public key (n, g) and encrypted vector. n is a 1024-bit number, and g has at most 2048 bits. The encrypted vector has d encrypted numbers, each with at most 2048 bits. Hence the total request message is at most 8064 bytes for 30-dimensional social coordinates. The reply message contains two encrypted numbers, each having at most 2048 bits. So the total size is within 512 bytes. Protocol 2 contains only one encrypted number in the reply but involves 4 additional messages as in proximity prefiltering (described in Section V-A) except the last message contains the dot product value instead of YES or NO, and Alice does not send γ, and the numbers are much bigger because one participant is now using the random vector. The total message size in our implementation is around 8900 bytes for protocol 1 and around 16K bytes for protocol 2. The exact value may vary depending on the encryption result and the random numbers.

The largest message among all protocols is the protocol 1 request message (8K bytes), which takes 3.96 ms and 0.195 mJ to transmit on average over 200 runs on the Droid. Hence, the communication cost for the protocols is not significant compared to the computation costs.

VI. RELATED WORK

Our work is related to the following research areas:

Social networks: The explosive growth of online social networks has attracted significant attention [1]–[3], [22]. Previous studies (*e.g.*, [19], [20], [31]) show that path-ensemble based proximity measures, such as Katz measure [16], rooted PageRank [19], [20], and escape probability [31], are effective in predicting links between users. [30] develops proximity embedding algorithm for efficient and accurate proximity estimation in large social networks. It is applicable to all path-ensemble based proximity measures, so we use it in our study.

Mobile social networks: The popularity of mobile social networks has increased rapidly. [9] describes social serendipity to perform matchmaking in mobile social networks. Loopt [21] is a mobile geo-location service that notifies users of friends' location and activities via detailed interactive maps.

Both schemes rely on a centralized server. Since users are often reluctant to share their personal information like location and status information with a server, this limits applicability. Mobiclique [28] is a middleware that allows mobile phone users to connect to others over ad-hoc networks to exchange social network identity information and forward messages. It assumes all users are trusted, and ignores privacy and security.

Privacy in online social networks: [4] applies attribute-based encryption to provide fine-grained access control to the personal information. [32] proposes a social attestation that certifies a social relationship between any two parties. The recipient of a social attestation can use it to prove to a third party (*e.g.*, an online system) that (s)he has a certain relationship with the sender and get access to the restricted content. These works are complementary to our protocols. [11] develops a cryptographic private matching technique that exploits friend-of-friend relationships to automatically generate whitelists for email senders, and [10] proposes enhancement. Since set intersection is a special case of dot product, our verifiable secure dot product protocol can be applied to their context to provide stronger security guarantees.

Privacy in wireless networks: Wireless network privacy is a serious concern due to ease of eavesdropping. [27] identifies several explicit identifiers in 802.11 MAC header that can be used to identify and track users. *SlyFi* [13] is an 802.11-like link layer protocol that obfuscates all transmitted bits to remove explicit identifiers and increase privacy. [8] provides flexible presence sharing between users with social relationships by broadcasting clique signals. It provides presence sharing among strangers by letting users broadcast opaque identifiers, which can be resolved to an identity at a centralized trusted broker. This is related to our virtual ID, but we use virtual ID as a public key to communicate directly with other users in the mobile social network.

Privacy-preserving computation: There are two classes of privacy-preserving computation: encryption-based techniques and obfuscation-based techniques. Encryption-based protocols like [12] are based on homomorphic encryption and are computationally expensive. The obfuscation based schemes [15], [34] are light weight but tend to leak some information. Privacy-preserving dot product protocols are important in the area of distributed range query computing and association mining [12], [15]. A unique feature of our approach is that it is both privacy-preserving and verifiable.

VII. Conclusion

With increasing popularity of mobile social networks, it is important to develop secure and practical protocols to enable users to effectively interact with each other. In this paper, we develop a secure friend discovery protocol for mobile social networks, and use both analysis and real implementation to demonstrate its feasibility and effectiveness. We hope that this paper will inspire other researchers to explore protocol design for mobile social networks, which is a rapidly growing area.

Acknowledgments: This research is supported in part by NSF Grants CNS-0916106, CNS-0546755, and CNS-0916309.

References

[1] L. A. Adamic, O. Buyukkokten, and E. Adar. A social network caught in the web. *First Monday*, 2003.

[2] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *Proc. of WWW*, 2007.

[3] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: Membership, growth, and evolution. In *Proc. of KDD*, 2006.

[4] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: An online social network with user-defined privacy. In *Proc. of ACM SIGCOMM*, Aug. 2009.

[5] R. M. Bell, Y. Koren, and C. Volinsky. Chasing $1,000,000: How we won the Netflix Progress Prize. *Statistical Computing and Statistical Graphics Newsletter*, 18(2):4–12, 2007.

[6] The code project: C# BigInteger class. http://www.codeproject.com/KB/cs/biginteger.aspx.

[7] Broadband Internet statistics. http://www.internetworldstats.com/dsl.htm.

[8] L. Cox, A. Dalton, and V. Marupadi. SmokeScreen: flexible privacy controls for presence sharing. In *Proc. of ACM MobiSys*, 2007.

[9] N. Eagle and A. Pentland. Social serendipity: Mobilizing social software. *IEEE Pervasive Computing*, Apr. 2005.

[10] M. J. Freedman and A. Nicolosi. Efficient private techniques for verifying social proximity. In *Proc. of IPTPS*, Feb. 2007.

[11] S. Garriss, M. Kaminsky, M. J. Freedman, B. Karp, D. Mazieres, and H. Yu. RE: Reliable Email. In *Proc. of NSDI*, 2006.

[12] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen. On private scalar product computation for privacy-preserving data mining. In *Proc. of ICISC*, 2004.

[13] B. Greenstein, D. McCoy, J. Pang, T. Kohno, S. Seshan, and D. Wetherall. Improving wireless privacy with an identifier-free link layer protocol. In *Proc. of MobiSys*, 2008.

[14] S. Hill, F. Provost, and C. Volinsky. Network-based marketing: Identifying likely adopters via consumer networks. *Statistical Science*, 2006.

[15] I. Ioannidis, A. Grama, and M. Atallah. A secure protocol for computing dot-products in clustered and distributed environments. In *Proc. of ICPP*, 2002.

[16] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 1953.

[17] F. Kerschbaum and O. Terzidis. Filtering for private collaborative benchmarking. In *Proc. of LNCS*, 2006.

[18] B. Krishnamurthy and C. E. Wills. On the leakage of personally identifiable information via online social networks. *ACM SIGCOMM Computer Communications Review*, Jan. 2010.

[19] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proc. of CIKM*, 2003.

[20] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.*, 2007.

[21] Loopt. http://www.loopt.com.

[22] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proc. of IMC*, 2007.

[23] Mobile Broadband Review 2010. http://mobile-broadband-services-review.toptenreviews.com/.

[24] Mobile social networking apps spark privacy concerns. http://channel.hexus.net/content/item.php?item=25288.

[25] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of EUROCRYPT*, 1999.

[26] Paillier's homomorphic cryptosystem (java implementation). http://www.csee.umbc.edu/~kunliu1/research/Paillier.html.

[27] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *Proc. of MobiCom*, 2007.

[28] A.-K. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot. Mobiclique: middleware for mobile social networking. In *Proc. of WOSN*, 2009.

[29] Powertutor : A power monitor for android-based mobile platforms. http://powertutor.org//.

[30] H. H. Song, T. W. Cho, V. Dave, Y. Zhang, and L. Qiu. Scalable proximity estimation and link prediction in online social networks. In *Proc. of IMC*, 2009.

[31] H. Tong, C. Faloutsos, and Y. Koren. Fast direction-aware proximity for graph mining. In *Proc. of KDD*, 2007.

[32] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman. Lockr: Better privacy for social networks. In *Proc. of CoNext*, Dec. 2009.

[33] Top 5 countries with most cell phone subscribers. http://www.associatedcontent.com/article/1764228/top_5_countries_with_m%ost_cell_phone.html?cat=15.

[34] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proc. of ACM SIGKDD*, 2002.

[35] Wikipedia. Social network. http://en.wikipedia.org/wiki/Social_network.

[36] Worldwide pricelist for iPhone 3G Plans. http://www.unwiredview.com/2008/07/16/worldwide-pricelist-for-iphone-3g%-plans/.

[37] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: Defending against Sybil attacks via social networks. In *Proc. of SIGCOMM*, 2006.