The Shy Mayor: Private Badges in GeoSocial Networks

Bogdan Carbunar[1], Radu Sion[2], Rahul Potharaju[3] and Moussa Ehsan[2]

[1] Florida International University Miami, FL
carbunar@cs.fiu.edu
[2] Stony Brook University, Stony Brook, NY
{sion,mehsan}@cs.stonybrook.edu
[3] Purdue University, West Lafayette, IN
rpothara@cs.purdue.edu

Abstract. Location based social or *g*eosocial networks (GSNs) have recently emerged as a natural combination of location based services with online social networks: users register their location and activities, share it with friends and achieve special status (e.g., "mayorship" badges) based on aggregate location predicates. Boasting millions of users and tens of millions of daily check-ins, such services pose significant privacy threats: user location information may be tracked and leaked to third parties. Conversely, a solution enabling location privacy may provide cheating capabilities to users wanting to claim special location status. In this paper we introduce new mechanisms that allow users to (inter)act privately in today's geosocial networks while simultaneously ensuring honest behavior. An Android implementation is provided. The Google Nexus One smartphone is shown to be able to perform tens of badge proofs per minute. Providers can support hundreds of million of check-ins and status verifications per day.

1 Introduction

Location based services offer information and entertainment services to mobile users, that rely on the geographical position of their mobile devices. A recently introduced but popular example, is the geosocial network (GSN) – a social network centered on the geographical position of its users. Services such as Foursquare [1], Yelp [2] or Gowalla [3] allow users to register or "check-in" their location, share it with their friends, leave recommendations and collect prize "badges". Badges are acquired by checking-in at certain locations, following a required pattern simultaneously with other users, i.e. multiplayer games, or obtaining the highest number of check-ins during a time window ("mayor" badge).

Besides keeping track of their friends' location, the user incentives for participation include receiving promotional deals, coupons and personalized recommendations. The main source of revenue for service providers lies in ad targeting. Boasting millions of users [4] and tens of millions of location check-ins per day [5], GSNs can provide personalized, location dependent ads. As such, the price of participation for users is steep: compromised location privacy. Service providers learn the places visited by each user, the times and the sequence of visits as well as user preferences (e.g., places visited more often) [6, 7]. The implications are significant as service providers may use this information in ways that the users never intended when they signed-up (e.g., having their location information shared with third parties [8, 9]).

While compromised privacy may seem a sufficient reason to avoid the use of such services, it may not be necessary. Instead, we propose here a framework where users themselves store and manage their location information. The provider's (oblivious) participation serves solely the goal of ensuring user correctness. This enables users to privately and securely check-in and acquire special location based status, e.g., in the form of badges. Badges are defined as aggregate predicates of locations. We then devise solutions to support a variety of such predicates, including (i) registering a pre-defined number of times at a location or set of locations, (ii) registering the most number of times (out of all the users) at a location and (iii) simultaneously registering with k other users at a location.

Given the recent surge of location privacy breaches and the ensuing liabilities issues [10], implementing privacy solutions may ultimately be in the service provider's best interest.

To this end, the problem is two-faceted. On one side, clients need strong privacy guarantees: The service provider should not learn user profile information, including (i) linking users to (location,time) pairs, (ii) linking users to any location, even if they achieve special status at that location and (iii) building user profiles – linking multiple locations where the same user has registered. On the other side, when awarding location-related badges the service provider needs assurances of client correctness. Otherwise, since special status often comes with financial and social perks, clients have incentives to report fake locations [11], copy and share special status tokens, or check-in more frequently than allowed.

We note that, despite being seemingly attractive, the simple use of client pseudonyms as a means to provide client privacy during check-ins and special status requests is vulnerable to profile based de-anonymization attacks [12, 13].

In this work we first define essential *privacy* and *correctness* properties for the aggregate location predicate problem. We then introduce SPOTR , a venue-oriented location verification protocol, that allows GSN providers to certify the locations claimed by users. SPOTR relies on single-use, 2 dimensional QR codes, displayed on devices inside participating venues. Furthermore, we propose three privacy-preserving solutions for the aggregate location predicate problem. The solutions deploy cryptographic techniques such as zero-knowledge proofs, quadratic residuosity constructs, threshold secret sharing and blind signatures. Clients collect special, provider-issued tokens during check-ins, which they either aggregate to build generic, non-traceable badges, or use to build zero-knowledge proofs of ownership. Client correctness is partly ensured by the use of blind signatures of single-use tokens.

We have implemented and evaluated the performance of our solutions on a Revision C4 BeagleBoard, Google Nexus One smartphones and a 16 quad-core server. Experimental results are extremely positive. The GSN provider can support thousands of check-ins and special status verifications per second, while a smartphone can build strongly secure aggregate location and correctness proofs in just a few seconds.

2 Related Work

Location Cloaking: Anonymization, pseudonimization and location and temporal cloaking techniques (introducing small errors in location reports in order to provide 1-out-of-k anonymity) have been initially proposed in [14], followed by a significant body of work [15–18]. These techniques are vulnerable to de-anonymization attacks [12, 13]: the address of a user that frequently reports a residential address may be identified by computing the intersection of the cloaked reports.

Location Verification: Saroiu and Wolman [19] introduced the location proof concept – a piece of data that certifies a receiver to a geographical location. The solution relies on special access points (APs), that are able to issue such signed proofs. Luo and Hengartner [20] extend this concept with client privacy, achieved with the price of requiring three independent trusted entities. Note that both solutions rely on the existence of specialized APs or cell-towers, that modify their beacons and are willing to participate and sign arbitrary information. To address the central management problems, Zhu and Cao [21] proposed the APPLAUS system, where co-located, Bluetooth enabled devices compute privacy preserving location proofs.

Proximity Alerts: Zhong et al. [22] have proposed three protocols that privately alert participants of nearby friends. Location privacy here means that users of the service can learn a friend's location only if the friend is nearby. Manweiler et al. [23] propose several cloaking techniques for private server-based location/time matching of peers. Narayanan et al. [24] proposed several other solutions for the same problem, introducing the use of location tags as a means to provide location verification.

Summary: Existing work has focused on (i) hiding user location from LBS providers and other parties and on (ii) enabling users to prove claimed locations. Besides proposing a novel, venue oriented approach for location verification, in this paper we focus on the next step, of anonymizing location aggregates defined by geosocial networks.

This paper extends our previous work [25] with a location verification solution, Spotr , detailed descriptions of the private aggregate location predicate protocols (*GeoBadge*, *GeoM* and *MPBadge*), proofs of correctness and privacy, details of Foursquare as well as implementation results of Spotr , *GeoBadge* and *GeoM*.

3 Model

3.1 The System

We consider a geosocial network provider, S. Each subscriber (or user) has an account with S. Subscribers are assumed to have mobile devices equipped with a GPS receiver and a Wi-Fi interface (present on most smartphones). To use the provider's services, a client application needs to be downloaded and installed. Subscribers can register and receive initial service credentials, including a unique user id; let Id_A denote the id of user A. In the following we use the terms *user* and *subscriber* to refer to users of the service and the term *client* to denote the software provided by the service and installed by users on their devices.

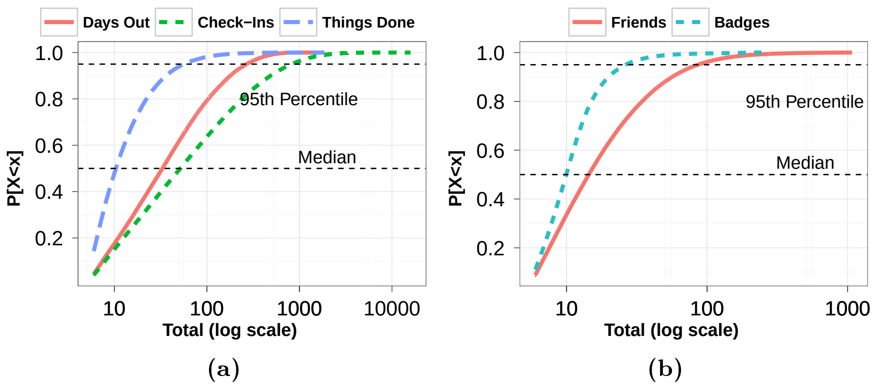


Fig. 1. Foursquare stats: (a) CDF of days out, check-ins and things done by users. (b) Badge and friends evaluation.

Foursquare: In the following, we model the online geosocial network provider S after the most popular in existence to date, Foursquare [1]. In Foursquare, users report their location, through *check-ins* at venues of interest, share it with friends (e.g., imported from Facebook or discovered and invited on Foursquare) and are awarded points and "badges". A user with more check-in days at a venue than anyone else in the past 60 days becomes the "Mayor" of the venue. Foursquare has partnered with a long list of venues (bars, cafes, restaurants, etc) to reward the Mayor with freebies and specials. Foursquare imposes a discrete division of time, in terms of *epochs*. A user can check-in at one venue at most once per epoch. This strategy has made Foursquare quite popular, with a constantly growing user base, which we currently estimate at over 14 million users.

In order to understand the utility of our solution to a GSN provider such as Foursquare, we have collected profiles from 781,239 randomly selected Foursquare users. Our first question was how active are Foursquare users. Figure 1(a) shows the CDF of the number of check-ins, days out (days the user was actively performing check-ins) and things done (e.g., reviews left for a venue) by users. Note that 45% of the collected users have between 80 and 950 check-ins, for between 50 and 300 days of activity (at this time Foursquare is 2 years and a half old).

This shows that many Foursquare users are very active. Our second question regards the popularity of badges in geosocial networks. Figure 1(b) shows the cumulative distribution function (CDF) of the number of badges earned by users as well as their friends. Note that 45% of the users (between the median and the 95th percentile) have between 10 and 50 badges and between 20 and 95 friends. This, coupled with the large numbers of check-ins reported strengthens our belief that private badge protocols are needed.

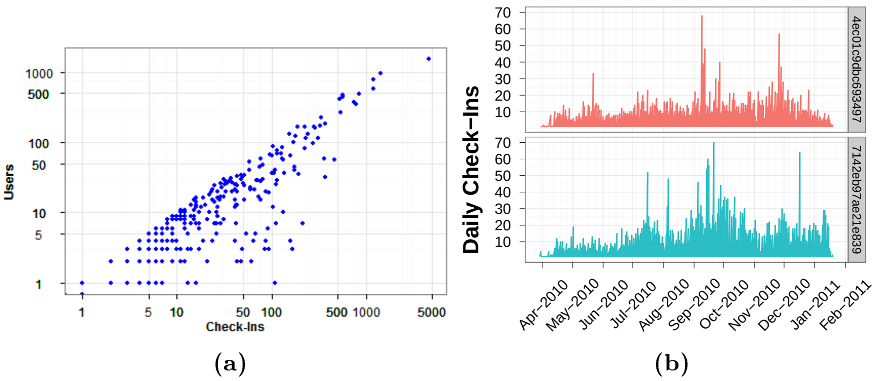


Fig. 2. (a) Scatterplot check-ins vs. users in a small town. (b) Per-venue check-in distribution over time for two random venues.

We corroborate the check-in data in a location-aware fashion: Figure 2(a) shows the scatter plot of check-ins vs. users in one of the most active locations in our dataset, the city of Babylon in Long Island, NY. Each point on the plot denotes a venue, the x axis shows the total number of check-ins recorded at the venue and the y axis shows the total number of users that have performed the check-ins. Note that a few venues record 1000-5000 check-ins, from more than 500 users. Most venues however range from a few tens to a few hundred check-ins and users. Finally, Figure 2(b) shows the evolution between August 2010 and February 2011 of the number of check-ins per day for two randomly selected venues. The number of check-ins range between 3 to almost 70 per day. Our conclusions are that Foursquare users are actively checking-in and venues record many daily check-ins. This data rich environment can be a goldmine for rogue GSN providers. Moreover, the number of recorded check-ins suggests that badges and mayorship are likely to become objects of contention. These points show that devising private and secure "badging" protocols is a problem of primary importance for GSNs.

Geo: A private GSN. A full-fledged privacy solution is composed of a set of protocols $Geo = \{Setup, RegisterVenue, Subscribe, CheckIn, StatVerify\}$. $Setup$ is executed initially by the service provider to generate system-wide parameters and $RegisterVenue$ is used to register a new venue with the provider S. $Subscribe$ is initiated by a client when registering with the service. $CheckIn$ is executed by a client to report its presence at a venue to S and $StatVerify$ is executed when the client has accumulated sufficient check-ins and claims its special status. Each operation returns -1 to report failure or 0 for success.

We support three special status types. First, *location badges* (see Section 6), issued after the client runs $CheckIn$ during k different epochs at a venue V (e.g., "local" badge in Foursquare [1]) or after the client runs $CheckIn$ at k different, select, locations (e.g., "adventurer" badge). Second, *mayorships* (see Section 7), issued when the client has the largest number of $CheckIn$ runs, at most one per epoch, in the past m epochs at a given venue V. m is a system parameter. Third, *multi-player badges* (see Section 8), issued when the client runs $CheckIn$ simultaneously with s other users at the same location. s is a system parameter.

3.2 Privacy and Correctness Properties

Server Model. The provider S is honest, yet curious. S follows the protocol correctly, but is interested in collecting tuples of the format (Id, V, T), where Id is a user id, V is a venue and T is a time value. To this end, it may collude with existing clients and generate Sybil clients to track users of interest. The provider has no interest in colluding with users to issue badges without merit. To achieve privacy, intuitively, the provider should learn nothing about Geo clients. First, this includes the venues at which users run the $CheckIn$ function, how many times and when they run $CheckIn$ (in total and for any venue). We note that this necessarily includes also hiding correlations between venues where a given client has run $CheckIn$. We formalize this intuition using games run between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. $\mathcal{A}$ controls the service provider and any number of clients, thus controls the initial parameter generation functionality (e.g., the *Setup* function). $\mathcal{A}$ shares public parameters with $\mathcal{C}$. $\mathcal{C}$ controls two clients C_0 and C_1. $\mathcal{C}$ initially runs the *Subscribe* function with $\mathcal{A}$ for the two clients and obtains their unique identifiers.

In a first CheckIn-Indistinguishability game, we model the adversary's inability to distinguish between clients during $CheckIn$ executions, even when the adversary controls an initial trace of $CheckIn$ executions. The game is defined for a given venue V.

CheckIn Indistinguishability (CI-IND). $\mathcal{A}$ generates l bits $c_1, .., c_l$ and sends them to $\mathcal{C}$. For each bit c_i, $\mathcal{C}$ executes $CheckIn(C_{c_i}(V), \mathcal{A})$. After processing all l bits, $\mathcal{C}$ flips a bit $b \in \{0, 1\}$ and runs $CheckIn(C_b, \mathcal{A})$. $\mathcal{A}$ outputs a bit b'. A solution is said to be CI-IND if the advantage of $\mathcal{A}$ in the CI-IND game, $Adv(\mathcal{A}) = |Pr[b = b'] - 1/2$—, is negligible.

In a second, StatVerify-Indistinguishability game, the adversary (e.g., service provider) should be unable to distinguish between clients running $StatVerify$, *even if* the adversary is able to trace client $CheckIn$ executions.

StatVerify Indistinguishability (SV-IND). $\mathcal{C}$ performs l $CheckIn$ and m $StatVerify$ operations on behalf of C_0 and C_1, as requested by $\mathcal{C}$. A $StatVerify$ operation succeeds only if special status has been achieved by the corresponding client in the previous $CheckIn$ runs. $\mathcal{A}$ generates $k > 2s$ new bits $c_1, .., c_k$ such that at least s of them are 0 and at least s of them are 1. $\mathcal{A}$ sends $c_1, .., c_k$ to $\mathcal{C}$. For each bit c_i, $\mathcal{C}$ runs $CheckIn(C_{c_i}(V), \mathcal{A})$. Finally, $\mathcal{C}$ flips a coin $b \in \{0, 1\}$ and runs $StatVerify(C_b(V, s), \mathcal{S})$. $\mathcal{A}$ outputs a bit b'. A solution is said to be SV-IND if the advantage of $\mathcal{A}$, $Adv(\mathcal{A}) = |Pr[b = b'] - 1/2|$, is negligible.

Note that even though the $CheckIn$ runs are executed for the same venue V, irrespective of the client, the SV-IND game is also suitable for the mayor badge – only one of the clients (C_b) will become mayor. However, for mayor badges, the value s needs to exceed the number of $CheckIn$ executions run on behalf of any client in the first step of the SV-IND game. Finally, we also need to allow the server to collect venue-based statistics:

Provider Usability. The service provider can count the number of $CheckIn$ executions for any venue and list the badges or mayorships awarded at any site.

Client Model. The client is assumed to be malicious. Malicious clients can be outsiders that are able to corrupt existing devices or may be insiders, i.e., subscribers, users that have installed the client. Malicious clients can try to cheat on their location (claim to be in a place where they are not [11]), attempt to prove a status they do not have, or disseminate credentials received from the server to other clients. The latter case includes any information received from the server, certifying presence at a specific location. Formally, we need a solution that has the following properties.

Status Safety. The challenger $\mathcal{C}$ controls the service provider and the adversary $\mathcal{A}$ controls any number of clients. The challenger runs first the $Setup$ protocol and provides $\mathcal{A}$ with its public parameters. $\mathcal{A}$ runs $Subscribe$ any number of times to generate clients. $\mathcal{A}$ then runs $CheckIn$ with $\mathcal{C}$ for any number of venues, but at most $k-1$ times for any venue. $\mathcal{A}$ runs $StatVerify$ with $\mathcal{C}$. The advantage of $\mathcal{A}$ is defined to be $Adv(\mathcal{A}) = Pr[StatVerify(C(params_C), S(priv_S)) = 1]$. We say that a solution is safe if $Adv(\mathcal{A})$ is negligible.

Note that a safe solution also prevents clients from running $CheckIn$ for venues where they are not located – otherwise $\mathcal{A}$ would succeed in $StatVerify$ with less than k $CheckIn$ runs at a site.

Token Non-distributability. No client or coalition thereof can use the same set of tokens more than once.

Token-Epoch Immutability. No client or coalition thereof can obtain more than one token per site per epoch.

4 Tools

Cryptographic Tools. We use semantically secure cryptosystems, as well as unforgeable signature schemes. Unforgeability is defined in terms of security "against one-more-forgery", where the user engaged in l runs of with the signer cannot obtain more than l signatures. We also use blind signatures with the standard (i) blindness and (ii) unforgeability properties. Blindness means that the signer learns nothing about the signed messages. Let BS denote the blind signature generation protocol. We use cryptographic hashes that are easy to compute and are (i) pre-image resistant, (ii) second pre-image resistant and (iii) collision resistant. We use $x \in_R S$ to denote the random choice of x from set S.

Anonymizers. Anonymizers or mix-nets [26, 27] are tools that attempt to make communication untraceable and unlinkable. Untraceability means that it should be infeasible to find the identity of the issuer of a given set of messages. Unlinkability implies the infeasibility of discovering pairs of communicating entities. We denote the anonymizer by Mix.

Anonymous Authentication. We rely on anonymous authentication techniques with revocation and identity escrow, e.g., [28], performed over Mix, to enable users to prove they are service subscribers.
QR-Assumption. Given a large composite $n = pq$, where p and q are safe primes and given n but not p and q, it is computationally hard to decide if any value v, whose Jacobi symbol $(v|n)$ is 1, is a quadratic residue or not. v is a quadratic residue if there exists a value y such that $y^2 = v \ mod \ n$.

5 Spotr : Secure Location Verification

Our work relies on the ability of the GSN provider to privately verify the claimed locations of clients. In this section, we propose Spotr , a solution that achieves this goal. Since venues have the most incentives to correctly reward users, Spotr relies on the co-operation of venue owners: owners need to install and operate a device inside their venues. We show however that simple, off-the-shelf equipment is sufficient and no Internet connectivity is required – thus imposing solely a one time investment.

Let Spotr$_V$ denote the device installed at venue V. Spotr$_V$ requires the owner of venue V to generate a public/private key, store the private key on Spotr$_V$ and report the public key to S, the GSN provider. S associates with each venue the owner's public key. Spotr relies on Quick Response Codes (QR codes), 2D barcodes consisting of black modules arranged in a square pattern on a white background, that can store up to 2,953 bytes. At any time, Spotr$_V$ displays a QR code encoding $T, \Delta T, S_O(H(T, ctr))$, containing the time when the QR code was generated, an expiration increment ΔT and the owner O's signature on these values. The following takes place during a check-in at venue V:
CheckIn$(C(Id, V, T, pub_S), S(priv_S, pub_O))$: The user approaches Spotr$_V$, snaps a picture of the displayed QR code and sends it, along with the venue identity, over Mix, to S. With the public key pub_O of the owner O of venue V, S verifies the correctness of the received signature, and that the current time is between $[T, T + \Delta T]$. If the verifications succeed, S validates the check-in. Otherwise, it returns -1. Spotr$_V$ changes the QR code to encode a fresh timestamp when either (i) the current time approaches $T + \Delta T$ or (ii) it detects that the current QR code has been read (see Section 9 for implementation details).

Spotr will be used as a building block by all subsequent solutions, GeoBadge, GeoM and MPBadge. Its security is proved as part of $GeoBadge$.

6 Geo-Badge

We now introduce $GeoBadge$, a private protocol that allows users to prove having visited the same location k times. At the end of the section we show how to adapt this solution to support private proofs of visiting k different places. In a nutshell, $GeoBadge$ works as follows: each subscribed client contacts the provider over the anonymizer Mix, authenticates anonymously, proves its current location and obtains a blindly signed, single use nonce and a share of a secret associated with the current venue. When k shares have been acquired (after k check-ins at the same venue) the client is able to reconstruct the secret - which is the proof required for the badge of the venue. The single use nonces prevent users from distributing received shares (or proofs).

GeoBadge extends *Geo* and provides the skeleton on which we build the subsequent solutions. Each client maintains a set Tk, storing all the tokens accumulated during *CheckIn* runs. When the client accumulates enough tokens in Tk to achieve special status, it runs $StatVerify$, aggregating the tokens in Tk. In the following we instantiate each protocol, executed between a client C and the GSN provider S.

Setup: The server chooses a large prime p and generates a random key K. The server publishes p and keeps K secret.

RegisterVenue$(C(), S(priv_S))$: The client C that registers venue V, called the owner of the venue, sends to S its public key. For each new venue V, S generates a secret M_V randomly. S uses a threshold secret sharing solution to compute shares of M_V, by generating a polynomial Pol of degree $k-1$ whose free coefficient is M_V: $Pol(x) = M_V + c_1x + c_2x^2 + ... + c_{k-1}x^{k-1}$. S keeps Pol's coefficients secret but publishes the degree k and the verification value $Ver_V = H(M_V H_K(V) \ mod \ p)$. S stores Pol's coefficients for V, along with the public key of V's owner - to be used as part of SPOTR (see Section 5).

Subscribe$(C(), S(pub_S, priv_S))$: The communication in this step is performed over Mix, to hide C's location from S. C runs the setup stage of the Anonymous Authentication protocol of Boneh and Franklin [28] to obtain tokens that allow it later to authenticate anonymously with the server.

CheckIn$(C(Id, V, T, pub_S), S(priv_S))$: Let time T be during epoch e. The following actions are performed by a client C and the service provider S:

– **Anonymous Authentication:** C runs the anonymous authentication procedure of Boneh and Franklin [28] to prove to S that it is a subscriber. This step is performed over Mix.

– **Location Verification:** C runs SPOTR (Section 5) to prove presence at V.

– **Token Generation:** S generates $x_e = H_K(e) \ mod \ p$ and computes $y_e = Pol(x_e) \ mod \ p$. S sends to C (as a reply over the anonymizer) the tuple $(x_e, c_e, S_S(E(R)))$, where $c_e = H_K(V)y_e \ mod \ p$ and the last field denotes the signed blinded nonce. C "unblinds" the signed nonce, $D(S_S(E(R))) = S_S(R) = s_e$ and stores (x_e, c_e, s_e) into its token set Tk.

StatVerify$(C(Id, V, k, Tk, pub_S), S(priv_S))$: Let $Tk = \{(x_1, c_1, S_S(R_1)), .., (x_k, c_k, S_S(R_k))\}$ – C has accumulated k tokens from S for a venue V. Let $l_j(x) = \Pi_{m=1..k, m \neq j} \frac{x - x_m}{x_j - x_m} \ mod \ p$ be the Lagrange coefficients. The following steps are executed, over Mix:

– C computes $SS = \Sigma_{j=1..k} c_j l_j(0)$. C verifies that $H(SS) = Ver_V$. If the verification fails, C outputs -1 and stops. Otherwise, it sends SS, along with the set of signed nonces, $(S_S(R_1), .., S_S(R_k))$ and the venue V to S.

– S verifies that (i) the k random values are indeed signed by it, (ii) that $R_1, .., R_k$ are unique and have not been used before and (iii) that $H(SS) = Ver_V$. If either verification fails, S outputs -1. Otherwise, S stores the values $R_1, .., R_k$, then issues a badge $S_S(\text{"}GeoBadge'', V, T_c)$ for the venue V, where T_c is the current issuance time. S sends this badge to C (as a reply over Mix).

Note that while described in different steps for clarity, the anonymous authentication and location verification steps in $CheckIn$ can be performed simultaneously, to avoid additional traffic and delays.

6.1 Analysis

Correctness. The following holds due to Lagrange interpolation:

$$SS = \sum_{j=1}^{k} c_j l_j(0) = H_K(V) \sum_{j=1}^{k} Pol(x_j) l_j(0) = H_K(V) Pol(0) = H_K(V) M_V$$

Theorem 61 *GeoBadge is CI-IND.*

Proof. (Summary) Following the CI-IND game, $\mathcal{A}$'s view consists of the outcome of $l+1$ anonymous authentication procedures, $l+1$ venue signatures (from QR codes) and $l+1$ blinded random values. The venue signatures carry no information identifying the client. The blinded random values are information theoretical secure. Thus, if $\mathcal{A}$ can distinguish between C_0 and C_1 in the last step of the game, we can build an adversary that has a non-negligible advantage against either (i) the anonymous authentication solution of Boneh and Franklin [28] or (ii) the untraceability property of Mix.

Theorem 62 *GeoBadge is SV-IND.*

Proof. (Summary) At the completion of the SV-IND game $\mathcal{C}$ is able to reconstruct $Y^{H_K(P)}$ for both C_0 and C_1. $\mathcal{A}$ has published a pre-commitment for $Y^{H_K(P)} - V_P$. Note that $\mathcal{C}$'s verification of $H(SS) = V_P$ prevents $\mathcal{A}$ from guessing b based on the value $\mathcal{C}$ to reconstruct during $StatVerify$. Thus, if the adversary has non-negligible advantage in the SV-IND game then we can also build an adversary that has non-negligible advantage against either (i) the untraceability property of Mix, (ii) the semantic security of the blinding algorithm E, or (iii) the information theoretic security of the threshold secret sharing mechanism.

Theorem 63 *GeoBadge provides Status Safety.*

Proof. (Summary) SPOTR efficiently prevents a single attacker from falsely claiming presence at V: without being present, the attacker is unable to predict or forge the signature displayed on SPOTR$_V$ (see the security against one-more-forgery of the signature scheme from Section 4). Then, if there exists an adversary that has non-negligible advantage in the Badge-Safety game we can build an adversary that has a non-negligible advantage against (i) the pre-image resistance property of hashes (inverting $V_P = H(Y^{H_K(P)})$) or (ii) the information theoretic threshold secret sharing technique (including combining shares generated at multiple sites).

Note that trivially $GeoBadge$ also provides the Token Non-distributability property – the use of the single use, server signed random nonces prevents more than one run of $StatVerify$ for a given set of tokens. The Token-Epoch Immutability property also holds (no colluding clients can obtain more than one token for a venue during any epoch e), since the pair (x_e, c_e) is a deterministic function of e.

6.2 Adventurer: The A-Badge

The "adventurer" badge is unlocked when the user registers at k different locations. *GeoBadge* can be easily tweaked to support this functionality: the provider assigns one share (one point of the polynomial Pol) to each venue that is part of the *ABadge* network. The free coefficient of Pol is the secret which unlocks the badge. Whenever a user checks-in at one venue, it receives the share associated with the venue. After visiting k venues, the user has k shares and can reconstruct the secret and unlock the badge. Note that multiple check-ins at the same venue will retrieve the same share, thus forcing the client to visit k *d*ifferent venues.

7 Geo-M

Using the Foursquare terminology, the user that has run *CheckIn* the most number of times, at a venue S, within the past m epochs, becomes the mayor of the place. We now propose *GeoM*, a solution that allows users to achieve this status with privacy, while allowing anyone to verify correctness. *GeoM* extends *GeoBadge*: First, it allows clients to prove any number of check-ins, not just a pre-defined value k. Second, the check-ins are time constrained: clients have to prove that all check-ins have occurred in the past m epochs. Finally, client issued proofs can be published by the provider to be verified by any third party, without the risk of being copied and re-used by other clients.

GeoM achieves these features by requiring the service provider to issue only one token for each venue during any epoch. When a user has accumulated k tokens for a venue, it proves to the provider that it has k out of the m tokens given in the past m epochs for that venue. The proof is in zero knowledge (ZK) and if it verifies is published by the server.

Setup: The server generates two large safe primes p and q and the composite $n = pq$. Let N denote n's bit length. S publishes n and keeps p and q secret.

RegisterVenue$(C(), S(priv_S))$: For each newly registered venue V, S generates a new random seed r_V and uses it to initialize a pseudo-random number generator G_V. During every epoch e_i, for the venue V, S generates a fresh random token t_i, using G_V, and publishes $t_i^2 \ mod \ n$.

CheckIn$(C(Id, V, T, q, pub_S), S(priv_S))$: Inherits the Anonymous Authentication and Location Verification steps from *GeoBadge*. If they succeed, let time T be within epoch e_i, when the provider's published token value is $t_i^2 \ mod \ n$. C generates a random nonce R, engages in a blind signature protocol with S and obtains $BS(R)$. S also sends to C the value t_i, the square root of the value published for the epoch e_i. C stores t_i in the set Tk along with the blinded signed nonce, $BS(R)$. All communication takes place over Mix.

StatVerify$(C(Id, V, k, Tk, pub_S), S(priv_S))$: Without loss of generality, let $T = \{(t_1, BS(R_1)), .., (t_k, BS(R_k))\}$ be the set of all tokens issued by S for venue V in the past m epochs and let $\mathcal{T}^2 = \{t_1^2, t_2^2, .., t_m^2\}$ denote the corresponding published values. Note that the membership of $\mathcal{T}^2$ changes during every epoch. The client and the server run the following steps s times (ZK proof of the client knowing k square roots of values from $\mathcal{T}^2$). If successful, at the end of the s steps S will be convinced with probability $1 - 2^{-s}$.

– C generates $y_1, .., y_m \in_R \{0,1\}^N$ and a random permutation π_1. C computes the set $M = \pi_1\{t_1^2 y_1^2, .., t_m^2 y_m^2\}$ and sends it to S. Note that C does not need to know $t_1, .., t_m$ to compute M.
– C generates $z_1, .., z_k \in_R \{0,1\}^N$ and a random permutation π_2 and computes the set $Proof = \pi_2\{t_1 z_1, .., t_k z_k\}$, which it sends to S.
– S flips a coin b and sends it to C.
– If b=0, C sends $y_1, .., y_m$ to S, which then verifies that for every $t_i^2 \in \mathcal{T}^2$, $t_i^2(y_i)^2$ occurs once in M.
– If b=1, C generates and sends $A = \pi_2\{a_1 = z_1^{-1}y_1, .., a_k = z_k^{-1}y_k\}$. S verifies that for every $p_i \in Proof$ and corresponding a_i, $(p_i a_i)^2$ occurs in M once.

If any step fails, S outputs -1 and stops. Otherwise, it generates a signed "mayor" token $S_S(\text{"}Mayor'', V, T_c)$ for venue V issued at time T_c and sends it to C. All communication in this step is done over Mix. To reduce delays, the ZK proof can be non-interactive – in the standard way, by making the challenge bits depend in an unpredictable way on the values sent to the server. This allows C to send the entire proof at once. S publishes the ZK proof for the current "mayor", which can be downloaded and verified by any third party.

7.1 Analysis

Theorem 71 *The $StatVerify$ protocol of $GeoM$ is a zero knowledge proof system of k square roots from $\mathcal{T}^2$.*

Proof. (Summary) To see that $GeoM$ is a proof system, we need to prove completeness and soundness.

Completeness – an honest server will be convinced by an honest client of the correctness of the proof. If b=0, S is convinced that M is obtained from $\mathcal{T}^2$ by multiplication with quadratic residues, y_i^2. That is, for each $t_i \in \mathcal{T}^2$, $t_i^2 y_i^2 \in M$. If b=1, S is convinced that C knows the square roots of k elements in M. This is because C can provide a_i values that satisfy $(p_i a_i)^2 = (t_i z_i z_i^{-1} y_i)^2 = t_i^2 y_i^2 \in M$. In conjunction, these two cases prove to S that C knows the square roots of k elements from $\mathcal{T}^2$ with probability $1 - 2^{-s}$.

Soundness – if the statement is false, no cheating client can convince an honest server that the statement is true, except with small probability. Without loss of generality, let us assume that C knows only $k-1$ square roots of $\mathcal{T}^2$, $t_1, .., t_{k-1}$. If C expects the challenge to be $b = 0$, C generates $y_1, .., y_m$ as in the protocol, builds M correctly but generates $Proof = \pi_2\{t_1 z_1, .., t_{k-1} z_{k-1}, z_k\}$, where z_k is random. If the challenge ends up being $b = 1$, C has to produce one a_j value that is equal to $y_j z_j^{-1}(t_j^2)^{1/2}$, for one $j \in k..m$. Due to the QR-Assumption, C is unable even to tell whether any t_j^2 is a quadratic residue or not. If C expects the challenge to be 1, it builds $M\pi_1 = \{t_1^2 w_1^2, .., t_{k-1}^2 w_{k-1}^2, w_k^2, .., w_m^2\}$, where the w_i's are random. It then build Proof to be
$Proof = \pi_2\{t_1 z_1, .., t_{k-1} z_{k-1}, z_k\}$. If $b = 1$, C can provide square roots for k values in M. If $b = 0$ however, C has to produce $m - k + 1$ values y_j such that $y_j = w_j(t_j^{-2})^{1/2}$, which contradicts again the QR-Assumption. The chance of a cheating client to succeed after s repetitions is 2^{-s}.

Zero Knowledge – if the statement is true, no cheating server learns anything except this fact. We prove this by following the approach from [29, 30]. Specifically, let S^* be an arbitrary, fixed, expected polynomial time server Turing machine. We generate an expected polynomial time machine M^* that, without being given access to a client C (or the square roots of any elements from $\mathcal{T}^2$, produces an output whose probability distribution is identical to the probability distribution of the output of $< C, S^* >$.

While we skip details due to space limitations, we note that M^* is built by using S^* as a black box. For each of the s steps of the protocol, M^* flips a coin a and builds the sets M and $Proof$ anticipating that the challenge bit b will equal a. It then feeds these values to S^*, which then outputs b. If $b = a$, M^* outputs the transcript of the transaction and moves to the next step. Otherwise, it repeats the current step. M^* terminates in expected polynomial time (each of the s steps is executed on average twice). The probability distributions of the output of $< C, S^* >$ and of M^* are identical, which is proved by induction.

Theorem 72 *GeoM is CI-IND and SV-IND.*

Proof. (Summary) The CI-IND proof is inherited from *GeoBadge*: *CheckIn* protocol differs solely in the provider's issuance of a square root value. For the SV-IND proof, we note that $StatVerify$ is a ZK proof system. Then, an adversary with advantage in the SV-IND game can be used to build an adversary against Mix's untraceability property.

Theorem 73 *GeoM provides Status Safety.*

Proof. (Summary) Results directly from Theorem 71: $StatVerify$ is a proof system of having k square roots from $\mathcal{T}^2$. A cheating client can succeed with probability 2^{-s}, where s is the number of proof iterations.

The single-use blindly signed nonces generated during *CheckIn* ensure the token non-distributability property of *GeoM*. *GeoM* trivially provides the token-epoch immutability property, as S issues a single token per venue per epoch.

8 Multi-Player: MP-Badge

The multi-player badge is issued when a user presents proof of co-location and interaction with $k - 1$ other users at a venue V. k is a parameter that may depend on the venue V. We now present $MPBadge$, an extension of *GeoBadge* that provides this functionality with privacy. $MPBadge$ relies on threshold signatures, where each client is able to provide a signature share and k unique signature shares generated at the same venue in the same epoch (see protocol $MP - CheckIn$). The shares can then be combined to produce a signed co-location proof. An additional difficulty here lies in the ability of an anonymous user to cheat: run *CheckIn* multiple times in the same epoch, obtain k signature shares and generate by itself the co-location proof. We solve this issue by allowing a user to run *CheckIn* only once per venue per epoch - using the blind signature generation, $BSGen$, protocol (see below).

Setup: The server S generates two large safe primes p and q and the composite $n = pq$. Let N denote n's bit length. S publishes n and keeps p and q secret.
RegisterVenue$(C(), S(priv_S))$: The following steps are executed:

– S stores a key table KT, indexed by venues and epochs. $KT[V, e]$ contains a unique key, used only for signing values for a venue V during epoch e. Let v denote the total number of venues supported.
– For each venue V and epoch e, S generates a value $M_{V,e} \in_R \{0, 1\}^N$ and a random polynomial $Pol_{V,e}$ with degree $k-1$, whose free coefficient is $M_{V,e}$. $M_{V,e}$ and $Pol_{V,e}$ are secret.

BSGen$(C(Id, e, pub_S), S(priv_S))$: Executed once per epoch e by each client C (when active) with provider S, over an authenticated channel. C generates v random values, one for each venue in the system, $R_1, .., R_v$. C and S engage in a blind signature protocol, where each R_i is blindly signed by S with $KT[P_i, e]$. S records the epochs when C has executed this step and returns -1 if C attempts to run this step twice for the same epoch. Otherwise, the client obtains $BS_{KT[P_i,e]}(R)$, $\forall i = 1..v$.
CheckIn$(C(Id, V, T, n, pub_S), S(priv_S))$: C and S run the Anonymous Authentication and Location Verification steps of *GeoBadge*. If they succeed, C sends $R, BS_{KT[V,e]}(R)$ to S over Mix – the values correspond to the venue V and epoch e where C runs *CheckIn*. S verifies that (i) R has not been used before and (ii) the validity of its signature. If either step fails, S returns -1. Otherwise, S stores R and generates a share of $M_{V,e}$: (x_e, y_e), where x_e is random and $y_e = Pol_{V,e}(x_e)$. S sends (x_e, y_e) to C as a reply over Mix, and C stores them.
MP-CheckIn$(C_1(Id_1, V, T), C_2(Id_2, V, T, x_{e,2}, y_{e,2}))$: This step is executed when a client C_1 contacts a co-located client C_2 to build a co-location proof for V during epoch e (containing current time T). The communication is done over Mix. C_1 contacts C_2 with the message $M = (\text{“}MPBadge'', V, e)$. If C_2 has already executed *CheckIn* at venue V and epoch e, let $(x_{e,2}, y_{e,2})$ be its share of $M_{V,e}$. C_2 then generates $\sigma_{e,2} = M^{y_{e,2}}\ mod\ n$ and sends back to C_1 the tuple $(x_{e,2}, \sigma_{e,2}, R_2, BS_{V,e}(R_2)\ mod\ n)$. R_2 is the value that C_2 has had the server blindly sign: $BS_{V,e}(R_2)$. C_1 stores these values in the set Tk.
StatVerify$(C(Id, V, k, Tk, e, pub_S), S(priv_S))$: Without loss of generality, let $Tk = \{(x_{e,i}, \sigma_{e,i}, R_i, BS_{V,e}(R_i)\}$, $\forall i = 1..k$. C and S run the following steps:

– C computes $\sigma = \prod_{i=1}^{k} \sigma_i^{l_i(0)} = M^{\Sigma_i y_{e,i} l_i(0)} = M^{M_{V,e}}$. C sends $\sigma, R_i, BS_{V,e}(R_i)$, for all k R_i values received from co-located clients to S over Mix.
– S verifies that (i) the time when the communication of the previous step has been initiated is within epoch e, (ii) that $(\text{“}MPBadge'', V, e)^{M_{V,e}} = \sigma$ and (iii) that all $BS_{V,e}(R_i)$ signatures verify for venue V during epoch e. S checks that the exact set of k revealed blind signatures has not been used before more than k-1 times: S records the set of k blind signatures and allows it to be used only k times. Subsequent uses of the tokens are allowed, as long as the newly revealed set contains at least one fresh blind signature. If any verification fails, S outputs -1 and stops. Otherwise, S generates an MPBadge: $S_S(\text{“}MPBadge'', V, e, T_c)$, where T_c is the time of issue, and sends it over Mix to C.

While we omit the proofs due to space constraints, we note that *MPBadge* is CI-IND and SV-IND.

9 Evaluation

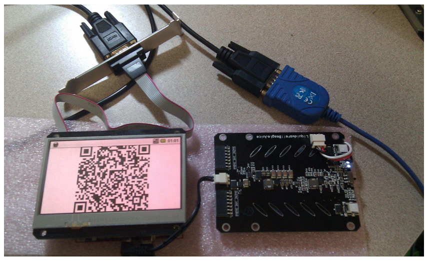


Fig. 3. Spotr on BeagleBoard.

In this section we study the efficiency of our solutions from the standpoint of both service provider and client. To this end we have implemented Spotr , *GeoBadge* and *GeoM* in Android and Java. All the results shown in the following are computed as an average over at least 10 independent runs.

Spotr Implementation: We have implemented Spotr on a Revision C4 of the BeagleBoard [31] system, featuring an OMAP 3530 DCCB72 720 MHz and a Google Nexus One smartphone featuring a 1 GHz Scorpion processor, Adreno 200 GPU and a Qualcomm QSD8250 Snapdragon chipset with 512 MB RAM. We use the ambient light sensor of the Nexus One to detect when anyone takes a picture of the displayed QR code (light level changes). Figure 3 shows a picture of the BeagleBoard displaying a generated QR code. The time to generate a QR code on the BeagleBoard is 50ms. The time to decode the QR code on the Nexus One is 190ms, at a distance of 20cm.

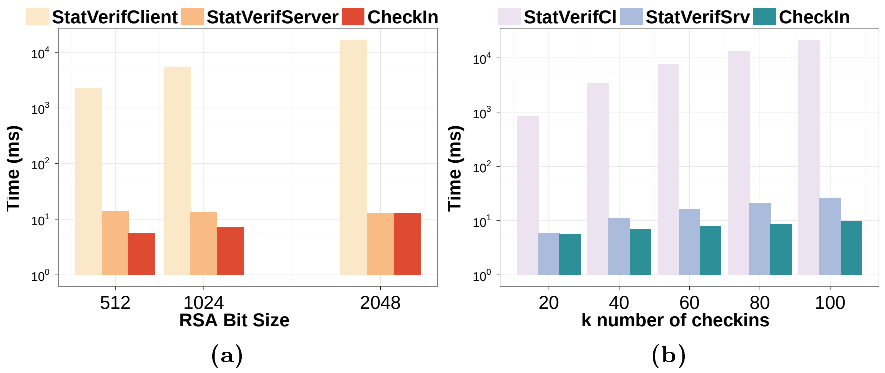


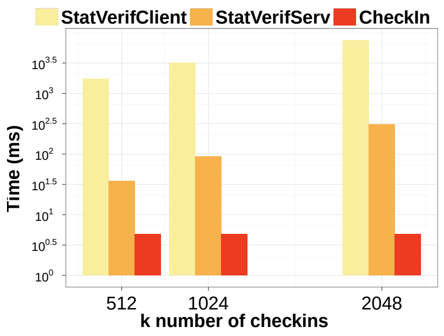
Fig. 4. (a) GeoBadge dependence on modulus size. (b) GeoBadge dependence on k, the check-in count.

In the following we describe our experiments with *GeoBadge* and *GeoM* on the Nexus One smartphone, when running the server side on a 16 quadcore server featuring Intel(R) Xeon(R) CPU X7350 @ 2.93GHz and 128GB RAM.

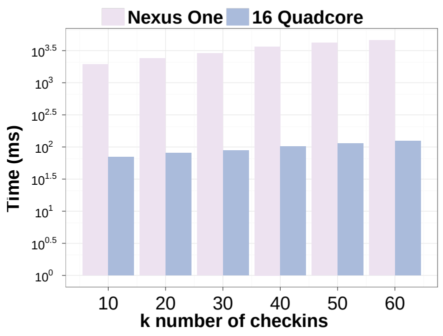
GeoBadge: We study the most compute-intensive functions of *GeoBadge*: *Setup*, the GSN provider side of *CheckIn*, the client and provider sides of

$StatVerify$. We investigate first the dependence on the modulus bit size. The *Setup* cost, a one time cost for the GSN provider, ranges from 277ms for 512 bit keys to 16.49s for 2048 bit keys. Figure 4(b) shows the performance of the remaining three components in milliseconds (ms) using a logarithmic y scale. The x axis is the modulus size, ranging from 512 to 2048 bits. The value of k, the number of $CheckIn$ runs required to acquire the badge is set to 50. On a single core, the $CheckIn$ cost, is 13ms even for a 2048 bit modulus size. Thus, the provider can support more than 4800 $CheckIn$ runs per second. The cost of the provider side of $StatVerify$ is almost constant for different key bit sizes, around 13ms – on an OpenSSL sample, the cost of performing one signature verification for 2048 bit is 0.1ms, thus dwarfed by the cost of string operations. Thus, the provider can support more than 4800 $StatVerify$ runs per second.

The client size of $StatVerify$ is 16.5s for 2048 bit keys, on the Nexus One. Figure 4(c) shows the performance dependency of the same protocols on k, the number of check-ins required, when the key size is set to 1024 bits. The client $StatVerify$ takes up to 21s when $k = 100$. However, the provider components are much faster: the $StatVerify$ takes less than 27ms, allowing the provider to support more than 2400 such operations per second. The $CheckIn$ cost is even smaller, less than 10ms for k=100, allowing more than 6500 simultaneous check-ins, or more than 560 million check-ins per day.



(a)



(b)

Fig. 5. $GeoM$: (a) Dependence on N, the modulus size, (b) StatVerify client and server side, function of k, the number of check-ins.

GeoM: For the next experiment, we studied $GeoM$. We have first tested key bit sizes ranging from 512 to 2048. A one time occurrence for the GSN provider, the *Setup* cost ranges from 227ms to 1.5s and is negligible. Figure 5(a) shows the performance of $CheckIn$ (server side) and $StatVerify$ (client and server side) in ms, as a function of the key bit size. The y axis shows the time in ms, in logarithmic scale. s, the number of proof rounds is set to 40, m, the number of past epochs is set to 60 and k, the number of $CheckIn$ runs is set to 30. The client side $StatVerify$, executed on the Nexus One platform , ranges from 1.7s to 7.5s. Since the provider is the bottleneck, the sensitive operations are $CheckIn$ and the provider side of $StatVerify$. These operations however are fast. Requiring a single table lookup and a signature generation, $CheckIn$ takes only 4.8ms. On a 16 quadcore server, the provider can support more than 13,000 check-ins per second. The provider side of $StatVerify$ is less compute intensive than the client side: it ranges from 36ms to 309ms (form 2048 bit keys).

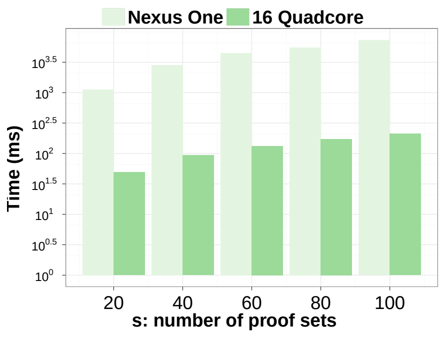


Fig. 6. StatVerify dependence on s, the number of proof iterations. y axis is time in milliseconds, in logarithmic scale.

We further evaluate the dependency of $StatVerify$ (client and server side) on the value of k when the modulus size N is 1024, m=60 and s=40. Figure 5(b) shows that the server side exhibits small linear increases with k, but is only 124ms when $k = m = 60$. The server can support thus 512 simultaneous $StatVerify$ runs per second. The client side is less then 4.6s even for 60 check-ins. Finally, Figure 5(c) shows the dependency of $StatVerify$ on the value of s, the number of proof sets. N is set to 1024, m is set to 60 and k is set to 30. Both costs are linear: up to 211ms for the provider and 7.2s for the client.

Summary. The server side overhead of $GeoBadge$ and $GeoM$ is small. The provider can support thousands of $CheckIn$s and $StatVerify$s per second. While on the order of a few seconds, the client side overhead of $StatVerify$ is not time sensitive and can be executed in the background.

10 Conclusions

We studied privacy issues related to aggregate location predicates in GSNs. We introduced new privacy and correctness properties and proposed solutions that privately and securely enable aggregate location predicates. We implemented and benchmarked a practical prototype.

References

1. Foursquare. `https://foursquare.com/`.
2. Yelp. `http://www.yelp.com`.
3. Gowalla. `http://gowalla.com/`.
4. Lauren Indvik. Foursquare Surpasses 3 Million User Registrations. `http://mashable.com/2010/08/29/foursquare-3-million-users/`.
5. Jolie O'Dell. Foursquare Day Sets Record with 3M+ Checkins. `http://mashable.com/2011/04/20/foursquare-day-2/`.
6. Chloe Albanesius. Apple location, privacy issue prompts house inquiry. PC Mag. `http://www.pcmag.com/article2/0,2817,2365619,00.asp`.
7. Jennifer Valentino-Devries. Google defends way it gets phone data. Wall Street Journal. `http://online.wsj.com/article/SB10001424052748703387904576279451001593760.html`, 2011.
8. Balachander Krishnamurthy and Craig E. Wills. On the leakage of personally identifiable information via online social networks. In *WOSN*, pages 7–12, 2009.

9. Balachander Krishnamurthy and Craig E. Wills. On the leakage of personally identifiable information via online social networks. *Computer Communication Review*, 40(1):112–117, 2010.
10. Josh Lowensohn. Apple sued over location tracking in iOS. Cnet News. `http://news.cnet.com/8301-27076_3-20057245-248.html/`, 2011.
11. Gpscheat! `http://www.gpscheat.com/`.
12. John Krumm. Inference attacks on location tracks. In *Pervasive*, 2007.
13. Philippe Golle and Kurt Partridge. On the anonymity of home/work location pairs. In *Pervasive*, 2009.
14. Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of MobiSys*, 2003.
15. Baik Hoh, Marco Gruteser, Ryan Herring, Jeff Ban, Dan Work, Juan-Carlos Herrera, Re Bayen, Murali Annavaram, and Quinn Jacobson. Virtual Trip Lines for Distributed Privacy-Preserving Traffic Monitoring. In *Proceedings of ACM MobiSys*, 2008.
16. Femi G. Olumofin, Piotr K. Tysowski, Ian Goldberg, and Urs Hengartner. Achieving Efficient Query Privacy for Location Based Services. In *Privacy Enhancing Technologies*, pages 93–110, 2010.
17. Xiao Pan, Xiaofeng Meng, and Jianliang Xu. Distortion-based anonymity for continuous queries in location-based mobile services. In *GIS*, pages 256–265, 2009.
18. Gabriel Ghinita, Maria Luisa Damiani, Claudio Silvestri, and Elisa Bertino. Preventing velocity-based linkage attacks in location-aware applications. In *GIS*, pages 246–255, 2009.
19. Stefan Saroiu and Alec Wolman. Enabling New Mobile Applications with Location Proofs. In *Proceedings of HotMobile*, 2009.
20. W. Luo and U. Hengartner. VeriPlace: A Privacy-Aware Location Proof Architecture. In *Proceedings of ACM SIGSPATIAL GIS*, 2010.
21. Z. Zhu and G. Cao. APPLAUS: A Privacy-Preserving Location Proof Updating System for Location-based Services. In *Proceedings of IEEE INFOCOM*, 2011.
22. Ge Zhong, Ian Goldberg, and Urs Hengartner. Louis, Lester and Pierre: Three Protocols for Location Privacy. In *Proceedings of PETS*, 2007.
23. Justin Manweiler, Ryan Scudellari, Zachary Cancio, and Landon P. Cox. We saw each other on the subway: secure, anonymous proximity-based missed connections. In *Proceedings of HotMobile*, 2009.
24. A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *Proceedings of NDSS*, 2011.
25. Bogdan Carbunar and Radu Sion. Private geosocial networking. In *Proceedings of the ACM SIGSPATIAL GIS*, 2011.
26. Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320, 2004.
27. Michael K. Reiter and Aviel D. Rubin. Anonymous web transactions with crowds. *Commun. ACM*, 42(2):32–38, 1999.
28. Dan Boneh and Matt Franklin. Anonymous Authentication With Subset Queries (Extended Abstract). In *in Proceedings of CCS*, 1999.
29. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1), 1989.
30. Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3), 1991.
31. G. Coley. Beagleboard system reference manual. *BeagleBoard. org, December*, 2009.