# Private Badges for Geosocial Networks

Bogdan Carbunar, Radu Sion, Rahul Potharaju, Moussa Ehsan

***Abstract*—Geosocial networks (GSNs) extend classic online social networks with the concept of location. Users can report their presence at venues through "check-ins" and, when certain check-in sequences are satisfied, users acquire special status in the form of "badges". We first show that this innovative functionality is popular in Foursquare, a prominent GSN. Furthermore, we address the apparent tension between privacy and correctness, where users are unable to prove having satisfied badge conditions without revealing the corresponding time and location of their check-in sequences. To this end, we propose several privacy preserving protocols that enable users to prove having satisfied the conditions of several badge types. Specifically, we introduce (i) GeoBadge and T-Badge, solutions for acquiring location badges, (ii) FreqBadge, for mayorship badges, (iii) e-Badge, for proving various expertise levels and (iv) MPBadge, for accumulating multi-player badges. We show that a Google Nexus One smartphone is able to perform tens of badge proofs per minute while a provider can support hundreds of million of check-ins and badge verifications per day.**

## I. Introduction

Location Based Services (LBS) provide users with information and entertainment applications centered on their geographical position. A recently introduced but popular LBS are Geosocial Networks (GSNs), social networks centered on the locations of users and businesses. GSNs such as Foursquare [1] and Yelp [2] 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 (i.e., venues), following a required pattern.

Besides keeping track of the locations of their friends, users rely on GSNs to receive promotional deals, coupons and personalized recommendations. For GSN providers however, the main source of revenue is location-based ad targeting. Boasting millions of users [3] and tens of millions of location check-ins per day [4], GSNs can provide personalized, location dependent ads. The more user information they are able to collect, the more accurate are their predictions.

Thus, the price of participation for users is compromised privacy, in particular, 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., the frequency distribution of their visits) [5], [6]. The service providers may use this information in ways the users never suspected when they signed-up (e.g., having their location shared with third parties [7], [8]).

Bogdan Carbunar is with the School of Computing and Information Sciences at the Florida International University, Miami, FL, USA. E-mail: carbunar@cs.fiu.com

Radu Sion and Moussa Ehsan are with the Computer Sciences Department at Stony Brook University, Stony Brook, NY, USA. E-mail: {sion,mehsan}@cs.stonybrook.edu

Rahul Potharaju is with the Computer Sciences Department at the University of Purdue, West Lafayette, IN, USA. E-mail: rpothara@purdue.edu

A preliminary version of this paper has appeared in ACNS 2012.

Opting out of GSN services seems to be a rational way to avoid compromised privacy (allowing stalking, theft [9]). In this paper however, we show that such radical measures may not be necessary. To this end, we introduce a framework that enables users to privately acquire GSN badges. In this framework. users are responsible for storing and managing their location information, and the provider's (oblivious) participation serves solely the goal of ensuring user correctness.

We define badges as aggregate location based predicates. We propose solutions to support a variety of such predicates, including (i) checking-in a pre-defined number of times at a location or set of locations, (ii) checking-in the most number of times (out of all the users) at a location, (iii) proving various expertise levels, and (iv) simultaneously checking-in with $k$ other users at a location.

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

The challenge consists of providing solutions that balance three requirements. On one dimension, 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 even (iii) building pseudonymous user profiles – linking multiple locations where the same "unknown" user has checked-in. On the second dimension, the service provider needs assurances of client correctness when awarding location-related badges. Otherwise, since special status often comes with financial and social perks, privacy would protect users that perpetrate fraudulent behaviors such as, reporting fake locations [11], duplicating and sharing special status tokens, or checking-in more frequently than allowed. On a third dimension, the provider needs to be able to collect certain user information. Being denied access to all user information discourages participation.

The use of client pseudonyms to provide client privacy during check-ins and special status requests is vulnerable to profile based de-anonymization attacks [12], [13]: Constructed pseudonymous profiles can be joined with residential and employment datasets to reveal profile owner identities.

Instead, in a first contribution, we introduce essential properties that need to be satisfied by private "badging" solutions. Informally, we define user *privacy* in terms of indistinguishability: an adversary controlling the service provider *and* any coalition of colluding users, should be unable to distinguish between any interactions with two registered (but not controlled) users. We then define *correctness*, to model the inability of users to claim special status without satisfying the associated spatial, temporal and frequency requirements. Furthermore, we introduce a *provider usability* property to model the ability of the provider to build popularity statistics for the venues supported (e.g., per-site check-ins and issued badges).

In a second contribution, we propose four solutions, for the the aggregate location predicates described above, that satisfy the defined properties. GeoBadge, allows users to privately prove having performed $k$ check-ins at one venue, where $k$ is a predefined parameter. FreqBadge extends GeoBadge with provably time-constrained check-ins as well as arbitrary values for $k$. e-Badge extends GeoBadge with the notion of levels of expertise, unlocked as the user performs more check-ins at new venues. MPBadge extends GeoBadge with the notion of simultaneous, co-located check-ins from multiple users. The complexity of MPBadge lies in the seeming contradiction between the ability of multiple clients to anonymously check-in at the same location and the ability of rogue users to launch Sybil attacks [14].

The solutions deploy cryptographic techniques such as zero-knowledge (ZK) 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 ZK proofs of ownership. Client correctness is partly ensured by the use of blind signatures of single-use tokens.

Instead of publishing acquired badges, and relinquishing privacy, our approach provides users with control over their badges. Users locally store them on their mobile devices and can prove ownership of their badges in a zero knowledge manner, to other interested parties.

We have implemented and evaluated the performance of our solutions on 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.

The paper is organized as follows. Section II summarizes related work. Section III describes the system model considered and defines the associated privacy and security requirements. Section IV describes the cryptographic tools used in our solutions. Section V presents GeoBadge, the private location badge solution and Section VI presents FreqBadge, the private mayorship solution. Section VII presents e-Badge, the private, multi-venue expertise badge solution and Section VIII present the private multi-player badge solution. Section IX describes our implementation results. Section X concludes.

## II. Related Work

This paper extends our previous work [15] with (i) an extensively modified FreqBadge solution, (ii) constructs supporting a new badge type (e-Badge), (iii) an analysis of MP-Badge, (iv) an extension of all the solutions with a protocol that enables proofs of badge ownership (ProveBadge), and (v) a better detailed system model.

**Location cloaking:** Anonymization, pseudonimization, location and temporal cloaking techniques (introducing errors in location reports to provide 1-out-of-k anonymity) have been initially proposed in [16]. Hoh et al. [17] proposed a location cloaking approach based on the concept of *virtual trip lines*, that when crossed, trigger a device location update. Olumofin et al. [18] propose a location cloaking based private information retrieval algorithm that enables mobile device users to privately retrieve points of interest around their location. Pan et al. [19] and Ghinita et al. [20] identified the important problem of preventing attacks that link even cloaked successive location reports. Pan et al. [19] rely on a trusted anonymizing proxy to maintain cloaking sets of active users, and update them as the users issue successive location reports. Ghinita et al. [20] propose both off-line solutions that report temporally cloaked pre-defined regions, and on-line solutions.

**Private geographic algorithms.** Eppstein et al. [21] introduced data-oblivious algorithms for secure multi-party computations (SMC) for location based services. The proposed techniques are relevant to geometric problems – convex hull, quadtrees, closest pair – and cannot be easily applied to solve the privacy issues we consider in this work. Ghinita et al. [22] propose a privacy robust geometric transformation for private matches on spatial datasets (e.g., geo-tagged data items).

**Location verification:** Saroiu and Wolman [23] introduced the location proof concept – a piece of data that certifies a receiver to a geographic location. The solution relies on special access points (APs), that are able to issue such signed proofs. APs add their location to their presence beacons and then generate location proofs upon client request, containing the signed client identity, AP identity, location and timestamp.

Luo and Hengartner [24] 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. Cellular providers are notorious for their unwillingness to collaborate and modify their protocols. Most AP owners have trouble setting up security features thus we envision that only few APs (if any) will provide this functionality – defeating the solution's applicability.

To address the central management problems, Zhu and Cao [25] proposed the APPLAUS system, where co-located, Bluetooth enabled devices compute privacy preserving location proofs. While the p2p approach can solve the central management problems (for a strongly Bluetooth-connected network), not many users enable this interface, due to lack of applications and associated power-drain.

**Proximity alerts:** Zhong et al. [26] 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. [27] propose several cloaking techniques for private server-based location/time matching of peers. Narayanan et al. [28] proposed several other solutions for the same problem, introducing the use of location tags as a means to provide location verification. Nielsen et al. [29] use secure multiparty computation techniques to address a similar problem. Hu et al. [30] address the problem of service providers delivering authenticated LBSs, while preserving the data being queried by clients. Our work is different, by enabling private and correct

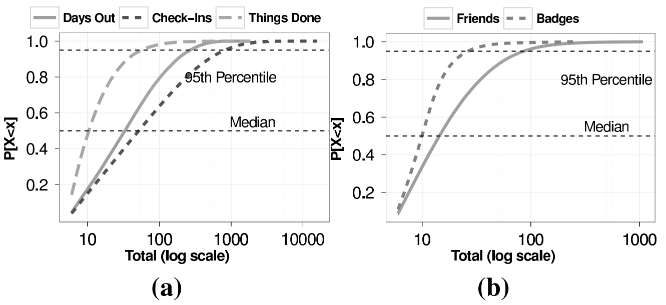


Fig. 1. Foursquare stats: (a) CDF of days out, check-ins and things done by users. (b) Badge and friends evaluation.

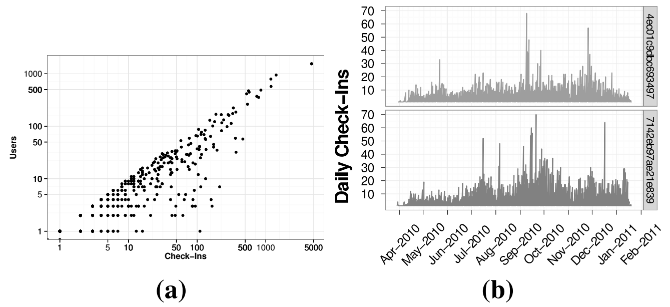


Fig. 2. (a) Scatterplot check-ins vs. users in a small town. (b) Per-venue check-in distribution over time for two random venues.

aggregate location predicates in GSNs.

**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. Instead, in this paper we focus on the next step, of anonymizing location aggregates defined by geosocial networks.

## III. Model

### A. The System

We consider a geosocial network provider, $S$, which we model after the most popular in existence to date, Foursquare [1]. 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.

Besides users, the geosocial network also supports a set of venues, which are businesses with a geographic location. Let $\overline{V}$ denote the set of all the venues registered with the system.

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 20 million users.

### B. Foursquare Data

In order to understand the need for our solutions, we have collected profiles from 781,239 randomly selected Foursquare users. For every user, we have gathered the user profile including the total number of friends, the total number of check-ins, the total number of days the user was out (days the user was actively performing check-ins) and the total number of things done (e.g., reviews left for a venue). 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 reported check-ins, leads us to conclude that Foursquare is a system worthy to evaluate our protocols.

To corroborate the check-in data in a location-aware fashion, we used a Foursquare feature that allows users to query the list of venues at a location using (latitude, longitude) pairs. Specifically, we started with a seed latitude and longitude (in our case, 40.000, -73.000, representing New York City). We then generated 5000 random coordinates around this coordinate pairs. For each newly generated coordinate pair, we queried Foursquare to collect all the venues near that location. 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. Thus, devising private and secure "badging" protocols becomes a problem of primary importance for GSNs.

### C. Geo: A Framework for Private GSNs

A full-fledged private GSN solution is composed of a set of protocols $Geo$ = $\{Setup$, $RegisterVenue$, $Subscribe$, $CheckIn$, $StatVerify$, $ProveBadge\}$, described in the following. We use the notation $Prot(P_1(args_1), .., P_n(args_n))$ to denote protocol $Prot$ run between participants $P_1, .., P_n$, each with its own arguments.

**Setup**$(S())$: Executed initially (only once) by the service provider $S$. The server produces public information $pub_S$ and private information $priv_S$. The server publishes $pub_S$.

**RegisterVenue**$(O(V), S(priv_S))$: Executed by the owner $O$ to register a new venue $V$ with the provider.

**Subscribe**$(C(), S(pub_S, priv_S))$: Executed once by any client $C$ that wants to register with the service. If the subscription fails, the server returns -1. Otherwise, the client receives a unique id and the server's public information $pub_S$.

**CheckIn**$(C(Id, V, T, pub_S), S(priv_S))$: Executed by a subscribed client with identifier $Id$, to report location $V$ at time $T$ to the provider $S$. $S$ verifies the correctness of $V$ and $T$ and returns -1 in case of failure. Otherwise, the client is issued a special token proving its presence at $V$ during $T$.

**StatVerify**$(C(Id, V, k, Tk, pub_S), S(priv_S))$: After accumulating sufficient tokens, the client runs $StatVerify$ with the server, for a specific venue $V$, providing its entire set of tokens, $Tk$. If the tokens prove that special status has indeed been achieved, the server issues a special status token (or badge), $B_V$, to the client. We support several badge types, introduced by Foursquare [1] and SCVNGR [31]:

- **Location Badge** (GeoBadge/T-Badge). GeoBage is issued after the client runs $CheckIn$ during $k$ different epochs at a venue $V$. T-Badge is issued after the client runs $CheckIn$ at $k$ different venues. GeoBadge and T-Badge model Foursquare badges such as "Newbie", "Local", "Adventurer", "Explorer" and "Superstar", see [32].
- **Expert Badge** (e-Badge). e-Badges support several levels of expertise. To achieve level 1 of expertise, the client needs to run $CheckIn$ at $k$ different, select locations, with a common background. A user having expertise level $L$ for an e-Badge can reach level $L + 1$ after performing $k$ more check-ins at similar (but different) locations. $k$ is a system parameter. This models several expertise badges from Foursquare (e.g., "Swimmie", "Wino", "Pizzaiolo", see [33]), where the rules are the same for all the areas of expertise: A user achieves level 1 for checking in at five unique places. From there, every level up is five more unique places.
- **Mayorship** (FreqBadge). Issued when the client has performed the largest number of $CheckIn$s, at most one per epoch, in the past $m$ epochs at a given venue. $m$ is a system parameter. FreqBadge models Foursquare "mayor" badges.
- **Multi-Player Badge** (MPBadge). Issued when the client runs $CheckIn$ simultaneously with $s$ other users at the same location. $s$ is a system parameter. The MP-Badge models Foursquare badges such as "Player Please! (Heart)", see [34].

**ProveBadge**$(C_1(pub_S, V, B_V), C_2(pub_S, V), S(priv_S, V))$: This protocol enables client $C_1$ to prove ownership of a badge $B_V$ for a venue $V$ to another client $C_2$. In order to preserve the privacy of $C_1$, following the ProveBadge execution, $C_2$ should not learn additional information about $C_1$ and should not be able to prove ownership of the badge to another client.

### D. Privacy and Correctness Properties

*1) Server Side:* We consider a provider $S$ that follows the protocols correctly. This implies for instance that the provider will not hand out incorrect information to users. However, we assume that $S$ 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. In order to achieve this goal, $S$ may collude with venues and existing clients and generate Sybil clients to track users of interest. The provider however does not collude with users to issue badges without merit. We do not consider physical attacks, such as, the server physically tracking individual users.

Intuitively, to achieve privacy, the provider should learn nothing about $Geo$ clients, including the venues and times at which a user runs the $CheckIn$ function, as well as her total and per-venue $CheckIn$ counts. 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, the set of venues 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. We use the notation $C_b(args)$ or $C_{c_i}$ to denote either client $C_0$ or client $C_1$ (according to the value of the bit $b$ or $c_i$), using input values $args$.

**CheckIn Indistinguishability (CI-IND).** $\mathcal{A}$ generates public information $pub_A$ (and corresponding private information $priv_A$), generates $l$ bits $c_1, .., c_l$, and $l + 1$ venue ids $V_1, .., V_l, V_{l+1}$, $V_i \in \overline{V}$, i=1..l+1, and sends them to $\mathcal{C}$. For each $i = 1..l$, $\mathcal{C}$ needs to run $CheckIn$ on behalf of client $C_{c_i}$, at venue $V_i$. $\mathcal{C}$ verifies that the time between two consecutive requests for the same client is sufficient to enable the client to travel the distance between the corresponding venues. If this condition is not satisfied, $\mathcal{C}$ ignores the request. Otherwise, it executes $CheckIn(C_{c_i}(Id_{c_i}, V_i, T_i, pub_A), \mathcal{A}(priv_A))$. After processing the $l$ requests, $\mathcal{C}$ makes sure that the distance between both $C_0$ and $C_1$'s last check-ins to venue $V_{l+1}$ can be physically traversed between the time of their last check-ins and the current time. If the verification fails, $\mathcal{C}$ stops the game. Otherwise, $\mathcal{C}$ generates a bit $b \in \{0, 1\}$ and runs $CheckIn(C_b(Id_b, V_{l+1}, T_{l+1}, pub_A, \mathcal{A}(priv_A))$. $\mathcal{A}$ outputs a bit $b'$. A solution provides CI-IND if the advantage of $\mathcal{A}$ in the CI-IND game, $Adv(\mathcal{A}) = |Pr[b = b'] - 1/2|$, is negligible.

**CI-IND Intuition.** The above definition models the claim of an adversary of being able to distinguish the client executing a $CheckIn$ protocol. For this, the challenger allows the adversary to request it to perform a number of $CheckIn$ operations on behalf of $C_0$ and $C_1$, two clients controlled by the challenger. The adversary also specifies the location where the check-in is to take place. Then, the challenger chooses privately one of the two clients and performs a $CheckIn$ on its behalf, at a venue chosen by the adversary. The adversary wins if it is able to guess the client that has performed the check-in, with probability significantly higher than 1/2. We note that the challenger verifies the feasibility of the check-ins: the fact that the adversary is not trying to win the game by making it impossible for a client to succeed in a check-in at a location.

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).** $\mathcal{A}$ generates public information $pub_A$ and sends it to $\mathcal{C}$ but keeps the private information $priv_A$ secret. The game has two steps. In the first step, $\mathcal{A}$ generates $k = 2s$ new bits $c_1, .., c_k$ such that $s$ of them are 0 and $s$ of them are 1. $\mathcal{A}$ also generates $k$ venue ids, $V_1, .., V_k$, $V_i \in \overline{V}$, i=1..k. $\mathcal{A}$ sends $c_1, .., c_k$ and $V_1, .., V_k$ to $\mathcal{C}$. For each i=1..k, $\mathcal{C}$ runs $CheckIn(C_{c_i}(Id_{c_i}, V_i, T, pub_A), \mathcal{A}(priv_A))$, only if the time between the previous $CheckIn$ of client $C_{c_i}$ and $T_i$ is sufficient to enable $C_{c_i}$ to travel the distance between the venue of the previous $CheckIn$ and $V_i$. At the end of this step, $\mathcal{C}$ verifies that $C_0$ and $C_1$ have performed the same number of check-ins at any venue $V_1, .., V_k$. If this verification does not succeed, $\mathcal{C}$ stops the game. In the second step, $\mathcal{A}$ sends to $\mathcal{C}$ a venue id $V \in \overline{V}$, such that the distance between the venue of the last $CheckIn$ of client $C_j$ (j=0,1) and $V$ can be physically traversed from the time of that $CheckIn$ to the current time. $\mathcal{C}$ generates a bit $b \in \{0, 1\}$ and runs $StatVerify(C_b(Id_b, V, T, pub_A), \mathcal{A}(priv_A)$. $\mathcal{A}$ outputs a bit $b'$. A solution is said to provide SV-IND if the advantage of $\mathcal{A}$, $Adv(\mathcal{A}) = |Pr[b = b'] - 1/2|$, is negligible.

**SV-IND Intuition.** The SV-IND game models the inability of $\mathcal{A}$, that controls the entire system with the exception of two clients $C_0$ and $C_1$, controlled by $\mathcal{C}$, to guess the identity of the client ($C_0$ or $C_1$) performing a $StatVerify$ operation. For this, in an initial step, $\mathcal{A}$ is allowed to request $\mathcal{C}$ to perform $CheckIn$ operations and specify the identity of the client and the venue where the check-in is to be performed. At the end of this step, $\mathcal{C}$ verifies that the two clients are equivalent: they have the same (badge) status at all the venues requested by $\mathcal{A}$. $\mathcal{A}$ secretly chooses one of the clients and executes $StatVerify$ on its behalf for one of the venues chosen by $\mathcal{A}$. $\mathcal{A}$ wins if it is able to guess the identity of the client with probability significantly larger than 1/2.

The following property models the ability of the server to collect venue-based statistics:

**Provider Usability.** The service provider can count the $CheckIn$ executions for any venue as well as list the issued badges and mayorships.

*2) Client Side:* 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.

Our solutions are not designed to handle private venues, venues that uniquely identify the user performing a check-in there (e.g., the user's home).

In the following game, $k$ is a system parameter that denotes the number of check-ins a user needs to perform in order to acquire special status (a badge).

**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 status safe if $Adv(\mathcal{A})$ is negligible.

**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.

## IV. Tools

**Hash functions and HMACs.** We use cryptographic hashes that are easy to compute and are (i) pre-image resistant, (ii) second pre-image resistant and (iii) collision resistant. Let $H(M)$ denote the hash of message $M$. Pre-image resistance means that given a hash value h it is hard to find any message M such that $H(M) = h$. Second pre-image resistance means that given a message $M_1$, it is hard to find another message $M_2$ such that $M_1 \neq M_2$ and $H(M_1) = H(M_2)$. Collision resistance means that it is hard to find *any* two messages $M_1$ and $M_2$ such that $M_1 \neq M_2$ and $H(M_1) = H(M_2)$.

We also use hash based message authentication codes, HMACs, that rely on cryptographic hashes and keys to authenticate messages [35]. Let $HMAC_K(M)$ denote the keyed message authentication code of message $M$. Two parties sharing a key $K$, can use the string $M, HMAC(K, M)$ to authenticate message $M$: only someone knowing key $K$ can generate $HMAC(K, M)$ and verify its authenticity for message $M$.

**Signatures and blind signatures.** We rely on unforgeable signature schemes. Let $Sig_X(M)$ denote the signature of a message $M$ by participant $X$. Unforgeability is defined in terms of security "against one-more-forgery", where the user engaged in $l$ runs of the signature algorithm with the signer cannot obtain more than $l$ signatures. We also make use of blind signatures [36], [37] that have the standard (i) blindness and (ii) unforgeability properties. Blindness means that the signer cannot learn information about the signed messages.

**Anonymizers.** We assume the existence of a network anonymizer, $Mix$, such as Tor [38]. Anonymizers or mix-nets [38], [39] are tools that make communication untraceable and unlinkable. Untraceability implies the infeasibility of finding the identity of the issuer of a given set of messages. Unlinkability implies the infeasibility of discovering pairs of communicating entities. Existing popular anonymizing tools include onion routing Tor [38] and Crowds [39].

**Anonymous authentication.** The authentication step allows the user to prove to the server that it is a subscriber. We rely on anonymous authentication techniques with revocation and identity escrow, e.g., [40], performed over $Mix$, to enable users to anonymously prove their service subscriber status. The solutions proposed by Boneh and Franklin [40] allow a user to prove in zero knowledge its membership to arbitrary subsets of users while allowing an escrow agent to reveal the identity of misbehaving users. We note that to minimize the communication overheads, the ZK proofs can be made non-interactive (e.g., the user computes the challenge based on verifiable values such as the current time and server status).

**The 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 $y^2 = v \ mod \ n$.

**Symmetric Private Information Retrieval.** A private information retrieval (PIR) protocol allows a user to retrieve an item from a server in possession of a database without revealing which item she is retrieving. Symmetric PIR (SPIR), introduces the additional restriction that the user may not learn any item other than the one she requested [41], [42].

**Zero knowledge (ZK) proofs.** ZK proofs are protocols that enable a prover, claiming to know that a statement is true, to prove this fact to a verifier, without allowing the verifier to learn any information that would allow her to prove the statement to anyone else. A ZK proof protocol needs to satisfy completeness, soundness and ZK properties. Completeness means that if the statement is true, an honest verifier will be convinced of this fact by an honest prover. Soundness means that if the statement is false, a cheating prover can convince the honest verifier that it is true only with negligible probability. ZK means that if the statement is true, even a cheating verifier learns nothing except this fact.

**Notation.** $x \in_R X$ is the random choice of $x$ from set $X$.

## V. Geo-Badge

GeoBadge is a private protocol that allows users to prove having visited the same location $k$ times (see Figure 3 for a high level diagram). The set of supported $k$ values is pre-defined, e.g., $k = 1$ for "Newbie", $k$ = 10 for "Adventurer", $k$ = 25 for "Explorer", etc, and is known by all client applications. At the end of the section we show how to adapt this solution to support T-Badges. 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, the client is able to reconstruct the secret, which is the proof required for the badge. The single use nonces prevent users from distributing received shares (or proofs).

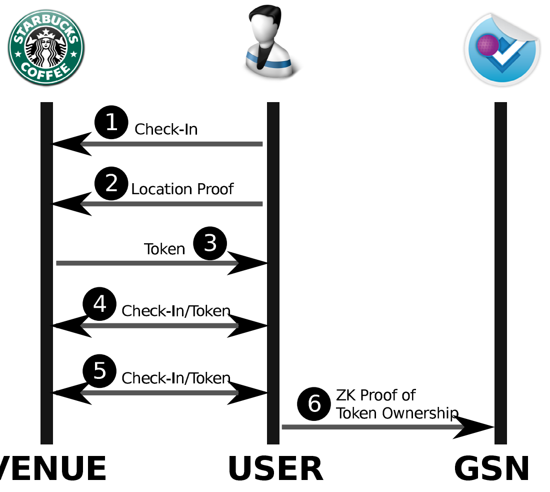


Fig. 3. High level overview of a private badge protocol.

GeoBadge extends $Geo$ and provides the skeleton on which we build the subsequent solutions. For instance, the anonymous authentication and location verification functions are only described for GeoBadge and inherited by FreqBadge and MPBadge. 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(S())**: Executed once in the beginning, by $S$. $S$ generates a large prime modulus $p$ that will be used to compute secret shares and publishes $p$. $S$ generates a random key $K$, that will be used for authentication purposes. $K$ is kept secret by $S$.

For each badge that requires $k$ check-ins, $S$ generates two large primes $p_k$ and $q_k$ such that $q_k|(p_k - 1)$. Let $G_{q_k}$ be the unique subgroup of $\mathbb{Z}^*_{p_k}$ of order $q_k$. Let $g_k$ be a generator of $G_{q_k}$. $S$ generates a fresh, random geo-badge $GB_k$ and computes the commitment value $CMT_k = g_k^{GB_k} \in G_{q_k}$. For each supported badge, $S$ publishes $p_k$, $q_k$, $g_k$ and $CMT_k$, but keeps secret $GB_k$.

**RegisterVenue**$(O(V), S(priv_S))$: The owner $O$ that registers venue $V$, sends to $S$ its public key. For each new venue $V$, for which the service provider offers badges (after $k$ $CheckIn$ runs) $S$ 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 $Ver_V = H(HMAC_K(V)M_V \ mod \ p)$. A client that reconstructs $Ver_V$, has proof of having achieved the special status (GeoBadge). $S$ stores $Pol$'s coefficients for $V$, along with the public key of $V$'s owner.

**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 [40] to obtain tokens that allow it later to authenticate anonymously with the server.

**CheckIn**$(C(Id, V, T, pub_S), S(priv_S))$: Let (current) 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 [40] to prove to $S$ that it is a subscriber. This step is performed over $Mix$.
- **Location Verification:** $C$ runs a location verification protocol [43] to prove presence at $V$.
- **Token Generation:** $C$ generates a fresh random value $R$ and sends the blinded $R$ to $S$, as $Obf(R)$ (obfuscated for instance using a modular multiplication, see Chaum's work [36] on blind signatures). $S$ computes $x_e = H(e)\ mod\ p$ and $y_e = Pol(x_e)\ mod\ p$. $S$ sends to $C$ the tuple $(x_e, c_e, Sig_S(Obf(R)))$, where $c_e = HMAC_K(V)y_e\ mod\ p$ and the last field denotes the blindly signed $R$. $C$ "unblinds" the signed nonce (see [36]), obtains $s_e = Sig_S(R)$ and stores $(x_e, c_e, s_e)$ into the set $Tk$.

**StatVerify**$(C(Id, V, k, Tk, pub_S), S(priv_S, k))$: Let $Tk = \{(x_1, c_1, Sig_S(R_1)),\ ..,\ (x_k, c_k, Sig_S(R_k))\}$. 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 $H(SS) = Ver_V$ (see the Correctness property in Section V-A). If the verification fails, $C$ outputs -1 and stops. Otherwise, it sends $SS$, along with the set of signed nonces, $(Sig_S(R_1), .., Sig_S(R_k))$ 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 sends the badge $GB_k$ (see $Setup$) to $C$ (over $Mix$).

**ProveBadge**$(C_1(pub_S, GB_k, V, p_k, q_k, g_k)$, $C_2(pub_S, V, p_k, q_k, g_k)$, $S(priv_S, V, CMT_k)$: $C_2$ retrieves $CMT_k$ from $S$. $C_1$ and $C_2$ engage in a zero knowledge protocol where $C_1$ proves knowledge of the $GB_k$, the discrete logarithm of $CMT_k$, for instance, using Schnorr's solution [44].

## A. Analysis

We now prove several properties of GeoBadge.

**Correctness.:** The following holds due to Lagrange interpolation:

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

We consider modified versions of the CI-IND and SV-IND games of Section III-D, where all the venues (chosen by $\mathcal{A}$) are identical. We now introduce the following results:

*Theorem 1:* GeoBadge is CI-IND.

*Proof:* (Sketch) Following the CI-IND game, $\mathcal{A}$'s view consists of the outcome of $l+1$ anonymous authentication procedures and $l+1$ blinded random values. The blinded random values are information theoretical secure. Then, 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 [40] or (ii) the untraceability property of $Mix$. ■

*Theorem 2:* GeoBadge is SV-IND.

*Proof:* (Sketch) At the completion of the SV-IND game $\mathcal{C}$ can reconstruct the $SS$ values for both $C_0$ and $C_1$. $\mathcal{A}$ has published a pre-commitment for $SS - Ver_V$. Note that $\mathcal{C}$'s verification of $H(SS) = Ver_V$ prevents $\mathcal{A}$ from guessing $b$ based on the value $\mathcal{C}$ reconstructs 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 blindness property of the blind signature algorithm, or (iii) the information theoretic security of the threshold secret sharing mechanism. ■

*Theorem 3:* GeoBadge provides Status Safety.

*Proof:* (Sketch) The use of a location verification solution [43] prevents the attacker from falsely claiming presence at $V$. Then, if there exists an adversary that has non-negligible advantage in the Status Safety game we can build an adversary that has a non-negligible advantage against (i) the pre-image resistance property of hashes (inverting $Ver_V = H(SS)$) or (ii) the information theoretic threshold secret sharing technique (including combining shares generated at multiple sites). ■

GeoBadge also provides the Token Non-Distributability property. The single use, server signed random nonces prevent more than one run of $StatVerify$ for a given set of tokens. The Token-Epoch Immutability property holds since the pair $(x_e, c_e)$ is a deterministic function of $e$.

## B. The Touring Badge (T-Badge)

The "adventurer" badge is unlocked when the user checks-in at $k$ different locations. $GeoBadge$ can be easily modified to support this functionality: the provider assigns one share (one point of the polynomial $Pol$) to each participating venue. 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 $k$ *different* venues. We note that multiple users could collude and combine their shares to obtain an "adventurer" badge, while none of them in isolation satisfies the condition. However, users may lack incentives for this attack: only one of the participants would receive the badge while the others waste their shares.

## VI. FreqBadge

Using the Foursquare terminology, the user that has run $CheckIn$ the most number of times, at a venue $SV$, within the past $m$ epochs, becomes the mayor of the place. Let $Mr_V$ denote the number of check-ins (at $V$) performed by the current mayor of $V$.

We introduce FreqBadge = $\{Setup$, $RegisterVenue$, $MaintainVenue$, $MaintainBadge$, $Subscribe$, $CheckIn$, $StatVerify$, $ProveBadge\}$, a solution that extends $Geo$ with two protocols: $MaintainVenue$ and $MaintainBadge$.

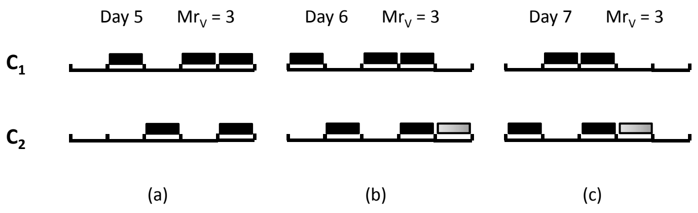


Fig. 4. Example timeline of mayorship evolution. $C_1$ and $C_2$ denote two clients that compete for mayorship at the same venue. Each bin denotes one day. A black or gray rectangle overlapping a day denotes a check-in performed by a client during that day.

FreqBadge allows clients to prove having performed any number of check-ins, not just a pre-defined value. The check-ins are time constrained: clients have to prove that all check-ins have occurred in the past $m$ epochs. Furthermore, 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.

## A. Overview

FreqBadge achieves these features in the following way. In the $MaintainVenue$ protocol, the service provider generates exactly one fresh token per epoch, for each supported venue $V$. When a client runs $CheckIn$ at $V$, it receives $V$'s token for the current epoch. The client stores the tokens accumulated for $V$ in the set $Tk_V$. At any time, for any venue $V$, the provider publishes and makes available upon request to any client, two values, (i) $Mr_V$, the number of tokens that the mayor of $V$ has proved to have accumulated in the past $m$ epochs, and (ii) $CMT_V$, a badge commitment value whose true nature we will reveal later.

If, during a $CheckIn$ run, a client's number of tokens, $|Tk_V|$, exceeds the current $Mr_V$, $StatVerify$ is invoked. The provider maintains a queue of $StatVerify$ requests: each new request is placed at the end of the queue and each request is processed in the order in which it was received. $StatVerify$ succeeds only if the client is able to prove to the provider that it knows at least $Mr_V + 1$ out of the $m$ tokens given in the past $m$ epochs for that venue. The proof is in zero knowledge. If the proof succeeds, it is published by the provider, along with an increased $Mr_V$ value, reflecting the new mayor's number of tokens. The provider then issues a private FreqBadge badge to the client, and publishes $CMT_V$, a commitment value for this badge.

If multiple clients initiate the $StatVerify$ protocol simultaneously, with the same number of tokens, only the first becomes the mayor: after the completion of the first client's $StatVerify$ protocol, the $Mr_V$ value is incremented. The second client's $StatVerify$ will not succeed, since its number of tokens does not exceed (but only equals) the new $Mr_V$ value. However, since the proof is in zero knowledge, the second client can safely reuse its tokens - they have not been revealed to the provider.

If a client needs to prove ownership of the FreqBadge for a venue $V$, it invokes the $ProveBadge$ protocol. The $ProveBadge$ is used to prove knowledge of the badge against

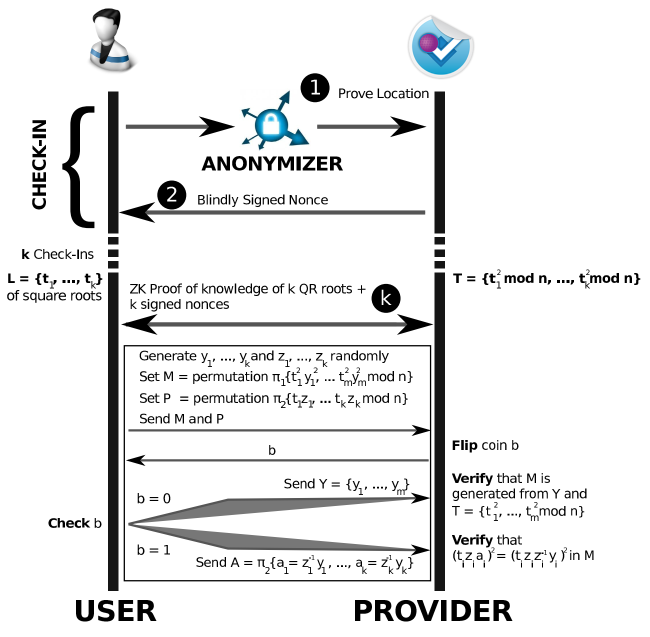

CHECK-IN
1 Prove Location
ANONYMIZER
2 Blindly Signed Nonce
k Check-Ins
L = {t_1, ..., t_k} of square roots
T = {t_1^2 mod n, ..., t_k^2 mod n}
ZK Proof of knowledge of k QR roots + k signed nonces
k
Generate y_1, ..., y_k and z_1, ..., z_k randomly
Set M = permutation π_1{t_1^2 y_1^2, ... t_m^2 y_m^2 mod n}
Set P = permutation π_2{t_1 z_1, ... t_k z_k mod n}
Send M and P
Flip coin b
b
Send Y = {y_1, ..., y_m}
b = 0
Check b
b = 1
Send A = π_2{a_1 = z_1^{-1} y_1, ..., a_k = z_k^{-1} y_k}
Verify that M is generated from Y and T = {t_1^2, ..., t_m^2 mod n}
Verify that (t_i z_i a_i)^2 = (t_i z_i z_i^{-1} y_i)^2 in M
USER
PROVIDER

Fig. 5. Diagram of FreqBadge.

$CMT_V$, in zero knowledge, that is, without the client actually revealing the badge.

The $MaintainBadge$ protocol is executed once per epoch by each active client $C$. For each venue $V$ where $C$ has performed a $CheckIn$, $C$ removes from the token set $Tk_V$ any token it has received $m$ epochs ago. It then contacts the provider to obtain the updated $Mr_V$ value. If $|Tk_V| > Mr_V$, $C$ initiates the $StatVerify$ protocol for $V$: it has become the mayor of $V$.

**Example.** Figure 4 shows an example of mayorship changes for a venue where two clients $C_1$ and $C_2$ contend for the position. $m$, the number of days over which the $CheckIn$ tokens are counted, is set to 5. After the first 5 epochs (Figure 4(a)), $C_1$ is the mayor, with 3 $CheckIn$ executions compared to $C_2$'s only 2. Thus, $Mr_V$ is set to 3. At the beginning of the 6th epoch (Figure 4(b)), the provider sets $Mr_V$ to 2. When $C_1$ is online, it runs $MaintainBadge$, detects it still has 3 tokens, thus exceeding $Mr_V$, invokes $StatVerify$ and maintains its FreqBadge. $Mr_V$ is then set back to 3. During the epoch, $C_2$ performs a new $CheckIn$. However, since its number of tokens does not exceed the $Mr_V$ value, it does not become the new mayor.

We note that if $C_1$ is not online during the 6th epoch, $C_2$ can become a mayor only after performing the new $CheckIn$ at $V$. At that time, $C_2$ has 3 tokens and $Mr_V$ = 2.

At the beginning of the 7th day (Figure 4(c)), $Mr_V$ is set to 2 and $C_1$ expires its least recent token. At this point, $C_1$ is still the mayor, since it has $Mr_V$ tokens: 2. However, as soon as $C_2$ comes online and runs $MaintainBadge$, it detects that its number of tokens exceeds $Mr_V$, invokes $StatVerify$ and becomes the new mayor of $V$.

## B. The Solution

We now describe each protocol of FreqBadge, illustrated in Figure 5.

**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**$(O(V), S(priv_S, Mr_V))$: For a newly registered venue $V$, $S$ generates a new random seed $r_V$ and uses it to initialize a pseudo-random number generator $G_V$. $S$ also generates two large primes $p_V$ and $q_V$ such that $q_V|(p_V - 1)$. Let $G_{q_V}$ be the unique subgroup of $\mathbb{Z}^*_{p_V}$ of order $q_V$. Let $g_V$ be a generator of $G_{q_V}$. $S$ publishes $p_V$, $q_V$ and $g_V$. $S$ also sets $Mr_V$ to 0: the venue has no mayor yet.

**MaintainVenue**$(S(priv_S))$: The protocol is run by the provider $S$ at the beginning of each epoch $e_i$. $S$ generates a fresh random token $t_i$, using $G_V$, and publishes $t_i^2 \ mod\ n$. $S$ decrements $Mr_V := Mr_V - 1$.

**MaintainBadge**$(C(Id, pub_S, e_i), S(priv_S, e_i)$: The protocol is run at the beginning of each epoch $e_i$ by each active client $C$, for each venue $V$ where $C$ has performed a $CheckIn$. Let $Tk_V$ denote the set of tokens received by $C$ at $V$. $C$ performs the following two actions:

- Remove from $Tk_V$ the token (if any) obtained during epoch $e_{i-m}$.
- Request from $S$ the current $Mr_V$ value. To prevent $S$ from learning the venues where $C$ has checked-in, this operation is done either over $Mix$, or using a PIR protocol. If $|Tk_V| > Mr_V$, $C$ invokes $StatVerifyC(Id, V, |Tk_V|, Tk_V, pub_S), S(priv_S))$.

**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 $t_i^2 \ mod\ n$. $S$ sends to $C$ the value $t_i$, the square root of the value published for the epoch $e_i$, along with $Mr_V$, the number of tokens of the current mayor of $V$. $C$ stores $t_i$ in the set $Tk_V$. If $|Tk_V| > Mr_V$, $C$ invokes $StatVerify(C(Id, V, |Tk_V|, Tk_V, pub_S, e_i), S(priv_S, e_i))$. All communication takes place over $Mix$.

**StatVerify**$(C(Id, V, k, Tk_V, pub_S, e_i), S(priv_S, e_i))$: All communication in this step is done over $Mix$. $C$ sends $k$ to $S$. If $k \leq Mr_V$, $S$ rejects the request and the protocol stops. Otherwise, without loss of generality, let $Tk_V = \{t_1, .., t_k\}$ be the set of all tokens retrieved by $C$ from $S$ for the venue $V$ in the past $m$ epochs. 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 $\pi_1$. $C$ computes the set $M = \pi_1\{t_1^2 y_1^2, .., t_m^2 y_m^2\}$ and sends it to $S$. $C$ needs not know $t_1, .., t_m$ to compute $M$.
- $C$ generates $z_1, .., z_k \in_R \{0,1\}^N$ and a random permutation $\pi_2$ 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, $S$ generates a fresh, random "mayor" badge $FB_V$ for venue $V$ and computes a commitment $CMT_V = g_V^{FB_V} \in G_{q_V}$. $S$ sends $FB_V$ and the signed commitment, $Sig_S(CMT_V, e_i)$ to $C$ and publishes $CMT_V$. Finally, $S$ updates $Mr_V$ to the value $k$.

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.

**ProveBadge**$(C_1(pub_S, FB_V, V, p_V, q_V, g_V)$, $C_2(pub_S, V, p_V, q_V, g_V)$, $S(priv_S, V, CMT_V)$: This protocol enables client $C_1$ to prove "mayorship" of a venue $V$ to another client $C_2$. $C_2$ retrieves $CMT_V$ from $S$. $C_1$ and $C_2$ engage in a zero knowledge protocol where $C_1$ proves knowledge of the discrete logarithm of $CMT_V$, for instance, using Schnorr's solution [44].

## C. Analysis

*Theorem 4:* The $StatVerify$ protocol of FreqBadge is a zero knowledge proof system of $k$ square roots from $\mathcal{T}^2$.

*Proof:* (Sketch) To see that FreqBadge 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**. The proof follows the approach from [45], [46]. 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 $< C, S^* >$.

$M^*$ 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. ■

Similar to the analysis of the GeoBadge protocol, here we also consider modified versions of the CI-IND and SV-IND games of Section III-D, where all the venues (chosen by $\mathcal{A}$) are identical. We now introduce the following results:

*Theorem 5:* FreqBadge is CI-IND and SV-IND.

*Proof:* (Sketch) 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: $\mathcal{A}$ can learn user information through (i) the proof and (ii) from the communication medium. However, Theorem 4 shows that $StatVerify$ is a ZK system. Furthermore, $Mix$ provides communication untraceability and unlikability (see Section IV). ■

*Theorem 6:* FreqBadge provides Status Safety.

*Proof:* (Sketch) Results directly from Theorem 4: $StatVerify$ 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. ■

FreqBadge trivially provides the token-epoch immutability property, as $S$ issues a single token per venue per epoch. FreqBadge does not provide token non-distributability. Introducing the blindly signed nonces of GeoBadge in FreqBadge to address this problem would not make sense: $S$ would be able to link two different runs of $StatVerify$ and break the SV-IND property.

## VII. e-Badge

The level 1 e-Badge is unlocked when the user checks-in at $k$ different locations having a common background (e.g., "swimmie", "wino", "pizzaiolo", see Section III-C). Each subsequent level of the e-Badge is reached when the user checks-in at $k$ new venues.

**Solution Overview.:** For each e-Badge, supporting $\overline{L}$ expertise levels, the provider generates $2\overline{L}-1$ secrets. Level 1 has only one secret, called the *outer* secret. Each level $L > 1$ has 2 secrets, the *outer* and *inner* secret. The outer secret of level $L$ is the xor between the inner secret of level $L$ and the outer secret of level $L-1$. To achieve expertise level $L$, a client needs to recover its outer secret.

The provider assigns a share of each of the $\overline{L}$ inner secrets to each qualifying venue. Thus, a venue receives $\overline{L}$ shares, each of a different secret. When a user subscribes, it receives $k$ *request tokens*, blindly signed by the provider. The request tokens enable the client to contact $k$ different venues and collect $k$ shares for the next desired level. Request tokens cannot be reused, thus preventing a user from collecting more than $k$ shares. Since they were blindly signed, the provider cannot link the request tokens to clients.

During $CheckIn$, the client uses a symmetric PIR protocol to privately collect a single share, without leaking the level desired. When the client recovers $k$ shares of the inner secret of the next level, it reconstructs the inner secret. It then combines it with the outer secret of its current level of expertise and recovers the outer secret of the next level. When the client reaches level $L$, it receives a new set of blindly signed request tokens, to enable it to acquire the next level $(L + 1)$ of expertise. We now detail each protocol of e-Badge.

**Setup**(S($\overline{L}$)): $\overline{L}$ is the number of expertise levels supported by the provider. $S$ chooses a large prime $p$ and generates a random key $K$. Similar to the $Setup$ of GeoBadge, $S$ generates a group $G$, with generator $g$. $S$ publishes $p$, $g$ and $G$ and keeps $K$ secret. For each supported e-Badge, $S$ generates a list of outer secrets $L_V = \{M_1, .., M_{\overline{L}}\}$, one for each supported expertise level, as follows:

- For level 1, generate a random value $M_1$. Use a threshold secret sharing solution to compute shares of $M_1$: generate a polynomial $Pol_1$ of degree $k-1$ whose free coefficient is $M_1$. Generate a random e-Badge for level 1, $eB_1$ and the commitment $CMT_1 = g^{eB_1} \in G$. Keep $Pol_1$'s coefficients and $eB_1$ secret. Publish the degree $k$ and the verification value $Ver_1 = H(M_1.HMAC_K(V)\ mod\ p)$. Store $Pol_1$'s coefficients. Publish $CMT_1$.
- For each subsequent level $L$, generate a random value as the outer secret $M_L$. Define the inner secret $\overline{M_L} = M_L \oplus M_{L-1}$. That is, the outer secret of level $L$ is the bitwise xor of the inner secret of level $L$ and the outer secret of level $L-1$. Use a threshold secret sharing solution to compute shares of $\overline{M_L}$ (generate a polynomial $Pol_L$ of degree $k-1$ whose free coefficient is $\overline{M_L}$. Keep $Pol_L$'s coefficients secret. Publish the verification value $Ver_L = H(M_L.HMAC_K(V)\ mod\ p)$. Store $Pol_L$'s coefficients. Generate a random e-Badge for level $L$, $eB_L$, and the commitment $CMT_L = g^{eB_L} \in G$. Keep $eB_L$ secret but publish $CMT_L$.

**RegisterVenue**$(O(\overline{L}, V, pub_O), S(priv_S))$: The owner $O$ that registers venue $V$ sends to $S$ its public key, $pub_O$. $S$ stores $pub_O$ along with $V$. If $V$ qualifies to provide an e-Badge, $S$ generates a share of a secret from each expertise level: Generate $x_V = H(V)\ mod\ p$ and $y_i = Pol_i(x_V)\ mod\ p$, for all $i = 1..\overline{L}$. $S$ stores $[x_V, y_i]$, $\forall i = 1..\overline{L}$, the shares of the secrets $M_1, \overline{M_2}, .., \overline{M_{\overline{L}}}$ along with $V$.

**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 [40] to obtain tokens that allow it later to authenticate anonymously with the server. Furthermore, $C$ generates $k$ random *request tokens* $rt_1, .., rt_k$. $C$ and $S$ engage in a blind signature protocol where $S$ blindly signs each request token for $C$. $C$ stores $Sig_S(rt_1)$, .., $Sig_S(rt_k)$ associated with the corresponding badge.

**CheckIn**$(C(Id, V, T, pub_S, L), S(priv_S))$: Let us assume that $C$ has an e-Badge at level $L-1$ and needs to acquire level $L$. The communication between the check-in client $C$ and $S$ takes place over $Mix$. If the venue $V$ qualifies for an e-Badge,

$C$ sends a yet unused, provider signed, request token to $S$. $S$ verifies its signature on the token and the fact that the token has not been used before (at any other venue). If the verifications fail, $S$ returns -1. Otherwise, $C$ and $S$ engage in a symmetric private information retrieval protocol [41], [42], allowing $C$ to retrieve a single share of the e-Badge, for the level $L$: ($X_V$, $y_L$). $C$ stores the share ($X_V$, $y_L$) along with the current secret for the level $L-1$ of the e-Badge, $M_{L-1}$. The set $Tk$ stores these values.

**StatVerify**$(C(Id, V, k, Tk, pub_S, L), S(priv_S))$: Let us assume that $C$ holds expertise level $L-1$ for the e-Badge and has performed $k$ more check-ins at qualifying venues. Thus, $C$ has a set $Tk$ storing $k$ shares of the inner secret $\overline{M_L}$. $C$ repeats the steps of $StatVerify$ of GeoBadge (over $Mix$) to reconstruct the inner secret $\overline{M_L}$. It then retrieves $M_L = \overline{M_L} \oplus M_{L-1}$ and presents the value to $S$. $S$ verifies the correctness of the value. If correct, it sends $eB_L$ to $C$, certifying e-Badge expertise level $L$. $S$ and $C$ engage in a protocol enabling $S$ to blindly sign $k$ new request tokens for $C$.

Due to lack of space, we omit the details of $ProveBadge$ that trivially extends the corresponding protocol of GeoBadge, with the exception that each expertise level has a different secret badge and corresponding commitment value.

## A. Analysis

Correctness is straightforward: a client at expertise level $L-1$, following the protocol correctly is able to retrieve $k$ shares of $\overline{M_L}$ and then recover $M_L$.

*Theorem 7:* e-Badge is CI-IND.

*Proof:* (Sketch) The communication between $C$ and $S$ during $CheckIn$ takes place over $Mix$. $C$ reveals a blindly signed request token not used before in order to perform this operation. $C$ uses a SPIR protocol to retrieve only one share of the secret needed. An adversary $\mathcal{A}$ with non-negligible advantage in the CI-IND game has the same advantage against either (i) the untraceability property of $Mix$, (ii) the blindness property of the blind signature algorithm, or (iii) the SPIR protocol in guessing the level accessed by $C$ with probability higher than $1/\overline{L}$. ∎

*Theorem 8:* e-Badge is SV-IND.

*Proof:* (Sketch) At the completion of the SV-IND game, $\mathcal{C}$ is able to reconstruct the secrets of both client $C_0$ and $C_1$ (for the same level). According to CI-IND, the adversary $\mathcal{A}$ is unable to identify clients performing $CheckIn$s. During the final $StatVerify$ run of the SV-IND game, $\mathcal{C}$ reveals only the outer secret of the level it wants to achieve, but no intermediate values. Thus, if $\mathcal{A}$ has an advantage in the SV-IND game, it has the same advantage against the (i) untraceability property of $Mix$ or (ii) the information theoretic security of the threshold secret sharing solution and the xor operation. ∎

*Theorem 9:* e-Badge provides Status Safety.

*Proof:* (Sketch) The use of a location verification protocol [43] prevents $\mathcal{A}$ from retrieving a share without being present at the venue. The use of the threshold secret sharing solution prevents $\mathcal{A}$ from reconstructing the secret of a level without completing $k$ additional $CheckIn$ operations. ∎

## VIII. Multi-Player: MPBadge

The multi-player badge is issued when a user presents a 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 $V$. This models a simplified form of the "Player Please!" badge of Foursquare, that is acquired when the user checks-in at the same location with 3 members of the opposite sex. We now present MPBadge, a privacy preserving solution that provides the co-location functionality of "Player Please! (Heart)" but without modeling the gender of the participants.

MPBadge relies on threshold secret sharing, where each client is able to provide a share of the secret. $k$ unique shares generated at the same venue in the same epoch (see protocol $MPCheckIn$) can 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**$(O(), S(priv_S))$: Perform the following steps:

- $S$ 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.

**Subscribe**$(C(), S(pub_S, priv_S))$: Inherited from GeoBadge.

**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 $Sig_{KT[P_i,e]}(R)$, $\forall i = 1..v$.

**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 $R, Sig_{KT[V,e]}(R)$ 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.

**MPCheckIn**$(C_1(Id_1, V, T, x_{e,1}, y_{e,1}),..,C_k(Id_k, V, T, x_{e,k}, y_{e,k}))$: This step is executed when $k$ clients $C_1$,..,$C_k$ are co-located. It enables them to build a co-location proof for $V$ during epoch $e$ (containing current time $T$). After performing a $CheckIn$ at venue $V$ and epoch $e$, let $(x_{e,i}, y_{e,i})$ be $C_i$'s share of $M_{V,e}$. Each device $C_i$ generates a random

MAC and IP address which it uses to setup an ad hoc network with co-located devices and also during subsequent communications. Each device $C_i$ generates the message $M = (\text{“}MPBadge'', V, e)$. $C_i$ generates $\sigma_{e,i} = M^{y_{e,i}}\ mod\ n$ and sends a multicast packet to all other devices containing the tuple $(x_{e,i}, \sigma_{e,i}, R_i, Sig_{V,e}(R_i)\ mod\ n)$. $R_i$ is the value that $C_i$ has had the server blindly sign, $Sig_{V,e}(R_i)$. Each client stores received tuples in its set $Tk$.

**StatVerify**$(C(Id, V, k, Tk, e, pub_S), S(priv_S))$: Without loss of generality, let $Tk = \{(x_{e,i}, \sigma_{e,i}, R_i, Sig_{V,e}(R_i)\}$, $\forall i = 1..k$. $C$ 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 $\sigma$, $R_i$, $Sig_{V,e}(R_i)$, 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 $Sig_{V,e}(R_i)$ 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: $Sig_S(\text{“}MPBadge'', V, e, T_c)$, where $T_c$ is the time of issue, and sends it over $Mix$ to $C$.

### A. Analysis

We extend the CI-IND game to also include the $MPCheckIn$ procedure: the adversary controls $a$ll clients except two clients $C_0$ and $C_1$, that are controlled by the challenger. The challenger then flips a coin $b$ and runs $CheckIn$ followed by $MPCheckIn$ with $\mathcal{A}$ for client $C_b$. $\mathcal{A}$'s advantage is defined the same, as the advantage over 1/2 in guessing the value of $b$. We introduce the following results.

*Theorem 10:* $MPBadge$ is CI-IND.

*Proof:* (Sketch) The blind signature generation step of $CheckIn$, executed at most once per epoch by any challenger controlled client, over an authenticated channel, retrieves a server signed nonce for each site registered in the system. During the subsequent token generation step, performed over $Mix$, the challenger reveals one signed nonce, along with the site of interest. If the adversary can link the blind signature and token generation steps with a non-negligible advantage (thus linking client to location) we can build an adversary that has the same advantage against (i) the blindness property of the blind signature scheme or (ii) the untraceability property of $Mix$. During $MPCheckIn$, the challenger sends an “identity neutral” message over $Mix$ to $\mathcal{A}$. Thus, any advantage of $\mathcal{A}$ can be converted into a similar advantage against the untraceability of $Mix$. ■

*Theorem 11:* $MPBadge$ is SV-IND.

*Proof:* (Sketch) Since we just proved that $MPBadge$ is CI-IND, the adversary's advantage can only be from the $StatVerif$ function. The communication in $StatVerif$ is performed over $Mix$ and contains an “identity neutral” $\sigma$ value along with $k$ pairs of random nonces and associated adversary generated blind signatures. The $k$ pairs have been generated during $MPCheckIn$ step, thus the adversary has no advantage from them. If at least one nonce is fresh (never used before), an adversary with an advantage in the SV-IND game following the $StatVerif$ run can be used to derive an advantage against either (i) the blindness of signature scheme, (ii) the untraceability of $Mix$ or (iii) the secure threshold signature scheme. ■

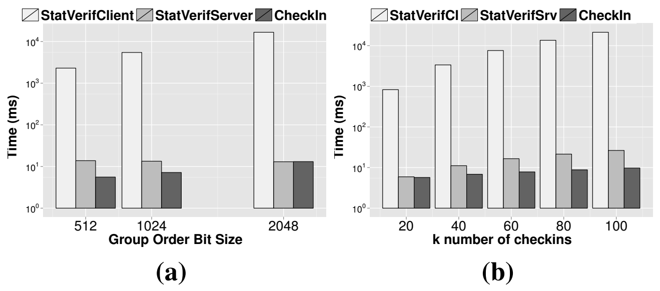


Fig. 6. GeoBadge dependence on (a) modulus size, (b) $k$, the check-in count.

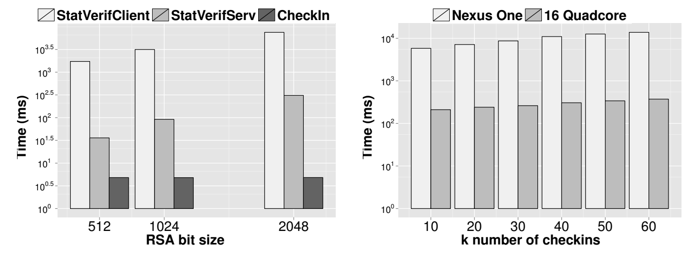
MPBadge is safe: If an adversary controls at most $k-1$ clients at a venue $V$, its advantage in the Safety game can be transformed into advantage against the information theoretical secure threshold secret sharing solution used to generate threshold signatures. MPBadge provides Token Non-Distributability, since $StatVerify$ succeeds only if $C$ provides a set of blindly signed nonces, at least one of which has never been used before.

## IX. Evaluation

We have implemented GeoBadge, FreqBadge and MPBadge in Android and Java and have tested the client side on the Nexus One smartphone and the server side on a 16 quadcore server featuring Intel(R) Xeon(R) CPU X7350 @ 2.93GHz and 128GB RAM. We have stress-tested the server side by sequentially sending multiple client requests. All the results shown in the following are computed as an average over at least 10 independent runs.

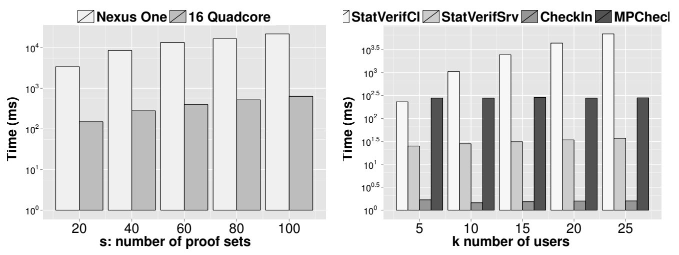
**GeoBadge:** We study the most compute-intensive functions of GeoBadge: $Setup$, the GSN provider side of $CheckIn$, the client and provider sides of $StatVerify$. We investigate first the dependence on the modulus bit size. The $Setup$ cost, a one time cost for the GSN provider, ranges from 277ms for 512 bit keys to 16.49s for 2048 bit keys.

Figure 6(a) 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. The cost of the provider side of $StatVerify$ is almost constant for different key bit sizes, also 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 $CheckIn$ or $StatVerify$ runs per second, or more than 412 million



(a) (b)

Fig. 7. FreqBadge: (a) Dependence on $N$, the modulus size, (b) StatVerify client and server side, function of $k$, the number of check-ins.



(a) (b)

Fig. 8. (a) FreqBadge: StatVerify dependence on $s$, the number of proof iterations. y axis is time in milliseconds, in logarithmic scale. (b) MPBadge dependence on $k$, the number of participants.

operations per day. The client side of $StatVerify$ takes 16.5s for 2048 bit keys, on Nexus One.

Figure 6(b) 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 $k = 100$. The provider components are much faster: the $StatVerify$ takes less than 27ms, allowing the provider to support more than 2400 such operations per second (more than 207 million ops per day). 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. In conclusion, GeoBadge imposes small overheads on the GSN provider – thousands of $CheckIn$ and $StatVerify$ can be performed per second. The client side overhead is reasonable as achieving special status is not a time constrained operation and can be performed in the background.

**FreqBadge:** In the next experiment we studied FreqBadge. 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 7(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 , requires between 1.7s to 7.5s. Since the provider is the bottleneck, the sensitive operations are $CheckIn$ and the provider side of $StatVerify$. These operations are fast: Requiring one table lookup and a signature generation, $CheckIn$ takes 4.8ms. On a 16 quadcore server, the provider can support more than 13,000 check-ins per second - more than 1.1 billion ops per day. The provider side of $StatVerify$ is less compute intensive than the client side: it ranges from 36ms to 309ms (for 2048 bit keys).

We further evaluate the dependency of $StatVerify$ (client and server side) on the value of $k$ when the modulus size $N$ is 2048, $m$=60 and $s$=40. Figure 7(b) shows that the server side exhibits small linear increases with $k$, and is only 372ms when $k = m = 60$. The server can support roughly 170 simultaneous $StatVerify$ runs per second or 14.5+ million per day. The client side overhead is around 13.8s even for 60 check-ins. Finally, Figure 8(a) shows the dependency of $StatVerify$ on the value of $s$, the number of proof sets. $N$ is set to 2048, $m$ is set to 60 and $k$ is set to 30. Even for 100 proof iterations, the cost is 633ms for the provider, enabling 6+ million daily runs. A client requires 21.2s to generate 100 proofs.

**MPBadge.** Finally, we study the dependence of the overhead of several MPBadge procedures on $k$, the number of participants. We set the modulus size to 2048 bits and range $k$ from 5 to 25. Figure 8(b) shows the performance of the server side $CheckIn$, the client and server sides of $StatVerify$ and the client side of $MPCheckIn$. On a single core the server side $CheckIn$ overhead is around 1.6ms and the server side $StatVerify$ is 37ms even for 25 participants. Thus, the provider can support 10,000 $CheckIn$s and 432 $StatVerify$s per second. The $MPCheckIn$ overhead on a smartphone is around 290ms, while the client side $StatVerify$ ranges from 230ms for 5 participants to 6.9s for 25 participants.

## X. Conclusions

In this paper we have studied privacy issues concerning popular geosocial network features, check-ins and badges. We have proposed several private protocols including (i) GeoBadge and T-Badge, for acquiring location badges, (ii) FreqBadge, for mayorship badges and (iii) MPBadge, for multi-player badges. Furthermore, we have devised e-Badge, a novel protocol that allows users to privately build expertise badges. We showed that GeoBadge, FreqBadge and MPBadge are efficient. The provider can support thousands of $CheckIn$s and hundreds of $StatVerify$s per second. A smartphone can build badges in a few seconds.

## References

[1] Foursquare. https://foursquare.com/.

[2] Yelp. http://www.yelp.com.

[3] Lauren Indvik. Foursquare Surpasses 3 Million User Registrations. http://mashable.com/2010/08/29/foursquare-3-million-users/.

[4] Jolie O'Dell. Foursquare Day Sets Record with 3M+ Checkins. http://mashable.com/2011/04/20/foursquare-day-2/.

[5] Chloe Albanesius. Apple location, privacy issue prompts house inquiry. PC Mag. http://www.pcmag.com/article2/0,2817,2365619,00.asp.

[6] Jennifer Valentino-Devries. Google defends way it gets phone data. Wall Street Journal. http://online.wsj.com/article/SB10001424052748703387904576279451001593760.html, 2011.

[7] Balachander Krishnamurthy and Craig E. Wills. On the leakage of personally identifiable information via online social networks. In *WOSN*, pages 7–12, 2009.

[8] 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.

[9] Please rob me. raising awareness about over-sharing. www.pleaserobme.com.

[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] John R. Douceur. The Sybil Attack. In *IPTPS*, pages 251–260, 2002.

[15] Bogdan Carbunar, Radu Sion, Rahul Potharaju, and Moussa Ehsan. The shy mayor: Private badges in geosocial networks. In *Proceedings of the 10th International Conference Applied Cryptography and Network Security (ACNS)*, pages 436–454, 2012.

[16] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of MobiSys*, 2003.

[17] 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.

[18] 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.

[19] Xiao Pan, Xiaofeng Meng, and Jianliang Xu. Distortion-based anonymity for continuous queries in location-based mobile services. In *GIS*, pages 256–265, 2009.

[20] 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.

[21] David Eppstein, Michael T. Goodrich, and Roberto Tamassia. Privacy-preserving data-oblivious geometric algorithms for geographic data. In *GIS*, pages 13–22, 2010.

[22] Gabriel Ghinita, Carmen Ruiz Vicente, Ning Shang, and Elisa Bertino. Privacy-preserving matching of spatial datasets with protection against background knowledge. In *GIS*, pages 3–12, 2010.

[23] Stefan Saroiu and Alec Wolman. Enabling New Mobile Applications with Location Proofs. In *Proceedings of HotMobile*, 2009.

[24] W. Luo and U. Hengartner. VeriPlace: A Privacy-Aware Location Proof Architecture. In *Proceedings of ACM SIGSPATIAL GIS*, 2010.

[25] Z. Zhu and G. Cao. APPLAUS: A Privacy-Preserving Location Proof Updating System for Location-based Services. In *Proceedings of IEEE INFOCOM*, 2011.

[26] Ge Zhong, Ian Goldberg, and Urs Hengartner. Louis, Lester and Pierre: Three Protocols for Location Privacy. In *Proceedings of PETS*, 2007.

[27] 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.

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

[29] Janus Dam Nielsen, Jakob Illeborg Pagter, and Michael Bladt Stausholm. Location Privacy via Actively Secure Private Proximity Testing. In *Proceedings of the International Workshop on Security and Social Networks (SESOC)*, 2012.

[30] Haibo Hu, Jianliang Xu, Qian Chen, and Ziwei Yang. Authenticating location-based services without compromising location privacy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 301–312, 2012.

[31] SCVNGR. http://www.scvngr.com/.

[32] The Full List of Foursquare Badges. http://www.4squarebadges.com/foursquare-badge-list/.

[33] Making badges a real reflection of your real-world expertise. http://blog.foursquare.com/2011/11/14/expertise/.

[34] How to get the Player Please! (Heart) Badge? http://www.4squarebadges.com/foursquare-badge-list/player-please-heart-badge/.

[35] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 1–15, 1996.

[36] Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.

[37] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography*, pages 31–46, 2003.

[38] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320, 2004.

[39] Michael K. Reiter and Aviel D. Rubin. Anonymous web transactions with crowds. *Commun. ACM*, 42(2):32–38, 1999.

[40] Dan Boneh and Matt Franklin. Anonymous Authentication With Subset Queries (Extended Abstract). In *in Proceedings of CCS*, 1999.

[41] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 151–160, 1998.

[42] Sanjeev Kumar Mishra and Palash Sarkar. Symmetrically private information retrieval. In *Proceedings of the First International Conference on Progress in Cryptology*, INDOCRYPT '00, pages 225–236, London, UK, UK, 2000. Springer-Verlag.

[43] Bogdan Carbunar and Rahul Potharaju. You unlocked the Mt. Everest Badge on Foursquare! Countering Location Fraud in GeoSocial Networks. In *To appear in Proceedings of the 9th IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS)*, 2012.

[44] Claus P. Schnorr. Efficient identification and signatures for smart cards. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 239–252, 1989.

[45] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1), 1989.

[46] 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.

**Bogdan Carbunar** is an assistant professor in the School of Computing and Information Sciences at the Florida International University. Previously, he held various researcher positions within the Applied Research Center at Motorola. His research interests include distributed systems, security and applied cryptography. He holds a Ph.D. in Computer Science from Purdue University.

**Radu Sion** is an Associate Professor of Computer Science at Stony Brook University. His research is in systems and security.

**Rahul Potharaju** is currently pursuing his Ph.D. at Purdue University. Prior to that, in 2009, he earned his M.S. in Computer Science from Northwestern University. He has over two years of industrial research experience working with Microsoft Research, Redmond and Motorola Applied Research Center. His current work focuses on large-scale Internet measurements, intrusion detection and security aspects of smartphones architectures and reliability aspects of data centers both from a hardware and a software perspective. A recurring theme in all his research is combining cross-domain techniques such as those from natural language processing with statistical machine learning and data mining to make surprising inferences in the networking and smartphone areas.

**Moussa Ehsan** is a computer science Ph.D. student in Stony Brook University. He earned his B.S. and M.S. degrees in computer engineering from Isfahan University of Technology (2005) and Sharif University of Technology in computer engineering (2009), respectively. He is currently working in the Stony Brook NSAC lab under the supervision of Prof. Radu Sion. His research interests include cloud computing and network security.