Privately Identifying Location Hotspots

Ananth Raghunathan Peter Chien Dan Boneh
Department of Computer Science, Stanford University
{ananthr,petchien,dabo}@cs.stanford.edu

April 11, 2012

Abstract

The ability to identify location hotspots – locations that are currently popular – is frequently used by location-based services to tune their recommendation and advertising systems. We present an efficient system that lets location-based services identify location hotspots, but without learning who is at those locations. The service also learns popular activities at hotspots, but learns nothing about who is engaging in these activities. Thus, our system strikes a balance between the service's need to mine location data and user privacy. The system is proven secure in a robust security model that protects against collusion between players and the server and protects against dishonest users who try to help a certain location appear more popular than it is.

1 Introduction

Location Privacy is an area of research that studies how to provide location based services without revealing the user's location to a 3rd party service. There is much work in this area (e.g. [BS03, CZBP06, NTL+11, PBBL11]), but very little is used in real products. One commonly cited reason is data mining — private location based services prevent service providers from mining user locations for improving the service or for monetizing it. In this paper we take a step towards showing that location privacy need not be in conflict with the need to mine location data.

In particular, we look at a specific data mining functionality that is frequently used to provide useful information to customers. Consider a location service such as Loopt or Google Latitude that tracks the locations of subscribers. The service wants to use this information to identify current hotspots where many customers are congregating. These hotspots often correspond to popular bars, concerts, movie releases, sporting events, etc. By correlating a known scheduled event at a particular location with its popularity the service can predict the popularity of this event at other locations and can use this information to recommend events to subscribers. More generally, the ability to identify hotspots is core to location services and they cannot use a privacy mechanism if it prohibits this functionality.

Currently, location services such as Loopt and Google Latitude learn the location of their subscribers and identify hotspots by simply counting the number of subscribers at a given location. They correctly identify hotspots, but in the process they also learn exactly who is at those hotspots.

In this paper we show that hotspots can be efficiently located, but without learning who is at those hotspots. More precisely, the service learns exactly how many people are at each location, but not who those people are. This information is sufficient for many hotspots marketing applications. While we focus on hotspots as an example, our thesis is that many data-mining algorithms can be efficiently implemented without learning anything other than the data-mining results. We chose to focus on the hotspots problem since we encountered that exact question in discussions with several location-based services.

Our approach to solving the hotspots question borrows from methods developed for electronic voting. The idea is that every time period, say every five minutes, the service sets up an election where all subscribers "vote" for their current (rounded) location. A property of voting systems is that the final tally becomes public, but the individual votes remain secret.

There are three standard approaches to electronic voting: (1) blind signatures [Cha88, FOO92], (2) encrypted counters [Ben87, SK94, CFSY96, CGS97, HS00], and (3) mix nets [Cha81, PIK93, SK95, Abe98]. Methods (2) and

(3) require non-colluding trustees for security: the trustees must be trusted not to decrypt mid-election ciphertexts. In our settings, these trustees do not exist since the only party who runs the election is the location-based service. Therefore, we chose to use the first approach based on blind signatures which is well suited for our settings since the same entity can serve as the registration authority authorizing users to vote and as the tabulation authority who counts location votes. In our system users register once by logging into the service and receive voting tokens enabling them to participate in many future location elections. When voting for their current location, users interact with the service over an anonymous channel. Our implementation uses Tor.

When voting for a location, users can embed additional data in the anonymous ballot that describes a current activity (e.g. the name of band that is playing or the type of dish they are eating). The service then learns the popularity of items at the location, but without learning who is contributing the data.

Security properties. §2.1 defines our exact threat model. Loosely speaking, our system ensures that every user can only vote once for their location in each election. Additionally, we prevent users from selling their voting tokens. The danger is that businesses might buy voting tokens from users and then use them to artificially make themselves appear more popular than they are. Our approach relies on the concept of *self enforcement* [DLN96]: when casting votes we require users to prove knowledge of their password (in zero-knowledge). Consequently, to sell a voting token, users must reveal their service password (e.g. Google password) to the business buying the token. This is likely to discourage users from selling tokens. Finally, we make it harder for users to lie about their location using the concept of *location tags* introduced in [NTL+11]. Location tags function as a location fingerprint that is difficult to determine without being present at the location.

To show that the scheme is practical we experimented with the system and describe our preliminary implementation in §6. The implementation is simple, efficient, and requires little infrastructure.

Related work. Popa, Blumberg, Balakrishnan and Li [PBBL11] describe a system that allows a central server to learn statistical information about user location. Their motivating example is learning the average speed of cars at particular points on the highway. Cars that pass through those points send their encrypted speed to the server using an additively homomorphic encryption. The server adds up all the encrypted speeds and then needs to ask a trustee to decrypt the results. The trustee (called an SM) decrypts the aggregate information and is trusted not to decrypt any other ciphertext. The authors propose to implement the trustee by distributing its key among the subscribers. Hence, while Popa et al. solve a more general problem than ours, their solution requires more infrastructure, and in particular requires a trustee.

By focusing on the hotspot problem we solve a more specific problem, but are able to give a simpler solution using voting based on blind signatures. In particular, we need very little infrastructure and there is no need for a separate trustee.

Pang et al.[PGK+09] describe *Wifi-Reports*, a collaborative service collecting information about access points (APs) which are analogous to locations in our application. Relevant data about APs is collected by users and reports are submitted to the service. The authors use blind signatures to ensure that each user can only publish one report for each location. However, the authors do not consider time-dependent updates of location information, which is crucial for our hotspots problem. In their system the same reporting *token* that allows a particular user to vote only for one location can be *re-used* to provide additional reports across different time periods. We solve this problem by binding tokens to time periods and do so without breaking privacy using zero-knowledge proofs.

Cornelius et al. [CKK+08] describe *Anonysense*, a privacy-aware collaborative sensing architecture. As in [PBBL11], the system requires a trusted Registration Authority to register users and implement group signatures. Moreover, the users in the security model are considered honest-but-curious and do not behave maliciously to falsify collected data.

More distant work is the paper by Cristofaro and Soriente [CS11]. Here, the authors describe PEPSI, a privacy-enhanced participatory sensing infrastructure where instead of aggregating information the model describes *sensors* that upload information and *queriers* who require information uploaded by the sensors. Unlike our settings, the authors do not consider malicious sensors and the security model requires the presence of two non-colluding services that respectively receive sensor information and answer queries. Under these assumptions much simpler identity-based encryption suffices for privacy.

2 Preliminaries

Notation: Let $\langle x, y, \ldots, z \rangle$ denote a tuple containing the elements $x, y, \ldots, z$ and $x \| y$ denote the *concatenation* of elements x and y. Let λ denote the security parameter. We use the following notation to denote an algorithm: output $\leftarrow \mathsf{Algorithm}(\text{input})$. For two distributions X and Y, let $X \approx Y$ denote the fact that the two distributions are computationally indistinguishable. And finally, we define a *negligible* function denoted by $\mathsf{negl}(\cdot)$, as any function that grows smaller than every inverse polynomial $p(\cdot)$. In other words, for every polynomial $p(\cdot)$, there exists an $N \in \mathbb{N}$ such that for all $n > N$, we have: $\mathsf{negl}(n) < 1/p(n)$. Equivalently, $\mathsf{negl}(n) = o(n^{-c})$ for every constant c.

2.1 Model

Our model comprises several *clients* and a *server*. Clients only communicate with the server and all communications in our system take place over the Internet. The server and client have a common list of locations with their unique identifiers (denoted by loc).

Clients in our scheme are people equipped with smartphones running our client software. The client software can automatically retrieve the location using services that determine geographic location from WiFi SSIDs[1]. Alternatively, users can manually enter their location. The latter system allows a dishonest client to wrongly report his/her location about which we elaborate in §2.5.

The server runs on a centralized *location service* that wishes to privately learn aggregate information about the number of people in each location so that it can identify locations with a lot of people (hotspots). Several such location-based services already exist (such as Loopt and Google Latitude). Clients are assumed to have an identifier and a password (denoted by id and pwd respectively) when they sign up with the service.

Our system requires the existence of an authenticated channel between the server and the client and another *anonymous* but authenticated channel. The former can be provided by SSL and an appropriate public-key infrastructure (such as the CA system on the Internet). The latter requires the use of special software such as Tor [DMS04]. We also require that the anonymous channel be authenticated, so in practice one can think of using SSL with Tor to achieve this.

In our system we also assume that the server and the clients are all synchronized and have a common clock. In the setting of smartphones, the clients and servers can use the Network Time Protocol [NTP] and contact NTP servers[2] to synchronize their clocks. With this synchronization, we assume that the scheme runs over a predefined time period (usually every five minutes). This ensures that the the servers accurately know the locations that are presently hotspots to within 10 minutes. The servers and the clients periodically run the protocol.

2.2 Functionality

At the end of each time period, the server must be able to compute the total number of clients at each location. The server learns nothing more. The clients learn nothing, but if required the server can publish the aggregated information (denoted $\mathsf{agglnfo}$ in this paper). It comprises of a list of locations and the number of people in each location.

2.3 Security

In our adversary model, we consider adversaries that control both the server and the client. An adversary might be a law enforcement agent coercing the service provider into revealing the user's location, or someone colluding with a dishonest server. Clients who try to break privacy might be stalkers who interact with the server. There might also be clients looking to flood the server with fake location information to artificially boost a location's popularity. We first look at the *ideal case* and then classify various adversary threats.

Ideal case. In this thought experiment, there is a trusted third party (TTP) that receives information from each client about his/her location information. The TTP verifies that clients send their information honestly (no more than once each time period and not masquerading as other clients). The TTP then sends $\mathsf{agglnfo}$ to the server and the server

[1] There are several services available such as Skyhook Wireless's *Loki* and Apple's in house services.

[2] One example is Apple's `http://time.apple.com`.

chooses to publish relevant hotspot information to the clients. Note that in our scheme we *do not need* a TTP. We just use it to describe the ideal functionality.

Dishonest server. A dishonest server (either coerced or otherwise) tries to interact maliciously with the clients to learn information it is not privy to which includes the location of a particular client during a particular time period and whether the same client reported two locations loc_1 and loc_2 during two time periods t_1 and t_2. In other words, the server should not be able to link up locations of the client (even if it has no information on *who* the actual client is[3]).

We define *location privacy* in a manner analogous to *semantic security*; roughly, the protocol should reveal no more information than revealed in the *ideal functionality*. For a more formal treatment, refer to §2.4.

Dishonest clients seeking to deanonymize other clients. We consider a dishonest client that interacts maliciously with a server to learn other clients' locations. We extend this to a dishonest client that *colludes* with the server.

Dishonest clients (or other users) seeking to falsify location information. We protect against a dishonest client that tries to interact with the server maliciously and (a) masquerade as a different client, or (b) anonymously provide its current location several times in the *same time period* to artificially *inflate* the popularity of a particular location. We extend these adversaries to "outside" users not participating in the system but with an interest in inflating location totals (such as owners of locations who want to artificially increase their popularity). For a more formal treatment, refer to §2.4. *Note:* A dishonest client that colludes with the server *can* inflate votes (see §2.5).

Dishonest clients who try to sell their location. And finally, we consider dishonest clients who sign up for the system and decide to allow third parties to send the server location information on their behalf. Thus, the server thinks it is interacting with a legitimate client but the location information is faked by third-parties with ulterior motives . This is different from inflating location data because the clients are *legitimately* allowed to send location information once every time period. We insist that the only party that can do this is the client itself. To enforce this, we note that a client is defined as a user registered with the service having an id and pwd. The scheme designed must accept location information *only* from someone who possesses a valid $\langle \mathsf{id}, \mathsf{pwd} \rangle$ combination. We term this *client accountability*.

2.4 Formal definitions of security

For security parameter λ, consider a protocol $\mathcal{P}(1^\lambda)$ between a set $\mathcal{C}$ of clients with identifiers $\{\mathsf{id}_1, \ldots, \mathsf{id}_N\}$ and a server S. Let $\mathcal{L}$ denote a list of locations $\{\mathsf{loc}_1, \ldots, \mathsf{loc}_m\}$. Let $\mathcal{D}$ denote all possible location data, i.e., the power set of all possible subsets of $\mathcal{C} \times \mathcal{L}$. For any aggregate hotspot information $\mathsf{agglnfo}$, let $\mathcal{D}_{\mathsf{agglnfo}} \subset \mathcal{D}$ denote all possible datasets that agree with $\mathsf{agglnfo}$. For a particular set of clients' location information $D \in \mathcal{D}$, let $\mathsf{view}(\mathcal{P}, D)$ denote the adversary's view in executing protocol $\mathcal{P}$ on D, i.e., the transcript of all interactions between the server and the client and depending on whether the adversary controls a client or a server, respective local computation. We define location privacy with this security game.

Security game IND-LP. The security game is defined by an adversary interacting with two experiments $\mathsf{Exp}_0(\mathcal{D})$ and $\mathsf{Exp}_1(\mathcal{D})$ with underlying possible location data $\mathcal{D}$. In the game, the adversary must distinguish between the transcripts of the protocol executed on two datasets (with the same aggregate information) of his/her choice.

1. The adversary chooses clients $\mathsf{id}_1, \ldots, \mathsf{id}_\ell$ to control. His/her subsequent view will include the local computation of these clients.
2. The challenger sets up $\mathcal{P}$ by choosing any secret or public keys required by $\mathcal{P}$ with security parameter λ and sends the public keys to the adversary.
3. The adversary chooses $\mathsf{agglnfo}$ and $D_0 \neq D_1 \in \mathcal{D}_{\mathsf{agglnfo}}$ and sends them across to the challenger. To avoid a trivial distinguishing attack, we require that the entries corresponding to the clients $\mathsf{id}_1, \ldots, \mathsf{id}_\ell$ remain identical in both D_0 and D_1.
4. The challenger runs $\mathcal{P}$ and constructs $\mathsf{view}(\mathcal{P}, D_0)$ and $\mathsf{view}(\mathcal{P}, D_1)$. In $\mathsf{Exp}_b(\mathcal{D})$, the challenger sends across $\mathsf{view}(\mathcal{P}, D_b)$ to the adversary.
5. The adversary does some local computation and outputs either 0 or 1.

[3]To see why, consider a scenario where a client runs the protocol at all times. At night, the server receives information that there is one person at a particular home location (thus, the person is most likely the resident). If the server can link this piece of information to the location of the same client at a future time period, then the server breaks the privacy of the client. For example, knowing that the same client visited a particular hospital or a particular store might lead to personal information that could be taken advantage of.

Let $\mathsf{LPAdv}[\mathcal{A}, \mathcal{P}, \mathcal{D}] = |\mathbf{Pr}[\mathcal{A}(\mathsf{Exp}_0(\mathcal{D})) = 1] - \mathbf{Pr}[\mathcal{A}(\mathsf{Exp}_1(\mathcal{D})) = 1]|$, denote the advantage of an adversary $\mathcal{A}$ in winning the IND-LP game against protocol $\mathcal{P}$. LPAdv denotes the ability of $\mathcal{A}$ to distinguish between D_0 and D_1. This leads to the following definition.

Definition 2.1 (Location Privacy)**.** A scheme $\mathcal{P}$ is said to be IND-LP secure with respect to possible datasets $\mathcal{D}$ if every polynomial time adversary $\mathcal{A}$, we have $\mathsf{LPAdv}[\mathcal{A}, \mathcal{P}, \mathcal{D}] < \mathsf{negl}(\lambda)$.

Security Game LUF. In the following game, the adversary must interact with the challenger and try to fool it into computing *incorrect* aggregate information.

1. The challenger sets up $\mathcal{P}$ as in step 1 of the IND-LP game (see previous security game).
2. The adversary chooses a client id_i and sends it to the challenger.
3. The challenger runs $\mathcal{P}$ and interacts with id_i for polynomially many time periods T. The adversary receives the view of the client and $\mathsf{agginfo}_1, \ldots, \mathsf{agginfo}_T$.
4. *Challenge:* The challenger chooses a dataset D' containing location information about all *other clients* id_j for all $j \neq i$ and the adversary chooses loc_i and interacts with the challenger to execute the protocol. Let $D := D' \cup \{\langle \mathsf{id}_i, \mathsf{loc}_i \rangle\}$ denote the dataset of client locations and $\mathsf{agginfo}(D)$ denote the agginfo computed by the ideal functionality (in other words, the *honest* hotspot information). The adversary wins the game if the server in $\mathcal{P}$ computes $\mathsf{agginfo}' \neq \mathsf{agginfo}$.

Let $\mathsf{LUFAdv}[\mathcal{A}, \mathcal{P}] = \mathbf{Pr}[\mathsf{agginfo}' \neq \mathsf{agginfo}]$ denote the advantage of an adversary $\mathcal{A}$ in winning the LUF game against protocol $\mathcal{P}$.

Definition 2.2 (Location Unforgeability)**.** A scheme $\mathcal{P}$ is said to be LUF secure if every polynomial time adversary $\mathcal{A}$, $\mathsf{LUFAdv}[\mathcal{A}, \mathcal{P}] < \mathsf{negl}(\lambda)$.

LUF with colluding clients. In Definition 2.2, we consider an adversary that only controls one client. However, the definition can be easily extended to consider ℓ colluding clients looking to forge location information. Instead of just id_i, we will allow the adversary to control several clients $\mathsf{id}_{i_1}, \ldots, \mathsf{id}_{i_\ell}$. As the adversary can directly affect $\mathsf{loc}_{i_1}, \ldots, \mathsf{loc}_{i_\ell}$, we modify $D := D' \cup \{\langle \mathsf{id}_{i_1}, \mathsf{loc}_{i_1} \rangle, \ldots, \langle \mathsf{id}_{i_\ell}, \mathsf{loc}_{i_\ell} \rangle\}$ and we require the adversary to interact with the server in executing $\mathcal{P}$ and fool it into computing $\mathsf{agginfo}' \neq \mathsf{agginfo}(D)$. In fact, in our scheme, the proof of location unforgeability will extend in a fairly straightforward manner to colluding clients.

Client accountability. We do not formally define it but the intuition is as follows: A scheme is said to have client accountability if every adversary we that successfully participates in the protocol and communicates a piece of location information to the server can be used by an *extractor* to recover a valid $\langle \mathsf{id}, \mathsf{pwd} \rangle$. This can be formalized along the lines of the security properties of zero knowledge proofs of knowledge (ZKPoKs) (see Appendix A.6).

2.5 Non-threats in our security model

In our security model, we do not provide location unforgeability when the server is dishonest. To see why this is a reasonable assumption, consider the ideal functionality. The server at the end of each time period in our protocol can compute aggregate location information. However, as the clients learn nothing except what is published by the server, the server can choose to *completely falsify* this information. In a real world setting, this would be analogous to a service like Google Latitude collecting information from all the users but choosing to publish their own list of purported hotspots. The only reason the server would be *disingenuous* in the publishing stage would be to favor one establishment over another. However, this would severely affect the integrity of the service and for the same reason a company like Google would be wary of artificially inflating page ranks of favored websites, we believe it is fair to assume honest behavior at the server's end when it comes to aggregating and publishing information.

Secondly, we allow the clients to choose their current location information. The thesis is that clients volunteer to join the scheme and help the server discover popular locations which would in turn be useful for themselves. Thus, there is no motivation for a client to falsify this information. Our security model guarantees that even a dishonest client that chooses to inform the server incorrectly only affects their location information for one time period and nothing more. With an overwhelming majority of honest clients, a single dishonest client can do little to upset aggregate data. This limitation can be overcome by using *location tags*, first introduced by Qui et al [QLE+07, QBLE09] (we elaborate in §5).

3 Scheme PrivLHS

At a high level, in PrivLHS, the clients are voters who vote their locations. In the registration stage, clients contact the server over a regular channel (not anonymous) to receive voting *tokens*. In the *voting* stage, each token allows the client to transmit his or her location and some *auxiliary data* over an anonymous channel exactly once every time period. The tokens are constructed to ensure that the client cannot sell his or her vote to a third party.

PrivLHS is closely related to a blind signature based voting system designed by Fujioka et al [FOO92] with two differences: (a) in our scheme, clients do not participate in every time-period which precisely makes it difficult to provide information integrity against a dishonest server[4], and (b) Fujioka et al consider two independent servers (called *Administrator* and *Counter*) whereas we only have one server which handles the registration and counting process. We describe the scheme using the following abstract cryptographic building blocks (with more details in Appendix A).

Message commitment scheme. [Ped91, CP92]
We require a *statistically hiding* and *computationally binding* message commitment scheme $\mathcal{MC} = (\mathsf{Commit}, \mathsf{Open})$, where $\mathsf{Commit}(m, r)$ commits to message m with randomness r (which can be subsequently opened to reveal m using Open).

One time signature. [EGM96]
We require an *existentially unforgeable* one-time signature scheme which we denote by $\mathcal{S}_{ot} = (\mathsf{Gen}_{ot}, \mathsf{Sign}_{ot}, \mathsf{Verify}_{ot})$. Here $\mathsf{sk}_{ot}, \mathsf{vk}_{ot}$ denote signing and verification keys respectively, and $\sigma_{ot}(m) \leftarrow \mathsf{Sign}_{ot}(\mathsf{sk}_{ot}, m)$ denotes a one-time signature. The algorithm $\mathsf{Verify}_{ot}(\mathsf{vk}_{ot}, \sigma_{ot}, m)$ outputs 1 if the signature is valid and 0 otherwise.

Many-time signature scheme. [GMR88]
We also require a many time signature scheme $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$. The syntax and correctness is identical to the one-time signatures presented above, so we skip the description. However, a many-time signature scheme satisfies a stronger security property of *existential unforgeability under chosen message attack:* it is infeasible for any polynomial time adversary given signatures on q *adaptively chosen messages* of his/her choice to produce a valid forgery (m', σ') for some $m' \notin \{m_1, \ldots, m_q\}$ the set of chosen message queries. We use $\sigma(m)$ (and $\sigma_{ot}(m)$ resp.) to denote a signature (one-time signature resp.) of the message m where the signing key is obvious in context.

Blind signature scheme [Cha83, PS96].
We require a blind signature scheme that satisfies the one-more unforgeability security property. Informally, $\zeta \leftarrow \mathsf{Blind}(m, r)$ and $\sigma(m) \leftarrow \mathsf{Retrieve}(\sigma(\zeta))$ are used to blind the message and retrieve a signature on the underlying message respectively. For a more formal treatment refer to Appendix A.

Cut-and-choose zero knowledge proofs.
Informally, a zero knowledge proof (ZKP) for a language is a protocol between a client and a server that allows a client to convince the server that a particular statement is a member of the language. Additionally, the protocol must not reveal *any more information about the statement*. In our scheme, we use the so-called cut-and-choose construction of a zero knowledge proof.

The cut-and-choose construction: Consider a client who wants to send across a blinded message $\zeta(m, r)$ where $m = \mathsf{id} \| \mathsf{vk}_{ot}$ to the server so that the server does not learn anything about vk_{ot}. An honest client always provides his/her correct id, whereas a dishonest client can try to trick the server into giving a signature on a different $\mathsf{id}' \neq \mathsf{id}$. To ensure that the vk_{ot} is still a secret, but m is well-formed (i.e., the id is honestly chosen), the client does the following. For a parameter ν, client first chooses ν different random keys $\mathsf{vk}_{ot}^{(1)}, \ldots, \mathsf{vk}_{ot}^{(\nu)}$. Next, for $i = 1$ to ν, the client computes $m_i = \mathsf{id} \| \mathsf{vk}_{ot}^{(i)}$ and $\zeta_i = \mathsf{Blind}(m_i, r_i)$, and sends across ζ_i to the server. The server chooses an i between 1 and ν at random (say ℓ) and asks the client to unblind the messages for all ζ_i *except* ζ_ℓ. The server then checks to see that $(m_i, \sigma(m_i))$ is a valid signature and that m_i is "well-formed" (contains the right id) for every $i \neq \ell$. If so, he is convinced that the final blinded message ζ_ℓ is well-formed as well and proceeds to sign it. A client can only cheat the server with a probability of $\frac{1}{\nu}$. For an informal proof of this protocol, please see Appendix A.5.

Zero knowledge proofs of knowledge (ZKPoKs) [CDS94].
In the voting stage of our scheme, we require a zero knowledge proofs of knowledge. In the cut-and-choose construction, we note that the statement "this message is well-formed" is true whenever $m = \mathsf{id} \| \mathsf{vk}_{ot}$ for some random vk_{ot}.

[4] See §2.5 and §5.

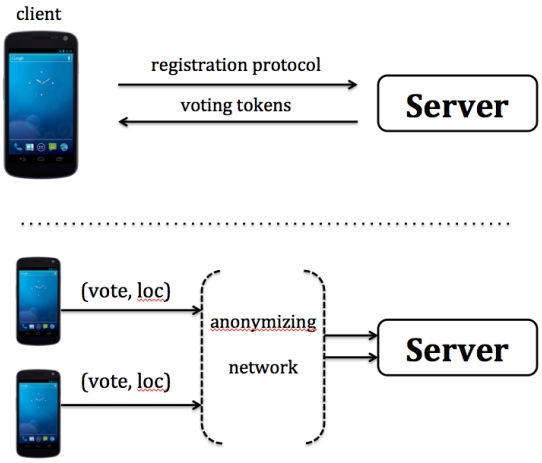

Figure 1: Architecture of PrivLHS.

The "knowledge" that the client possesses regarding the truth of this statement is the value of vk_{ot}. Now, we *could* have a client who knows that m is well-formed but does not know the value of vk_{ot}. Zero knowledge proofs of knowledge convince a server (in zero-knowledge) that not only is the statement true, but the client proving the statement has some underlying "knowledge" about this truth.

Usual ZKPoKs comprise a protocol between the client and the server that runs in three rounds (in Appendix C we show how to make the protocol non-interactive). The client sends across a commitment to the statement com. The server responds with a random challenge challenge and the client sends back a response to the challenge response. The server can then verify that the proof of knowledge is valid.

We use the Schnorr construction [Sch89] for proof of knowledge of *discrete log* and apply it to the Chaum-Pedersen commitment scheme [CP92]. See Appendix B for the details.

In the description of various randomized algorithms above, we explicitly mention the randomness r for completeness. However, in the scheme description below, for ease of readability, we drop the randomness whenever it is obvious in context.

We first describe the system setup and then the two stages of the scheme. The client registers itself with the server to receive voting *tokens*. The registration protocol is run *only once* and for every time-period, the voting protocol is executed by the client and server. After T time periods, the server and client must run the registration protocol once again for the subsequent voting stage. Let aux denote any auxiliary information that a client wishes to communicate to the server.

3.1 System Setup

The clients and server agree upon honestly generated public parameters for the message commitment scheme, signature schemes, and the ZKPoK. During setup, the server generates a signing and a verification key $(\mathsf{sk}_S, \mathsf{vk}_S) \leftarrow \mathsf{Gen}(1^\lambda)$ and publishes vk_S. Let $\sigma_S(m) \leftarrow \mathsf{Sign}(\mathsf{sk}_S, m)$ denote the server's signature on message m.

3.2 Registration stage

Every client runs the registration protocol (Figure 2) with the client over an authenticated (but not anonymous) channel. At the end of the protocol, each client computes a token τ to be used *once* during the subsequent voting stage.

The client and server repeat this protocol T times *in parallel* to generate $\tau_1, \ldots, \tau_T$ to be used for T time periods. When generating T tokens, the server must make sure that the serial numbers have the correct time stamps correspond-

Registration protocol in PrivLHS.

Step 1. The client chooses a serial number denoted by $\mathsf{s\#}$. The serial number has a time stamp to denote a time period and a unique ballot number. Next, the client commits to his/her id and pwd using a message commitment scheme. We denote this by $\mathsf{Commit}(\mathsf{id}\|\mathsf{pwd})$. And finally, the client generates a pair of one-time signing and verification keys $(\mathsf{sk}_{ot}, \mathsf{vk}_{ot}) \leftarrow \mathsf{Gen}_{ot}(1^\lambda)$ that is used to sign the location during voting.

Step 2. The client constructs a message $m := \langle \mathsf{s\#}, \mathsf{vk}_{ot}, \mathsf{Commit}(\mathsf{id}\|\mathsf{pwd})\rangle$. A message is *well-formed* if it satisfies the following properties:

- The serial number $\mathsf{s\#}$ does not repeat and uses the right time stamp.
- $\mathsf{Commit}(\mathsf{id}\|\mathsf{pwd})$ is an *honest commitment* to the ID and password.

The client requires the signature of the server on m to produce a voting token that will be used in the voting stage. However, if the server learns vk_{ot}, it can subsequently link it to id. To prevent this, we first blind the message to get $\zeta \leftarrow \mathsf{Blind}(m)$. To ensure that despite being blinded, the message is well-formed, the client and the server run a *cut-and-choose* protocol as described earlier. At the end of the protocol, the client receives $\sigma_S(\zeta)$.

Step 3. The client recovers $\sigma_S(m) \leftarrow \mathsf{Retrieve}(\sigma_S(\zeta))$ and stores a voting *token* $\tau = \langle m, \sigma_S(m)\rangle$ to be used subsequently.

Figure 2: Registration protocol in PrivLHS.

ing to the time periods $1, \ldots, T$. A cheating client who tries to get multiple voting tokens for the same time period can be detected by the server when checking whether m is well-formed or not.

3.3 Voting stage

In the registration stage the client receives T voting tokens and can execute T voting stage protocols successfully with the server. The client and the server run the protocol in Figure 3 over an *anonymous* channel. In time period i, the client uses τ_i. Without voting tokens, as location privacy demands that the clients be anonymous, the server cannot check in anyway whether a particular id does not fake location information. In particular, it cannot distinguish between the following scenarios: (a) two clients voting from the same location, and (b) one dishonest client voting twice. Voting tokens preserve anonymity but allow a client to only vote once. To ensure that a client can only vote once, and cannot sell his/her vote, we use the protocol in Figure 3.

Self-enforcement through commitments. If we omit the commitment scheme, a dishonest client can compute a well-formed m and a corresponding $\sigma_S(m)$ and then *sell* his/her token to a third party by giving them $\langle \tau, \mathsf{sk}_{ot}\rangle$. To avoid this, in rounds 3 and 4 of the voting protocol, the client must *prove* to the server in zero knowledge that he/she can open the commitment (and therefore knows $\langle \mathsf{id}, \mathsf{pwd}\rangle$). Thus, to sell a token, the third party must also learn a client's $\langle \mathsf{id}, \mathsf{pwd}\rangle$. This is sensitive information that is easy to misuse and serves as a powerful disincentive. This is an application of the principle of *self-enforcement* [DLN96].

At the end of the voting stage, each honest client successfully transmits his/her location to the server without revealing their id. The server can then tally each vote to compute $\mathsf{aggInfo}$ and use this to compute which hotspot information. As the entire computation is lightweight and the amount of transmitted information small, this protocol can be repeated a large number of times for very small time periods. Small time periods ensure that the information about popular locations is reasonably up-to-date. After the current time period expires, the counters are reset to zero and the server repopulates the number of people at each location by repeating the above protocol with participating clients.

Removing interactions. Rounds 3 and 4 can be done concurrently with rounds 1 and 2, but are presented separately for ease of exposition. In fact, the ZKPoK and the cut-and-choose zero knowledge proof, though presented as interactive proofs, can be replaced by *non-interactive* zero knowledge proofs [BFM88]. These use the Fiat-Shamir [FS86] paradigm: the client himself/herself constructs the $\mathsf{challenge}$ by using a hash function with the appropriate range. These NIZKs are secure when the hash function is modeled as a random oracle [BR93]. (For more details, see Appendix C)

Voting protocol in PrivLHS.

Round 1 (client). The client first computes $\sigma_{ot}(\mathsf{loc}\|\mathsf{aux}) \leftarrow \mathsf{Sign}_{ot}(\mathsf{sk}_{ot}, \mathsf{loc}\|\mathsf{aux})$. Recollect that $m = \langle \mathtt{s\#}, \mathsf{vk}_{ot}, \mathsf{Commit}(\mathsf{id}\|\mathsf{pwd})\rangle$ and $\tau = \langle m, \sigma_S(m)\rangle$. Next, the client sends to the server *over an anonymous channel* a 'vote' (denoted by ξ) comprising:

$$\xi := \langle \underbrace{\mathtt{s\#}, \mathsf{vk}_{ot}, \mathsf{Commit}(\mathsf{id}\|\mathsf{pwd})}_{m}, \sigma_S(m), \mathsf{loc}\|\mathsf{aux}, \sigma_{ot}(\mathsf{loc}\|\mathsf{aux})\rangle.$$

Round 2 (server). The server upon receiving the vote verifies the following.

- s# is a valid serial number (for the current time period).
- The one-time signature on loc‖aux is valid, i.e., $\mathsf{Verify}_{ot}(\mathsf{vk}_{ot}, \sigma_{ot}(\mathsf{loc}\|\mathsf{aux}), \mathsf{loc}\|\mathsf{aux}) = 1$.
- The one-time verification key is valid. This can be checked by verifying the signature $\sigma_S(m)$ using its own public key sk_S.

Round 3 (client). To prove that the vote is not sold, as discussed above, the client executes a zero knowledge proof of knowledge protocol. Let c denote $\mathsf{Commit}(\mathsf{id}\|\mathsf{pwd})$. Here $\mathcal{R}$ is the set of tuples $\langle c, (\mathsf{id}\|\mathsf{pwd}, r)\rangle$ such that $(\mathsf{id}\|\mathsf{pwd}) \leftarrow \mathsf{Open}(c, r)$, i.e., c contains a commitment to id‖pwd which can be opened by the client. Let $\Pi^{\mathcal{R}}$ be the corresponding ZKPoK for language $\mathcal{R}$. The client executes the commitment round of $\Pi^{\mathcal{R}}$ and sends across com to the server.

Round 4 (server). Given c and the first round of the ZKPoK com, the server computes a challenge challenge and sends it across to the client.

Round 5 (client). The client responds to challenge with response according to the protocol $\Pi^{\mathcal{R}}$. If the server accepts in the protocol, then loc is a valid location and the server updates the counter for the location loc. The server uses the auxiliary data as required.

Figure 3: Voting protocol in PrivLHS.

4 Security of PrivLHS

4.1 Location Privacy

For the proof of security, we assume the client only transmits loc, but depending on what auxiliary data the client wishes to transmit to the server, the proof of the full system can be shown in a straightforward manner. However, one must be careful to ensure that aux itself doesn't leak any additional information about id. Also, the IND-LP definition must be suitably modified to require indistinguishability between any two datasets that agree with aggInfo *as well as any aggregate* aux *information*.

Two datasets D_0 and D_1 are said to be *neighbors* if there exists two tuples in D_0 $\langle \mathsf{id}, \mathsf{loc}\rangle$ and $\langle \mathsf{id}', \mathsf{loc}'\rangle$ such that upon swapping the id's we get D_1. In other words, the two datasets are only one swap away. Note that D_0 and D_1 both produce the same aggInfo.

Theorem 4.1. *In* PrivLHS*, suppose that for all polynomial time adversaries, the blind signature scheme is* blinding*, the commitment scheme* hiding*, and the zero knowledge proof of knowledge protocol satisfies the* zero knowledge *property, then in each time period, for all polynomial time adversaries controlling the server, and all* neighboring *datasets* D_0, D_1*,* $\mathsf{view}(D_0, \mathsf{PrivLHS})$ *and* $\mathsf{view}(D_1, \mathsf{PrivLHS})$ *are (computationally) indistinguishable.*

We first list a couple of corollaries of this theorem and then give the proof.

Corollary 4.1. PrivLHS *is IND-LP secure against dishonest servers.*

Proof. Consider adversarially chosen datasets D_0 and D_1. Let n denote the number of clients in $\mathcal{C}$. One can go from D_0 to D_1 through at most n intermediate datasets $I_0 = D_0, I_1, \ldots, I_{n-1}, I_n = D_1$ such that I_j and I_{j+1} are neighbors. To see why, consider the following. First order the datasets so that they are in increasing order of id. Next, swap the location of id_1 in D_0 with the appropriate id so that $\langle \mathsf{id}_1, \cdot\rangle$ entry matches D_1. This constitutes intermediate I_1 which is D_0's neighbor. Now repeat the process always swapping with increasing id's noting that at the j^{th} stage, $\mathsf{id}_1, \ldots, \mathsf{id}_j$ match with the corresponding entries in D_1. From Theorem 4.1, as I_j and I_{j+1}

are neighbors, it follows that $\mathsf{view}(I_j, \mathsf{PrivLHS}) \approx \mathsf{view}(I_{j+1}, \mathsf{PrivLHS})$. A standard hybrid argument shows that $\mathsf{view}(D_0, \mathsf{PrivLHS}) \approx \mathsf{view}(D_1, \mathsf{PrivLHS})$ which completes the proof. ∎

Extending Corollary 4.1 to colluding clients. Consider a more powerful adversary that also is able to control clients. This represents a real world scenario where the server colludes with a few clients to break the privacy of others. Recollect the definition of IND-LP in the presence if dishonest clients. An adversary gets to choose two datasets with the same $\mathsf{aggInfo}$, D_0 and D_1 but now it is required that the locations of the corrupted clients be identical. For obvious reasons distinguishing between two scenarios where corrupted clients themselves move around is trivial.

To extend Corollary 4.1 to this case, note that we only require to modify elements in D_0 and D_1 that *differ*. As the elements corresponding to the corrupted id's remain the same for both D_0 and D_1, it is never the case that a corrupted id is involved in defining the intermediaries. Local computation of corrupted clients is identical for neighboring datasets and we can use Theorem 4.1 to state:

Corollary 4.2. $\mathsf{PrivLHS}$ *is IND-LP secure against a dishonest server that also colludes with clients.*

Now, we give the proof of the theorem.

Theorem 4.1. We describe the proof as a series of hybrid games. Let $\mathsf{id}_0, \mathsf{id}_1$ denote the swapped identifiers in the two datasets. In D_0 we have $\langle \mathsf{id}_0, \mathsf{loc}_0 \rangle, \langle \mathsf{id}_1, \mathsf{loc}_1 \rangle$, whereas in D_1 we have $\langle \mathsf{id}_0, \mathsf{loc}_1 \rangle, \langle \mathsf{id}_1, \mathsf{loc}_0 \rangle$.

In Game_0, the challenger interacts with the adversary as in Exp_0 in the IND-LP game. The adversary gets to see $\mathsf{view}(D_0, \mathsf{PrivLHS})$.

In Game_1, the view of the registration stage remains the same as in Game_0. In the voting stage, the challenger does the following: for client id_0, instead of sending across the six tuple with $\mathsf{Commit}(\mathsf{id}_0 \| \mathsf{pwd}_0)$, the challenger cooks up another honest commitment $\mathsf{Commit}(\mathsf{id}_1 \| \mathsf{pwd}_1)$, signs the modified message m' to get $\sigma_S(m')$ and sends the modified six tuple across. Note that commitments to both id_0 and id_1 are *valid*. The challenger also correspondingly modifies the ZKPoK to produce another valid ZKPoK. For id_1, the challenger repeats the same procedure as above using a commitment and ZKPoK for $\mathsf{Commit}(\mathsf{id}_0 \| \mathsf{pwd}_0)$.

In Game_2, the challenger interacts with the adversary as in Exp_1 in the IND-LP game. The adversary gets to see $\mathsf{view}(D_1, \mathsf{PrivLHS})$.

Now we show that if the blind signature scheme is statistically *blinding*, the commitment scheme *hiding*, and the zero knowledge proof of knowledge satisfies the *zero knowledge* property, then the adversary's views in Game_0 and Game_1 (and in a largely analogous manner, in Game_1 and Game_2) are indistinguishable. We do not consider the ordering between id_0's and id_1's interactions with the server in the transcript. The anonymity of the channel during the voting stage will not allow the adversary to learn which part of the transcript corresponds to which player and the order does not matter.

Now let us look at the adversary's view. Across Game_0 and Game_1, in the view, loc for each client remains the same. Also, the challenger setting up the system can faithfully compute $\sigma_S(\cdot)$ as required. Thus we ignore these two components.

In Game_0, for $b \in \{0, 1\}$, let m_b denote $\langle \mathtt{s\#}, \mathsf{vk}_{ot}^{(b)}, \mathsf{Commit}(\mathsf{id}_b \| \mathsf{pwd}_b) \rangle$ and $\mathcal{T}(m_b)$ denote the transcript of the corresponding ZKPoK.

In Game_1, let m'_b denote $\langle \mathtt{s\#}, \mathsf{vk}_{ot}^{(b)}, \mathsf{Commit}(\mathsf{id}_{1-b} \| \mathsf{pwd}_{1-b}) \rangle$. The relevant part of the adversary's view therefore comprises:

$$\mathsf{Game}_0 : \langle \underbrace{\mathsf{Blind}(m_0), \mathsf{Blind}(m_1)}_{\text{from the reg. stage}}, \underbrace{m_0, m_1, \mathcal{T}(m_0), \mathcal{T}(m_1)}_{\text{from the voting stage}} \rangle$$

$$\mathsf{Game}_1 : \langle \underbrace{\mathsf{Blind}(m_0), \mathsf{Blind}(m_1)}_{\text{from the reg. stage}}, \underbrace{m'_0, m'_1, \mathcal{T}(m'_0), \mathcal{T}(m'_1)}_{\text{from the voting stage}} \rangle$$

We need to show that these two views are indistinguishable. The hiding property of the commitment scheme ensures that $\langle m_0, m_1 \rangle \approx \langle m'_0, m'_1 \rangle$. However, the adversary is also given auxiliary information (blinded messages and a zero knowledge proof of knowledge). Given two message μ_1, μ_2, without knowing whether they are m_0, m_1

or m_0', m_1', we simulate the rest of the corresponding view. As the blind signature is statistically blinding, there is a distribution that is *independent* of the underlying message (perhaps depending only on the public parameters of the scheme) that is statistically close to the blinded message. We sample two values β_1, β_2 from that distribution. Next, consider the transcripts $\mathcal{T}(\mu_1), \mathcal{T}(\mu_2)$. From the zero knowledge property, $\mathcal{T}(\mu)$ only has information that μ has an honest commitment and there exists a simulator that successfully simulates $\mathcal{T}(\mu)$ whenever this is true. Given μ_1, μ_2 we use this simulator to construct $\mathcal{T}(\mu_1), \mathcal{T}(\mu_2)$.

The tuple $\langle \beta_1, \beta_2, \mu_1, \mu_2, \mathcal{T}(\mu_1), \mathcal{T}(\mu_2) \rangle$ simulates the rest of the adversary's view. The ZKPoK allows us to replace commitments with other commitments that are honestly constructed. And a statistically blind signature ensures that $\mathsf{Blind}(m)$ leaks no information about m even in the presence of auxiliary information (such as m itself). Therefore, any adversary that can distinguish between the views in Game_0 and Game_1 given $\langle \beta_1, \beta_2, \mu_1, \mu_2, \mathcal{T}(\mu_1), \mathcal{T}(\mu_2) \rangle$ can be used to distinguish between commitments to m_0, m_1 and m_0', m_1'. However, the commitment scheme is *hiding*, and we conclude that the adversary *cannot* distinguish between his views in Game_0 and Game_1.

In a similar manner, one can show that Game_1 and Game_2 are also indistinguishable to an adversary. The proof follows exactly the same approach except we end up with $\mathsf{vk}_{ot}^{(0)}$ for id_1 and $\mathsf{vk}_{ot}^{(1)}$ for id_0. However, note that vk_{ot} is chosen randomly by each player. Therefore, swapping the two vk_{ot}'s (and correspondingly modifying the one-time signature on the location) does not allow the adversary to distinguish between the two transcripts because they're chosen uniformly at random and therefore independently from the rest of the view.

Thus, we've established that for all polynomial time adversaries, $\mathsf{Game}_0 \approx \mathsf{Game}_1 \approx \mathsf{Game}_2$ which concludes the proof of Theorem 4.1.

∎

4.2 Location Unforgeability

Theorem 4.2. *Any adversary with non-negligible LUF advantage against* PrivLHS *with high probability* must *either (a) break the one-time existential unforgeability of the one-time signature scheme, or (b) break the one-more unforgeability of the blind signature scheme.*

From this it follows:

Corollary 4.3. *Suppose that the underlying signature schemes are unforgeable (existential and one-more respectively), then* PrivLHS *is LUF secure.*

Proof of Theorem 4.2. Consider a server executing PrivLHS and interacts with an adversary breaking the LUF security game. Let D denote the *true* dataset and $\mathsf{agglnfo}(D)$ denote the corresponding hotspot information. At the end of the voting stage, the server receives m votes anonymously $\xi_1, \xi_2, \ldots, \xi_m$ and computes $\mathsf{agglnfo}'$. We know $\mathsf{agglnfo}' \neq \mathsf{agglnfo}(D)$ (as the adversary breaks the LUF game). Consider the various ways the adversary can vote such that $\mathsf{agglnfo}'$ is calculated incorrectly. The adversary could have multiple votes for the same id for the same time period. If the adversary only votes once, as all other clients are considered honest, the only other way to falsify $\mathsf{agglnfo}'$ is to modify another client's vote. We consider these two possibilities:

1. $m \neq n$. In this case, there are at least two *valid* ξ's (say) ξ_i and ξ_j that share the same id. There are two messages m_i, m_j such that $\sigma_S(m_i)$ and $\sigma_S(m_j)$ are valid. We argue that with high probability one of the two messages, say m_j, was never queried in the registration stage.

   If m_j *was* queried in the registration stage, from the cut-and-choose protocol, it follows that with probability at least $\frac{\nu-1}{\nu}$, m_j is well formed. As m_i and m_j share the same serial number $\mathsf{s\#}$, the server would not have produced (the blinded) signatures for both m_i and m_j. Thus m_j could not be successfully queried in the registration stage.

   From this, it follows that the adversary must have used the blind signature scheme and produced a 'one-more forgery' to get a valid signature of m_j breaking the blind signature scheme.

2. $m = n$ and the adversary modifies an honest client's vote. Let this vote be ξ_i. If the adversary uses its own sk_{ot} and vk_{ot} in constructing m_i, then $\sigma_S(m_i)$ would not be valid unless the adversary produces a one-more forgery

on the blind signature scheme. If the adversary uses the honest client's vk_{ot} and produces a different location $\mathsf{loc}' \neq \mathsf{loc}_i$, then to successfully produce $\sigma_{ot}(\mathsf{loc}')$ the adversary must break the unforgeability of the one-time signature scheme.

This completes the proof of Theorem 4.2. ∎

Extending Theorem 4.2 to colluding clients. We can extend the proof of Theorem 4.2 to include colluding clients. This is fairly straightforward. Controlling more than one client gives the adversary no extra power (a discerning reader will observer in the proof above that, in fact, controlling even one adversary does not impart an advantage to the adversary). Recollect from the definition of LUF that it does not matter what choices of loc corrupted clients make. We only consider it a forgery if the adversary corrupts aggInfo of the *remaining* clients. Thus, we can honestly communicate with and simulate the interactions and votes corresponding to corrupted clients without using the interaction to forge a one-time or a blind signature. Even if we consider multiple clients, the proof will come down to the two possibilities discussed above.

4.3 Client accountability

In order to prevent clients from selling their location information to third parties, we use the principle of *self-reliance* [DLN96]. We show that any adversary that successfully gives location information (fake or otherwise) on a client's *behalf* must know the clients id and pwd. We then argue that under ordinary circumstances, clients will be wary of giving out their id and pwd to third parties as this information is sensitive and can be used maliciously against them. This serves as a useful disincentive. The server already knows the clients id and pwd, therefore clients wouldn't be wary of interacting with the server.

Theorem 4.3. *(Informal) Assuming the one-more unforgeability of the blind signature scheme, with high probability any adversary that successfully submits a vote ξ must know a client's* id *and* pwd.

Informal Proof. The proof follows almost directly from the *special soundness* property of the ZKPoK. Assuming the unforgeability of the signature schemes, any valid vote ξ must have been queried for during the registration stage and therefore must be well-formed with high probability (otherwise the adversary produces a one-more forgery). As the vote is valid, and the underlying message well-formed, ξ must have a valid commitment $\mathsf{Commit}(\mathsf{id}\|\mathsf{pwd})$ of some client *and* a successful zero knowledge proof of knowledge that the commitment is valid (see steps 3–5 in the voting stage). The special soundness property of the zero knowledge proof of knowledge guarantees the existence of an *extractor* that interacts with the prover (in this case the adversary) and extract a valid witness (in this case the valid witness is an opening to the commitment). This witness has both the client's id and pwd and that completes the proof. (The extractor is described in the proof of Theorem B.1). ∎

5 Extensions

5.1 Location unforgeability against a dishonest server

Recollect from §2.5, we do not provide location unforgeability against a dishonest server. In theory, along the lines of the construction in [FOO92], we could modify our system to also allow users to *audit* the published hotspot information at a later time. If there are any discrepancies in the published hotspot information, the server can publish in addition to location information the one-time verification key and signature of the location. Given $\langle \mathsf{loc}, \mathsf{vk}_{ot}, \mathsf{Sign}_{ot}(\mathsf{loc}) \rangle$, clients check to see that the signature is valid and that *their* vk_{ot} appears somewhere in the list. If a client's vote has been discarded by a dishonest server, the client can then go to a suitable authority and publish $m, \mathsf{Sign}_S(m)$ from his/her voting token where m contains vk_{ot}. The client must also prove (not in zero knowledge) that the commitment is valid. If the signature scheme used is one-more unforgeable and the message well-formed, this proves to the authority that the server has discounted the client's vote. Thus, a dishonest server's behavior can be detected.

Although this method is reasonable for election schemes as elections are held infrequently, the fact that our system runs periodically every five minutes makes it infeasible for clients to audit the system in that short period. Moreover,

in the rebuttal process, a client reveals his/her identity. Getting around this will add further rounds of communication and make the scheme unnecessarily complicated and is therefore left for future work.

5.2 Using location tags to discover dishonest clients

Recollect that we also allow clients to falsify their location information as long as they vote for the location at most once per time period. Briefly, a location tag is a secret associated with a point in space and time, mainly a set of *location features* derived from signals present in the nearby physical environment. In [NTL+11], the authors go into details about constructing location tags that are *reproducible* and *unpredictable*. Reproducibility ensures that the location tags constructed by two or more clients in a particular location always have at least some threshold number of features in common. Unpredictability ensures that clients that are not in a particular location cannot predict its location tag to greater than the threshold accuracy. Location tags can be constructed from WiFi broadcast packets, WiFi access point IDs, Bluetooth devices in the vicinity, GPS data, GSM data, and even audio.

Location tags allow a server to check if a client is falsifying his/her location information. Recollect that in a vote ξ, the clients can place auxiliary information about the location in a private manner (especially if $\mathtt{aux}$ is only a function of location). To detect clients that might be falsifying their location information, the server can check to see if the location tags for each vote with the same location all share at least a threshold number of location features. If there are location tags that deviate beyond this threshold, then they can be discarded. This is a rather simple fix that works only when locations have large number of votes (for example, a location with two conflicting tags will not reveal which of the two clients voted falsely) but can be made more robust. We leave this for future work. We conclude the section by noting that our construction allows us to also extend the notion of security accommodating more adversary threats, but make the scheme more cumbersome to deploy in practice.

6 Implementation

To experiment with the system we implemented the main parts and measured their performance. The implementation runs on Motorola Droid Android phones, which have 550 Mhz ARM processors and 256 MB of RAM, and on a virtual private server (VPS) on slicehost with 256 MB of RAM and 10 GB of disk space. To test bandwidth requirements, we implemented the most bandwidth-intensive operations—the registration and zero knowledge protocols. We wrote all the code in Java, both for the Android client and the server; we used Google and Android libraries for getting user locations and the Bouncy Castle library for implementing the cryptographic primitives.

To run the voting protocol, we used a time period of five minutes. The information retrieved was therefore at most ten minutes out-of-date. The cut-and-choose protocol was instantiated with $\nu = 100$ messages (of which 99 messages at random were opened up). This means that a dishonest client will be detected with %99 certainty. The registration protocol was run with $T = 10$ which allowed the clients to transmit up to 10 locations before refreshing their voting tokens.

For the anonymous channel in the voting protocol, we used Tor. To access Tor on the Android phones, we used Orbot, the standard Tor client for Android. To ensure authenticity of all communications (especially to protect against possibly snooping Tor exit nodes), the servers and clients communicated over SSL.

We implemented both the one-time signature and the blind signatures using RSA-PSS [BR96]. To ensure adequate security, we used a RSA modulus of 2048 bits. For the purposes of testing, we used 192-byte long client ids during registration. We used 64 byte ballot $\mathtt{s\#}$'s, with the last 8 bytes being time. The implementation is about 2600 lines of code, including a lightweight user interface.

Performance. Recall that our system uses Tor for anonymity when reporting location information to the server. The latency introduced by Tor has little impact on our system – it only affects when the server receives location ballots – and can be safely ignored. We note that Tor, by itself, does not solve the hotspot problem since it does not prevent users from casting multiple votes per election or lying about their location.
The following table summarizes the performance of different parts of the system:

	Time	Bandwidth
Client registration	$< 1s$	12 KB per token
Voting for a location	$< 1s$	4.7 KB per vote

The 12KB per token during registration is primarily due to the zero knowledge proofs. After registration, when voting for a location the phone sends out about 4.7KB of data, which even for heavy users who use the system for 8 hours a day every day will end up consuming about a total of 22.8 MB of bandwidth each month. While noticeable, this bandwidth is far less than current lowest tier data plans (200 MB/month). Bandwidth can be lowered by reducing the frequency of elections (e.g. once every 30 minutes instead of every 5 minutes) and limiting the time of day when elections run (e.g. limit to evenings and weekends only).

One of the most pressing demands today of mobile applications is to conserve the battery. We tested the client using *PowerTutor* and observed that it uses approximately 3.0 mW (excluding power used by the screen). This is less power than several applications including Orbot.

7 Conclusion

We showed how to identify location hotspots, and popular activities at those hotspots, without learning who is at those hotspots. Our scheme PrivLHS extends to more general notions of aggregate information where each participating entity is only allowed to contribute information a limited number of times. We give formal notions for location privacy, location unforgeability, and client accountability and show that PrivLHS satisfies all the above requirements. We briefly described a few extensions that can enhance security.

PrivLHS is non-interactive and uses simple cryptographic constructions. We implemented the system in Java on the Android platform and showed that it doesn't require much bandwidth, power, or memory to run on these smartphones.

References

[Abe98] Masayuki Abe. Universally verifiable mix-net with verification work indendent of the number of mix-servers. In *EUROCRYPT*, pages 437–447, 1998.

[Ben87] Josh Benaloh. "verifiable secret-ballot elections", phd thesis, yale university, 1987.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with rsa and rabin. In *EUROCRYPT*, pages 399–416, 1996.

[BS03] Alastair Beresford and Frank Stajano. Location privacy in pervasive computing. *IEEE security and privacy*, pages 1536–1268, 2003.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.

[CFSY96] Ronald Cramer, Matthew K. Franklin, Berry Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. In *EUROCRYPT*, pages 72–83, 1996.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*, pages 103–118, 1997.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[Cha83] David Chaum. Blind signature system. In *CRYPTO*, page 153, 1983.

[Cha88] David Chaum. Elections with unconditionally-secret ballots and disruption equivalent to breaking rsa. In *EUROCRYPT*, pages 177–182, 1988.

[CKK+08] Cory Cornelius, Apu Kapadia, David Kotz, Daniel Peebles, Minho Shin, and Nikos Triandopoulos. Anonysense: privacy-aware people-centric sensing. In *MobiSys*, pages 211–224, 2008.

[CP92] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO*, pages 89–105, 1992.

[CS97] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical Report No. 260, Dept. of Computer Science, ETH Zürich, March 1997.

[CS11] Emiliano De Cristofaro and Claudio Soriente. Short paper: Pepsi - privacy-enhanced participatory sensing infrastructure. In *WISEC*, pages 23–28, 2011.

[CZBP06] Reynold Cheng, Yu Zhang, Elisa Bertino, and Sunil Prabhakar. Preserving user location privacy in mobile data management infrastructures. In *Privacy Enhancing Technologies*, pages 393–412, 2006.

[DLN96] Cynthia Dwork, Jeffrey B. Lotspiech, and Moni Naor. Digital signets: Self-enforcing protection of digital information (preliminary version). In *STOC*, pages 489–498, 1996.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320, 2004.

[EGM96] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *J. Cryptology*, 9(1):35–67, 1996.

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *AUSCRYPT*, pages 244–251, 1992.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *EUROCRYPT*, pages 539–556, 2000.

[NTL+11] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, and Dan Boneh. Location privacy via private proximity testing. In *NDSS*, 2011.

[NTP] https://en.wikipedia.org/wiki/Network_Time_Protocol.

[PBBL11] Raluca A. Popa, Andrew J. Blumberg, Hari Balakrishnan, and Frank H. Li. Privacy and accountability for location-based aggregate statistics. In *ACM Conference on Computer and Communications Security*, pages 653–666, 2011.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[PGK+09] Jeffrey Pang, Ben Greenstein, Michael Kaminsky, Damon McCoy, and Srinivasan Seshan. Wifi-reports: improving wireless network selection with collaboration. In *MobiSys*, pages 123–136, 2009.

[PIK93] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *EUROCRYPT*, pages 248–259, 1993.

[PS96] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In *ASIACRYPT*, pages 252–265, 1996.

[QBLE09] D. Qui, D. Boneh, S. Lo, and P. Enge. Robust location tag generation from noisy location data for security applications. *The Institute of Navigation International Technical Meeting*, 2009.

[QLE+07] D. Qui, S. Lo, P. Enge, D. Boneh, and B. Peterson. Geo-encryption using loran. *The Institute of Navigation International Technical Meeting*, 2007.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pages 239–252, 1989.

[SK94] Kazue Sako and Joe Kilian. Secure voting using partially compatible homomorphisms. In *CRYPTO*, pages 411–424, 1994.

[SK95] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In *EUROCRYPT*, pages 393–403, 1995.

A Cryptographic Primitives

In this section, we give more details about the various cryptographic primitives used in the scheme.

A.1 Message commitment scheme [Ped91, CP92]

A message commitment scheme $\mathcal{MC} = (\mathsf{Commit}, \mathsf{Open})$ runs in two phases. In the *commitment* phase, the client runs $(c, \mathsf{sk}) \leftarrow \mathsf{Commit}(m, r)$ which takes a message m and randomness r and produces a commitment to the message c and produces a secret key sk that is used subsequently to open the commitment. In the *opening* phase, to open the commitment, the client runs $(m, r) \leftarrow \mathsf{Open}(c, \mathsf{sk})$ which takes a commitment c, the secret key sk and outputs the message m and the randomness r used in the commitment.

A secure bit commitment scheme satisfies the following properties:

- *hiding:* a commitment c reveals no information about the message m, and
- *binding:* it is infeasible for a person committing to c to produce an opening (m', r') such that $m' \neq m$. In other words, c *binds* the user to the message m.

A.2 One time signature [EGM96]

A one time signature scheme $\mathcal{S}_{ot} = (\mathsf{Gen}_{ot}, \mathsf{Sign}_{ot}, \mathsf{Verify}_{ot})$ is an *efficient* signature scheme that is secure when used exactly once. The key generation algorithm outputs a pair of one-time signing and verification keys upon input the security parameter, i.e., $(\mathsf{sk}_{ot}, \mathsf{vk}_{ot}) \leftarrow \mathsf{Gen}_{ot}(1^\lambda)$. On input the signing key and a message m, Sign_{ot} outputs a signature, i.e., $\sigma_{ot}(m) \leftarrow \mathsf{Sign}_{ot}(\mathsf{sk}_{ot}, m)$. Verify_{ot} outputs 1 or 0 on input a signature and a corresponding message, i.e., $\{0, 1\} \leftarrow \mathsf{Verify}_{ot}(\mathsf{vk}_{ot}, \sigma_{ot}, m)$.

A secure one-time signature satisfies the following properties:

- *correctness:* for every $(\mathsf{sk}_{ot}, \mathsf{vk}_{ot}) \leftarrow \mathsf{Gen}_{ot}(1^\lambda)$, and for every message m, $\sigma_{ot}(m)$ is a valid signature, i.e., $\mathsf{Verify}_{ot}(\mathsf{vk}_{ot}, \sigma_{ot}(m), m) = 1$, and
- *one-time existential unforgeability:* it is infeasible for any polynomial time adversary given *one* signature on any message of his/her choice to produce a valid *forgery* on any *other* message. A valid forgery is a pair (m', σ') for some $m' \neq m$ such that $\mathsf{Verify}_{ot}(\mathsf{vk}_{ot}, \sigma', m') = 1$.

A.3 Blind signature scheme [Cha83, PS96].

As in [FOO92], we require a blind signature scheme to ensure the server *cannot link* players across the two stages. A blind signature scheme in addition to the three algorithms described above comprises two additional algorithms, Blind and Retrieve. Blind takes a message m and blinds it using randomness r to produce a blinded message ζ and a state τ that allows one to retrieve a signature subsequently, i.e., $(\zeta, \tau) \leftarrow \mathsf{Blind}(m, r)$. Usually, it suffices to have $\tau = r$. Retrieve on input a signature $\sigma(\zeta)$ and state τ retrieves a valid signature of the underlying message m, i.e., $(m, \sigma(m)) \leftarrow \mathsf{Retrieve}(\sigma(\zeta), \tau)$.

A secure blind signature scheme satisfies the following properties:

- *correctness:* as in the definition of one-time signatures,
- *blinding:* ζ, the blinding of the message m for uniformly chosen r statistically hides the message m, and
- *one-more unforgeability:* as the blinding and retrieving property allows an adversary to retrieve a signature of a message from every signature query we cannot expect the scheme to be existentially unforgeable. Instead, we rely on a weaker notion that states that for any q it is infeasible for a polynomial time adversary given q signatures on adaptively chosen message queries, produce $q + 1$ signatures (i.e., recover one more signature).

A.4 Zero knowledge proofs [GMR89]

For a class of languages $\mathcal{L}$, a zero knowledge proof is an interactive protocol that is carried out between two parties, the *prover* and the *verifier* (the client and server respectively). The prover has a statement $x \in \mathcal{L}$ and must *convince* the verifier that $x \in \mathcal{L}$ without revealing any more information about x itself. Informally, a zero knowledge proof satisfies the following properties:

- *completeness:* for every true statement in the class, an honest server *accepts* the clients proof with high probability,
- *soundness:* a dishonest client cannot convince a server of the truth of a *false* statement, except with a very low probability, and
- *zero knowledge:* for every possible (dishonest) server, and every possible true statement, there is a *simulator* that given no information about the statement successfully simulates the *transcript* of an interaction between the server and the honest client proving the true statement. In other words, when interacting with a possibly dishonest server, the transcript of the interaction leaks no information other than the truth of the statement.

A.5 Informal proof of the security of the cut-and-choose construction

Correctness: This construction satisfies correctness in a fairly straightforward manner.

Soundness: To see that the construction is sound, we note that a server chooses ℓ uniformly at random from 1 to ν and only accepts if all $\nu - 1$ messages after opening are well-formed. Thus, in order to cheat and get a signature on an *ill-formed* message, the client must guess ℓ in advance. Therefore, the cheating client only succeeds with probability $\frac{1}{\nu}$.

Zero-Knowledge: And finally, to see that the transcript leaks no information about $\mathsf{vk}_{ot}^{(\ell)}$, we show informally how a simulator, without knowing anything about $\mathsf{vk}_{ot}^{(\ell)}$, can simulate a real transcript. Choose a random ℓ and for $i \neq \ell$ choose random $\mathsf{vk}_{ot}^{(i)}$ and construct a well-formed m_i and ζ_i. For $i = \ell$, simply choose a random value for ζ_ℓ. Now the transcript is as follows: In the first round, the transcript has ζ_i. This looks like a real interaction because except for ζ_i they are honestly constructed and ζ_ℓ, chosen randomly looks like a blinded signature. The second round, the server returns ℓ, which the simulator chose uniformly from 1 to ν as in a real scheme. And in the final round, $\nu - 1$ blinded messages are opened and valid signatures are sent across to the server by the simulator. The simulated transcript therefore has the *same distribution* as a real transcript and the scheme is zero knowledge.

A.6 Zero knowledge proofs of knowledge (ZKPoKs) [CDS94].

Abstractly, we consider a binary relation $\mathcal{R} = \{\langle S, \kappa \rangle\}$ for which membership can be tested in polynomial time. It is instructive to think of S encoding a statement with corresponding knowledge κ. For any S, its *witness set* $\kappa(S)$ is set of κ's such that $(S, \kappa) \in \mathcal{R}$. A ZKPoK protocol Π_{zkpk} is (usually) a three round protocol[5] comprising the following interactions. In the first round the client sends across a commitment[6] $\mathsf{com}(S, r)$ that depends on the statement S and some randomness r. In the second round, the server returns (what is usually a random) challenge. And in the third round, the client responds with a $\mathsf{response}(S, r, \mathsf{challenge}, \kappa)$. After this, the server does some local computation and outputs 1 (to indicate that the proof is valid) or 0 (to indicate an invalid proof).

In addition to correctness and zero-knowledge as defined above, zero knowledge proofs of knowledge have a stronger notion of soundness:

(b) *special soundness:* the length of challenge is such that the number of challenges is superpolynomial in λ (the security parameter), and for any client, given two conversations between the client and the server $(\mathsf{com}, \mathsf{challenge}, \mathsf{response})$ and $(\mathsf{com}, \mathsf{challenge}', \mathsf{response}')$, when $\mathsf{challenge} \neq \mathsf{challenge}'$, an element of $\kappa(S)$ can be computed in polynomial time.

Although special soundness is less general than the standard definition, all known proofs of knowledge have this property, or a slightly weaker variant that involves a small number of correct challenge-responses.

B ZKPoK for Chaum-Pedersen

In this section we show a zero knowledge proof of knowledge for opening a Chaum-Pedersen commitment. First, we review the Chaum-Pedersen commitment scheme.

B.1 Chaum-Pedersen commitment [CP92]

Setup: To setup the system, choose a group $\mathbb{G}$ of prime order p over which discrete log is hard. Next, choose a random generator g and a random $h \in G$. This completes system setup.

$\mathsf{Commit}(m, r)$: Given a message $m \in \mathbb{Z}_p$, and random $r \xleftarrow{R} \mathbb{Z}_p$ and set $c = g^m \cdot h^r$ and $\mathsf{sk} = r$.

$\mathsf{Open}(c, \mathsf{sk})$: To open the commitment c to the message m, simply publish $(m, \mathsf{sk}) = (m, r)$. For correct m and r, one can verify that the opening is correct by checking whether $c \stackrel{?}{=} g^m \cdot h^r$.

The commitment scheme is hiding because h^r is distributed uniformly in $\mathbb{G}$ and therefore hides g^m. The binding property follows from the discrete log assumption over $\mathbb{G}$ (in particular, given g, h it is hard to find $a \in \mathbb{Z}_p$ such that $g^a = h$).

B.2 ZKPoK

The following proof of knowledge is adapted largely from more general zero knowledge proofs of *representations* as defined in [CS97]. Recollect from PrivLHS, that given public parameters g, h from the Chaum-Pedersen commitment scheme, $\mathcal{R}$ is the set of tuples $(c, (m, r))$ where $c = \mathsf{Commit}(m, r)$. The server is given the commitment c and the public parameters of the Chaum-Pendersen scheme g, h. The zero knowledge proof of knowledge $\Pi^{\text{cp}}_{\text{zkpk}}$ that the client knows m, r that open c proceeds as follows:

- *Round 1:* The client picks random $r_1, r_2 \xleftarrow{R} \mathbb{Z}_p$ and sends across to the server the values $R_1 = g^{r_1}, R_2 = h^{r_2}$.
- *Round 2:* The server picks random $\gamma \xleftarrow{R} \mathbb{Z}_p$ and returns this as the challenge.
- *Round 3:* The client computes $z_1 = r_1 + \gamma \cdot m \,(\text{mod } p)$ and $z_2 = r_2 + \gamma \cdot r \,(\text{mod } p)$ and (z_1, z_2) comprises the client's response.

[5] These are often called Σ-protocols.

[6] Not to be confused with the message commitment algorithm $\mathsf{Commit}(\cdot, \cdot)$ although they are closely related.

- The server receives z_1, z_2 and checks to see if $R_1 \cdot R_2 \cdot c^\gamma \stackrel{?}{=} g^{z_1} \cdot h^{z_2}$. If so, the server outputs 1 (accepts); else it outputs 0 (rejects).

Theorem B.1. *Π^{cp}_{zkpk} is a secure zero knowledge proof of knowledge.*

Proof Outline. This is a standard argument in crypto literature and we will only briefly mention the ideas.

Correctness. Correctness follows rather trivially. As $c = g^m \cdot h^r$, it can be worked out that $R_1 \cdot R_2 \cdot (g^m h^r)^\gamma = g^{z_1} \cdot g^{z_2}$ for all R_1, R_2, and γ.

Zero knowledge. The simulator that can simulate client-server transcripts that are indistinguishable from a real interaction is very similar to the simulator in [Sch89]. The way the simulator works is to *cheat* and choose γ first. Given an honest c the simulator then chooses R_1, R_2 and z_1, z_2 (without knowing r_1, r_2 or of course the underlying m, r) such that $R_1 \cdot R_2 \cdot c^\gamma = g^{z_1} \cdot h^{z_2}$. It can be shown that for random γ and honest c, the transcript the simulator produces $\langle R_1, R_2, \gamma, z_1, z_2 \rangle$ is distributed in a manner indistinguishable from the real transcript.

Special soundness. To show that a client that always causes a server to verify can be used to *extract* the message m (and randomness r), we use the *rewinding trick* (described indirectly in the definition of special soundness previously). We rewind the client to receive two accepting transcripts such that the clients initial commitment is the same (which requires rewinding). In this case, we have R_1, R_2 and *two* $\langle \gamma_1, z_{11}, z_{12} \rangle, \langle \gamma_2, z_{21}, z_{22} \rangle$. From this we can compute $m = (z_{11} - z_{12}) \cdot (\gamma_1 - \gamma_2)^{-1} \pmod{p}$ which extracts the message m as required. ∎

C Removing interactions using a random oracle

The interactions in PrivLHS arise from the cut-and-choose zero knowledge proof of well-formedness of the message (during registration) and the three-round ZKPoK that the client can open the commitment (during voting).

Blum, Feldman and Micali [BFM88] introduced the concept of *non-interactive* zero knowledge proofs (NIZKs). A standard way to construct NIZKs by removing interactions in (interactive) zero knowledge proofs (and proofs of knowledge) is to use the Fiat-Shamir paradigm [FS86]. This involves constructing the challenge in the second round of both protocols (a random ℓ and a random challenge respectively) by hashing the commitment in the first round using a hash function $H(\cdot)$ that takes strings and maps them to the appropriate domain (integers between 1 and 100 and the challenge-domain respectively). Now, instead of a three round protocol involving interactions between the client and the server, the client himself/herself computes the challenge, the response to this challenge, and transmits to the server the values for all three rounds (commitment, challenge, and response). This collapses the three round protocol down to a one round (non-interactive) protocol. These NIZKs remain secure when $H(\cdot)$ is modeled as a *random oracle* [BR93].