Distributed Approaches for Location Privacy

by

Ge Zhong

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

With the advance of location technologies, people can now determine their location in various ways, for instance, with GPS or based on nearby cellphone towers. These technologies have led to the introduction of location-based services, which allow people to get information relevant to their current location. Location privacy is of utmost concern for such location-based services, since knowing a person's location can reveal information about her activities or her interests.

In this thesis, we first focus on location-based services that need to know only a person's location, but not her identity. We propose a solution using location cloaking based on k-anonymity, which requires neither a single trusted location broker, which is a central server that knows everybody's location, nor trust in all users of the system and that integrates nicely with existing infrastructures. Namely, we suggest having multiple brokers, each deployed by a different organization (e.g., an operator of a cellphone network) and each knowing the location of only a *subset* of users, with the subsets being disjoint. The servers and a user can jointly determine the cloaked area based on k-anonymity. We present and analyze two protocols both of which exploit the same idea above. The evaluation of our sample implementation demonstrates that one of the protocol is sufficiently fast to be practical, but the performance of the other protocol is not acceptable for its use in practice.

In addition to the distributed k-anonymity protocol which serves as a general solution for location privacy when users' identities and fine-grained location are not required, we then propose four protocols—Louis, Lester, Pierre and Wilfrid— for a specific, identity required, location-based service: the nearby-friend application, where users (and their devices) can learn information about their friends' location if and only if their friends are actually nearby. Our solutions do not require any central trusted server or only require a semi-trusted third party that dose not learn any location information. Moreover, users of our protocol do not need to be members of the same cellphone provider, as in existing approaches. The evaluation on our implementation shows that all of the four protocols are efficient.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Urs Hengartner, for his guidance and support. In the last two years, I learned a lot from him. I could not have completed this thesis without his help. I would also like to thank Dr. Douglas Stinson and Dr. David Jao for their careful reviews and valuable comments.

I would like to acknowledge Dr. Ian Goldberg for his cooperation and contribution in our research.

Special thanks to my parents for their love, support and encouragement, and to Haitian Zhao for her love.

Last but not least, I would like to thank all other CrySP Lab members for spending the two memorable years with me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Location-Based Service

With the advance of location technologies, various positioning systems can determine people's locations with high accuracy, such as GPS, nearby cellphone towers, and wireless access points. The receiver of all these positioning systems can be integrated into a small mobile device with limited computing power and storage, which can report location information to people almost anywhere anytime. These technologies have led to the introduction of location-based services (LBS), which provide people information relevant to their current locations. There are various types of existing LBS, such as:

Navigation service. This service provides directions to a user-defined target location.

Location-based traffic and weather alerts. This service provides traffic and weather information to users in real time. It could be combined with the navigation service to avoid traffic congestion or road hazards in bad weather.

Nearby-information service. This service provides information in a user's vicinity that could be of interest to the user such as locations of nearby restaurants, gas stations, and advertisement of any nearby services.

Nearby-friends service. This service, also called buddy tracking, notifies a user when some of his/her friends are nearby.

Children, elderly parents, or car finders. This service provides the locations of a user's children, elderly parents, or vehicles when they are lost.

Emergency management service. This service provides emergency responders with the location of the emergency to help them reach the location as soon as possible. For example, E-911.

Moreover, there are many other types of location-based services and probably many new ones will appear for future demand. These location-based services will become more and more involved in everyone's daily life.

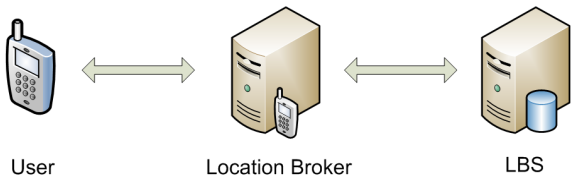

Figure 1.1: General System Architecture

1.2 Location Privacy

Location-based services improve the quality of life by providing people with useful information related to their locations, but also invoke people's concerns about potential threats. The utmost concern of location-based services is location privacy. Most people would not agree to report regularly updated location information to some third parties where the usage of this information is out of the knowledge and control of the people themselves. To protect location privacy, we must first define what location privacy is. Privacy could mean different things in different context. Location privacy can be defined as the claim of individuals to determine for themselves when, how and to what extent location information about them is communicated to others [23]. In short, people should have the control of their own location information.

People's locations just like their home address, telephone number, ages and medical history, are considered as private information, so people want themselves to control over it. Failures to protect location privacy will result in at least three key negative effects [23]:

- **Location-based spam:** Malicious parties could send unsolicited advertisement to users based on users' location.
- **Personal safety:** Theft, stalking or physical attacks are more likely to happen if people's locations are leaked to a malicious party.
- **Intrusive inferences:** People's activities and interests could be inferred by the locations they have visited.

Therefore, protecting people's location privacy is a very basic requirement in the environment of ubiquitous positioning devices and location-based services. Researchers have investigated various approaches to protect people's location privacy.

1.3 Existing Approaches

In most existing location-based services, the system architecture, as shown in Figure 1.1, consists of three entities: the user, the location broker and the location-based service. The user provides the location broker his/her fine-grained location

and a location-based query. The location broker performs the location privacy enhancing algorithm on the fine-grained location. Then the location broker sends the location-based query with the processed location to the location-based service. The location-based service finds the best answer to the query and returns it to the user via the location broker. The location broker could also give the processed location to the user, then the user sends the location-based query with the processed location directly to the location-based service.

In this section, we provide a general overview of the existing approaches used to enhance location privacy in the system described above.

1.3.1 Access Control

As a traditional security mechanism, access control is used to protect location privacy in some existing solutions. Access control allows one user to impose restriction on the ability of other users to retrieve information. It is usually used in the location-based services that require location sharing among users, such as a nearby-friends service. A wide variety of access control methods have been developed for different environment, such as encryption-based access control and role-based access control [5]. This approach requires a centralized architecture where the central server has all users' location information. The access control mechanism is applied to protect the private data on the server. However, the central server itself may become a single point of failure that could leak private information. We want to avoid the central server that has all location information, which is not a goal of access control. Moreover, access control basically requires a user to reveal her identity, which is not necessary in many location-aware applications. It would be preferred that users can hide their identities from the location-based service when identity information is not needed.

1.3.2 Pseudonyms

A person who accesses a location-based service might be tempted to reveal her precise location. Due to location privacy concerns, people want to hide their identities from the location-based service. One approach is to use pseudonyms. In the system shown in Figure 1.1, pseudonyms could be generated by the user herself or by the location broker. The user could use different pseudonyms for every request or for different time periods. However, if the location is associated with this user (e.g., her home), just knowing the location enables the service to re-identify the user. Moreover, since users' moving pattern usually follows a smooth path, changing pseudonyms becomes ineffective. Here, a location-based service can easily link new pseudonyms with previous pseudonyms. For example, if at time t_i, we know a user at location l_i and moving toward a direction with velocity $\vec{v_i}$, and later on at time t_{i+1} we see a query at $l_i + \vec{v_i}(t_{i+1} - t_i)$, then it is most likely that this query is from the same user. To prevent this attack, Beresford and Stajano proposed Mix

zones [6]. In their approach, users can communicate with a location-based service in an *application zone*, but they are not allowed to do so when they are in a *mix zone*. Assuming users change to a new, unused pseudonym whenever they enter a mix zone, the location-based service that sees a user emerging from the mix zone cannot distinguish that user from any other who was in the mix zone at the same time and cannot link people going into the mix zone with those coming out of it. Some other solutions use silent periods [33, 44], which are essentially a temporal version of mix zones. These techniques, which prohibit any user to communicate with the services either in a predefined area or in a period of time, inevitably sacrifice the quality of service, and the actual degree of anonymity that they can provide is much related to the configuration detail, which is hard to control. Moreover, not all classes of location-based service can accept pseudonyms. Some services, like buddy tracking, require real identity.

1.3.3 Obfuscation

The idea of this approach is to obfuscate the location of users in order to introduce confusion about the identification of the actual user. The obfuscation task is performed by the user or with the help of the location broker as shown in Figure 1.1. The assumption made in obfuscation is that if a location-based service requires much less spatial and temporal resolution than the underlying positioning system provides, then users can still get high quality of service by providing relatively coarse-grained location. This assumption works well in most services, and we will talk about different techniques to perform the obfuscation below. However, obfuscation cannot solve all problems. Some services require fine-grained location, such as buddy tracking.

Dummy Location

Some researchers propose using dummy location as the obfuscation technique [38, 59]. In this technique, a user sends true position information with several false position information (dummies) to a location-based service, which provides the result for each position information. The user then extracts the true result from the dummy results. The difficulty of this approach is that it is hard to ensure that the dummy locations appear to be realistic in both the short and long term, and that the distance deviation between the dummy paths and the real path is large enough. Moreover, the higher the degree of privacy, the more dummies are needed, and the communication cost is also higher.

Basic Location Cloaking

The other proposed way to obfuscate location is location cloaking. The idea of location cloaking is that instead of using a fine-grained location that can easily identify

a user, the user sends more coarse-grained, less invasive location information to a service. In basic cloaking, a user simply determines an area that has a predetermined size (e.g. four city blocks) and that contains her current location and sends the area to the location-based service. The service will return information about the entire area, and the user will discard irrelevant information. Presumably, there are other users in the coarse-grained location area. Then the attacker cannot distinguish the real sender from any other user who was in the same area at the same time [16]. The problem of basic cloaking is that the user does not know whether there are enough users in the predetermined location area in advance. For example, in a rural, less populated region, it is highly probable that the predetermined area only contains the user herself. If the user is the only user in the area or the total number of users in the area is so small that the attacker can distinguish them with high probability, then privacy is breached.

Location Cloaking based on k-Anonymity

A clever way to do location cloaking that avoids the problem of basic cloaking is location cloaking based on k-anonymity [53]. Here, a user's current location is cloaked such that there are at least $k-1$ other users within the cloaked area. A location-based service learns only the cloaked area, and the user remains anonymous within the set of k users.

Location cloaking based on k-anonymity has been studied extensively [5, 7, 17, 28, 30, 29, 32, 35, 46, 48]. Traditionally, this approach has been implemented with the help of a central trusted server, the location broker. Users register their current location with the location broker. Whenever a user wants to access a location-based service, she has the location broker compute a cloaked area that has the k-anonymity property. Then, the location broker contacts the location-based service on the user's behalf. The drawback of this approach is that the location broker knows everybody's location. Users must trust it not to leak their location information to unauthorized parties, maybe inadvertently. In short, the location broker is a single point of failure, which we want to avoid.

More recent research has proposed to get rid of the location broker and to have (nearby) users jointly compute a cloaked area that has the k-anonymity property. Then, the user (or, for increased privacy, another user on her behalf) contacts the location-based service. The drawback of this approach is that all existing solutions trust users to implement the proposed solution faithfully and not to leak location information learned during the computation. Whereas this requirement might hold in a closed environment, where users know each other, it will be difficult to satisfy in more open environments.

Another drawback of both the centralized and the distributed approach is that neither of them integrates well with existing infrastructures for location-based services. Namely, many existing location-based services are targeted at cellphone users, since the operator of a cellphone network knows the current location of *its*

customers and can provide this information to a location-based service. However, there is no single entity that knows the location of *all* cellphone users across all cellphone networks, as required by the centralized approach. In case of the distributed approach, it fails to take advantage of the already existing location information that an operator has about its customers.

1.3.4 Summary

As we can see, if identity and fine-grained location are not required by a service, location cloaking based on k-anonymity is the state-of-art approach for location privacy. It avoids the possibility of linking in pseudonyms and guarantees the anonymity within the set of k users. However, some services, like buddy tracking, require users' identity and fine-grained location. In this case, we could use a specialized location privacy technique (see Chapter 3). Furthermore, no matter what kind of a location-based service and what approach it uses for location privacy, we want to avoid introducing a central server that has everybody's location information.

1.4 Contributions

1.4.1 Distributed k-Anonymity

In this thesis, we first focus on location-based services that need to know only a person's location, but not her identity. We propose a solution in Chapter 2 using location cloaking based on k-anonymity, which requires neither a single trusted location broker nor trust in all users of the system and that integrates nicely with existing infrastructures. Namely, we suggest having multiple brokers, each deployed by a different organization (e.g., an operator of a cellphone network) and each knowing the location of only a *subset* of users, with the subsets being disjoint. The servers and a user can jointly determine the cloaked area based on k-anonymity.

We present and analyze two protocols of this approach. Both protocols exploit homomorphic encryption (see Section 2.3). In the first protocol, homomorphic encryption is used to construct the underlying secure greater-than protocol, which compares two input values securely without letting the other party know its own input (see Section 2.3.3). Then using homomorphic addition, users can calculate the sum of the number of people in their area reported by individual location brokers and compare it with their privacy threshold k. Users can also register their location to any one of the location brokers and switch the registration to another broker as they wish. Therefore, k-anonymity is assured for users, and no single location broker can track a user (if the broker is not the user's cellphone provider). In the second protocol, homomorphic encryption is used to construct the underlying secure threshold set-union protocol, which computes the threshold union of private sets without leaking the elements that appear less than the threshold times in the

union (see Section 2.9). Each location broker constructs a set which contains n copies of the identifier of a cell if the cell contains n people. Then using the secure threshold set-union protocol, a location broker can learn whether the number of people in an area is larger than the given privacy threshold k.

We implemented and evaluated the two protocols. We show that the first protocol is sufficiently fast to be practical, but the performance of the second protocol is not acceptable for its use in practice.

1.4.2 Buddy Tracking

In addition to the distributed k-anonymity protocol which serves as a general solution for location privacy when users' identities and fine-grained location are not required, we then propose four protocols—Louis, Lester, Pierre and Wilfrid [1]—in Chapter 3, for a specific, identity required, location based service: the nearby-friend application, where users (and their devices) can learn information about their friends' location if and only if their friends are actually nearby. Our solutions do not require any central trusted server or only require a semi-trusted third party that does not learn any location information. Moreover, users of our protocol do not need to be members of same cellphone provider, as in existing approaches.

Our protocols also take advantage of homomorphic encryption. The first three protocols use homomorphic addition to calculate the square of the distance between Alice and Bob. In the Louis protocol, the square of the distance is compared with the square of the threshold distance r by a semi-trusted third party. In the Lester protocol, Alice has to solve a discrete log to get the square of the distance. She can solve it quickly only if the distance is small. In the Pierre protocol, Alice and Bob use more coarse-grained location information. Alice can learn Bob's location only if the square of the distance is 0, 1 or 2. The last protocol, Wilfrid, does not need to calculate the distance between Alice and Bob. It gives the users more flexible ways to define their nearby area. Alice can learn Bob's location only if their nearby area intersects.

We implemented and evaluated the four protocols. The experiments show that all of the four protocols are efficient. They can be run on a device that is capable to perform public key cryptoscheme.

[1] Our protocols are named after four former residents of 24 Sussex Drive, Ottawa.

Chapter 2

Two Distributed k-Anonymity Protocols

2.1 Introduction

In this chapter, we focus on location-based services that need to know only a person's location, but not her identity. Example services in this category are services that return road maps, nearby places (e.g., restaurants or gas stations), or current traffic conditions. As we already mentioned in Section 1.3.2, a service that learns only a person's location might still be able to re-identify the person [32]. For example, the location could be associated with the person (e.g., her home) or the location corresponds to a place that is under physical surveillance by the location-based service. Once a service has re-identified a person, the service can connect the dots and build a detailed location profile for this person (assuming the person uses the service in a continuous way).

As explained in the previous chapter, location cloaking based on k-anonymity is the most promising general approach to protect location privacy when a user's identity and fine-grained location are not required by the service. However, both centralized and distributed approaches have their drawbacks. In this chapter, we propose a solution that requires neither a single trusted server nor trust in all users of the system and that integrates nicely with existing infrastructures. Namely, we have multiple servers, each deployed by a different organization (e.g., an operator of a cellphone network) and each knowing the location of only a *subset* of users (e.g., the operator's customers), with the subsets being disjoint. When a user wants to access a location-based service, she cloaks her area and asks each server for the number of people in this area. In a naïve solution, the servers simply give her these numbers, she sums them up and, if the sum is at least k, she accesses the location-based service. However, this approach has the flaw that the user could track people if the number of people in a queried area is small. For example, if the user learns that there is only a single person in an area and nobody in the surrounding areas, the user can likely follow the path of the person when the person leaves the area

and enters one of the surrounding areas. As soon as the person enters an area that is associated with her identity or that is under surveillance by the user, the user can re-identify her.

Our solution avoids this problem with the help of cryptography and ensures that a user cannot learn the number of people in an area reported by a server. Moreover, the user can learn only whether the sum of these numbers is at least k. A user can also change the location broker that she reports her location to as she wishes, so no single server could track a user. This approach satisfies the result of Barkuus and Dey [4]'s study on people's concern about location privacy: "Concern about location privacy can be dependent on the type of application, with applications that track users' movements over a period of time causing more concern than simple positioning application". Therefore, on the side of a location-based service provider, users' privacy is protected by location cloaking based on k-anonymity, and on the side of a location broker, users' privacy is protected by the ability to switch between location brokers.

The contributions of this chapter are as follows:

- First, we introduce a distributed k-anonymity protocol for location privacy in which a user collaborates with multiple servers and a third party to learn whether there are at least k people in her area. Nobody, not even the servers and the third party, can learn the total number of people in the area.
- Second, we present a protocol that prevents users from registering multiple times with different servers and hence from skewing the total number of users in area.
- Third, we present a sample implementation of our protocol. In its evaluation, we demonstrate that our protocol can be implemented efficiently.
- Fourth, we present and analyze another distributed k-anonymity protocol, which also has multiple servers but no third party, though its performance is not as good as the previous protocol.

In Section 2.2, we discuss related work in the area of k-anonymity and location privacy. Our protocol exploits the Paillier cryptosystem [50] and Blake and Kolesnikov's protocol for strong conditional oblivious transfer [8], which we review in Section 2.3. In Section 2.4, we present our system and threat model of our first protocol. We introduce the first protocol in Section 2.5 and evaluate it in Section 2.6. We give a security analysis in Section 2.7 and study some deployment issues in Section 2.8. In Section 2.9, we present the second distributed k-anonymity protocol, which exploits privacy-preserving set operations, and explain why its performance is not as good as the previous protocol.

2.2 Related Work

Samarati and Sweeney [53] propose k-anonymity to enable the release of person-specific information from a database while maintaining individuals' privacy. Previous research has applied k-anonymity to the release of location information that occurs when a user queries a location-based service. We first discuss related work that is based on a central trusted server, then we review distributed approaches.

Gruteser and Grunwald [32] propose both spatial cloaking and temporal cloaking of location information. In the former, a user's location is cloaked such that there are at least $k-1$ other users in the cloaked area. In the latter, sending of a query to a location-based service is delayed until at least $k-1$ other users have visited the user's location. In this chapter, we concentrate on spatial cloaking, but our approach also applies to temporal cloaking. Gruteser and Grunwald have a trusted "location anonymizer" perform the cloaking based on a quadtree. Upon a query, the location anonymizer subdivides space into quadrants until it finds a quadrant that contains the query issuer and fewer than $k-1$ other users. The parent quadrant becomes the cloaked area. Gedik and Liu [28] let users have personalized values of k, and the cloaked area corresponds to the minimum bounding rectangle of k users. Mokbel et al. [48] observe that this approach can leak information about a user's location (e.g., some users will be on the boundary of the rectangle). They use a balanced quadtree that is traversed bottom-up for better performance until a quadrant with at least k users is found. In our approach, we choose the bottom-up strategy and allow users to personalize k.

Beresford [5] finds that, if a location-based service is familiar with the cloaking algorithm and knows the locations of all users within the cloaked area, the service could infer the identity of the query issuer from the shape of the cloaked area. Namely, this happens when the cloaked area generated for the query issuer is different from the cloaked areas that would have been generated for the other users in the cloaked area. Kalnis et al. [35] and Bettini et al. [7] later re-discover this finding. Kalnis et al. and Mascetti and Bettini [46] present (centralized) cloaking algorithms that are not susceptible to this attack. In our approach, we leave it up to a user to decide what kind of cloaking algorithm to use. She can use either an algorithm similar to Mokbel et al.'s algorithm that does not necessarily guarantee her privacy, but is easy to compute, or an algorithm similar to Mascetti and Bettini's that is robust in terms of privacy, but more expensive.

Chow et al. [17] propose the first distributed approach for location k-anonymity. A user who wants to access a location-based service broadcasts a message with Bluetooth or WiFi. Nearby users respond to this message with their current location. If the number of responses is smaller than $k-1$, the user repeats the process, but has the nearby users forward the message, maybe iteratively. The user then computes her cloaked location and, for increased privacy, asks a nearby user to send her query for the cloaked location to the location-based service. Ghinita et al. [30] show that this approach often fails to achieve location privacy, since the query issuer tends to be in the center of the cloaked area. The same authors [29] later propose to use

a distributed hash table based on Chord [56]. Chord requires that a user knows at least the positions of the two users that immediately follow and precede her in the hash table. Furthermore, for robustness reasons, a user also needs to know the positions of $log_2(n)$ other users, where n is the number of users. In summary, the proposed distributed approaches for location k-anonymity have the drawback that nodes can learn location information about other nodes, so the nodes have to trust each other [29].

Kapadia et al. [36] propose "statistical k-anonymity". They assume the global availability of statistical data about the number of people who are present in an area with high probability at a particular time of the day. When a user wants to access a location-based service, she independently decides based on this data whether her area is likely to be visited by at least k people. The drawbacks of this approach are that there remains a chance that there fewer than k people in the area and the requirement of extensive data collection (across different communication technologies and providers and during different times of the day, days of the week,...) to compute the provided statistical data, which raises privacy issues of its own. Our approach is always accurate and requires no such data collection.

The idea of replacing a single trusted infrastructure component with multiple components, each of them having only a limited view of the overall system and run by a different organization, has been exploited by other privacy-enhancing technologies, such as Tor [21]. Similar to Tor, our solution takes advantage of a directory server that advertises infrastructure components being part of the system to users of the system.

2.3 Preliminaries

Our protocols use the techniques of public-key cryptography, but we require the cryptosystem used to have a special algebraic property: that it is *additive homomorphic*. An additive homomorphic cryptosystem is one in which, given only $\mathcal{E}(m_1)$ and $\mathcal{E}(m_2)$, one can efficiently compute $\mathcal{E}(m_1 + m_2)$. In this section, we review one such cryptosystem and a threshold version of it. Furthermore, we review a protocol for securely computing the greater-than predicate.

2.3.1 Paillier

In the Paillier cryptosystem [50], like in the RSA cryptosystem, a user Alice selects random primes p and q and constructs $n = pq$; plaintext messages are elements of $\mathbb{Z}_n$. Unlike RSA, however, ciphertexts are elements of $\mathbb{Z}_{n^2}$. Alice picks a random $g \in \mathbb{Z}_{n^2}^*$ and verifies that $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$ exists, where $\lambda = \text{lcm}(p-1, q-1)$ and $L(x) = (x-1)/n$. Alice's public key is then (n, g) and her private key is (λ, μ).

To encrypt a message m, another user Bob picks a random $r \in \mathbb{Z}_n^*$ and computes the ciphertext $c = \mathcal{E}(m) = g^m \cdot r^n \bmod n^2$. To decrypt this message, Alice computes

$\mathcal{D}(c) = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$, which always equals m.

Given $\mathcal{E}(m_1) = g^{m_1} \cdot r_1^n \bmod n^2$ and $\mathcal{E}(m_2) = g^{m_2} \cdot r_2^n \bmod n^2$, Bob can easily compute $\mathcal{E}(m_1 + m_2) = \mathcal{E}(m_1) \cdot \mathcal{E}(m_2) \bmod n^2 = g^{m_1+m_2} \cdot (r_1 r_2)^n \bmod n^2$, and given $\mathcal{E}(m_1) = g^{m_1} \cdot r_1^n \bmod n^2$ and m_2, Bob can easily compute $\mathcal{E}(m_1 \cdot m_2) = (\mathcal{E}(m_1))^{m_2} \bmod n^2 = g^{m_1 \cdot m_2} \cdot (r_1^{m_2})^n \bmod n^2$.

In the rest of this chapter, $\mathcal{E}_A(\cdot)$ is the Paillier additive homomorphic encryption function using public key A.

2.3.2 Threshold of Paillier

A threshold decryption scheme is a protocol that allows any subset of $t+1$ out of l entities, or servers, to decrypt a ciphertext, but that disallows the decryption if fewer than t servers participate in the protocol. The Paillier cryptosystem can be extended to support threshold decryption [25]. This version is used in the second distributed k-anonymity protocol that we investigate. The details of the threshold version of Paillier are as follows:

Let $\Delta = l!$ where l is the number of servers.

Key generation algorithm. Choose an integer n, product of two strong primes p and q, such that $p = 2p' + 1$ and $q = 2q' + 1$ and $\gcd(n, \varphi(n)) = 1$. Set $m = p'q'$. Let β be an element randomly chosen in $\mathbb{Z}_n^*$. Then randomly choose $(a, b) \in \mathbb{Z}_n^* \times \mathbb{Z}_n^*$ and set $g = (1+n)^a \times b^n \bmod n^2$. The secret key $SK = \beta \times m$ is shared with the Shamir scheme: let $a_0 = \beta m$, randomly choose t values a_i in $\{0, \ldots, n \times m - 1\}$ and set $f(X) = \sum_{i=0}^{t} a_i X^i$. The share s_i of the i^th server P_i is $f(i) \bmod nm$. The public key PK consists of g, n and the value $\theta = L(g^{m\beta}) = am\beta \bmod n$. Let $VK = v$ be a square that generates of cyclic group of squares in $\mathbb{Z}_{n^2}^*$. The verification keys VK_i are obtained with the formula $v^{\delta s_i} \bmod n^2$.

Encryption algorithm. To encrypt a message M, randomly pick $x \in \mathbb{Z}_n^*$ and compute $c = g^M x^n \bmod n^2$

Share decryption algorithm. The i^{th} player P_i computes the decryption share $c_i = c^{2\Delta s_i} \bmod n^2$ using his secret share s_i. He makes a proof of correct decryption which assures that $c^{4\Delta} \bmod n^2$ and $v^\Delta \bmod n^2$ have been raised to the same powers s_i in order to obtain c_i^2 and v_i. Fouque et al. [25] explains this proof in more detail.

Combining Algorithm. If fewer than t decryption shares have valid proofs of correctness the algorithm fails. Otherwise, let S be a set of $t+1$ valid shares and compute the plaintext

$$M = L(\prod_{j \in S} c_j^{2u_{0,j}^S} \bmod n^2) \times \frac{1}{4\Delta^2\theta} \bmod n$$

where $u_{0,j}^S = \Delta \times \prod_{j' \in S \setminus \{j\}} \frac{j'}{j'-j} \in \mathbb{Z}$

2.3.3 Secure Greater Than

Blake and Kolesnikov [8] propose the Greater Than - Strong Conditional Oblivious Transfer (GT-SCOT) protocol. The protocol has two participants, a receiver and a sender. The receiver and sender have private inputs x and y, respectively. The sender has two secrets, $s_0 \in D_S$ and $s_1 \in D_S$, where D_S is a subset of $\mathbb{Z}_n$. The sender wants to send s_0 to the receiver if $x < y$ and s_1 if $x > y$, but is oblivious about which secret is sent. In short, the sender cannot learn whether $x > y$.

In the secure greater-than problem, the receiver (but not the sender) simply learns whether $x > y$. As observed by Blake and Kolesnikov, if at least one of s_0 and s_1 is known to the receiver, a solution for the secure greater-than problem follows immediately from the GT-SCOT protocol.

The GT-SCOT protocol requires a semantically secure additive homomorphic encryption scheme with large message domains. As suggested by Blake and Kolesnikov, we will use the Paillier scheme. In the protocol, the two numbers to be compared are encrypted bit by bit with the receiver's public key. The sender finds the most significant bit that is different in the two numbers without learning its position. The sender then obliviously assigns s_0 or s_1 to that bit and randomizes all other bits. Finally, the sender permutes the vector of encrypted values to prevent the receiver from learning the position of that bit and sends the vector to the receiver. The details of the protocol are as follows:

1. The receiver sets up the Paillier encryption scheme and chooses her public and private key. She randomly encrypts each bit x_i of x, where x_i denotes the i^{th} most significant bit in the n-bit binary representation of x, with her public key, R, and sends $(R, \mathcal{E}_R(x_1), ..., \mathcal{E}_R(x_n))$ to the sender.

2. The sender computes the following, where $i = 1..n$:

   (a) an encryption of the difference vector d, where $d_i = x_i - y_i$.

   (b) an encryption of the flag vector f, where $f_i = x_i \, XOR \, y_i = (x_i - y_i)^2 = x_i - 2x_iy_i + y_i$.

   (c) an encryption of vector γ, where $\gamma_0 = 0$ and $\gamma_i = 2\gamma_{i-1} + f_i$.

   (d) an encryption of vector δ, where $\delta_i = d_i + r_i(\gamma_i - 1)$, where $r_i \in_R \mathbb{Z}_n$.

   (e) a random encryption of vector μ, where $\mu_i = \frac{s_1 - s_0}{2}\delta_i + \frac{s_1 + s_0}{2}$.

   The sender sends a random permutation $\pi(\mathcal{E}_R(\mu_1), ..., \mathcal{E}_R(\mu_n))$ to the receiver.

3. The receiver obtains $\pi(\mathcal{E}_R(\mu_1), ..., \mathcal{E}_R(\mu_n))$, decrypts it, and determines the output as follows: if μ contains a single $v \in D_S$, output v, otherwise abort.

In a *random* encryption, we ensure $r \neq 1$ in the randomness part of the Paillier encryption scheme. Otherwise, we set $r = 1$, which makes the scheme much faster.

In step 2 of the protocol, the sender introduces randomness only for the last computation, before sending the result to the receiver. Blake and Kolesnikov show that for properly chosen parameter values, value v obtained by the receiver equals the desired secret with high probability, that is, the probability of a false result or an abort is negligible.

For the secure greater-than problem, we do not require the oblivious assignment in step 2(e). Instead, we observe that with high probability exactly one of the elements of vector δ computed in step 2(d) will equal 1 if $x > y$ and -1 if $x < y$. All elements will have random values if $x = y$. Therefore, we will leave away step 2(e) in our implementation, compute a *random* encryption of vector δ in step 2(d), send a permutation of it to the receiver, and modify the receiver accordingly.

For the general GT-SCOT problem, if $x = y$, all elements of vector δ in step 2(d) will be random, which fails the secret assigning operation of step 2(e). Therefore, the case of $x = y$ is no longer allowed. This can be enforced by mapping, for instance, $x \mapsto 2x$, $y \mapsto 2y + 1$. The mapping can be done entirely by the sender.

There are other protocols to solve the secure greater-than problem, for example, by Fischlin [24]. We choose Blake and Kolesnikov's protocol because it has both better communication complexity and better computation complexity in our application scenario (see Section 2.8.2). The secure greater-than problem is different from Yao's millionaire problem [58], where two millionaires want to determine which of them is richer without revealing their net worth to each other. In the millionaire problem, both parties can learn the final result, which must be avoided in our application (see Section 2.5.1). Moreover, solutions for the millionaire problem require at least six communication steps [10], whereas Blake and Kolesnikov's protocol requires only three.

2.4 System and Threat Model

In this section, we present our system and threat model.

2.4.1 System Model

We give an overview of our system in Figure 2.1. For scalability reasons, there are multiple *coverage areas*, where a coverage area corresponds to the area covered by a particular instantiation of our system (e.g., a city or a state). A coverage area is divided into a well-defined grid of equally sized, square cells. The width of a cell is chosen such that, for most cells, there is a realistic chance that multiple users can be located in the cell. For example, a cell could have a width of 100 meters. Moreover, there are four classes of parties: location brokers, users, secure comparison servers, and a directory server.

A location broker keeps track of the current location (i.e., the current cell) of a *subset* of the users in the coverage area. There are multiple location brokers,

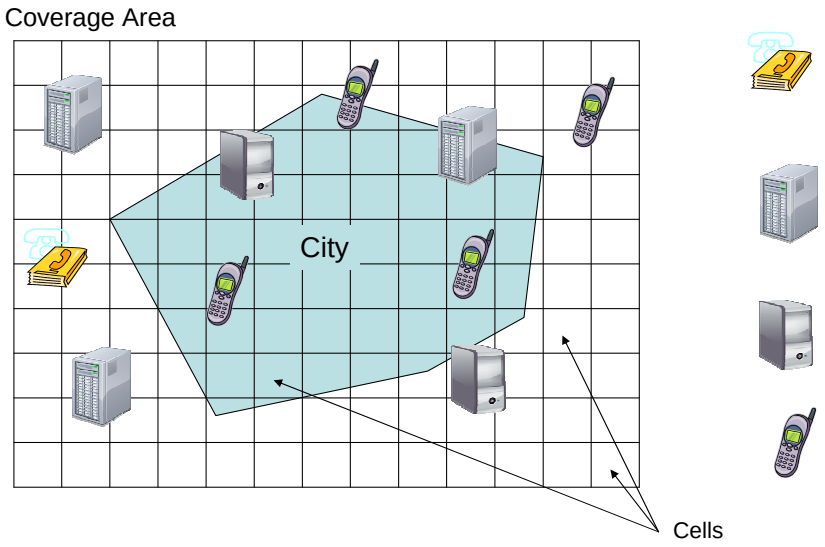

Figure 2.1: System Model. A user registers her location with a location broker, whose contact information is provided by the directory server. She can then learn whether there are at least k users in her cell by contacting all location brokers and one of the secure comparison servers.

each keeping track of the location of a *different* subset of users. Each broker is maintained by a different organization. For example, the operator of a cellphone network could maintain a location broker that keeps track of the location of the operator's customers in the coverage area. A location broker does not necessarily provide coverage for all cells in the coverage area. For example, whereas a broker maintained by a cellphone network operator would likely cover most cells, a broker operated by the provider of a WiFi network would provide coverage only for a subset of the cells.

Users carry mobile devices (e.g., a cellphone or a laptop) with them that can locate themselves (e.g., using GPS or nearby WiFi base stations) and that can access the Internet. A user registers her current location (i.e., her current cell) with a location broker of her choice. Likely, if the provider of the communication service exploited by the mobile device runs a location broker, the user will (maybe implicitly) register her location with this broker, since the provider already knows or at least has an estimate of the user's location.

A secure comparison server acts as the sender in the GT-SCOT protocol explained in Section 2.3.3 during a cloaking operation, Namely, a user interacts with all location brokers and one of the secure comparison servers to figure out whether there are at least k users who have registered the user's current cell as their location across *all* location brokers. (See Section 2.5 for the detailed protocol.) If this is not the case, the user can repeat the protocol for a superset of cells that contains her current cell, maybe in an iterative way. Each secure comparison server is maintained by a different organization. An organization can maintain both a location broker

and a secure comparison server.

The directory server publishes contact information for the location brokers and for the secure comparison servers in the coverage area. Moreover, it publishes coverage information for location brokers, that is, which broker provides coverage for which cells in the coverage area. This way, users can choose a location broker to register with and a secure comparison server to run the GT-SCOT protocol with.

2.4.2 Threat Model

In our threat model, the location brokers and the secure comparison servers are honest-but-curious, that is, they honestly follow our protocol, but are curious about learning location information. We discuss malicious brokers and servers in Section 2.7.

A location broker can learn the location and the number of users in a cell that have registered this cell as their location with this particular broker. However, a location broker should not learn the *total* number of users in a cell that have registered this cell as their location across *all* location brokers. Knowing this number might make possible tracking attacks, similar to the one presented in Section 2.1. Similarly, a location broker should not learn the location of users who register their location with any other location broker. A location broker can learn the identity of a user when she registers with the broker; this assumption is also made in the earlier work and some brokers already have this information (e.g., an operator of a cellphone network). However, a location broker should not learn the identity of a user through the user's query, if the user is not registered with the broker.

Each location broker is run by a different organization. We assume that these organizations do not collude with each other. Legal means (e.g., privacy laws or a contract between a user and a location broker) can be used to enforce this assumption. Technical enforcement means make less sense here, since today's cellphone network operators know their customers' location and identity and could potentially share this information with each other. For the same reason, we assume that location brokers do not collude with users.

A secure comparison server should neither learn a user's location nor the total number of registered users in a cell. This implies that the server should not learn the individual numbers for each location broker, either (except using back channels if a secure comparison server is run by the same organization as a location broker). Moreover, a secure comparison server should not learn a user's privacy preference, k. This preference might leak information about a user's privacy attitudes. Worse, it might allow a secure comparison server to learn the number of users in an area (see Section 2.5).

A secure comparison server might collude with other secure comparison servers to learn users' location. Due to the same reason mentioned above, we assume that secure comparison servers do not collude with location brokers (except in the

implicit case where a broker and a server are run by the same organization, here it can learn at most the location of users registered with this broker). Finally, we assume that secure comparison servers do not collude with users.

A user can learn only her own location and whether the number of people in her cell (or superset of cells) is at least k. Users have only one mobile device, which they carry with them and which faithfully reports the user's location to a broker. A broker is able to authenticate the device, which makes it possible to detect if a user tries to register with the broker multiple times. These assumptions are also made in the earlier work. We discuss malicious users in Section 2.7.

The directory server should not learn any location information about users. The server might misbehave (e.g., list a single location broker multiple times for a single cell or fail to vet location brokers or secure comparison servers (see Section 2.8.1)).

2.5 Distributed k-Anonymity Protocol

In this section, we describe how a user learns whether there are at least k users in her area. We also explain how we defend against users that try to register with multiple location brokers at the same time.

2.5.1 Distributed Greater Than

The goal of a user is to learn whether there are at least k registered users (including herself) in the user's *query area*, where k is a value chosen by the user and where the query area initially corresponds to the user's current cell. If the user learns that there are fewer than k users in this cell, she can enlarge the query area to a superset of cells that contains the user's current cell and re-execute the protocol for the enlarged area. This process can be repeated multiple times. As mentioned in Section 2.2, a user can choose between different types of enlargement algorithms, which trade off between privacy and cost. A user is expected to register her current cell as her location with one of the location brokers, but there is no need for the user to register additional cells when enlarging the query area.

To learn whether there are at least k users in a particular query area, a user first needs to identify the location brokers that provide coverage for (maybe parts of) the query area. The user must not ask the directory server for a list of brokers that provide this coverage, else the server could learn the user's location. Instead, the user should download the entire directory (or recent changes to it) from the server on a regular basis, such as once a day. The directory is signed, which prevents the directory server from misbehaving.

The user then interacts with the relevant location brokers and a secure comparison server, as illustrated in Figure 2.2. Before we discuss the protocol in detail, let

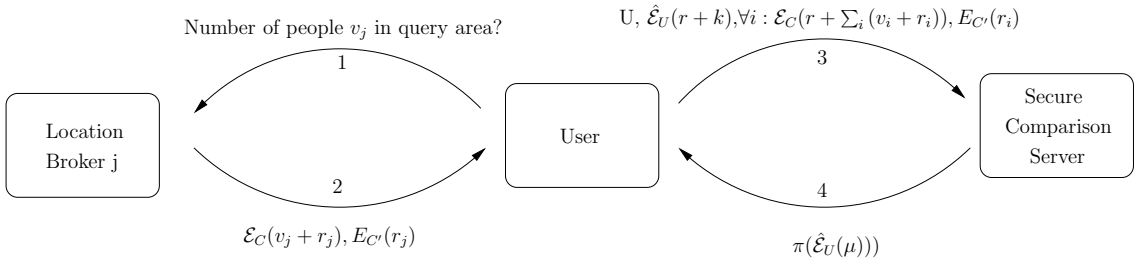

Figure 2.2: Distributed k-anonymity protocol. U and C_l are the Paillier public key of the user and the secure comparison server, respectively. C_l' is the RSA public key of the secure comparison server. $\mathcal{E}_{\mathcal{A}}(\cdot)$ denotes regular Paillier encryption with public key A. $\hat{\mathcal{E}}_A(\cdot)$ denotes *bit-by-bit* Paillier encryption with public key A. $E_A(\cdot)$ denotes RSA encryption with public key A. $\pi(\cdot)$ is a random permutation. The ciphertexts $E_{C'}(r_i)$ are also timestamped and signed (not shown).

us give an overview: The user first asks each broker covering the query area for the number of users who have registered a cell in the query area as their current location with this particular broker. A broker gives this number to the user in a covert way (i.e., the user cannot learn it). Then, the user sums up the received numbers without learning the sum and, with the help of one of the secure comparison servers, determines whether this sum is at least k.

While this protocol looks simple at first sight, there remain several challenges that need to be addressed. Let us now discuss our protocol and the challenges in detail. In the first stage, the user contacts each relevant location broker. The user and a contacted broker jointly choose a secure comparison server, l, among the set of secure comparison servers listed by the directory server. This choice needs to be consistent for a particular user across a run of our protocol (see Section 2.7.3 for details). If there are v_j users in the query area who have registered with broker j, broker j encrypts v_j with public key C_l, as published by the directory server, and sends $\mathcal{E}_{C_l}(v_j)$ to the user. The user then calculates $\mathcal{E}_{C_l}(r + \sum_i v_i)$ using the additive homomorphic property of the encryption scheme, where r is a random number generated by the user that will keep the total number of users hidden from secure comparison server l.

In the second stage, the user could simply send $\mathcal{E}_{C_l}(r + \sum_i v_i)$ and $\mathcal{E}_{C_l}(r + k)$ to secure comparison server l, which would decrypt and compare the two values and inform the user of the result. Since both the sum and the user's privacy preference, k, are obscured with r, the server can learn neither of them. However, this solution is flawed, because it might reveal the total number of users to a secure comparison server and a location broker that are run by the same organization. Assume that the location broker is the only broker that covers the query area. Here, based on the knowledge of $\sum_i v_i$ (where the sum covers only one broker), the broker and the server can jointly determine r, which allows them to compute k. In turn, once they know a user's k, the server and the broker can infer the total number of registered people in any query area chosen by the user, as long as the user's choice of k is static

and the query area is covered by the broker. The coverage condition guarantees that the broker will be contacted by the user and hence can learn the query area. Otherwise, the server and the broker could learn only the total number of people, but not for which query area. To avoid these information leaks, we need to hide the user's input to the comparison, $r+k$, from the secure comparison server. Moreover, we also need to hide the result of the comparison from the server, else the server could still infer $r + k$ in case it is found to be equal to $r + \sum_i v_i$. The GT-SCOT protocol allows us to perform the comparison in this way. Here, the user sends only $\mathcal{E}_{C_l}(r + \sum_i v_i)$ to the server. The server uses $r + \sum_i v_i$ as the sender's input, y, to the GT-SCOT protocol. The user uses $r + k$ as the receiver's input, x. The protocol guarantees that the server will not learn the result of the comparison.

The GT-SCOT protocol allows the user to distinguish between three cases: $r + \sum_i v_i < r + k$, $r + \sum_i v_i = r + k$, and $r + \sum_i v_i > r + k$. However, k-anonymity does not distinguish between the equality and the larger-than case. Moreover, telling a user that there are precisely k people in the query area enables tracking attacks, similar to the one outlined in Section 2.1. To avoid this leak, we have the secure comparison server compute and compare encryptions of $2*(r+\sum_i v_i)+1$ and $2*(r+k)$, which avoids the equality case. The computation of $2*(r+\sum_i v_i)+1$ can be done in plaintext, and the computation of the bit-by-bit encryption of $2*(r+k)$ can be done by appending an encryption of zero on the bit-by-bit encryption of $(r + k)$, i.e. left shift one bit.

A remaining flaw of the protocol is that, using binary search, a user might still be able to learn the precise number of users in a cell. Namely, the user could present $\mathcal{E}_{C_l}(r + \sum_i v_i)$ multiple times to the secure comparison server, maybe in re-randomized form or with a different value of r each time. By adjusting the value of k in each run of the GT-SCOT protocol, the user can perform a binary search for the actual value of $\sum_i v_i$. Previous research has not considered this attack. In the traditional approach with a central trusted server, this server learns the query area and the identity of a user, which could allow the server to detect the attack. In our scenario, this is more difficult for the secure comparison server since the query area remains hidden from it. To prevent this attack, we use a ticket-based solution. Instead of sending $\mathcal{E}_{C_l}(v_j)$ to the user, a location broker sends $\mathcal{E}_{C_l}(v_j + r_j)$ and a ticket that contains a signed and timestamped copy of $E_{C'_l}(r_j)$, where r_j is a random number changing with each request and $E_{C'_l}(\cdot)$ is the RSA encryption function using public key C'_l of the secure comparison server. The secure comparison server will decrypt all $E_{C'_l}(r_j)$ and subtract $\sum_i r_i$ from $r + \sum_i (v_i + r_i)$. The server also remembers tickets till their expiration date and refuses to re-use a ticket seen previously. This way, the user cannot re-use old tickets and their corresponding ciphertext, $\mathcal{E}_{C_l}(\sum_i v_i + r_i)$. Also, the user cannot use a new ticket with an old ciphertext, since the r_i value will be different, meaning the secure comparison server cannot compute the correct value for the secure greater-than operation and the user cannot learn any useful information from this operation. The validity period of a ticket is determined by a location broker. It reflects the query frequency allowed in traditional approaches based on a central trusted server.

Otherwise, a user could simply get new tickets and corresponding ciphertexts from the broker, as often as required by the binary search.

Our protocol gracefully deals with crashes of a location broker or of a secure comparison server. In the first case, the user contacts the remaining brokers, which might still report a sufficient number of registered users. To work around the second case, we can let the user and the broker choose a set of candidate secure comparison servers, instead of only a single one. The drawback of this approach is that it increases a user's chance of success in case of collusion with a secure comparison server (see Section 2.7.3). Over time, the directory server will learn of the crash of a location broker or of a secure comparison server and will remove it from the directory.

2.5.2 Defence Against Multiple Registrations

As mentioned in Section 2.4.2, consistent with earlier work, our threat model assumes that a location broker can detect attempts by a user to register with the broker multiple times in parallel. Having multiple location brokers, as it is the case in our solution, introduces a new vulnerability. Namely, a user could register multiple times, but each time with a *different* location broker. This way, other users might be wrongly told that their k-anonymity preference is satisfied. There are both technical and non-technical controls for this vulnerability. Charging money is an example of a non-technical control. Namely, if location brokers are maintained by operators of cellphone networks as a service to customers, a user would would have to buy multiple cellphones and plans to register in parallel with multiple brokers, which makes the attack expensive. In the remainder, we present a technical control that does not make any assumptions about the underlying communication technology. Since we control the vulnerability, our threat model for user behaviour can remain identical to the model from the earlier work.

There are two naïve approaches to prevent a user from registering with different location brokers concurrently. In the first one, a location broker contacts the other location brokers whenever a user registers and inquires whether the user has already registered with one of them. Apart from (potentially solvable) privacy concerns, this approach has the more fundamental problem of being expensive in terms of performance. The second approach has each broker keep records of its registered users. Periodically, the brokers compare records, ideally in a privacy-preserving way, and try to detect misbehaving users. The main problems of this approach are that it requires record keeping, which raises obvious privacy concerns, and that it is retroactive.

Our solution gets around these problems. It is based on e-cash [13] and is outlined in Figure 2.3. Initially, a user gets one (and only one) coin from the bank. To ensure that a user cannot withdraw multiple coins from the bank, a user must register with the bank with her real identity, and the bank needs to authenticate the

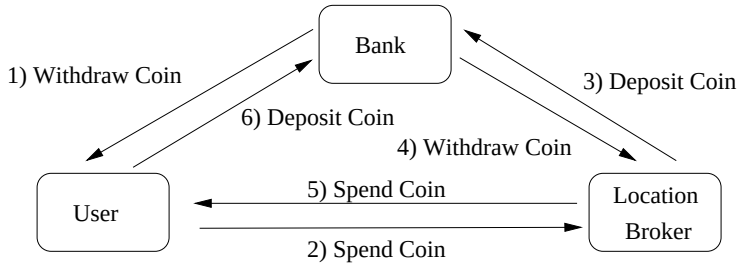

Figure 2.3: Defence against multiple registrations based on e-cash. By being given only one coin, a user can register only once. The user is returned her coin when de-registering.

user before giving the coin. The role of the bank can be assumed by the directory server or by an external party.

When a user wants to register with a location broker, she spends her coin at the broker. Since the user has only one coin, registering with another broker amounts to double spending of the coin. The other broker detects this double spending when depositing the coin, either immediately in case of an online e-cash scheme [14, 15], or in a delayed way in case of an offline scheme [12]. In the former case, the broker will deny registration. In the later case, the user will be banned from the system (see Section 2.7.3).

Whenever a location broker deposits a user's (valid) coin, the broker also withdraws a fresh coin from the bank. (Alternatively, since location brokers are not malicious, the bank can periodically give an entire set of coins to the broker for increased performance.) When the user wants to de-register, she asks the broker to spend this coin by giving it to the user. The user then deposits the coin and withdraws a fresh coin from the bank, which she can later spend at another location broker.

In case a location broker crashes, a registered user will not be able to register with a new broker. Here, we have the user contact the bank with a proof of registration issued by the broker. The bank will then issue a new coin to the user. When the broker comes back up, it will re-synchronize with the bank. If a malicious location broker refuses to refund a coin when the user de-registers, the user can also contact the bank and the directory server to revoke the location broker and get her new coin.

The benefits of our solution are that it does not require all location brokers to be contacted for a registration and that location brokers do not need to keep records of registered users after de-registration. Moreover, since e-cash is anonymous, the bank cannot learn which users register with which location broker, and a location broker cannot learn where a user has registered previously. There are no special assumptions about the underlying e-cash scheme; any scheme (e.g., Camenisch et al.'s efficient scheme [12]) can be used, which makes our solution easier to deploy.

2.6 Evaluation

In this section, we evaluate the distributed greater than protocol introduced in Section 2.5.1. We first examine the cost of contacting a location broker, followed by the cost of contacting a secure comparison server.

We implemented our protocol using the OpenSSL [49] and NTL [55] libraries. The key size for RSA and Paillier is 1024 bits. The cipher stack in TLS is AES128 in CBC mode with ephemeral Diffie-Hellman key exchange. We deploy a location broker and a secure comparison server on a 2.4 GHz Intel Xeon Dual Core running Linux 2.6.24. The user has a slow laptop (a ThinkPad T43 with a 2 GHz Intel Pentium M running Linux 2.6.22) to approximate the capabilities of a modern smartphone. Communication runs over WiFi and is protected against eavesdroppers with TLS using Diffie Hellman for forward secrecy.

2.6.1 Location Broker

We examine the performance of querying a location broker for the number of people in the query area and of adding this number to an existing (encrypted) sum. In the experiment, when a user connects to a location broker, the location broker sends back a homomorphically encrypted random value. The user then performs a homomorphic addition. We repeat the experiment ten times and report mean and standard deviation.

The overall delay experienced by the user is 39.7 ± 0.7 ms. When taking a closer look at the delay, we find that it takes 29.2 ± 0.5 ms to set up a TLS connection, which includes client and server authentication. The server takes 7.6 ± 0.0 ms to homomorphically encrypt a random value. The cost of the homomorphic addition operation by the user is negligible. In summary, setting up the TLS connection is about 3.8 times as expensive as the homomorphic encryption of a random value.

In practice, the user will likely contact multiple location brokers. Apart from the addition operation, whose cost is negligible, the brokers can be contacted in parallel. If this is not feasible for the user's device, the overall delay will be linear in the number of location brokers. We envision that this number is small (5-10 brokers) in most scenarios. This number reflects the number of cellphone and WiFi network operators providing coverage for the query area, which tends to be small. In addition, there might be a small number of independently operated location brokers (see Section 2.8.1).

The user also needs to homomorphically encrypt the random value that she will add to the (encrypted) sum of users reported by the location brokers. This encryption takes 67.2 ± 0.5 ms. As expected, the encryption operation is slower on the laptop than on the server. However, as opposed to the other operations, this encryption can occur offline. Moreover, the user can use an encrypted value multiple times for a secure comparison server.

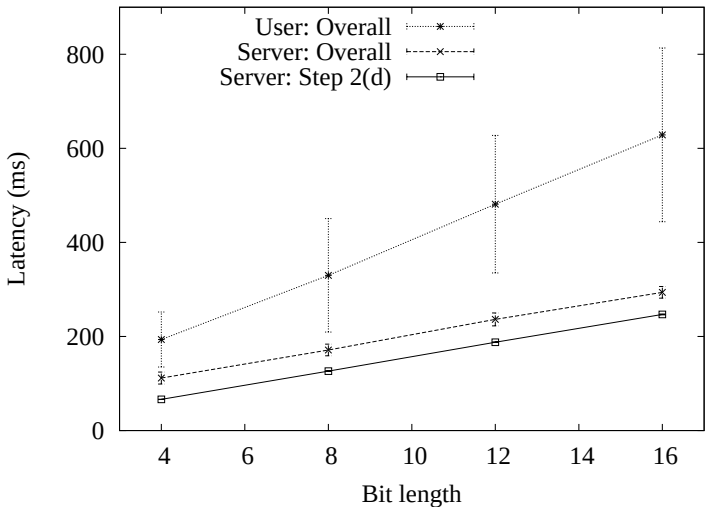

Figure 2.4: Latency experienced by the secure comparison server and the user in relationship to the bit length used in the GT-SCOT protocol. We show mean and standard deviation.

2.6.2 Secure Comparison Server

We evaluate the performance of the GT-SCOT protocol for different bit lengths of the bit-by-bit homomorphic encryption. We vary the bit length between 4 and 16 and perform fifty runs for each configuration.

We present our results in Figure 2.4. The bottom graph shows the cost of step 2(d) of the protocol. We single out this step, since it is the most expensive one, whereas the cost for steps 2(a)-(d) turns out to be negligible. (Remember that we leave away step 2(e) in our implementation and instead perform a random encryption in step 2(d).) The cost for step 2(d) varies between 66.3 ± 0.3 ms for a bit length of 4 and 247.9 ± 0.9 ms for a bit length of 16. The middle graph corresponds to the overall cost by the server. In addition to the cost of step 2(d), it also includes the cost of setting up a TLS connection, Paillier decryption of the total number of users, and steps 2(a)-(c). Finally, the top graph shows the overall latency, as experienced by the user. It varies between 193.5 ± 58.4 ms for a bit length of 4 and 628.6 ± 184.7 ms for a bit length of 16. The overall latency corresponds to the overall cost by the server plus the cost of step 3 of the GT-SCOT protocol, which takes 77.5 ± 47.7 ms for a bit length of 4 and 330.4 ± 179.6 ms for a bit length of 16. The standard deviation is large, because the user decrypts the result vector element-by-element and stops as soon as she recovers either 1 or -1.

In our implementation, we let a user choose the bit length. In practice, we expect that bit lengths between 8 and 12 will be used mainly, depending on the number of location brokers covering the query area and the maximum number of reported users for the query area (which is different from the number of registered users in the query area). If there are a bits in total, we can support up to 2^c location

brokers and up to $2^b - 1$ reported users per location broker per query area, where $a = b + c + 2$. A user informs a broker of her choice of b; if there are more than $2^b - 1$ registered users in the query area, the broker simply reports $2^b - 1$ users. The two remaining bits leave space for adding the random number, r, chosen by the user to the total number of users, which will consume at most $b+c+1$ bits, and for allowing the secure comparison server to double the resulting sum to avoid the equality case. For example, for a bit length of 8, we can support up to 8 location brokers and 7 reported users per broker per query area. Here, the overall latency experienced by the user is 330.0 ± 120.7 ms. For a bit length of 12, we can support up to 16 location brokers and 63 reported users per broker per query area. Here, the overall latency experienced by the user is 481.2 ± 146.2 ms. In short, we expect the overall latency to be noticeable, but tolerable.

The user also needs to perform a bit-by-bit encryption of the sum of her privacy preference and of her chosen random value. The cost of this operation varies between 210.4 ± 0.8 ms for a bit length of 3 and 864.4 ± 115.0 ms[1] for a bit length of 15. However, as opposed to the other operations, this encryption can be done offline. Moreover, the user can use an encrypted value multiple times for a secure comparison server.

2.7 Security Analysis

In this section, we first review how our protocol defends against the threats listed in Section 2.4.2. Our threat model assumes that location brokers and secure comparison servers are honest-but-curious, but not malicious, and that users are not malicious. In the remainder of the section, we discuss how our system can be extended to defend against malicious parties.

2.7.1 Threat Analysis

In our protocol, location brokers do not interact with other location brokers, so they cannot learn the location of users in the query area who are registered with other location brokers, not even their total number.

A secure comparison server learns only the obscured total number of users in the query area, which provides no useful information. Moreover, it cannot learn the outcome of the comparison. A server also gains no benefit from colluding with other secure comparison servers

A user learns only whether the total number of users in her query area is at least k and no other information. The tickets prevent her from learning the actual number of users with a binary search.

[1]The large variation is an artifact of using a bit length of 15. The variation is small for a bit length of 16, which has a larger mean. We are investigating this behaviour.

The directory server cannot learn any location information, because users do not retrieve individual records for their current cell from the server. The published directory is signed, which prevents the directory server from misbehaving.

2.7.2 Malicious Servers and Brokers

Assume that a malicious secure comparison server fails to correctly execute some of the steps in the GT-SCOT protocol. While it is not possible for the server to learn the total number of users, due to the randomness added by the user, the server could misbehave with the intent to give the user the impression that there are at least k registered users in an area, even if this is not the case, or vice versa.

We could address this concern by adding zero-knowledge proofs to each step of the protocol, proving that the step was executed faithfully. For proving correct application of the Paillier cryptosystem, we can exploit zero-knowledge proofs suggested by Damgård and Jurik [19]. Furthermore, Groth [31] proposes an efficient scheme for proving in zero-knowledge the correctness of a permutation of homomorphic encryptions. As it turns out, this scheme still requires three additional rounds of interactions between the prover and the verifier, which makes it expensive for mobile devices.

Therefore, in our scheme, we choose a retroactive approach. We have a secure comparison server keep a record of its actions, such as the random values used in its encryption and permutation operations. Furthermore, the server has to sign all its generated messages. If users suspect misbehaviour, they, likely in collaboration with the directory server, can force the secure comparison server to reveal the logged information and can validate the server's computations.

Similar to the secure comparison server, a malicious location broker can misbehave while executing our cryptographic protocol. In particular, a broker can encrypt a value that is different from the actual number of users registered in an area. It is possible to ask a broker to keep a record of all its actions. However, this record would include user registrations, which is problematic in terms of privacy. We prefer a less privacy-invasive approach. If users suspect misbehaviour by a location broker (and misbehaviour by a secure comparison server can be excluded, based on the above mechanism), they report the set of location brokers from which they retrieved information to the directory server. Over time, this will allow the directory server to single out a particular location broker.

2.7.3 Malicious Users

Malicious users report wrong locations to a location broker. As it turns out, a complete defence against this attack is likely impossible. A determined attacker can give her mobile device to another user or simply tamper with the location reporting mechanism on her mobile device. A user could also acquire multiple

devices, maybe under different identities, and use them to register multiple times. As mentioned in our threat model (see Section 2.4.2), these threats are not new to our system; they also arise in previous schemes. In this section, we outline some mechanisms that make these attacks harder.

A location broker might be able to detect wrongly reported locations. For example, if a broker is controlled by the operator of a WiFi network, the operator can ensure that a reported location is close to the WiFi access point from which the registration request was sent. An operator of a cellphone network can verify whether the reporting device is close to a particular cellphone tower.

We can also defend against malicious users by requiring users to sign up to our system before being able to interact with a location broker. The sign-up server, which could be identical to the directory server, can require physical identification, which reduces the danger of a user signing up multiple times. However, this approach makes the system more difficult to use. An alternative is to ask the user for a credit card number, including her name and billing address. This option becomes especially attractive if the system charges its users in the first place. Billing itself can become a mechanism for reducing misbehaviour, because an attacker might not have the necessary resources for a (large-scale) attack.

Signed-up users are given a credential. A user shows this credential to a location broker when registering. Therefore, a user does not remain anonymous to the broker she is registered with. This approach is consistent with earlier work and also reflects current business practices (e.g., the operator of a cellphone network typically knows the identity of its customers). While there are anonymous credential schemes, they would make detection of malicious users harder. Malicious users are banned from the system by having their credential revoked or by not having their expired credential renewed. However, as we indicated in the threat model, a user should remain anonymous to the brokers that she is not registered with. If a user queries a location broker other than the one she is registered with, the broker might learn the user's location (based on the query content) and identity. There are multiple ways to prevent this: First, instead of querying the other location brokers directly, the user could have the broker where it is registered with act as a proxy for sending queries. So brokers would trust each other to forward only authorized queries. Second, in addition to the real query, the user could send dummy queries to the other location brokers. Third, we could use an anonymous authentication scheme for querying the other location brokers. For example, a group signature scheme seems sufficient. Any authorized user will be able to issue a signed query. If there is misbehaviour, the client's identity can be revealed.

2.7.4 Collusion between Users and Secure Comparison Servers

Having multiple secure comparison servers distributes load and avoids a single point of failure. Our threat model assumes that users and secure comparison servers do not collude. As it turns out, if collusion did happen, having multiple secure

comparison servers would limit its impact. Collusion needs to be avoided because it would allow a user to learn the total number of registered users in her query area.

As mentioned in Section 2.5.1, we need to pick one of the secure comparison servers at the beginning of a protocol run. If there is risk of collusion, we cannot let a user choose a server. Instead, we need to choose a server such that, over time, the risk of a user working together with a colluding server is limited by k/n, where n corresponds to the number of secure comparison servers, with k of them colluding with the user. Therefore, we cannot statically assign a secure comparison server to a user, since we might be unlucky and pick a colluding one. Moreover, a user might decide not to trust servers maintained by particular organizations, and she might refrain from using our system if we forced her to use such a server all the time.

Another strategy is to have each location broker randomly choose a secure comparison server upon a user's request. However, this strategy has two flaws: First, our protocol requires that all brokers choose the same server, which will likely not be the case for this strategy. Second, if a user is assigned a non-colluding server, she can repeat her request until a colluding server is chosen. To address these flaws, we need an assignment scheme that, within a particular time frame, has all brokers assign the same server to a particular user. The length of the time frame should be chosen such that the impact of using a malicious server within the entire duration of the time frame is limited (e.g., the time frame should be shorter than a day) and such that if a user decides to perform an attack at a particular moment in time, her expected waiting time till she is being assigned a colluding server is so long that the attack environment (e.g., locations of users) will likely have significantly changed by then (e.g., the time frame should be longer than a minute).

We now present our algorithm for choosing a secure comparison server. For simplicity, we assume that there are $n = 2^m$ secure comparison servers and that there is a cryptographic hash function, $h()$, with output length m. Time is split into epochs. Epochs, in turn, are split into intervals, where each epoch has 2^m intervals. EP indicates the current epoch (starting at 0), and IV the current interval (starting at 0 for each epoch). The duration of an interval is one hour. Brokers are loosely synchronized, and they have a unique, static identifier, ID, for each user (e.g., the user's SIM card number in case of a cellphone-based query). The index of the secure comparison server for user I in epoch EP at interval IV is now computed as $h(EP||IV||ID)$. In this scheme, within an epoch, a user will be assigned different servers, but likely in a different order for each epoch. This makes it even more difficult for the user to collude with a particular server, since she will get to collude with the server at a different interval in each epoch, if at all.

In general, to minimize the risk of collusion, we do not let random people deploy a secure comparison server. Instead, the directory server should vet a server before listing it, similar to the limited vetting done by a directory server in Tor.

2.8 Discussion

In this section, we give some guidelines that help a user choose among the set of location brokers during registration. Then, we discuss the choice of the underlying secure greater than protocol for our distributed k-anonymity protocol.

2.8.1 Choice of Location Broker

From a global point of view, having multiple location brokers provides more privacy than traditional approaches based on a central trusted server, because there is no longer a single party that knows all users' location. This is an inherent benefit of our scheme and lets it integrate nicely with existing infrastructures. For example, every operator of a cellphone network knows the location of its customers. However, there is no single entity that knows the location of all cellphone users.

From a single user's point of view, our approach does not necessarily provide more privacy. In particular, if a user always registers her location with the same broker, this broker will have complete information about the user's whereabouts. In some scenarios, it actually makes sense for a user to always register with the same broker, and registering with different brokers would reduce the user's privacy. Namely, many users carry a cellphone with them that is always on. In order for the operator of the cellphone network to be able to route phone calls to the cellphone, the operator needs to know the cellphone's current location. In environments with a dense deployment of cell towers, this information is very accurate. Here, assuming the operator provides a location broker, it makes most sense for the user to register with this broker. Registering with a location broker operated by somebody else just has another party know the user's location.

However, there are scenarios where it does make sense for a user to register her location with different location brokers over time to prevent a single broker from learning her location profile. These are scenarios where cellphone towers are sparse and a cellphone can determine its fine-grained location with the help of GPS, where a cellphone network operator does not provide a location broker, or where a user takes advantage of different communication providers over time (e.g., a company's WiFi during work hours and WiFi in a Starbucks over lunch). We now discuss some user strategies for choosing a location broker.

For many users, their movement patterns are regular (e.g., there is a commute to work in the morning at always roughly the same time and a reverse commute in the evening). A user could randomly choose a location broker whenever she switches cells. However, especially if the set of candidate brokers is small, this strategy could still result in a location broker ultimately obtaining a nearly complete picture of the user's daily commute pattern. Namely, a location broker can simply piece together location information gathered during different days for the same user, based on the time when a user enters or leaves a cell.

A better strategy for a user is to assign a single location broker to each cell that is part of her daily movement patterns and to always use the assigned broker when registering in a cell. In the example, a user registers with her company's location broker during work hours and with a third-party broker during lunch (assuming Starbucks does not provide a broker). This way, a broker will not be able to establish complete location profiles for a user. The drawback of this approach is that a user needs to remember which broker is assigned to which cell. However, her mobile device can assign and remember brokers on her behalf.

When choosing a location broker, especially among brokers that do not already have location information about a user, the user must decide which of the candidates she trusts not to leak her location information. To let a user establish this trust, the directory server should vet a broker before listing it, and the users could also participate in the vetting process and report malicious location brokers if they are detected later on. (As stated in Section 2.7.4, the directory server might also vet secure comparison servers.) Moreover, the directory server should keep the number of brokers per coverage area small since each broker needs to be contacted for a query. Obvious candidates for running a location broker in a coverage area are organizations that already have location information about people in this area, such as cellphone network operators or WiFi providers. Other candidates are organizations that have an interest in the well-being or privacy of people (e.g., a municipality or the Electronic Frontier Foundation). Unlike in the case of Tor, where any individual can contribute a relay, we do not expect individuals to run location brokers due to privacy concerns.

2.8.2 Choice of Secure Greater Than

There are other protocols to solve the secure greater-than problem, for example, by Fischlin [24]. Same as Blake and Kolesnikov's protocol, it requires only two communication steps. However, instead of using the Paillier cryptoscheme, it uses the Goldwasser-Micali (GM) cryptoscheme. In the GM encryption scheme, the plaintext is one bit, either 0 or 1. The homomorphic properties of the GM scheme are different from the ones of the Paillier scheme. One can compute the exclusive-or of two encrypted bits and flip an encrypted bit. The GM scheme could also be extended to support the homomorphic "AND" property, but the size of ciphertext will be λ times larger, where λ is the new introduced error parameter, which must be sufficiently large such that $2^{-\lambda}$ is small enough.

Blake and Kolesnikov present a comparison between the two secure greater-than protocols on the computation and communication complexity. Let N be the size of the plaintext domain and n be the length of inputs x and y in binary. The communication complexity of Blake and Kolesnikov's protocol is about $\lambda/4$ times better than Fischlin's: $4n \log N$ vs $(\lambda+1)n \log N$ bits, but Blakes and Kolesnikov's protocol has higher computation cost: $16n \log N$ vs $8n\lambda$. Although it is asymptotically slower, with secure but smaller key size, such as 1024 bits and a reasonable

choice of λ, such as 40, Blake and Kolesnikov's protocol still has lower computation cost. Therefore, Blake and Kolesnikov's protocol is better in our case.

2.9 Using Privacy-Preserving Set Operations

In the previous protocol, collusion between a secure comparison server and a user allows the user to learn the number of people in an area. We could avoid this threat based on a distributed secure greater-than protocol that gets rid of the secure comparison server and instead has the location brokers jointly participate in the protocol. To the best of our knowledge, no such protocol is yet known to exist.

We now investigate an approach that is completely distributed. The approach is based on the privacy-preserving set operations proposed by Kissner and Song [39]. In particular, we could use the Over-Threshold Set-Union protocol, or the Threshold-Perfect-HBC protocol for an even higher degree of privacy. In the Threshold-Perfect-HBC protocol, all players learn which elements appear in the union of the players' private input multisets at least a threshold number t times, and nothing else. In the Over-Threshold Set-Union protocol, all player can also learn the number of times these elements appeared in the union of players' private inputs.

We give a brief overview of the Over-Threshold Set-Union and Threshold-Perfect-HBC protocols in Section 2.9.1, and then describe how to use them to construct our second distributed k-anonymity protocol in Section 2.9.2. We analyze and evaluate its performance in Section 2.9.3.

2.9.1 Brief Overview of Set-Union Protocols

We first explain how private sets are represented in the protocols. Suppose there are n players. We denote the private input set of player i as S_i, and $|S_i| = k (1 \leq i \leq n)$. Let the domain of the elements in these sets be P. Let R denote the plaintext domain of the homomorphic encryption, such as Paillier. The protocol requires that R be sufficiently large that an element drawn uniformly from R has only negligible probability of representing an element of P. This is easy to achieve because the plaintext domain is usually quite large, e.g. 2^{1024}, and the domain of set elements usually is much smaller.

In their protocols, sets are represented by polynomials. The polynomial ring $R[x]$ consists of all polynomials with coefficients from R. Let $f \in R[x]$, such that $f(x) = \sum_{i=0}^{deg(f)} f[i]x^i$, where $f[i]$ denotes the coefficient of x^i in the polynomial f. Suppose a multiset $S = \{S_j\}_{1 \leq j \leq k}$, the representation of S, $f \in R[x]$ is $f(x) = \prod_{1 \leq j \leq k}(x - S_j)$. Therefore, an element $a \in S$ if and only if $f(a) = 0$ and $a \in P$. The element a appears in the multiset b times if $(x-a)^b \mid f \wedge (x-a)^{b+1} \nmid f$.

Three basic multiset operations: union, intersection, and element reduction can be computed using the polynomial representation of sets. Let f and g be the polynomial representation of set S and T. Then:

Union $S \cup T = f * g$

Intersection $S \cap T = f * r + g * s$, where r and s are random polynomials of degree $deg(f)$ with coefficients chosen independently and uniformly from R.

Element Reduction This operation (denoted $Rd_d(S)$) is defined as follows: for each element a that appears b times in S, it appears $\max\{b-d, 0\}$ times in the resulting multiset. It can be proved that $Rd_d(S) = \sum_{j=0}^{d} f^{(j)*F_j*r_j}$ where $f^{(j)}$ is the j^{th} derivative of f, r_j's are random polynomials of degree $deg(f)$, and each F_j is any polynomial of degree j, such that $\forall_{a \in P} F(a) \neq 0 (0 \leq j \leq d)$ and $gcd(F_0, \ldots, F_d) = 1$. Random polynomials of degree $0, \ldots, d$ in $R[x]$ can satisfy this property with overwhelming probability.

The computation on polynomials above is in plaintext. We have to use a trusted third party to perform the computation and to reveal only the final result, since anyone who has the polynomial f in plaintext can solve the equation $f(x) = 0$ to get the private elements. However, with homomorphic encryption, we can implement the protocols without a third party. The computation on polynomials above only uses three operations on polynomials: addition, multiplication, and the formal derivative. All of these operations can be performed on encrypted polynomials using homomorphic encryption.

For $f \in R[x]$, the encryption of polynomial f, $E_{pk}(f)$, is the ordered list of the encryption of its coefficients under the additively homomorphic cryptosystem: $E_{pk}(f[0]), \ldots, E_{pk}(f[deg(f)])$. Then we can compute the following operations on encrypted polynomials without the knowledge of the private key:

Sum of encrypted polynomial: Let $g = f_1 + f_2$, then $E_{pk}(g[i]) = E_{pk}(f_1[i]) +_h E_{pk}(f_2[i])$ $(0 \leq i \leq max\{deg(f_1), deg(f_2)\})$

Product of an unencrypted polynomial and an encrypted polynomial: Let $g = f_1 * f_2$. If we know f_2 and the encryption of f_1, then $E_pk(g[i]) = (f_2[0] \times_h E_pk(f_1[i])) +_h (f_2[1] \times_h E_pk(f_1[i-1])) +_h \ldots +_h (f_2[i] \times_h E_pk(f_1[0]))$ $(0 \leq i \leq deg(f_1) + deg_(f_2))$.

Derivative of an encrypted polynomial: Let $g = \frac{d}{dx} f_1$, then $E_{pk}(g[i]) = (i+1) \times_h E_{pk} f_1[i+1]$ $(0 \leq i \leq deg(f_1) - 1)$.

Evaluation of an encrypted polynomial at an unencrypted point: Given the encryption of polynomial f_1, we can compute the encryption of $a = f_1(b)$ by calculating $E_pk = (b^0 \times E_{pk}(f_1[0])) +_h (b^1 \times E_{pk}(f_1[1])) +_h \ldots +_h (b^{deg(f_1)} \times E_{pk}(f_1[deg(f_1)]))$

Using encrypted polynomials to compute the basic set operations is the basic idea of Kissner and Song's protocols. The two protocols that we could use in our distributed k-anonymity protocol are the Over-Threshold Set-Union protocol and the Threshold-Perfect-HBC protocol.

The idea behind the Over-Threshold Set-Union protocol is that each player first calculates the polynomial representation of her own private set. The first player encrypts her polynomial using homomorphic encryption and sends the encrypted

polynomial to next player. The next player multiplies her polynomial to the polynomial her just received from the previous player and then sends the resulting polynomial to the following player. The next player repeats this process until the first player receives and distributes the final result. At this stage, the roots of the final encrypted polynomial are actually the union of the private sets. All players then perform a re-randomization and element reduction on the final polynomial, so only the elements that appear more than t times are still the roots of the polynomial. Finally, all players evaluate and decrypt the polynomial.

The underlying homomorphic cryptosystem of both protocols is the threshold version of Paillier (see Section 2.3.2), so all players can perform group decryption on the final result and prevent a certain number of dishonest players in the protocol. We present the details of the Over-Threshold Set-Union protocol below:

Input: There are $n \geq 2$ honest-but-curious players, $c < n$ dishonestly colluding, each with a private input set S_i, such that $|S_i| = k$ The threshold number of repetitions at which an element appears in the output is t. $F_0, \ldots, F_{t-1}$ are fixed polynomials of degree $0, \ldots, t-1$ which have no common factors or roots representing elements of P.

1. Each player $i = 1, \ldots, n$ calculates the polynomial $f_i = (x - (S_i)_1) \ldots (x - (S_i)_k)$
2. Player 1 sends the encryption of the polynomial $\lambda_1 = f_1$ to Player 2
3. Each player $i = 2, \ldots, n$
   (a) receives the encryption of the polynomial λ_{i-1} from player $i-1$
   (b) calculates the encryption of the polynomial $\lambda_i * f_i$
   (c) sends the encryption of the polynomial λ_i to player $i+1 \bmod n$
4. Player 1 distributes the encryption of the polynomial $p = \lambda_n = \prod_{i=1}^{n} f_i$ to players $2, \ldots, c+1$
5. Each player $i = 1, \ldots, c+1$
   (a) calculates the encryption of the $1, \ldots, t-1$th derivatives of p, denoted $p^{(1)}, \ldots, p^{(t-1)}$
   (b) chooses random polynomials $r_{i,0}, \ldots, r_{i,t-1} \leftarrow R^{nk}[x]$
   (c) calculates the encryption of the polynomial $\sum_{l=0}^{t-1} p^{(l)} * F_l * (\sum_{i=1}^{c+1} r_{i,l})$ and send it to all other players.
6. All players perform a group decryption to obtain the polynomial $\Phi = \sum_{l=0}^{t-1} p^(l) * F_j * (\sum_{i=0}^{c+1} r_{i,l})$.
7. Each player $i = 1, \ldots, n$ for each $j = 1, \ldots, k$

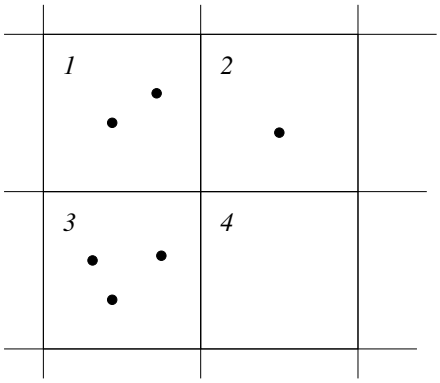

Figure 2.5: An example of location registration in four cells.

(a) choose a random element $b_{i,j} \leftarrow R$

(b) calculates $u_{i,j} = b_{i,j} \times \Phi((S_i)_j) + (S_i)_j$

8. All players distribute/shuffle the element $u_{i,j}$ $(1 \leq i \leq n, 1 \leq j \leq k)$, such that each player learns all of the elements, but does not learn their origin. Each element $a \in P$ that appears b time in the shuffled elements is an element in the threshold set that appears b times in the players' private inputs.

The idea of Threshold-Perfect-HBC protocol is basically the same, so it shares most steps (step 1 to 5) with the Over-Threshold Set-Union protocol. For more details about the Threshold-Perfect-HBC protocol, we refer readers to Kissner and Song's original paper.

2.9.2 Protocol Description

The system and threat model of the second distributed k-anonymity protocol is similar to that of the first protocol except that it does not require the secure comparison server. In this protocol, we assume that the location area is divided into cells and that each cell has a unique identifier, which is shared by all independent location brokers. If n users registered a particular cell as their current location with a location broker, the location broker includes the unique identifier of the cell n times in its private set. For example, in Figure 2.5, the identifier of a cell is in the upper-left corner of the cell, and each black dot represents one registration for the corresponding cell. In the figure, there are 2 people in cell 1, only 1 person in cell 2, 3 people in cell 3, and no people in cell 4, so the polynomial constructed by the broker is $f(x) = (x-1)^2 * (x-2) * (x-3)^3$. If a user wants to know whether there are at least k users in a cell, the location brokers jointly perform the Perfect Threshold Set-Union protocol with the threshold set to k. As a result, the servers learn which cell identifiers appear at least k times in the union of the private sets, that is, which cells contain at least k users. For better computational and communication performance, the location brokers can perform Over-Threshold Set-Union

Encryption	Homomorphic addition	Homomorphic multiplication
54.9 ± 2.6 ms	0.031 ± 0.002 ms	27.3 ± 1.3 ms

Table 2.1: Performance of Threshold Paillier 1

Share decryption	Proof generation	Proof verification
54.2 ± 2.6 ms	108 ± 5.2 ms	125 ± 5.8 ms

Table 2.2: Performance of Threshold Paillier 2

protocol instead with the cost of leaking the total number of users in a cell when the cell contains more than t users.

2.9.3 Performance

As mention before, the underlying homomorphic cryptosystem of the Over-Threshold Set-Union protocol and the Threshold-Perfect-HBC protocol is the threshold version of Paillier. Therefore, its performance is essential to the performance of the two privacy-preserving set protocols.

We implemented the threshold version of Paillier using NTL+GMP and evaluated it on a Pentium 4 3GHz desktop. We assume that all servers are honest, that is, all proofs are valid, and the threshold is the number of servers minus one. The key size we chose for the experiment is 1024 bits, and we repeat the experiment 100 times. Table 2.1 shows the time needed to perform encryption, homomorphic addition and multiplication on random plaintexts of 1024-bit length. The performance of these operations is independent of the number of servers. Table 2.2 shows the time needed to perform share decryption, proof generation and proof verification. These operations must be performed by each server in the decryption process. In Figure 2.6, we draw the performance of the combining algorithm with respect to the number of servers.

From the tables and figure above we can see that the performance of the threshold Paillier alone is acceptable. The most expensive parts are the proof generation and verification. However, in our threat model, we assume honest-but-curious players. Therefore, to save computation time, we could delay or omit the proof generation and verification.

Unfortunately, the two private set operation protocols rely too heavily on the underlying homomorphic cryptosystem to have practical performance. The most expensive part of the Over-Threshold Set-Union and Threshold-Perfect-HBC protocols is step 5 where the protocols perform element reduction operation. Moreover, all other privacy-preserving set operations in Kissner and Song's paper that require element reduction operation also suffer from this performance drawback and can only be used when quick response is not required.

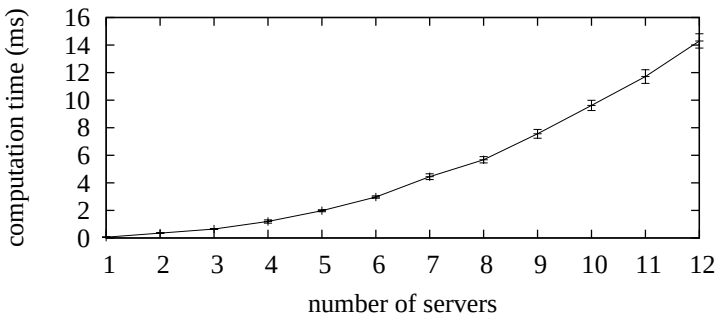

Figure 2.6: Computation time of the combine algorithm

In the element reduction operation, the encrypted polynomial and its encrypted derivatives are multiplied by random polynomials r_j of the same degree using homomorphic multiplication. The coefficient of r_j is randomly chosen from $\mathbb{Z}_n$. If the public key size of the threshold Paillier is 1024 bits, the coefficients of r_j is also 1024 bits long. The homomorphic multiplication is done by modular exponentiation. To multiply two polynomials of degree d, we need $2 * \sum_{i=1}^{d+1} i - (d+1)$, that is, $(d+1)^2$ modular exponentiation operations. If the protocol needs to perform reduction by k, where k is much smaller than d, then about $(k+1) * (d+1)^2$ modular exponentiation operations are needed in total. A single modular exponentiation operation $a^e \bmod n$ has the complexity of $O(\log e)$. In the case of element reduction e is the random coefficient, whose bit length is always in the worst case as explained earlier. In our implementation, we use 1024 bits key size. A single modular exponentiation takes about $27.3ms$. Assume a very small system with 5 location brokers, 10 users registered on each broker, and the privacy threshold is only 2. Then the encrypted polynomial has degree 50 and the reduction factor is 2, so it takes at least $213s$ to finish the element reduction part in step 5c based on approximation using the above formula and ignoring other relatively inexpensive operations. With other overhead, step 5c takes about $236s$ in our real implementation.

2.10 Conclusions and Future Work

We have presented two protocols for location privacy based on k-anonymity that require neither a single trusted server nor users to trust each other. Our sample implementation of the two protocols and their evaluation have shown that the first protocol is sufficiently fast to be practical, but the performance of the second protocol is not acceptable for its use in practice. Moreover, we have addressed several deployment concerns.

In terms of future work, we could implement and analyze the underlying e-cash scheme, and integrate our protocol into a platform for location-based services. With a real platform, we can study the protocol's usability in practice more throughly. For example, we can study how users will switch between location brokers and

answer questions like: How much human intervention is required in this task? Can a program help a user to determine how to switch between brokers intelligently without leaking privacy? This future work is important for us to build a practical and easy to use privacy-preserving location-based service.

Chapter 3

Four Protocols for Nearby-Friend Application

3.1 Introduction

The potential of location-based services, together with rising interest in social-networking applications, has led to the introduction of buddy-tracking applications. For example, Boost Mobile, a US cellphone service targeted at young people, offers the Loopt Service [45], which alerts users of nearby friends. The drawback of the Loopt Service is that it is bound to a particular cellphone network and wireless technology. MIT's iFIND project [47] works around this problem by introducing a distributed buddy-tracking application, where a person's WiFi device determines its location and shares this information with the person's friends. While it is possible to exploit this approach for alerting people of nearby friends, its disadvantage is that the friends always learn each other's location, regardless whether they are actually nearby; that is, the approach may reveal more information than desired. What we really want is a distributed buddy-tracking application where users (and their devices) can learn information about their friends' locations if and only if their friends are actually nearby. In the rest of this thesis, we call this problem the *nearby-friend problem*.

We present four protocols—Louis, Lester, Pierre and Wilfrid—for solving the nearby-friend problem. The Louis protocol requires a semi-trusted third party that does not learn any location information. The Lester protocol does not need a third party, but has the drawback that a user might be able to learn a friend's location even if the friend is in an area that is no longer considered nearby by the friend. However, this can happen only if the user is willing to invest additional work. The Pierre protocol does not have this disadvantage at the cost of not being able to tell the user the precise distance to a nearby friend. The Wilfrid protocol gives users more flexible ways to define their nearby area instead of using the threshold distance.

Our protocols can run on wireless devices with limited communication and computation capabilities. The Louis protocol requires four communication steps, whereas the Lester, Pierre and Wilfrid protocols require only two steps. Furthermore, the evaluation of our sample implementation shows that the cost of running our protocols is comparable to the cost of setting up a TLS [20] connection.

The rest of this chapter is organized as follows. In section 3.2, we discuss previous approaches to solve the nearby-friend problem. Our protocols exploit homomorphic encryption, which we review in section 3.3. We present the Louis, Lester, Pierre and Wilfrid protocols in sections 3.4, 3.5, 3.6 and 3.7, respectively, and compare their features in section 3.8. The first three of our protocols were published in the Seventh Privacy Enhancing Technologies Symposium [60].

3.2 Related Work

Location cloaking has been a popular approach for providing location privacy [16, 28, 32, 48]. Here, an individual's device or a third party cloaks the individual's location before giving it to the provider of a location-based service. Cheng et al. [16] study location cloaking for a service that alerts people of nearby friends. For each individual, the service provider knows only that the individual is within a particular region, but not where exactly. The authors develop a metric for describing the quality of an answer received from the service. This metric allows an individual to trade off privacy for better answer quality. A drawback of this approach is that the service provider learns some location information. Our protocols do not require such a third party. (In the Louis protocol, the third party does not learn any location information.) Furthermore, if a friend is nearby, our protocols will always return a positive answer and there is no doubt about the quality of the answer.

The nearby-friend problem is an instance of a secure multiparty computation problem, where multiple parties jointly compute the output of a function without learning each other's inputs. We next examine two previous approaches based on secure multiparty computation that are applicable to solving the nearby-friend problem.

Køien and Oleshchuk [42] present a secure two-party protocol for the point-inclusion problem. The protocol allows Alice to learn whether a point chosen by Bob is in a polygon determined by Alice, without Bob revealing the point to Alice and without Alice revealing the polygon to Bob. We could exploit this protocol for letting Alice know whether Bob is nearby. Namely, Alice determines the circle around her current location that corresponds to the area that she considers nearby and approximates the circle with a polygon; Bob picks the point that corresponds to his current location. However, Køien and Oleshchuk's protocol has a flaw: Alice can learn Bob's location by choosing a degenerate polygon. Assume that Bob's location is $z = (\alpha, \beta)$ and Alice's polygon is represented by $P = \{g_i(x, y) | i = 1, \ldots, n\}$ where $g_i(x, y)$ is the function for one of the edges. Location z is inside P if and only if

$g_i(\alpha, \beta) > 0$ for all $i = 1, 2, \ldots, n$. Briefly, in the protocol, Bob uses homomorphic encryption to calculate $r_i = E(g_i(\alpha, \beta))$, $e_1 = \sum r_i$ and $e_j = r_{i_1} \cdot r_j$ for $j = 2, \ldots, n$, where r_{i_1} is one of r_i's randomly picked by Bob. Bob then permutes all of the e_i's and sends them to Alice. Alice concludes that Bob is inside the polygon P if all decrypted e_i's are positive. There are different ways for Alice to choose a degenerate polygon. For example, if there are only two different edges $g_a(x, y)$ and $g_b(x, y)$ in the polygon and all the other edges are identical to $g_a(x, y)$, the decryption of the e_i's can only have three possible values: $(n-1) * g_a(\alpha, \beta) + g_b(\alpha, \beta)$, $g_a(\alpha, \beta) * g_b(\alpha, \beta)$, and $g_a(\alpha, \beta) * g_a(\alpha, \beta)$ when $r_{i_1} = E(g_a(\alpha, \beta))$, or two possible values: $(n-1) * g_a(\alpha, \beta) + g_b(\alpha, \beta)$ and $g_a(\alpha, \beta) * g_b(\alpha, \beta)$ when $r_{i_1} = E(g_b(\alpha, \beta))$. In either case, Alice can usually distinguish these values by their number of repetition in the e_i's and solve (α, β) from these values. Bob cannot detect degenerate polygons, assuming the underlying encryption scheme is semantically secure, so this protocol is not adequate for solving the nearby-friend problem.

Atallah and Du [2] also study the point-inclusion problem. Their protocol lets both Alice and Bob learn whether Bob's point is in Alice's polygon. The protocol is based on solving the secure two-party scalar product problem and the secure two-party vector dominance problem [2]. With the help of a semi-trusted third party, the first problem can be solved in three communication steps [22]. The solution of the second problem is based on solving Yao's millionaire problem [58]. The most efficient constant-round protocol for solving this problem requires six communication steps [10]. With a semi-trusted third party, the problem can be solved in three communication steps [11]. Our Louis protocol, which needs a semi-trusted third party, lets Alice know in four communication steps whether Bob is nearby and requires one additional step to inform Bob of this result. The Lester, Pierre and Wilfrid protocols each require two communication steps to let Alice learn whether Bob is nearby. To let Bob know whether Alice is nearby, these protocols also require one additional step. In summary, to achieve the same result as Atallah and Du's protocol, our protocols require fewer communication steps and the Lester, Pierre and Wilfrid protocols do not need a third party at all.

Kerschbaum [37] presented a pseudonymization technique for timestamps that is distance preserving. I.e. given two pseudonymized timestamps one can compute the distance δ, if δ is below or equal to an agreed threshold d and one cannot compute δ if $\delta \geq 2d$. Kerschbaum also extends the technique for two-dimensional spacial data. Therefore, it can be used to solve our near-by-friend problem. The protocol requires a third party, Trudy, to perform the comparison. In one dimension, the idea of the protocol is as follows: Think of timestamps as points on a scale from left to right. The scale is divided into equal-sized sections by grid points. For each of their timestamps t, Alice and Bob compute the two grid points closest to the timestamp and the distance to the two grid points. Suppose l is the lower one, u is the upper one, m is the distance to l and v is the distance. They both send the timestamp tuple $\mathbf{t} = \langle MAC(l, s), m, MAC(u, s), v \rangle$ to Trudy. Then if there are two tuples, t and t', that as a common MAC value, which means they share a grid point, then their distance is less than $2d$ and can be calculated by $m - v'$. If the two tuples do

not have a common grid point, then their distance must be larger than d. A similar construction can be applied for the two-dimensional case. Compared with our Louis protocol, which also requires a semi-trusted third party, our protocol has the advantage in precision. In the Louis protocol, the two parties can learn precisely if their distance is less than d and nothing else. However, in Kerschbaum's protocol, when the distance is between d and $2d$, the protocol sometimes can determine their distance and sometimes cannot, depending on whether their positions have a common grid point or not.

3.3 Homomorphic Encryption

As in the previous Chapter, our protocols also take advantage of additive homomorphic cryptosystem. In particular, our protocols in this Chapter use two of these systems. One is Paillier, which we already reviewed in Section 2.3.1, and the other is CGS97, which we review here.

3.3.1 CGS97

Cramer, Genarro and Schoenmakers [18] present the CGS97 scheme. This is a variant on El Gamal, where we have (public) large primes p and q such that $q|p-1$. Plaintexts are elements of $\mathbb{Z}_q$ and ciphertexts are elements of $\mathbb{Z}_p \times \mathbb{Z}_p$. Alice's private key is a random element $a \in \mathbb{Z}_q$ and her public key is $A = g^a \bmod p$.

To encrypt a message m, Bob picks a random $r \in \mathbb{Z}_q$ and computes $(c_1, c_2) = \mathcal{E}(m) = (g^r \bmod p, A^{r+m} \bmod p)$. To decrypt this message, Alice finds $A^m = c_2 \cdot c_1^{-a} \bmod p$ and computes m as the discrete log of that value with the base of A, mod p. Note that this can only be done if M, the number of possible plaintext messages, is small. In that event, the Pollard lambda, or "kangaroo", method [51] can find m in time $O(\sqrt{M})$. Note that the original CGS97 scheme is slightly different from our version. The encryption of m in original version is to compute $(c_1, c_2) = \mathcal{E}(m) = (g^r \bmod p, A^r g^m \bmod p)$ and the decryption is to solve the discrete log with the base of g. Compared with the original version, our version improves the performance by saving one exponentiation while preserving the same level of security. Also, our version prevents the adversary from pre-computing lots of powers of g for a dictionary attack.

Given $\mathcal{E}(m_1) = (g^{r_1} \bmod p, A^{r_1+m_1} \bmod p)$ and $\mathcal{E}(m_2) = (g^{r_2} \bmod p, A^{r_2+m_2} \bmod p)$, Bob can easily compute $\mathcal{E}(m_1 + m_2) = (g^{r_1+r_2} \bmod p, A^{r_1+r_2+m_1+m_2} \bmod p)$ by pointwise multiplication mod p, and given $\mathcal{E}(m_1) = (g^{r_1} \bmod p, A^{r_1+m_1} \bmod p)$ and m_2, Bob can easily compute $\mathcal{E}(m_1 \cdot m_2) = (g^{r_1 \cdot m_2} \bmod p, A^{r_1 \cdot m_2 + m_1 \cdot m_2} \bmod p)$ by pointwise exponentiation mod p.

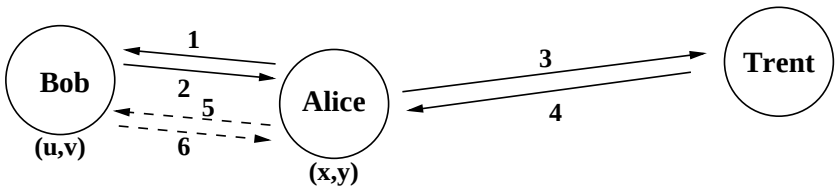

Figure 3.1: System model of the Louis protocol. The dashed arrows indicate the optional second phase.

3.4 The Louis Protocol

There are three participants in the Louis protocol: Alice, Bob and Trent. Alice and Bob are friends and Alice wants to know whether Bob is nearby. Alice considers Bob nearby if he is within a circle of some radius r centered around Alice. Alice informs Bob of r and Bob can refuse to participate in the protocol if he considers it to be too large. Trent acts as a third party and helps Alice and Bob decide whether they are nearby. Unlike other protocols for implementing location-based services that exploit third parties [16, 28, 32, 45], the Louis protocol does not allow Trent to learn any location information about either Alice or Bob.

Our protocol consists of two phases. In the first phase, Alice and Bob jointly solve the nearby-friend problem and Alice learns whether Bob is nearby. If this is the case, Alice and Bob inform each other of their locations in the (optional) second phase of the protocol. Alice and Bob cannot learn each other's locations if they are not nearby.

Alice and Bob can misbehave and input fake locations into the protocol. However, the detection of misbehaviour by one of them will likely affect their friendship, so they are less likely to misbehave. We discuss the detection of misbehaviour by Alice or Bob, and of cheating by the third party Trent in section 3.4.3.

3.4.1 Protocol Description

We assume that a location can be mapped to two-dimensional coordinates and that the mapping is known to Alice and Bob. Let Alice's location be (x, y) and Bob's be (u, v). By the definition above, they are nearby if $\sqrt{(x-u)^2 + (y-v)^2} < r$. Equivalently, we can check the sign of $d = (x-u)^2 + (y-v)^2 - r^2$. In particular, Bob is near Alice if $d < 0$.

Figure 3.1 presents the two communication channels used in our system model. The first is between Alice and Bob, and the second is between Alice and Trent. Alice also acts as a relay of the communication between Bob and Trent. The benefit of this approach is to hide Bob's identity from Trent, thus improving privacy. We assume that the two secure communication channels are set up before our protocol begins.

The protocol consists of two phases. The first phase lets Alice determine whether Bob is nearby. If this is the case, the (optional) second phase lets Alice and Bob learn each other's locations. In our protocol, $\mathcal{E}_A(\cdot)$ is the Paillier additive homomorphic encryption function using Alice's public key, $\mathcal{E}_T(\cdot)$ is a (non-homomorphic) public-key encryption function using Trent's public key, $\mathcal{H}(\cdot)$ is a cryptographic hash function, $sig_A(m)$ is Alice's signature on message m, and similarly with $sig_T(m)$.

1. **First phase:** Alice determines her location (x, y) and her desired radius r, and picks a random salt s_A.

   Alice→Bob: $\mathcal{E}_A(x^2 + y^2)$, $\mathcal{E}_A(2x)$, $\mathcal{E}_A(2y)$, r, $\mathcal{H}(x \parallel y \parallel s_A)$

2. Bob checks the value of r. If he thinks r is too large, he aborts the protocol. Otherwise, he determines his location (u, v), picks a random value k and computes

   $$\mathcal{E}_A(d + k) = \frac{\mathcal{E}_A(x^2 + y^2) \cdot \mathcal{E}_A(u^2 + v^2) \cdot \mathcal{E}_A(k)}{(\mathcal{E}_A(2x))^u \cdot (\mathcal{E}_A(2y))^v \cdot \mathcal{E}_A(r^2)},$$

   Bob also chooses a random salt s_B.

   Bob→Alice: $\mathcal{E}_A(d + k)$, $\mathcal{E}_T(k)$, $\mathcal{H}(u \parallel v \parallel s_B)$, $\mathcal{H}(k)$.

3. Alice decrypts $\mathcal{E}_A(d + k)$.

   Alice→Trent: $d + k$, $\mathcal{E}_T(k)$, $sig_A(d + k)$, $sig_A(\mathcal{E}_T(k))$

4. Trent decrypts $\mathcal{E}_T(k)$ and verifies Alice's signatures. Next, he computes d. If $d < 0$, Trent sets $answer = \text{'YES'}$ else $answer = \text{'NO'}$.

   Trent→Alice: $answer$, $sig_T(answer \parallel sig_A(d + k) \parallel sig_A(\mathcal{E}_T(k)))$.

5. Alice verifies Trent's signature. Next, if $answer == \text{'YES'}$, she knows that Bob is nearby. Alice terminates the protocol if Bob is not nearby or if only the first phase of the protocol is run. Otherwise:

   Second phase: Alice reveals her location to Bob:

   Alice→Bob: $answer$, $d+k$, $sig_A(d+k)$, $sig_A(\mathcal{E}_T(k))$, $sig_T(answer \parallel sig_A(d+k) \parallel sig_A(\mathcal{E}_T(k)))$, x, y, s_A.

6. Bob verifies all signatures. He then computes $\mathcal{H}(x \parallel y \parallel s_A)$ and compares the hash value with the one provided by Alice in step 1. He also uses (x, y) to compute $d + k$ and compares it to the value received. If the values do not match, Bob aborts the protocol. Otherwise Bob reveals his location to Alice:

   Bob→Alice: u, v, s_B, k.

7. Alice computes $\mathcal{H}(u \parallel v \parallel s_B)$ and $\mathcal{H}(k)$ and compares the values with the hash values provided by Bob in step 2. Alice also computes $d + k$ based on (x, y), (u, v), and k and verifies whether it equals the decrypted value of $\mathcal{E}_A(d + k)$.

	Alice	Bob	Trent
TLS connection time	516 ± 2 ms	255 ± 4 ms	256 ± 2 ms
Computation time	635 ± 4 ms	175 ± 4 ms	41 ± 0.6 ms

Table 3.1: Runtime of the Louis protocol.

Note that our protocol checks whether $d < 0$. In the Paillier cryptosystem, d will be an element of $\mathbb{Z}_n$, so to check this condition, we ensure that n is sufficiently large, and we say $d < 0$ if $n/2 < d < n$.

3.4.2 Measurements

We implemented our protocols using the OpenSSL [49] and NTL [55] libraries. We chose RSA for the non-homomorphic encryption and signature functions. The key sizes of all the cryptographic functions are 2048 bits. Our hash function is SHA-256, and the cipher stack in TLS is AES256 in CBC mode with ephemeral Diffie-Hellman key exchange. (The ephemeral keys can be used in the Lester, Pierre and Wilfrid protocols, below.) We evaluated these protocols on a 3.0 GHz Pentium 4 desktop. We ran the protocol one hundred times and measured TLS connection-setup time and overall computation time for each protocol participant. Table 3.1 shows our results.

With 2048-bit keys, it takes about a quarter second to set up a TLS connection. Alice initiates two TLS connections, which takes about half a second. Trent's computation time is very small. The major burden is on Alice, who takes about 0.6 s; Bob's computation time is less than one third of Alice's. In short, if a mobile device can set up a TLS connection, it should be able to finish the Louis protocol in comparable time or shorter.

3.4.3 Analysis

The Louis protocol can directly detect scenarios where Alice and Bob reveal other locations than the ones they committed to. We next explain how Alice and Bob can discover other kinds of misbehaviour.

Alice detects misbehaviour by Bob or Trent. If Alice detects suspicious behaviour, such as not spotting nearby Bob though she was told that he is nearby, and if only the first phase of the protocol has been run, Alice asks Bob to execute the second phase. If Bob refuses, Alice will suspect that Bob misbehaved. Otherwise, Alice proceeds as follows:

If Alice is told by Trent that Bob is nearby, but then fails to spot Bob at his released, nearby location, Alice will realize Bob's misbehaviour. If the released location is not nearby, Alice asks Bob to reveal the random values that he used in

his calculations and repeats the calculations. If the results are not identical to the ones released by Bob, Bob must have misbehaved. Otherwise, there was cheating by Trent.

If Alice is told by Trent that Bob is not nearby, but then spots him in her vicinity, she proceeds in a similar way. Namely, if the location released by Bob is not nearby, Bob must have misbehaved. If it is nearby, Alice repeats Bob's calculations, as explained above, to detect cheating by Trent.

Finally, if step 7 in the protocol fails, Alice also repeats Bob's calculations to discover who misbehaved.

Bob detects misbehaviour by Alice or Trent. If the second phase of the protocol is not run, Bob does not learn any location information about Alice, which makes it impossible for him to detect misbehaviour. However, Bob can refuse to answer multiple queries from Alice if they arrive within a very short time. These queries could be part of a probing attack, where Alice knows a set of likely locations for Bob and uses each of them for invoking the protocol.

If the second phase of the protocol is run and Bob detects suspicious behaviour, Bob uses mechanisms similar to Alice's to discover misbehaviour.

Alice or Bob collude with Trent. Our protocol cannot detect collusion, where Trent tells the value of d to one of the parties. However, Alice and Bob can jointly choose the third party, which reduces the risk of collusion.

3.5 The Lester Protocol

The Louis protocol allows Alice and Bob to learn each other's locations if and only if they are nearby, but it requires the participation of Trent. In our second protocol, Lester, we do away with the need for Trent. However, this comes at some small costs. First, the information disclosure is now only one-way; that is, Alice learns about Bob's location, but not vice versa. Alice and Bob could of course run the protocol a second time, with the roles reversed, to mutually exchange information. (Note that this requires only one extra message, since the resulting two messages from Bob to Alice can be combined.) Second, Alice learns less exact information about Bob; she only learns the distance between them, although this may actually be a benefit, depending on the context.

3.5.1 Protocol Description

This protocol uses the CGS97 cryptosystem of section 3.3.1. Recall that this cryptosystem has an unusual property: the amount of work Alice must do in order to decrypt a message depends on the number of possible messages. We use this property to our advantage in this protocol.

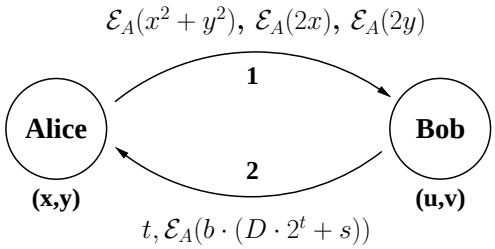

Figure 3.2: An overview of the Lester protocol. t is the *workfactor* chosen by Bob and s is the random salt of length t. $D = (x-u)^2 + (y-v)^2$ is the square of the distance between Alice and Bob.

The Lester protocol is very simple, and its overview is presented in Figure 3.2. Let a and b be Alice and Bob's private keys, and $A = g^a$ and $B = g^b$ be their public keys. Note that these keys may be ephemeral; if Alice and Bob are communicating via TLS [20], for example, they can use the key pairs from an ephemeral Diffie-Hellman key exchange. Alice and Bob can each calculate $C = A^b = B^a$. Alice sends Bob $\mathcal{E}_A(x^2+y^2)$, $\mathcal{E}_A(2x)$, $\mathcal{E}_A(2y)$. Bob picks a *workfactor* t (see below) and a random salt s of length t, and sends to Alice $t, \mathcal{E}_A(b \cdot (D \cdot 2^t + s))$, where $D = (x-u)^2 + (y-v)^2$ is the square of the distance between Alice and Bob. Alice receives this message, and can calculate $A^{b \cdot (D \cdot 2^t + s)} = C^{D \cdot 2^t + s}$.

If Alice wants to learn whether Bob is closer than some threshold distance r away, she uses the kangaroo method [51] to determine $D \cdot 2^t + s$ if it is in the range $[0, r^2 \cdot 2^t]$. This can be done in time $O(r \cdot 2^{t/2})$ and space $O(t \log r)$. Other methods to calculate discrete logarithms, such as Baby-Step-Giant-Step [54], can solve this problem in the same order of time with better constants, but with exponentially larger space requirements. Considering space constraint is crucial on mobile devices, the kangaroo method is the better choice here, and it can be easily parallelized to take advantage of multicore architectures. If this discrete logarithm step is successful, shifting the result by t bits yields D. The effect of Bob including a factor of b in his response to Alice is that Alice's discrete logarithm calculation is to the base of the ephemeral C rather than A. This prevents Alice from doing a certain amount of reusable precomputation derived from a predetermined base.

Bob should choose t so that he is comfortable with the amount of work Alice would have to do in order to discover the distance between them. This will likely depend on things Bob knows about his friend Alice, such as the computational capacity of Alice's cellphone.

3.5.2 Measurements

The runtime of the Lester protocol is dominated by Alice's computation of the discrete log of $C^{D \cdot 2^t + s}$ to the base of C. In Figure 3.3, we plot this time against the workfactor value t, chosen by Bob. For fixed r, we expect this runtime to scale as $2^{t/2}$ and the log plot shows that this is indeed the case. This gives Bob a fair

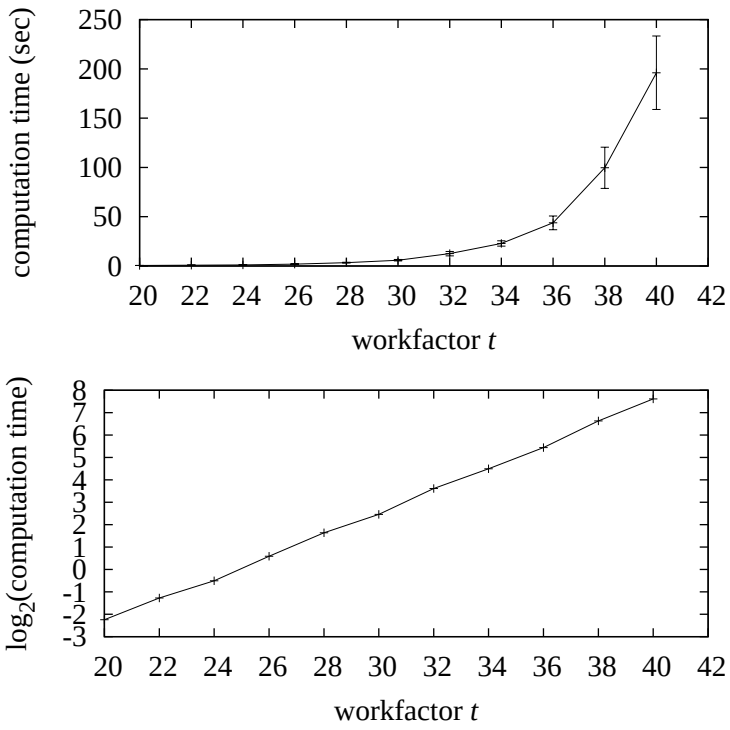

Figure 3.3: Alice's computation time in the Lester protocol.

amount of control over the amount of work Alice will need to do to find the distance between them: in our setup, if $t = 20$, Alice needs only about a quarter of a second, and if $t = 40$, Alice needs a few minutes of computation time. If this is not enough, Bob could choose even larger values, and the exponential nature of the runtime means he can make Alice work a very long time with only a small increase in t.

We measured Bob's computation time, on the other hand, to be 175 ± 2 ms, comparable to that of the Louis protocol, and this value is independent of t.

3.5.3 Analysis

This protocol has no way to detect if Alice or Bob use incorrect locations as their input. This could allow Alice to confirm a guess of Bob's location simply by entering that guess as her own location and seeing if the protocol successfully finds Bob to be very nearby. Alice could also check specific ranges of large values of D. For example, if locations are measured in metres, she could check whether Bob is between 10000 and 11000 m away for about the same cost as checking whether he is between 0 and 4600 m away. Of course, the former ring represents a much more widely spread out geographical area, and knowing only that Bob is in that ring probably gives less useful information to Alice. An exception is when Alice knows a few places that Bob is likely to be: his home, his work, etc.; she can then confirm those guesses with minor difficulty. Note that Bob has a little bit of extra power: not only can

he choose a large t if he suspects Alice is probing for his exact location, but he can also effectively refuse to participate in the protocol, without letting Alice know. He does this by returning an *unconditional negative*; that is, an encryption of a random value instead of the correct response. This makes it extremely probable that Alice's discrete log computation will fail. If Bob wants to be extra careful, he should be sure to avoid revealing he has done this to side channels, such as timing differences [40]. Conversely, he could return an *unconditional positive* by returning an encryption of a small number rather than the result of his calculation. If Alice cares, she can prevent the latter by adding a random value k to her $x^2 + y^2$ and dividing Bob's response by $C^{k \cdot 2^t}$. Of course, as in the Louis protocol, Alice is likely to notice if Bob claims to be nearby but is not.

Another downside of this protocol is that Bob only has very coarse control over the threshold distance; he can choose how much work Alice would have to do in order to discover that he was, say, 500 metres away, but with only twice as much work, Alice could discover that Bob was 1000 metres away. A minor modification to the Lester protocol, however, can make Alice's work be quadratic in the threshold distance instead of linear. Instead of the CGS97 cryptosystem, the Boneh-Goh-Nissim cryptosystem [9] can be used. This protocol has the same properties (additive homomorphic; decryption takes $O(\sqrt{M})$ time) as CGS97, but also allows calculations of encryptions of *quadratic* functions, in addition to linear ones. With this system, Bob could compute $\mathcal{E}_A(D^2 \cdot 2^t + s)$ for a random salt s between 0 and $(2D + 1)2^t - 1$, and Alice's work to find the distance to Bob will be $O(r^2 \cdot 2^{t/2})$. However, the level of privacy that the Lester protocol provides can be affected by the computing power of devices and the actual implementation. For example, a user could use a faster computer and faster implementation to solve discrete logarithms. Therefore, we do not really suggest its use in practice if this problem cannot be controlled.

3.6 The Pierre Protocol

Our third protocol, Pierre, solves the above problems with the Lester protocol and gives Bob more confidence in his privacy. On the other hand, if Alice and Bob are nearby, the Pierre protocol will inform Alice of that fact, but will give her much less information about Bob's exact location.

3.6.1 Protocol Description

In this protocol, Alice picks a *resolution distance* r, roughly analogous to the threshold distance r in the previous protocols. Alice and Bob then express their coordinates in (integral) units of r; that is, if Alice's true position is (x, y), then for the purposes of this protocol, she will use coordinates $(x_r, y_r) = (\lfloor \frac{x}{r} \rfloor, \lfloor \frac{y}{r} \rfloor)$, and similarly for Bob. This has the effect of dividing the plane into a grid, and Alice and Bob's location calculations only depend on the grid cells they are in; see Figure 3.4.

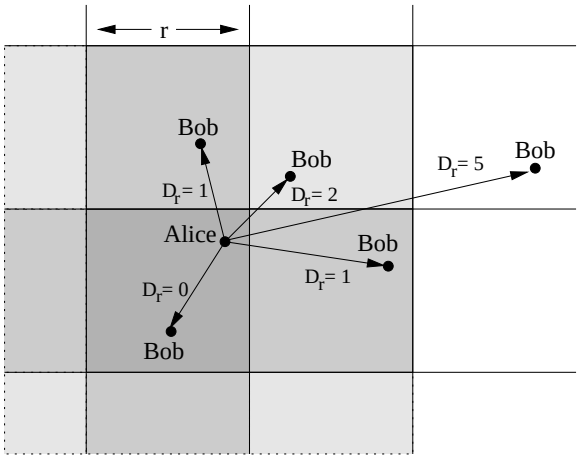

Figure 3.4: Grid distances in the Pierre protocol. The x and y distances between Alice and Bob are measured in grid cells (integral units of r), and $D_r = (\Delta x_r)^2 + (\Delta y_r)^2$. Alice can determine whether Bob is in the dark grey, medium grey, or light grey area, but no more specific information than that.

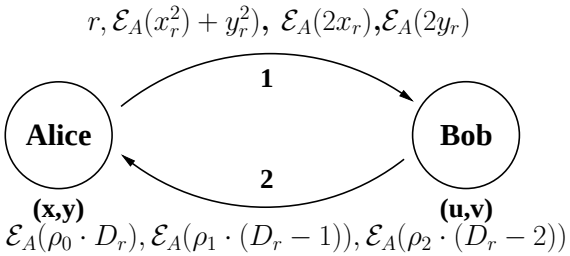

Figure 3.5: An overview of the Pierre protocol. ρ_0, ρ_1 and ρ_2 are three random elements picked by Bob in $\mathbb{Z}_p^*$ and $D_r = (\Delta x_r)^2 + (\Delta y_r)^2$.

This protocol can use either of the homomorphic cryptosystems we have mentioned. It turns out that CGS97 is slightly more efficient, so we will use the notation of that system. As with the Lester protocol, Alice and Bob's public keys can be the ephemeral ones generated during TLS setup.

An overview of Pierre protocol is presented in Figure 3.5. Alice first sends to Bob $r, \mathcal{E}_A(x_r^2 + y_r^2)$, $\mathcal{E}_A(2x_r)$, $\mathcal{E}_A(2y_r)$. Bob then picks three random elements ρ_0, ρ_1, ρ_2 of $\mathbb{Z}_p^*$ and replies with $\mathcal{E}_A(\rho_0 \cdot D_r), \mathcal{E}_A(\rho_1 \cdot (D_r - 1)), \mathcal{E}_A(\rho_2 \cdot (D_r - 2))$, where $D_r = (x_r - u_r)^2 + (y_r - v_r)^2$ is the square of the distance between Alice and Bob, in integral units of r. As in the Lester protocol, if Bob is uncomfortable with Alice's query, either because of her choice of r, her frequency of querying, or some other reason, Bob can reply with encryptions of three random values, ensuring Alice will not think he is nearby.

Note that $\rho_0 \cdot D_r = 0$ if Alice and Bob are in the same grid cell and is a random element of $\mathbb{Z}_p^*$ otherwise. Similarly, $\rho_1 \cdot (D_r - 1) = 0$ if Alice and Bob are in adjacent grid cells and random otherwise, and $\rho_2 \cdot (D_r - 2) = 0$ if Alice and Bob are in diagonally touching grid cells and random otherwise.

	Alice	Bob
TLS connection time	256 ± 3 ms	257 ± 1 ms
Computation time	384 ± 4 ms	354 ± 3 ms

Table 3.2: Runtime of the Pierre protocol.

In CGS97, it is easy for Alice to check whether a received ciphertext (c_1, c_2) is an encryption of 0: this is the case exactly when $c_2 = {c_1}^a \bmod p$, where a is Alice's private key. Therefore, with this protocol, Alice can tell when Bob is in the same, adjacent, or diagonally touching grid cell (and learns which is the case), but she learns no more specific information than that.

3.6.2 Measurements

We measured the computation time of the Pierre protocol using 2048-bit keys for both TLS and CGS97; the results are shown in Table 3.2. For comparison, we also show the time to set up an TLS connection between Alice and Bob. The computation times shown are for the worst-case situation; that is, Alice and Bob are not nearby.

We can see that the computational cost of the Pierre protocol is only slightly more expensive than setting up TLS; this suggests that the protocol would be reasonable to run on mobile devices.

3.6.3 Analysis

As with the other protocols, we cannot prevent Alice from using an incorrect location in order to try to confirm a guess of Bob's location. However, in the Lester protocol, as mentioned above, Alice can try to verify a number of guesses with a single query to Bob. This is not the case in the Pierre protocol; each protocol run tells Alice only whether Bob is near the location she entered, and she can extract no other information from Bob's reply.

Like the Lester protocol, the Pierre protocol can gain a minor benefit from using the Boneh-Goh-Nissim cryptosystem. Bob can combine two of his responses and reply with, for example, $\mathcal{E}_A(\rho_1 \cdot D_r \cdot (D_r - 1)), \mathcal{E}_A(\rho_2 \cdot (D_r - 2))$. If the first ciphertext decrypts to 0, then Alice knows that D_r is either 0 or 1, but not which. This gains Bob a small amount of privacy, and at the same time slightly decreases the size of his reply, even taking into account that Boneh-Goh-Nissim is elliptic curve based.

A more dramatic benefit could be gained by using a *ring homomorphic* encryption system; that is, a system in which, given $\mathcal{E}(x)$ and $\mathcal{E}(y)$, one can efficiently compute both $\mathcal{E}(x + y)$ and $\mathcal{E}(x \cdot y)$. With such a system, Bob could reply with the single ciphertext $\mathcal{E}_A(\rho \cdot D_r \cdot (D_r - 1) \cdot (D_r - 2))$. Bob could also include more

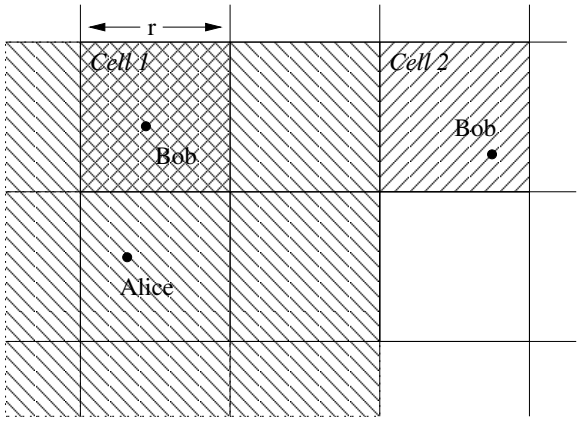

Figure 3.6: An example of cells being defined by coordinates. Alice's input is her nearby nine cells, and Bob's input is *Cell 1* or *Cell 2*.

factors of $(D_r - i)$ inside the encryption while reducing r and be able to more accurately approximate a circle around Alice by using more grid cells of a smaller size. Unfortunately, no secure ring homomorphic cryptosystem is yet known to exist.

3.7 The Wilfrid Protocol

In our fourth protocol, Wilfrid, instead of setting a threshold distance r, Alice and Bob can independently define which cell(s) of area that they consider as nearby. They are nearby if and only if their input cell(s) intersect. The protocol will inform Alice of the intersecting cell(s).

3.7.1 Protocol Description

In this protocol, we assume that the location area is divided into cells. Cells can be defined by the following ways:

- **Coordinates**: cells are defined in the same way as in the Pierre protocol. Users use coordinates $(x_r, y_r) = (\lfloor \frac{x}{r} \rfloor, \lfloor \frac{y}{r} \rfloor)$, where r is the *resolution distance*. In this way, cells are equally sized square. A user could use multiple nearby cells as her input set. For example, in Figure 3.6, Alice chooses the nearby nine cells as her input, and Bob uses just one cell as his input. If Bob's input is *cell 1*, there is an intersection, so they are considered as nearby. If Bob's input is *cell 2*, there is no intersection, then they are not nearby.
- **Description**: cells are uniquely defined by their description. For example, we can use the string "Waterloo–University of Waterloo–Davis Centre" to define the area where Davis Centre is. The description is shared by all users of the protocol.

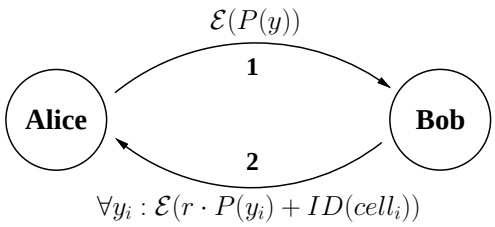

Figure 3.7: An overview of the Wilfrid protocol.

- **Granularity**: cells have different levels of granularity, so a user could use multiple cells of different levels of granularity to form her input set. For example, "Waterloo–University of Waterloo–Davis Centre", "Waterloo–University of Waterloo" and "Canada–Ontario–Waterloo" are three cells at different level of granularity. They could form Alice's input set. If Bob's input set is "Waterloo–University of Waterloo–Mathematics Centre", "Waterloo–University of Waterloo" and "Canada–Ontario–Waterloo", then Alice will learn that Bob is in the University of Waterloo, Ontario, but she cannot learn Bob is in which building, except that Bob is not in Davis Centre.

Using which way to define cells could depend on the specific application and users' preference. By one of the above ways, each cell is given a unique identifier, which composes the input to the protocol.

The protocol uses the idea of the private matching protocol proposed by Freedman et al. [27], and it is actually a private set intersection protocol, which is similar to the private set intersection protocol proposed by Kissner and Song [39]. (We presented Kissner and Song's set union protocol in Section 2.9.) We choose Freedman's protocol for its better performance in our setting. Just like Pierre, this protocol can use also use either of the homomorphic cryptosystems we have mentioned, and we choose CGS97 for its higher efficiency.

As shown in Figure 3.7, the system model and the number of communication steps of the Wilfrid protocol are the same as in the Lester and Pierre protocols. The protocol works as follows:

1. Alice defines a polynomial P whose roots are hash values of the k_A identifiers of the cells that she considers as nearby, i.e. given $x_i \leftarrow \mathcal{H}(ID(cell_i^{Alice}))$ Alice computes:

$$P(y) = (x_1 - y)(x_2 - y)\dots(x_{k_A} - y) = \sum_{u=0}^{k_A} a_u y^u$$

   Alice encrypts these k_A coefficients under her public key and sends the resulting ciphertext to Bob.

2. Bob uses the homomorphic properties of the encryption system to evaluate the polynomial on each hash value of the k_B identifiers of the cells that he considers as nearby, i.e. $\forall y_i \leftarrow \mathcal{H}(ID(cell_i^{Bob}))$ Bob computes:

$$\mathcal{E}(P(y_i)) = \mathcal{E}(a_0)(\mathcal{E}(a_1)(\ldots \mathcal{E}(a_{k_B})^{y_i})^{y_i})^{y_i}$$

Bob then multiplies each $P(y_i)$ result by a fresh random number r and adds it to the corresponding identifier of the cell that he considered as nearby:

$$\mathcal{E}(r \cdot P(y_i) + ID(cell_i))$$

Bob then returns this set to Alice.

3. Alice decrypts each element of this set. She can recover the identifier of the cell only if $P(y_i) = 0$. Otherwise, the resulting decryption appears random. Alice checks if the identifier is valid and whether it is one of the identifiers of the cells that she considers as nearby.

When using CGS97 as the underlying homomorphic cryptoscheme, we need to use it in combination with the standard ElGamal to avoid solving a discrete logarithm in decryption. Specifically, we need to use CGS97 in all other steps except when calculating $\mathcal{E}(r \cdot P(y_i) + ID(cell_i))$ in Step 2. Suppose $P(y_i) = 0$, after calculating the intermediate result of $\mathcal{E}(r \cdot P(y_i))$ we have an encryption of zero in the form of $\langle g^r, A^{r+0} \rangle = \langle g^r, A^r \rangle$. Then when adding $m = ID(cell_i)$ to the encryption, instead of using CGS97 to calculate $\langle g^r, A^{r+m} \rangle$, we use standard ElGamal to calculate $\langle g^r, A^r m \rangle$. As a result, we can recover m directly using standard ElGamal decryption, which avoids solving m in A^m.

3.7.2 Measurements

The key size we use for both TLS and CGS97 is 2048 bits, which is the same as in the implementation of the previous protocols. However, we only use the higher 64 bits of SHA-1 as the hash value of an identifier, since a 64-bit long identifier is enough to represent two GPS coordinates with accuracy in the order of meters, that is, the number of possible identifiers is less than 2^{64}. Therefore, there is no need to use longer hash values, and the shorter hash can improve the performance of step 2. In our implementation, when both Alice and Bob's input size is 5, the computation time of step 2 is reduced from 636ms to 529ms, and it can be reduced even more when the input size is larger. In step 3, Alice does not need to know $\mathcal{E}(P(y_i))$ for checking whether $P(y_i) = 0$. We limit the length of an identifier to be fewer than 64 ASCII characters, then if the length of the plaintext of $\mathcal{E}(r \cdot P(y_i) + ID(cell_i))$ is fewer than 64 bytes, with overwhelming probability $P(y_i) = 0$, and Alice gets $ID(cell_i))$.

Freedman et al. [26] also suggest a performance optimization to achieve a more significant reduction of overhead by allowing Alice to use multiple low-degree polynomials and then allocating input values to polynomials by hashing. However, in our case, the input size is usually small, so this optimization is not implemented.

Alice's input size	1	2	3	4	5
Alice's Comp. time (ms)	180 ± 2	270 ± 2	360 ± 3	450 ± 2	541 ± 3
Alice's input size	6	7	8	9	10
Alice's Comp. time (ms)	631 ± 3	722 ± 4	812 ± 4	903 ± 4	992 ± 4

Table 3.3: Alice's computation time in step 1

Bob's input size	1	2	3	4	5
Alice's Comp. time (ms)	45 ± 1	90 ± 1	136 ± 1	181 ± 2	226 ± 2
Bob's input size	6	7	8	9	10
Alice's Comp. time (ms)	271 ± 2	316 ± 2	362 ± 2	407 ± 3	452 ± 3

Table 3.4: Alice's computation time in step 3

Since the TLS connection time of the Wilfrid protocol is the same as in the Pierre protocol, our measurements focus on the computation time of Alice and Bob. Alice's computation time in step 1 only depends on the size of Alice's input, which is shown in Table 3.3, and Alice's computation time in step 3 only depends on the size of Bob's input, which is shown in Table 3.4. Bob's computation time in step 2 depends on the input size of both Alice and Bob. Figure 3.8 shows Bob's computation time for every input size from one to ten.

From the measurement results, the Wilfrid protocol is efficient for small input size, which is practical since the number of nearby cells usually is also small. When the input size is one for both parties, the computation time is only 225ms for Alice and 93ms for Bob. For a fair comparison with the Pierre protocol, we assume that, in the Wilfrid protocol, cells are defined by their coordinates, and Alice's input is her nearby nine cells and Bob's input is the one cell that he is in. The case is just the same as in Figure 3.6. (We avoid Bob also using the nearby nine cells, in which case Alice might be able to infer Bob's current cell from the intersecting cell(s) even if Bob is not in one of the nine cells that Alice considers as nearby.) In this setup, Alice's computation time is worse in the Wilfrid protocol, 948ms vs. 384ms, but Bob's computation time is better, 119ms vs. 354ms. Although the overall performance of the Wilfrid protocol is worse than the Pierre protocol, Alice can have more flexible ways to define her nearby area in the Wilfrid protocol.

3.7.3 Analysis

As in previous protocols, Alice can guess Bob's location by using an incorrect location. In the Wilfrid protocol, Alice can try to verify a number of guesses with a single query to Bob. However, the more number of guesses for one query, the higher the degree of the polynomial to be sent to Bob. For privacy and performance reasons, Bob could limit the degree of the polynomial that he is willing to accept. If he thinks that the degree of the polynomial is too high, he can effectively refuse to participate in the protocol by returning an *unconditional negative*, as we suggested

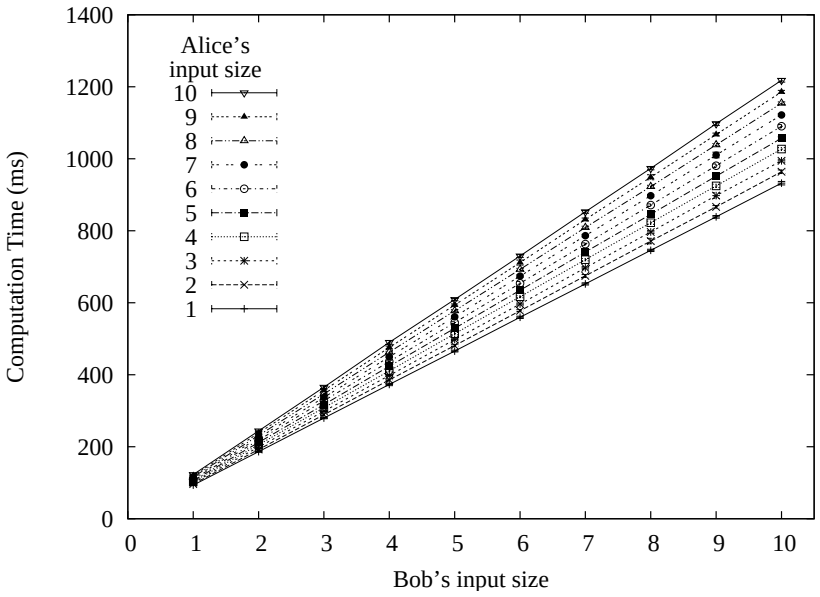

Figure 3.8: Bob's computation time in the Wilfrid protocol

in Section 3.5.3, which is an encryption of a random value.

If Bob decides to cheat, he can either use incorrect locations to evaluate the polynomial or just skip the polynomial evaluation step and send the encryption of whatever he likes to Alice. In the first case, if there is no intersection between Bob's incorrect input and Alice's input, no party will be hurt; if there is an intersection, Alice thinks that Bob is nearby. After Alice fails to spot Bob, Alice will realize Bob's misbehaviour. In the second case, If Bob sends random encryptions to Alice, the result is the same as in the no intersection case before. If Bob sends encryptions of valid identifiers of cells to Alice, Alice can detect Bob's misbehaviour by noticing that the recovered identifiers are not a subset of her input. If they happened to be a subset of Alice's input, the result is the same as the intersection case before. Therefore, there is no effective attack that Bob could launch in the Wilfrid protocol.

3.8 Comparison of the Protocols

In each of our four protocols we say Alice *succeeds* if she discovers Bob is nearby. In some of the protocols, if Alice succeeds, she also learns extra information about Bob's location. For convenience of discussion, we hereafter assume that, in the Wilfrid protocol, cells are defined by their coordinates, and Alice's input is her nearby nine cells and Bob's input is the one cell that he is in, which is the same assumption we made when comparing its performance with the Pierre protocol. We have set up each of our four protocols so that if Alice and Bob are within a distance r of each other, Alice will succeed. In the Louis protocol, the inverse is also true: if Alice and Bob are slightly more than distance r apart, Alice will not succeed.

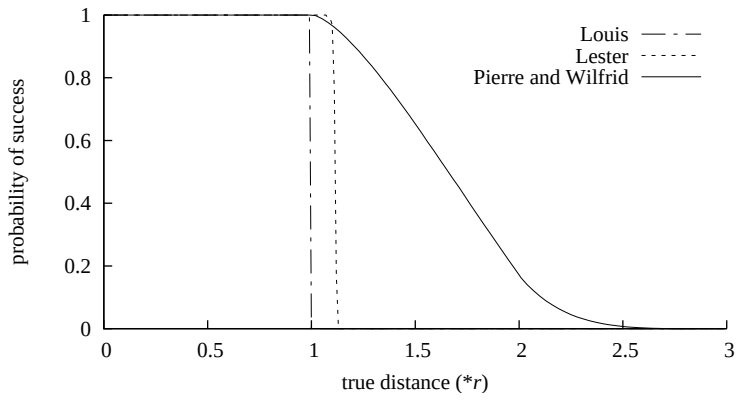

Figure 3.9: Success probabilities of the four protocols, as a function of the actual distance between Alice and Bob (as a multiple of r).

Protocol	Louis (first phase only)	Louis (both phases)	Lester	Pierre	Wilfrid
Extra info. learned by Alice	none	Bob's exact location	Bob's exact distance	Bob's grid cell distance	Intersecting cell
Requires 3^{rd} party	✓	✓			
Bob learns r	✓	✓		✓	✓
Bob learns Alice's location		✓			
Comm. steps	4	6	2	2	2

Table 3.5: Feature comparisons of our four protocols

This behaviour does not match realistic use models, however; it is unlikely that Alice will want to learn if Bob is 199 m away, but not if Bob is 201 m away. In our other three protocols, the probability that Alice succeeds does not fall to 0 as soon as Bob is slightly further than r away; rather, it gradually drops to 0 as Bob gets further, reaching 0 at some outer threshold distance r_{out}. That is, if Bob's distance from Alice is less than r, Alice will certainly succeed; if his distance is greater than r_{out}, Alice will certainly not succeed, and between those values, Alice's probability of success gradually decreases. This seems to fit better with what Alice is likely to want.

In Figure 3.9 we plot Alice's success probability against Bob's distance from her (in units of r), for each of the four protocols. As you can see, all four protocols succeed with probability 1 when the distance is less than r. The success probability of the Louis protocol drops immediately to 0 at that point, while the other protocols fall to 0 more gradually. The success probability of the Lester protocol starts dropping slowly as the distance increases past r, but then has a rapid decrease to 0 soon after; this is due to the fact that the kangaroo method for finding discrete logarithms has a small chance of succeeding, even if the logarithm in question is

outside the expected exponent range. By the assumption about Wilfrid protocol before, the configuration of Wilfrid and Pierre protocol are the same. The success probability of the Pierre and Wilfrid protocol, on the other hand, decreases to 0 gradually as the distance increases from r to $r_{out} = 2\sqrt{2}r$; this last value is the maximum distance by which Alice and Bob can be separated and still be in diagonally touching cells.

In Table 3.5 we summarize the properties of our four protocols. For each, we indicate what additional information Alice learns about Bob's location in the event that the protocol succeeds, and whether the protocol requires the participation of a third party. We also indicate whether Bob learns Alice's choice of r, whether Bob learns any information about Alice's location, and the number of communication steps.

3.9 Conclusion

We have presented four protocols to solve the nearby-friend problem without requiring a third party that learns location information. Compared to previous work, our protocols require fewer rounds of computation. Moreover, we have demonstrated their feasibility with a sample implementation and its evaluation.

Alerting people of nearby friends is only one of many possible location-based services. A topic of further investigation is what other services can be built with the techniques exploited in this chapter.

Chapter 4

Conclusion and Future Work

In the thesis, we investigated distributed approaches for location privacy in location-based services. We first focused on location-based services that need to know only a person's location, but not her identity. We presented a solution using location cloaking based on k-anonymity, which requires neither a single trusted location broker nor trust in all users of the system and that integrates nicely with existing infrastructure. In our solution, we suggested having multiple brokers, each deployed by a different organization (e.g., an operator of a cellphone network) and each knowing the location of only a *subset* of users, with the subsets being disjoint. The servers and a user can jointly determine the cloaked area based on k-anonymity. We implemented two protocols, both of which exploit the same idea above. The performance of one of the two protocols is sufficiently fast to be practical. To study the protocol's usability in practice becomes immediate future work.

Instead of distributing the task of the central location broker among multiple location brokers, another path for future work is to hide location information from the central broker in the first place. For example, users can register with and query the central location broker without letting the broker learn a user's location. During registration, a user sends a vector consisting of homomorphically encrypted values to the broker. All entries in the vector are encryptions of zero, except the entry for the user's current location, which is an encryption of one. The broker maintains the sum of all received vectors. (The user must also submit a vector upon leaving a cell.) For answering queries, the broker and a user rely on an oblivious transfer scheme [52], similar to the one suggested by Kohlweiss et al. [41] for retrieving location-specific information from a location-based service, but augmented such that the user learns only whether the current number of users in a cell is at least k. To the best of our knowledge, no such protocol is yet known to exist. Furthermore, this approach can be computationally expensive, since the broker needs to update each of the cells whenever a user (de-)registers with the broker. Therefore, one future work could be to find an efficient protocol to achieve the above idea.

In addition to the distributed k-anonymity protocol, we then presented four protocols—Louis, Lester, Pierre and Wilfrid—for a specific, identity required location-

based service: the nearby-friend application. The four protocols are also distributed and do not require a third party that learns location information, and all of the four protocols are proved to be efficient.

The experiment of our protocols is run on a desktop machine, which has a decent Pentium CPU. However, the mobile device usually is a smartphone, which only has a 400 MHz ARM CPU recently. The cryptographic performance of ARM processor is also worse than that of x86 processor in clock for clock comparison. Therefore, to be efficient on smartphones, our protocols may need further optimization. Although as indicated by our experiment, the performance of our protocols would not be much slower than setting up a TLS/SSL connection, which a smartphone can do efficiently, we envisioned the improvement on computation and communication complexity by using the Elliptic Curve implementation of the underlying cryptoschemes.

One emerged area of future work is to get rid of the semi-trust third party, Trent, in the Louis protocol. One possible way is to use Blake and Kolesnikov's GT-SCOT protocol, which we introduced in Section 2.3.3. As in the original Louis protocol, Bob sends Alice $\mathcal{E}_A(d+k)$, then Alice gets $d+k$ after decrypting it. At this stage, Alice and Bob could run the GT-SCOT protocol using $d+k$ and k as their input correspondingly. They are nearby if and only if the GT-SCOT protocol determines $d+k<k$. However, the GT-SCOT protocol requires bit-by-bit encryption of its inputs. For efficiency, we should limit the input length to 16 bits, but 16 bits are unlikely to be sufficient if Alice and Bob are not nearby, since distances are squared. To reduce the input length, we could use coarser granularities, but in that case we could just use the Pierre protocol, which is more efficient.

Besides the nearby friend application, there are other location-based services that also require a user's identity such as children, elderly parent or car finders. We could investigate how the techniques used in this thesis can improve location privacy in these services and if there are other specially designed protocols that are a better fit than general solutions for a location-based service.

References

[1] J. Al-Muhtadi, R. Campbell, A. Kapadia, M. D. Mickunas, and S. Yi. Routing Through the Mist: Privacy Preserving Communication in Ubiquitous Computing Environments. In *Proceedings of 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 65–74, July 2002.

[2] M. J. Atallah and W. Du. Secure Multi-party Computational Geometry. In *Proceedings of 7th International Workshop on Algorithms and Data Structures*, pages 165–179, August 2001. 39

[3] E. Bangerter, J. Camenisch, and A. Lysyanskaya. A Cryptographic Framework for the Controlled Release of Certified Data. In *Proceedings of 12th International Workshop on Security Protocols*, pages 26–28, April 2004.

[4] L. Barkuus and A. Dey. Location-based services for mobile telephony: A study of users' privacy concerns. In *Proceeding of 9th IFIP TC13 International Conference on Human-Computer Interaction (INTERACT)*, 2003. 9

[5] A. R. Beresford. Location privacy in ubiquitous computing. Technical Report 612, Computer Laboratory, University of Cambridge, January 2005. 3, 5, 10

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

[7] C. Bettini, S. Mascetti, X. S. Wang, and S. Jajodia. Anonymity in Location-based Services: Towards a General Framework. In *Proceedings of 8th International Conference on Mobile Data Management (MDM 2007)*, pages 67–79, May 2007. 5, 10

[8] I. F. Blake and V. Kolesnikov. Strong Conditional Oblivious Transfer and Computing on Intervals. In *Proceedings of ASIACRYPT 2004*, pages 515–529, December 2004. 9, 13

[9] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *Theory of Cryptography (TCC) '05, Lecture Notes in Computer Science 3378*, pages 325–341. Springer-Verlag, 2005. 47

[10] F. Brandt. Efficient Cryptographic Protocol Design based on Distributed El Gamal Encryption. In *Proceedings of 8th International Conference on Information Security and Cryptology (ICISC)*, pages 32–47, December 2005. 14, 39

[11] C. Cachin. Efficient Private Bidding and Auctions with an Oblivious Third Party. In *Proceedings of 6th ACM Conference on Computer and Communications Security*, pages 120–127, November 1999. 39

[12] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact E-Cash. In *Proceedings of EUROCRYPT 2005*, pages 302–321, May 2005. 21

[13] D. Chaum. Blind Signatures for Untraceable Payments. In *Proceedings of CRYPTO '82*, pages 199–203, August 1982. 20

[14] D. Chaum. Blind Signature System. In *Proceedings of CRYPTO '83*, pages 153–156, August 1983. 21

[15] D. Chaum. Security without Identification: Transaction Systems to Make Big Brother Obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985. 21

[16] R. Cheng, Y. Zhang, E. Bertino, and S. Prabhakar. Preserving User Location Privacy in Mobile Data Management Infrastructures. In *Proceedings of 6th Workshop on Privacy Enhancing Technologies (PET 2006), Lecture Notes in Computer Science 4258*, pages 393–412. Springer-Verlag, June 2006. 5, 38, 41

[17] C.-Y. Chow, M. F. Mokbel, and X. Liu. A Peer-to-Peer Spatial Cloaking Algorithm for Anonymous Location-based Services. In *Proceedings of 14th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS'06)*, pages 171–178, November 2006. 5, 10

[18] R. Cramer, R. Gennaro, and B. Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *Advances in Cryptology—Eurocrypt '97, Lecture Notes in Computer Science 1233*, pages 103–118. Springer-Verlag, 1997. 40

[19] I. Damgård and M. Jurik. A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System. In *Proceedings of 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 119–136, February 2001. 25

[20] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, `http://www.ietf.org/rfc/rfc4346.txt`, April 2006. 38, 45

[21] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of 13th USENIX Security Symposium*, pages 303–319, August 2004. 11

[22] W. Du and Z. Zhan. A Practical Approach to Solve Secure Multi-party Computation Protocols. In *Proceedings of 2002 Workshop on New Security Paradigms Workshop*, pages 127–135, September 2002. 39

[23] M. Duckham and L. Kulik. Location Privacy and Location-Aware Computing. In J. et al. Drummond, editor, *Dynamic and Mobile GIS: Investigating Changes in Space and Time*, pages 35–52. CRC Press:Boca Raton, FL USA, 2006. 2

[24] M. Fischlin. A Cost-Effective Pay-Per-Multiplication Comparison Method for Millionaires. In *Proceedings of RSA Security 2001 Cryptographer's Track*, pages 457–471, April 2001. 14, 29

[25] Poupard G. Fouque, P. and Stern J. Sharing Decryption in the Context of Voting or Lotteries. In *FC '00: Proceedings of the 4th International Conference on Financial Cryptography*, pages 90–104, London, UK, 2001. Springer-Verlag. 12

[26] M.J. Freedman, K. Nissim, and B. Pinkas. Efficient Private matching and Set Intersection. In *Advances in Cryptology–EUROCRYPT 2004*, May 2004. 52

[27] S. Garriss, Kaminsky M., Freedman M., Karp B., Maziéres D., and H. Yu. RE: Reliable Email. In *Proceedings of the 3rd Symposium on Networked Systems Design & Implementation (NSDI'06)*, May 2006. 51

[28] B. Gedik and L. Liu. Location Privacy in Mobile Systems: A Personalized Anonymization Model. In *Proceedings of 25th International Conference on Distributed Computing Systems (ICDCS 2005)*, pages 620–629, June 2005. 5, 10, 38, 41

[29] G. Ghinita, P. Kalnis, and S. Skiadopoulos. MobiHide: A Mobile Peer-to-Peer System for Anonymous Location-Based Queries. In *Proceedings of Proceedings of 10th International Symposium on Spatial and Temporal Databases (SSTD 2007)*, pages 221–238, July 2007. 5, 10, 11

[30] G. Ghinita, P. Kalnis, and S. Skiadopoulos. PRIVÉ: Anonymous Location-Based Queries in Distributed Mobile Systems. In *Proceedings of 16th International World Wide Web Conference (WWW2007)*, pages 371–380, May 2007. 5, 10

[31] J. Groth. A Verifiable Secret Shuffle of Homomorphic Encryptions. In *Proceedings of 6th International Workshop on Practice and Theory in Public Key Cryptography*, pages 145–160, January 2003. 25

[32] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *Proceedings of 1st International Conference on Mobile Systems, Applications, and Services (MobiSys 2003)*, pages 31–42, May 2003. 5, 8, 10, 38, 41

[33] T. Jiang, H. Wang, and Y.-C. Hu. Preserving Location Privacy in Wireless LANs. In *Proceedings of the 5th International Conference on Mobile Systems, Applications, and Service (MobiSys)*, June 2007. 4

[34] T. Jiang, H. J. Wang, and Y.-C. Hu. Preserving Location Privacy in Wireless LANs. In *Proceedings of 5th International Conference on Mobile Systems, Applications, and Services (MobiSys 2007)*, pages 246–257, June 2007.

[35] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preserving Anonymity in Location Based Services. Technical Report TRB6/06, School of Computing, The National University of Singapore, 2006. 5, 10

[36] A. Kapadia, N. Triandopoulos, C. Cornelius, D. Peebles, and D. Kotz. AnonySense: Opportunistic and Privacy-Preserving Context Collection. In *Proceedings of 6th International Conference on Pervasive Computing (Pervasive 2008)*, pages 280–297, May 2008. 11

[37] F. Kerschbaum. Distance-Preserving Pseudonymization for Timestamps and Spatial Data. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society (WPES)*, October 2007. 39

[38] H. Kido, Y. Yanagisawa, and T. Satoh. An Anonymous Communication Technique using Dummies for Location-based Services. In *Proceedings of the 2nd IEEE International Conference on Pervasive Services (ICPS)*, 2005. 4

[39] L. Kissner and D. Song. Privacy-Preserving Set Operations. In *Proceedings of CRYPTO 2005*, pages 241–257, August 2005. 30, 51

[40] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology—CRYPTO '96, Lecture Notes in Computer Science 1109*, pages 104–113. Springer-Verlag, August 1996. 47

[41] M. Kohlweiss, S. Faust, L. Fritsch, B. Gedrojc, and P. Preneel. Efficient Oblivious Augmented Maps: Location-Based Services with a Payment Broker. In *Proceedings of 7th Privacy Enhancing Technologies Symposium (PET 2007)*, pages 77–94, June 2007. 57

[42] G. M. Køien and V. A. Oleshchuk. Location Privacy for Cellular Systems; Analysis and Solutions. In *Proceedings of 5th Workshop on Privacy Enhancing Technologies (PET 2005), Lecture Notes in Computer Science 3856*, pages 40–58. Springer-Verlag, May/June 2005. 38

[43] T. Kölsch, L. Fritsch, M. Kohlweiss, and D. Kesdogan. Privacy for Profitable Location Based Services. In *Proceedings of 2nd International Conference on Security in Pervasive Computing (SPC 2005)*, pages 164–178, April 2005.

[44] M. Li, K. Sampigethaya, and Poovendran R. Huang, L. Swing & Swap: User-Centric Approaches Towards Maximizing Location Privacy. In *Proceedings*

of 5th Workshop on Privacy in the Electronic Society (WPES), pages 19–28, October 2006. 4

[45] Loopt, Inc. loopt - Live In It. `http://www.loopt.com/`. Accessed February 2007. 37, 41

[46] S. Mascetti and C. Bettini. A Comparison of Spatial Generalization Algorithms for LBS Privacy Preservation. In *Proceedings of International Workshop on Privacy-Aware Location-based Mobile Services (PALMS)*, May 2007. 5, 10

[47] MIT SENSEable City Lab. iFind. `http://ifind.mit.edu/`. Accessed February 2007. 37

[48] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The New Casper: Query Processing for Location Services without Compromising Privacy. In *Proceedings of 32nd International Conference on Very Large Data Bases (VLDB 2006)*, pages 763–774, September 2006. 5, 10, 38

[49] The OpenSSL Project. OpenSSL: The Open Source toolkit for SSL/TLS. `http://www.openssl.org`. Accessed February 2008. 22, 43

[50] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proceedings of EUROCRYPT '99*, pages 223–238, May 1999. 9, 11

[51] J.M. Pollard. Monte Carlo Methods for Index Computation (mod p). *Mathematics of Computation*, 32(143):918–924, July 1978. 40, 45

[52] M. O. Rabin. How to Exchange Secrets with Oblivious Transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981. 57

[53] P. Samarati and L. Sweeney. Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression. Technical Report SRI-CSL-98-04, SRI International, 1998. 5, 10

[54] D. Shanks. Class number, a theory of factorization, and genera. *Proceedings of Symposia in Pure Mathematics*, 20:415–440, 1971. 45

[55] Victor Shoup. NTL: A Library for doing Number Theory. `http://www.shoup.net/ntl`. Accessed February 2008. 22, 43

[56] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003. 11

[57] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[58] A. C. Yao. Protocols for Secure Computations. In *Proceedings of 23rd IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982. 14, 39

[59] T.-H. You, W.-C. Peng, and W.-C Lee. Protecting Moving Trajectories with Dummies. In *Proceedings of the first International Workshop on Privacy-Aware Location-based Mobile Services*, May 2007. 4

[60] G. Zhong, I. Goldberg, and U. Hengartner. Louis, Lester and Pierre: Three Protocols for Location Privacy. In *Proceedings of Seventh Privacy Enhancing Technologies Symposium (PET 2007), Ottawa, Canada*, June 2007. 38