Karlsruhe Reports in Informatics 2019,7
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

On Preserving Secrecy in Mobile Social Networks

Gabriela Suntaxi, Aboubakr Achraf El Ghazi,
Klemens Böhm

2019

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

On Preserving Secrecy in Mobile Social Networks

GABRIELA SUNTAXI, ABOUBAKR ACHRAF EL GHAZI, and KLEMENS BÖHM, Karlsruhe Institute of Technology, Germany

Location-based services are one of the most important services offered by mobile social networks. Offering this kind of services requires accessing the physical position of users together with the access authorizations, i.e., who is authorized to access what information. However, these physical positions and authorizations are sensitive information which have to be kept secret from any adversary, including the service providers. As far as we know, the problem of offering location-based services in mobile social networks with a revocation feature under collusion assumption, i.e., an adversary colludes with the service provider, has not been studied. In this paper, we show how to solve this problem in the example of range queries. Specifically, we guarantee any adversary, including the service provider, is not able to learn (1) the physical position of the users, (2) the distance between his position and that of the users, and (3) whether two users are allowed to learn the distance between them. We propose two approaches namely two-layer symmetric encryption and two-layer attribute-based encryption. The main difference between the first and the second approach is that they use, among other encryption schemes, symmetric and attribute-based encryption, respectively. Next, we prove the secrecy guarantees of both approaches, analyze their complexity and provide experiments to evaluate their performance in practice.

Additional Key Words and Phrases: mobile social networks; access control; location-based services

1 INTRODUCTION

1.1 Motivation

Mobile Social Networks (mSNs) like Foursquare or Badoo have become popular in the last years. Similarly to traditional social networks, mSNs allow users to create virtual communities to share content, but they also let users share their physical position with other users. Having access to the physical position of the users, mSNs offer location-based services (LBS) such as querying friends within a given distance. In this kind of network, each user specifies who is authorized to learn information about his physical position, i.e., users establish authorization relationships with others. Next, given the dynamic relationships between users that are inherent to human behavior, revocation of such privileges is a fundamental feature of mSNs.

To deliver services in mSNs different architectures with various components have been proposed [10]. However, the three major components are service providers, mobile users, and network infrastructure. Depending on the features supported by mSNs, the service providers can be dedicated servers (e.g., location server, access control server, video sharing server, VoIP server) which provide services to the users through the network infrastructure [10]. Mobile users (e.g., mobile phones, wearable devices) receive data or service results from the service providers. The network infrastructure is used to transfer data from a source (e.g., service providers) to a destination (e.g., mobile users). Figure 1 illustrates a mSN architecture consisting of users and two service providers: the LBS provider and the access control server (ACS). This system architecture is enough to provide typical LBS and has been considered by existing work in the area [17, 18, 30]. Here, the LBS provider stores the physical positions of the users, and the ACS stores the authorization relationships.

The physical positions of the users and authorization relationships are sensitive information that should be kept secret from any adversary, including the service providers. In fact, this information is particularly sensitive since it can be used to infer further personal information. For example, the

Authors' address: Gabriela Suntaxi, gabriela.suntaxi@kit.edu; Aboubakr Achraf El Ghazi, elghazi@kit.edu; Klemens Böhm, klemens.boehm@kit.edu, Karlsruhe Institute of Technology, Am Fasanengarten 5, Karlsruhe, 76131, Germany.

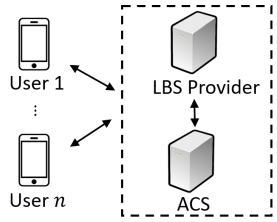

Fig. 1. mSN System Architecture

physical positions of the users can be used to infer their state of health or personal preferences [1]. Consequently, this information has to be kept secret from unauthorized users, i.e., confidentiality.

Confidentiality is usually facilitated by trusting the entity in the system that manages the access policies, i.e., the ACS. However, recent privacy breaches on existing social networks, such as the Facebook-Cambridge Analytica scandal [9], have put this into question. In such attacks, the ACS has allowed unauthorized entities, intentionally or unintentionally, to access private information. Collusion attacks where adversaries, including malicious users, collude with the service provider to gain unauthorized access to information are an important problem in mSNs.

This paper studies how to facilitate LBS in mSNs while providing secrecy guarantees to the users under collusion assumption. We focus on one specific service, namely querying friends within a given distance. As motivated earlier, we also cover revocation. Regarding collusion, we study the case of pairwise collusion in which a user colludes with either the LBS provider or the ACS to try accessing information that he is not unauthorized to see. In Section 2.2, we define our adversary model. Regarding secrecy guarantees, we provide users with the following guarantees:

- $G_{position}$: The physical positions of users are kept secret from any adversary.
- $G_{distance}$: Given a user u, only entities they themselves have authorized can learn the distance between their physical position and the one of u.
- $G_{authorization}$: Given two users u and v, an adversary will not be able to learn whether u or v are allowed to learn the distance between them.

Existing work in the area [17, 18, 33], either does not consider collusion attacks, or their system architectures assume trusted authorities. This however does not solve the collusion problem, but only shifts it to the trusted entity. Furthermore, these approaches do not provide a rigorous specification of the collusion strategy as part of their adversary model, and they consider weaker adversaries, as we explain in Section 2.2. Next, none of the existing work we are aware of does protect $G_{authorization}$ against both the LBS provider and ACS.

1.2 Challenges

Encryption techniques are an effective way to keep the information secret from all but those who are authorized to access it, without the presence of a trusted entity [21]. However, offering the secrecy guarantees $G_{position}$, $G_{distance}$ and $G_{authorization}$ under our collusion assumption with a revocation feature in mSNs is challenging. First, the weak computing power of mobile devices requires mobile users to outsource any heavy computation task. This not only restricts the choice of encryption schemes that can be used but also the design alternatives for a solution.

Second, because of the collusion assumption, the typical encryption guarantee that only holders of the right key can decrypt the information does not hold anymore. If an entity A that is not authorized to decrypt a given ciphertext c colludes with an authorized entity B, A can get the key from B to decrypt c or send c to B to decrypt it and get back the plaintext. To solve this issue, one

can add further entities in the system architecture and use multi-layer encryption. Then, one needs to assign the keys of each encryption layer to the entities so that in case of collusion, unauthorized entities cannot gain access to the information. That is, either the unauthorized entities cannot access the ciphertext or they cannot access at least one key of a non-colluding entity to be able to decrypt all encryption layers. Indeed, multi-layer encryption with an adequate key distribution among the entities of the system is the core idea of our solution, as we explain in Section 4. The idea of multi-layer encryption has been used previously in systems such as CryptDB [27]; however, the context is different from ours. In CryptDB, each encryption layer gives a higher security level but reduces the capability of performing computation over the encrypted data. Users in CryptDb basically have all the keys to decrypt the encryption layers, and during query processing, the users decrypt the layers until they reach one that allows processing a given query.

Third, in a multi-user setting, such as our mSN scenario, several owners want to share information with authorized users. To this end, one can consider two main encryption schemes, namely symmetric encryption and attribute-based encryption (ABE). Both encryption schemes have advantages and disadvantages when comparing them to each other. The advantage of symmetric encryption schemes is the efficiency in the encryption/decryption operations. In the multi-user setting, however, key management and revocation are known problems of this kind of scheme which could overcome its advantage if not implemented appropriately. Section 4.1 features an example illustrating that seemingly simple solutions based on symmetric encryption schemes are not adequate, i.e., the illustrated solution has a huge resource consumption in terms of storage and CPU usage, and users have to be online during revocation. In contrast, an approach that uses ABE has two advantages: One does not need to deal with multiple ciphertexts, and key management and revocation are straightforward. This is because a single ciphertext is generated based on an access control policy so that only users whose attributes fulfill the access policy can decrypt it. However, the encryption/decryption process using ABE involves bilinear pairing and exponentiation operations, which in general has a significant impact on performance [24]. Indeed, we show that both approaches can solve the problem of sharing information in the multi-user setting if implemented appropriately. However, based on the advantages and disadvantages of both encryption schemes, it is not obvious to determine which one performs better in our specific scenario.

1.3 Contributions

In this paper, we propose two approaches, which combine existing encryption schemes, to allow users of mSNs to query friends within a given distance. Both approaches include a revocation feature and provide users with the secrecy guarantees $G_{position}$, $G_{distance}$ and $G_{authorization}$ under the collusion assumption. First, we describe our problem and our adversary model. Next, we describe our approaches, namely two-layer symmetric encryption (*2lSE*) and two-layer attribute-based encryption (*2lABE*). The main difference between the first and the second approach is that they use, among other encryption schemes, symmetric and attribute-based encryption, respectively. We prove that both approaches fulfill our secrecy guarantees. To evaluate the performance of our approaches, we provide complexity analyses of them. Our analyses tell us which approach is better at each entity involved in the system. Next, we conduct experiments to validate the results of our complexity analyses and finally determine which approach performs better in practice. Next to other insights, although with the second solution the key management is more straightforward than with the first solution, and the service provider does not have to store multiple encrypted copies for each message, we have found that our first solution is on average twice as efficient in our scenario. Therefore, we propose to consider the *2lSE* approach, which not only solves the secrecy problem existing in mSNs but also is more performant than the *2lABE* approach.

2 PROBLEM FORMULATION

2.1 System Architecture

mSNs require the following information for each user u to provide LBS: (1) the physical position of u denoted by $p_u = (x_u, y_u)$, (2) the set of users who have allowed u to learn the distance between their physical positions and the one of u, $Grantor_u$, and (3) the set of users to whom u has allowed to learn the distance between his position p_u and their physical positions, called the set of grantees of u, $Grantee_u$. As mentioned in Section 1, we consider a mSN system consisting of users and two service providers: the LBs provider and the ACS, which store the physical position and the sets of grantors and grantees of each user, respectively. In addition to these entities, there is a key authority responsible for key issuing.

Each user u can request to update his position p_u and to allow or to revoke access from users at any time. Users have to send access requests, Definition 2.1, to other users to establish who can learn the distance between their physical positions. To avoid that the servers learn information from the stored data, the users have to encrypt the information before outsourcing it. To obtain LBS, users have to send their queries to the ACS. The ACS and the LBS provider interact with each other to compute the query result, which the ACS sends to the user. See Section 4 for details. LBS support different types of queries. In this paper, we focus on range queries, one of the most important queries in mobile networks [35]. Let U and $dist(p_u, p_v)$ denote the set of all users and the distance between the physical positions of users u and v, respectively.

Definition 2.1 (Access request). An **access request** $accessReq : U \times U \rightarrow \{true, false\}$ is a function that takes as input two users $u, v \in U$ and indicates that user u requests v for permission to *learn* the distance between their physical positions. The function $accessReq(u, v)$ outputs *true* if v authorizes the access request. Otherwise, it outputs *false*. We call a user who authorizes an access request and a user who receives an authorized access request, grantor and grantee, respectively.

Definition 2.2 (Range query). Given a user u and a distance d, a **range query**, $Range(u, d)$, is a query that returns the users who are located within a distance d from u and who have authorized u to learn the distance between their physical positions and the one of u. Formally, $Range(u, d) = \{v \in U \mid v \in Grantor_u \wedge dist(p_u, p_v) \leq d\}$.

2.2 Adversary Model and Secrecy Guarantees

To formally define an adversary model under a collusion assumption, one needs to specify four aspects [14]: (1) the collusion strategy, i.e., when or how entities come under the control of the adversary, (2) the adversarial behavior strategy, i.e., the actions that the colluding entities might take, (3) the computational strategy, i.e., the computational complexity the adversary is assumed to have, and (4) the protocol execution strategy, i.e, how many times the adversary is allowed to execute the protocol. Next, we describe briefly the main alternatives for each of the four aspects of the adversary model and specify our model. We specify our model based on a *just-strong-enough* principle, in which the selected alternatives for each of the four aspects are just strong enough to model real-world adversaries under a collusion assumption with revocation capability, as we explain in the remaining of this section. Studying more complex settings which consider stronger adversary models is future work.

Collusion strategy: There exist two main collusion strategies: static and adaptive [14]. In the static strategy, the adversary is given a set of entities to collude with, and the honest entities remain honest during the protocol execution. In the adaptive strategy, the adversary can collude with any entity in the system during the protocol execution. Here, we consider the static strategy. Although the static strategy is weaker than the adaptive one, developing highly efficient schemes that are

secure under the static strategy serves as an important step for constructing secure schemes under the adaptive one [14]. We refer to any entity that participates in a collusion as an adversary.

Adversarial behavior strategy: There exist three main adversarial behavior strategies: semi-honest, covert, and malicious [14]. In the semi-honest strategy, every colluding entity follows the protocol specification, i.e., each entity performs the tasks assigned to it correctly. However, adversaries can access to the state of all colluding parties and try to learn information from it. In the covert strategy, adversaries may deviate from the protocol specification if honest entities do not detect them. The covert strategy represents many real-world scenarios such as financial or political settings, where the involved entities, i.e., companies or individuals, cannot afford the embarrassment, loss of reputation and law punishment associated with being caught cheating. In the malicious strategy, the colluding entities can deviate arbitrarily from the protocol specification, according to the instructions of the adversary. Since the security guarantees provided by schemes in the semi-honest strategy are weak, and schemes that offer security guarantees under the malicious strategy are not efficient enough to be implemented and used in practice [2, 14], we opt for the covert strategy, which represents real-world adversaries. However, since we aim to cover realistic scenarios where unauthorized entities try to gain access to information they are not allowed to, we restrain the covert strategy further to limit adversaries to deviate from the protocol only to advantage themselves, but they will not disadvantage any entity in the system. Advantaging and disadvantaging an entity means to give the entity access to information that the entity is not allowed to access and to deny access to information that the entity is allowed to access, respectively.

Computational strategy: There exist two main computational strategies: polynomial and unbound. In the first one, adversaries run in polynomial time, while in the second one, they do not have computational limits. Similar to existing works [2, 17], we consider the polynomial strategy.

Protocol execution strategy: There exist two main protocol execution strategies: stand-alone and concurrent-composition [7]. In the stand-alone strategy, the adversary is allowed to execute the protocol a single time, while in the concurrent-composition one, he can execute the protocol several times. Considering adversaries in the concurrent-composition strategy is a harder problem to solve; however, having a scheme for the stand-alone strategy can be used to design schemes for the concurrent-composition one [14]. Following our *just-strong-enough* principle for specifying the adversary model, we consider the stand-alone strategy, and we extend it to fulfill our needs. In our extension, we allow the execution of the protocol twice, which let us evaluate the secrecy guarantees met after a revocation takes place. Before describing the extension, we explain the need for it. In a single protocol execution, the adversary can (1) collude with the entities of the system based on the defined collusion strategy, (2) send a set of queries, and (3) get their respective answers based on the information stored at each entity of the system at the moment of the protocol execution. In our scenario, the information stored by the entities of the system includes, among others, the encrypted sets of grantors and grantees of the users. The information stored at each entity is not modified during the protocol execution. However, to evaluate the secrecy guarantees of a scheme under revocation, one needs to change the authorizations, i.e., adjust the set of grantors and grantees of the users involved in the revocation. Therefore, after updating the information, we need to allow the adversary to execute the protocol a second time. In the second protocol execution, the adversary is allowed to repeat the steps (2)-(3) of the first execution, but it is not allowed to collude anymore with any entity. The purpose of the second execution is to evaluate whether a revoked user can gain access to information that he is not authorized to access anymore but that he could access on the first execution. We call this extension the twofold-composition strategy.

To summarize our adversary model, regarding the collusion, adversarial behavior, computational and protocol execution strategies, we choose the static strategy, the covert strategy, where adversaries are allowed to deviate from the protocol only to advantage themselves, the polynomial

strategy and the twofold-composition strategy, respectively. In the remaining of the paper, when we refer to adversaries, we imply adversaries with the power specified in our adversary model. Obviously, neither a user is an adversary of himself nor are the entities that a user has allowed them freely to access his data.

Having specified our adversary model, it only remains to define the setting of the static strategy, i.e., specify the possible collusion scenarios.

Static strategy setting: On each execution of the protocol there are four entities: the key authority, a user, the ACS and the LBS provider. First, the key authority is an honest entity. We note that the key authority is only responsible of issuing keys during the registration of users in the system, it does not participate in any other phase of the protocol specification, and it does not store any information. Details of our protocol are in Section 4. Thus, considering the key authority as an honest entity does not shift the collusion problem to it. Second, the LBS provider, ACS and the user can be adversaries. However, we limit ourselves to the case of pairwise collusion. This restriction is in line with the selected adversary behavior strategy, i.e., the covert strategy. Since three entities participate in the critical phases of the protocol execution, i.e, access request, query and revocation phases, at least one honest entity is needed to be able to detect the deviation from the protocol of the adversaries. Finally, in line with existing approaches [18, 25, 33, 37], we consider that: (1) the LBS provider and ACS do not collude with each other, and (2) the ACS and LBS provider cannot identify the users by observing their IP addresses in the connections.

Secrecy Guarantees: Based on our adversary model, we aim to offer the secrecy guarantees $G_{position}$, $G_{distance}$ and $G_{authorization}$ stated in Section 1.

3 ENCRYPTION SCHEMES USED

In this section, we define the encryption schemes that we consider in our approaches. In Section 4, we explain how we apply them in our scenario.

Definition 3.1 (Symmetric Encryption Scheme). A **symmetric encryption scheme** $\mathcal{SE}$ = (KGen, Enc, Dec) consists of three algorithms: (1) A key generation algorithm KGen that returns a key k. (2) An encryption algorithm Enc which can be probabilistic or deterministic and takes as input the key k and a plaintext m to return a ciphertext c. (3) A deterministic decryption algorithm Dec that takes as input the key k and a ciphertext c to return a plaintext m such that $\mathsf{Dec}(\mathsf{k}, \mathsf{Enc}(\mathsf{k}, m)) = m$.

We write $\mathsf{Enc}(\mathsf{k}, m)$ and $\mathsf{Dec}(\mathsf{k}, c)$ for the operations of encrypting m under key k and decrypting c under key k, respectively.

Definition 3.2 (Asymmetric Encryption Scheme). An **asymmetric encryption scheme** $\mathcal{AE}$ = (KGen, Enc, Dec) consists of three algorithms: (1) A key generation algorithm KGen that returns a pair of public and secret keys (pk, sk). (2) A probabilistic encryption algorithm Enc which takes as input the public key pk and a plaintext m to return a ciphertext c. (3) A deterministic decryption algorithm Dec that takes as input the secret key sk and a ciphertext c to return a plaintext m, such that $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$.

Definition 3.3 (Somewhat Homomorphic Encyption Scheme). A **somewhat homomorphic encryption scheme** *SHE* is an asymmetric encryption scheme in which the message space is a ring $(R, +, \cdot)$ and the ciphertext space is also a ring $(R, \oplus, \otimes)$ such that for all messages $m1, m2 \in R$, and all pair of keys (pk, sk), $m1 + m2 = \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m1) \oplus \mathsf{Enc}(\mathsf{pk}, m2))$, and $m1 \cdot m2 = \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m1) \otimes \mathsf{Enc}(\mathsf{pk}, m2))$. A SHE supports limited computations on ciphertext, i.e., one can perform a limited number of addition and multiplication operations.

We also use Ciphertext-Policy attribute based encryption (CP-ABE) [5]. In CP-ABE, the data is encrypted based on an access policy, which itself consists of constraints on user attributes, like

$role = \text{"student"} \wedge memberOf = \text{"Project1"}$. The encryption is done so that only users who fulfill the access policy can decrypt. Formally:

Definition 3.4 (Ciphertext-Policy Attribute based Encryption Scheme). A **CP-ABE scheme** consists of four algorithms (*Setup*, KGen, Enc, Dec) (1) A setup algorithm, *Setup*, that selects two cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$ and use them to generate and return a public key pk and a master key mk. (2) A key generation algorithm KGen that takes as input a set of attributes ω_u associated with a user u and the master key mk, to return a secret key sk_{ω_u}. (3) A probabilistic encryption algorithm Enc that takes as input a message m, an access policy γ and the public key pk to return a ciphertext c_γ. (4) A deterministic decryption algorithm Dec which takes as input the public key pk, a secret key sk_{ω_u} associated with the set of attributes ω_u and a ciphertext c_γ to return a message m if ω_u satisfies the access policy γ; otherwise it returns an error message $\perp$. We write $\mathsf{Enc}(\mathsf{pk}, m_\gamma)$ for the operation of encrypting m under the access policy γ and key pk, and $\mathsf{Dec}((\mathsf{pk}, \mathsf{sk}_{\omega_u}, c_\gamma))$ for the operation of decrypting c_γ under the pair of keys $(\mathsf{pk}, \mathsf{sk}_{\omega_u})$.

Note that general CP-ABE schemes do not guarantee the security of the access policy used to encrypt a given ciphertext c, i.e., an entity who has access to c can learn who is authorized to decrypt c. Learning this information is against the secrecy guarantee $G_{authorization}$ that we aim to offer. However, we consider stronger CP-ABE schemes with *hidden policy* as proposed in [19].

4 OUR APPROACH

To fulfill the secrecy guarantees, users have to encrypt the information before outsourcing it to the ACS and the LBS provider. There are different possibilities to do so.

Mainly due to the decryption and revocation overhead at the users-side, we do not consider naive solutions such as the one of Example 4.1. The illustrated solution has the following shortcomings: it affects the storage capacity and limited processing resources of mobile devices, and revocation is not only not efficient for data owners, but it also requires authorized users to be online. Given a set S, let $|S|$ denote the cardinality of S.

Example 4.1. Assume that each user u encrypts his name with a key k_u, stores his encrypted name at the LBS provider, and distributes k_u to all authorized users. Such a solution has several problems. First, each user u has to store as many keys as grantors. Second, during query processing, u receives as result a set of encrypted names corresponding to users who fulfill the query condition. Since u stores one key for each of his grantors and u does not know which key to use to decrypt each ciphertext, the decryption process has a worst case complexity of $\mathcal{O}(|Grantor_u|^2)$. Third, if u wants to revoke access from a user, u has to generate a new key $k_{u'}$, encrypt his name with $k_{u'}$, replace his encrypted name at the LBS provider with the new ciphertext and distribute $k_{u'}$ to all still authorized users.

Next, we show how to use and combine existing cryptographic techniques to implement a scheme under our secrecy guarantees. We come up with two approaches. To ease the explanation of them, we first start by describing two basic schemes, called basic two-layer symmetric encryption, *basic 2lSE*, and basic two-layer attribute-based encryption, *basic 2lABE*. Our basic schemes meet our secrecy guarantees under a weaker adversary model than the one defined in Section 2.2. With it, we weaken the protocol execution strategy by considering the stand-alone strategy instead of the twofold-composition strategy, i.e., the protocol is executed only once. We then show how to extend the basic schemes to meet our secrecy guarantees under our actual adversary model, Section 2.2.

The main difference between our basic schemes lies on the cryptographic schemes used to encrypt the names which are sent as query answers. The basic schemes consist of four phases, namely

initialization phase, registration phase, access request phase, and query phase. In Sections 4.1 and 4.2, we explain these phases for the *basic 2lSE* and the *basic 2lABE* schemes, respectively.

4.1 Basic two-layer symmetric encryption (*basic 2lSE*)

4.1.1 Initialization phase. In this phase, the key authority generates and distributes keys. The initialization of the system happens only once. The entities involved in this phase are the key authority, the ACS, and the LBS provider. The key authority generates three pairs of keys $(\mathsf{pk}_{LBS}, \mathsf{sk}_{LBS})$, $(\mathsf{pk}_{ACS}, \mathsf{sk}_{ACS})$, and $(\mathsf{pk}_H, \mathsf{sk}_H)$. These keys are used later during the registration, access request, and query phases. The key authority sends the secret keys sk_H and sk_{ACS} to the ACS and the secret key sk_{LBS} to the LBS provider. The key authority chooses the integers p and g of Diffie-Hellman key exchange protocol (DH), Definition 4.2. The DH protocol is used in Sections 4.1.2 - 4.1.4 to generate and share secret keys between a pair of users.

Definition 4.2 (Diffie-Hellman key exchange). The **Diffie-Hellman** key exchange (DH) is a protocol which allows two parties, A and B, that have no prior knowledge of each other to establish a shared secret key jointly. The protocol is as follows: First, a trusted party chooses and publishes two integers p and g, where p is large, e.g., 512 bits, and g is a primitive root modulo p.[1] Second, the parties A and B choose the secret integers, a and b, respectively. Next, A computes $Z_A \equiv g^a \pmod p$ and sends Z_A to B. B computes $Z_B \equiv g^b \pmod p$ and sends Z_B to A. Finally, A computes the shared key $k_{ba} \equiv {Z_B}^a \pmod p$. B computes the shared key $k_{ab} \equiv {Z_A}^b \pmod p$. The shared key value is $k_{ba} \equiv {Z_B}^a \pmod p \equiv (g^b)^a \pmod p \equiv g^{ab} \pmod p \equiv (g^a)^b \pmod p \equiv {Z_A}^b \pmod p \equiv k_{ab}$.

4.1.2 Registration phase. In this phase, new users are registered in the system. Registering a user u in the system involves four entities, the key authority, the ACS, the LBS provider, and the user u.

Figure 2 illustrates the steps of this phase. First, the key authority sends to u (1) two identifiers id_u^{ACS} and id_u^{LBS}, (2) a secret key k_u, (3) the public keys pk_{LBS} and pk_{ACS}, and (4) the integers p and g. Second, u selects an integer number η_u and compute the value $Z_u \equiv g^{\eta_u} \pmod p$. We call η_u and Z_u the secret and public numbers of u, respectively. These two numbers are used as part of the DH protocol in the access request and query phases, as we explain in Sections 4.1.3 and 4.1.4. Next, u stores at the LBS provider his identifier id_u^{LBS}, his encrypted position $\mathsf{Enc}(\mathsf{pk}_H, p_u)$,[2] and his encrypted public number $\mathsf{Enc}(\mathsf{pk}_{ACS}, Z_u)$. The use of SHE allows the LBS provider to compute the encrypted square distance between the encrypted positions of two users. The LBS provider cannot decrypt any of the ciphertexts because it does not have the secret keys to do it. Finally, u stores at the ACS his identifier id_u^{ACS} and two empty sets $Grantor_u$ and $Grantee_u$. Information is added to these two sets in the access request phase, Section 4.1.3. We note that neither the LBS provider nor the ACS knows the link between users and their identifiers.

4.1.3 Access request phase. In this phase, a user u calls the function $accessReq(u, v)$. If user v authorizes the access request, i.e., $accessReq(u, v) = true$, v stores encrypted information at the ACS and the LBS provider, as we will explain in this Section. The providers use this information to process queries sent by u. Before explaining the steps of this phase, let us analyze briefly how query processing works to understand the information that v has to store. Example 4.3 illustrates two design alternatives, *querying-filtering* and *filtering-querying*, to answer a given range query.

Example 4.3. Think of an LBS provider and an ACS. Assume that for each user u, the LBS provider stores his encrypted physical position using SHE, and the ACS stores the set of grantors $Grantor_u$. To answer a given range query $Range(u, d)$, different designs alternatives are conceivable.

[1] A primitive root modulo p is an integer g such that $g \pmod p$ has multiplicative order $p - 1$.

[2] In reality, we encrypt a given position p_u as $\mathsf{Enc}(\mathsf{pk}_H, x_u)$ and $\mathsf{Enc}(\mathsf{pk}_H, y_u)$.

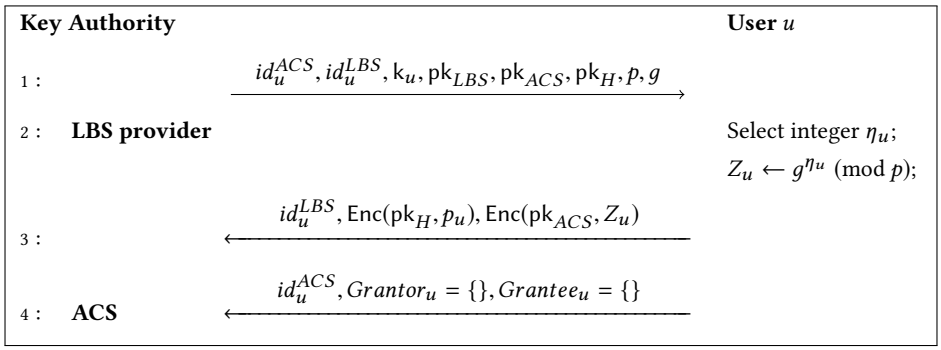

Fig. 2. User Registration Phase - *basic 2lSE*

In particular, one can consider two designs namely *querying-filtering* and *filtering-querying*. With the first alternative, the LBS provider executes, first, the query and then sends the result to the ACS to filter it based on the set $Grantor_u$. With the second alternative, the ACS sends, first, the set $Grantor_u$ to the LBS provider and then the LBS provider executes the query using only the physical positions of users in $Grantor_u$. With both alternatives, since the LBS provider stores encrypted physical positions, it cannot use indexing, like B-tree or R-tree, for spatial query processing. Then in terms of performance, the *querying-filtering* alternative is not a suitable option because the LBS provider would need to compute the encrypted square distances between the encrypted position of u and the ones of all the users in the system.

Due to performance reasons, as explained in Example 4.3, we opt for the *filtering-querying* approach. Then the set of grantors of each user u has to contain information that allows the LBS provider to reduce the computation cost during query execution. Specifically, if $accessReq(u, v) = true$, v has to add, among other information, his encrypted identifier id_v^{LBS} in the set $Grantor_u$, and to store his encrypted name at the LBS provider. The encrypted names of the users are sent as query answers, as we explain in Section 4.1.4. The access request phase involves the following entities: the users that are part of the access request, u and v, the LBS provider, and the ACS.

Figure 3 illustrates the steps of this phase. We denote the concatanation of strings a and b by $a\|b$. First, user u calls the function $accessReq(u, v)$ and sends to user v his identifier id_u^{ACS} and his public number Z_u. If $accessReq(u, v) = true$, v computes the shared key $\mathsf{k}_{uv} \equiv {Z_u}^{\eta_v} \pmod p$ and selects two random numbers $r_{u_v}^{ACS}, r_{u_v}^{LBS} \in \mathbb{Z}$. Next, v encrypts his name using two layers of encryption. The shared key k_{uv} is used for the inner layer of encryption, and the public key pk_{ACS} is used for the outer layer of encryption. Then, v stores at the LBS provider the resulting ciphertext, $\mathsf{Enc}(\mathsf{pk}_{ACS}, \mathsf{Enc}(\mathsf{k}_{uv}, v))$, together with the random number $r_{u_v}^{LBS}$. The LBS provider cannot decrypt the ciphertext, even if it colludes with any of the users, because none of them has the key to decrypt the outer layer of encryption. Secrecy proofs are in Section 5. Next, v sends to the ACS the identifier of u, id_u^{ACS} together with a tuple t which consists of two elements: the random number $r_{u_v}^{ACS}$ and the ciphertext $\mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS} \| r_{u_v}^{LBS})$, i.e., $t = \left\langle r_{u_v}^{ACS}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS} \| r_{u_v}^{LBS}) \right\rangle$. The ACS adds the tuple t to the set $Grantor_u$. The ciphertext, which is part of tuple t, is sent by the ACS to the LBS provider during query processing. This ciphertext can be decrypted only by the LBS provider. The LBS provider uses the decrypted information as an index to recover the encrypted name, encrypted position, and encrypted public number of v. Note that only v knows his identifier id_v^{LBS}, then he is the only one who can add his encrypted id to the set of grantors of other users. Finally, to revoke access, v stores in his set of grantees, $Grantee_v$, at the ACS, the ciphertext

$c = \mathsf{Enc}(\mathsf{k}_v, u \| id_u^{ACS} \| r_{u_v}^{ACS} \| r_{u_v}^{LBS})$. The ciphertext c contains the name of u and index information that allows v to revoke access from u, i.e., v can delete (1) the tuple t added in the set $Grantor_u$ and (2) the encrypted name stored at the LBS provider. Only v knows the key to decrypt c.

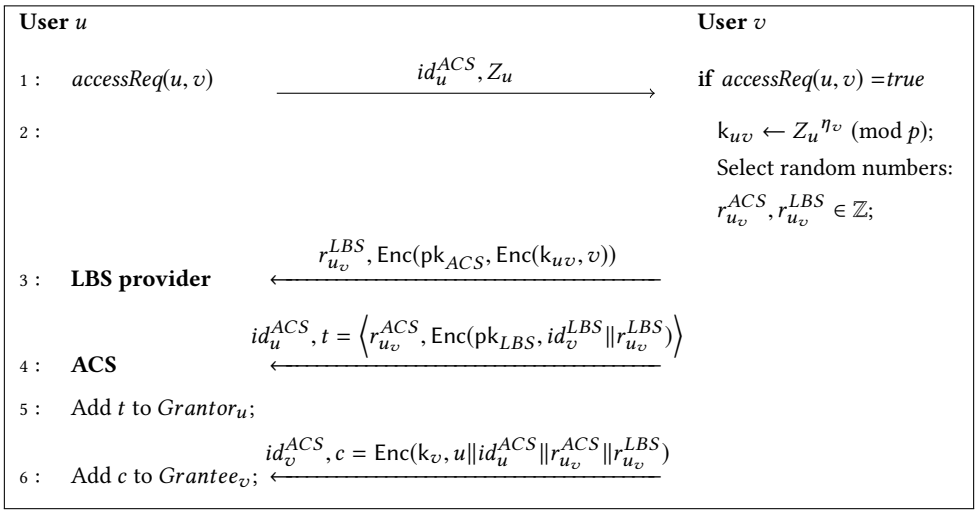

Fig. 3. Access Request Phase - *basic 2lSE*

4.1.4 Query phase. In this phase, users send range queries and get back query answers. The query phase involves three entities, the querying user u, the ACS, and the LBS provider.

We use the following notation: Given n strings, $st_1, \cdots, st_n$, and the string $st = st_1 \| st_2 \| \cdots \| st_n$, the function $get(st, i)$, where $1 \leq i \leq n$, returns the i-th string in st. Given a tuple t which consist of n elements, we use t_i, where $1 \leq i \leq n$, to denote the i-th element of tuple t.

Figure 4 illustrates the steps of this phase. First, the querying user u sends his identifier id_u^{ACS}, the constraint d of the range query and his encrypted position $\mathsf{Enc}(\mathsf{pk}_{LBS}, (\mathsf{Enc}(\mathsf{pk}_H, p_u))$. u encrypts his position using two layers of encryption because the encrypted position is sent thought the ACS to the LBS provider, and the ACS knows the key sk_H to decrypt the SHE. The outer layer of encryption prevents the ACS from learning the position of u. Second, using the id id_u^{ACS}, the ACS retrieves the set of grantors of u, $Grantor_u$. Recall that each tuple t in $Grantor_u$ consists of two elements, a random number and a ciphertext. The ACS constructs a set C which contains the ciphertext of each tuple t in $Grantor_u$, i.e., t_2. The ACS sends C and the encrypted position of u to the LBS provider. Third, the LBS provider decrypts each ciphertext c in C and obtains the plaintext st, which consists of an identifier id concatenated with a random number $rand$. The LBS provider searches and retrieves the encrypted position and the encrypted public number corresponding to the identifier id. We use p_{id} and Z_{id} to denote the retrieved encrypted position and encrypted public number, respectively. Then using p_{id} and the encrypted position of u, the LBS provider computes the encrypted square distance between them, $\mathsf{Enc}(\mathsf{pk}_H, dist(p_{id}, p_u)^2)$. The LBS provider also searches and retrieves the encrypted name, corresponding to the number $rand$. We denote this ciphertext by $name_{rand}$. Then, it creates a tuple e which contains three elements: $name_{rand}, Z_{id}$ and $\mathsf{Enc}(\mathsf{pk}_H, dist(p_{id}, p_u)^2)$, and adds e to the result set Res. Fourth, the LBS provider sends the set Res to the ACS. Fifth, for each tuple $e \in Res$, the ACS decrypts the element e_3, i.e., the encrypted square distance. If the decrypted distance is less or equal than d^2, the ACS (1) decrypts e_1 and e_2,

i.e., the outer layer of the encrypted name and the encrypted public number, (2) creates a tuple l containing the decrypted information, and (3) adds l to the set of answers *Ans*. Then the ACS sends *Ans* to u. Finally, for each tuple l in *Ans*, u (1) uses l_2, i.e., the public number, and his secret number η_u to compute the shared key $\mathsf{k}_{shared} \equiv {l_2}^{\eta_u} \pmod{p}$, and (2) decrypts l_1, i.e., the usernames , using k_{shared}. The decrypted names correspond to users that fulfill the query condition.

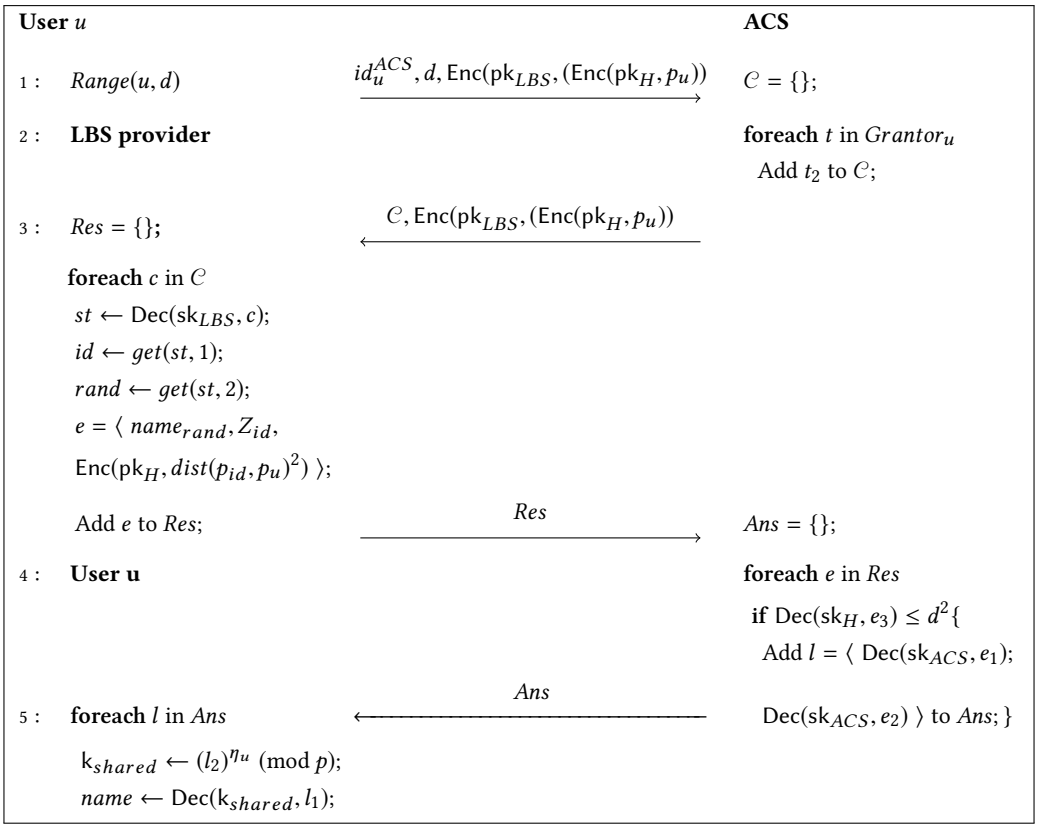

Fig. 4. Query Phase - *basic 2lSE*

4.2 Basic two-layer attribute-based encryption (*basic 2lABE*)

To facilitate the understanding, before describing the *basic 2lABE* approach, we specify the differences between our two approaches.

With the *basic 2lSE*, the LBS provider stores, for each user u, $|Grantee_u|$ copies of the encrypted name of u. Each ciphertext is generated using a shared key between u and each v in $Grantee_u$. The DH protocol is used to generate shared keys. In contrast, with the *basic 2lABE* there is no need of storing multiple copies of the encrypted name or sharing keys. Instead, each user encrypts his name using CP-ABE, Definition 3.4. Because of the properties of CP-ABE, each user receives a secret key based on his set of attributes, and only users who fulfill the access policy can decrypt a given ciphertext. In Sections 4.2.2 and 4.2.3, we specify the access policy and the set of attributes used for the encryption and decryption process.

Next, we describe the initialization, registration, access request, and query phases of the *basic 2lABE*. For brevity, we only describe the differences between the *basic 2lABE* and *basic 2lSE* schemes.

4.2.1 Initialization phase. This phase differs from that of the *basic 2lSE*, Section 4.1.1, only in the selection of the parameters p and g of the DH protocol. Since the *basic 2lABE* uses CP-ABE, there is no need of using the DH protocol for generating and sharing keys. Hence, the key authority generates a public key pk_{ABE} and a master key mk_{ABE}. When a user u registers in the system, the key authority uses the master key mk_{ABE} to generate a secret key for u. Then, it sends to u, among other information, the generated secret key and the public key pk_{ABE}. The key authority keeps mk_{ABE} secret from all entities.

4.2.2 Registration phase. Before explaining the steps of this phase, let us analyze how CP-ABE works in our scenario. The key authority has to generate a secret key for each user u using attributes associated with u. In our scenario, we consider only the attribute *name* of the users. That is because the access policy used in this work considers only the names of the users to establish who can decrypt a given ciphertext c. Otherwise, if other attributes are used, they just have to be included in the set of attributes associated with the users. Given a user u, the access policy defined by u, γ_u, is a disjunction of terms of the form (*name=value*), where *name* is a user attribute, and *value* refers to an atomic value.

Example 4.4. Consider a user u and a message m, which u wants to encrypt and allow to all users in the set $Grantee_u$ to decrypt it. u can use the usernames to (1) identify the users in his set $Grantee_u$, and (2) generate the access policy γ_u needed to encrypt m. Assume that $Grantee_u$ consists of users v and w. u can specify the access policy $\gamma_u = \big((name = v) \vee (name = w)\big)$ and generate the ciphertext $c = \mathsf{Enc}(\mathsf{pk}_{ABE}, m_{\gamma_u})$. γ_u indicates that c can be decrypted by users whose name is v or w. The secret keys of users v and w are generated based on their attributes, i.e., for instance, the secret key of v, sk_{ω_v} is generated based on the set of attributes $\omega_v = \{name = v\}$. Consequently, only users whose secret key fulfill the access policy γ_u will be able to decrypt c.

This phase differs from that of the *basic 2lSE*, Section 4.1.2, in the following: First, the key authority uses the master key mk_{ABE} and the set of attributes of the registering user u, $\omega_u = \{name = u\}$ to generate the secret key sk_{ω_u}. Second, the key authority, instead of sending to u the parameters p and g, sends the the keys pk_{ABE} and sk_{ω_u}. Third, u does not need to (1) select a secret number η_u and (2) compute and store at the LBS provider the encrypted public number part of the DH protocol. Instead, u stores at the LBS provider, apart from his identifier id_u^{LBS} and his encrypted position, his encrypted name $\mathsf{Enc}(\mathsf{pk}_{ACS}, \mathsf{Enc}(\mathsf{pk}_{ABE}, u_{\gamma_u}))$, with access policy $\gamma_u = \big((name = u)\big)$. Note that the access policy γ_u specifies that only u can decrypt his encrypted name. That is because, u has not authorized any access request, yet.

4.2.3 Access request phase. This phase differs from that of the *basic 2lSE*, Section 4.1.3, in the following: First, since the *basic 2lABE* does not use the DH protocol, user u does not need to send his public number Z_u, and user v does not need to compute the shared key k_{uv}. Instead, v has to update his access policy γ_v by adding to the disjunction the term "$(name = u)$". Second, v does not need to generate the random number $r_{u_v}^{LBS}$. That is because, the LBS provider stores, together with the identifier id_v^{LBS}, a single ciphertext containing the encrypted name of v. Then, it is not necessary to have an index, i.e., $r_{u_v}^{LBS}$, to retrieve the encrypted name of v that u can decrypt. Therefore, v selects only one random number, $r_{u_v}^{ACS} \in \mathbb{Z}$, instead of two. As a consequence, v does not have to include $r_{u_v}^{LBS}$ in the sets $Grantor_u$ and $Grantee_v$. That is, the tuple t added to the set $Grantor_u$ is $t = \big\langle r_{u_v}^{ACS}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS})\big\rangle$, instead of $t = \big\langle r_{u_v}^{ACS}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS} \| r_{u_v}^{LBS})\big\rangle$, and the ciphertext c added to the set $Grantee_v$ is $c = \mathsf{Enc}(\mathsf{k}_v, u \| id_u^{ACS} \| r_{u_v}^{ACS})$, instead of $c =$

$\mathsf{Enc}(\mathsf{k}_v, u \| id_u^{ACS} \| r_{u_v}^{ACS} \| r_{u_v}^{LBS})$. Third, v encrypts and updates at the LBS provider his name using, for the inner layer of encryption, the public key pk_{ABE} and the access policy γ_v, instead of using the shared key k_{uv} for each $u \in Grantee_v$.

4.2.4 Query phase. This phase differs from that of the *basic 2lSE*, Section 4.1.4, in the following: First, the set of ciphertexts C that the LBS provider receives from the ACS contains only encrypted identifiers, instead of encrypted identifiers concatenated with random numbers. Second, the LBS provider does not need to retrieve the public numbers, and it does not use the decrypted random numbers to retrieve encrypted names. Instead, it only uses the decrypted identifiers to retrieve the encrypted positions and encrypted names. Third, the set of tuples that the ACS receives from the LBS provider does not contain encrypted public numbers. So it does not need to decrypt them. Fourth, the querying user u does not need to compute shared keys to decrypt the ciphertexts received as query answer. Instead, u uses his secret key sk_{ω_u}.

4.3 Extending the basic schemes

Having the basic schemes, *basic 2lSE* and *basic 2lABE*, we now show how to extend them to meet our secrecy guarantees $G_{position}$, $G_{distance}$ and $G_{authorization}$ under our actual adversary model defined in Section 2.2. That is, we consider the twofold-composition strategy in which the protocol is executed twice. The only problem of the basic schemes under the twofold-composition strategy relies on the revocation process. With both approaches, a revoked user, in case of collusion with the LBS provider could gain access to unauthorized information as explained in Examples 4.5. In case of collusion with the ACS, there is no information leakage, as we prove in Section 5.

Example 4.5. Consider a user u who has a single grantor v. Assume that u colludes with the LBS provider. Assume now that u sends a range query. As part of the query processing, the LBS provider receives from the ACS encrypted information containing the identifier id_v^{LBS}. Assume further that the LBS provider sends the identifier id_v^{LBS} to u (as part of the collusion). Now u knows that id_v^{LBS} is the identifier of user v. Next, assume that v revokes access to u. However, u can regain access by following himself the steps of the access request phase. Specifically, with the *basic 2lSE*, u can follow himself all the steps of the access request phase that the grantor v should execute, Section 4.1.3. With the *basic 2lABE*, u can follow himself the steps of the access request phase required to store information at the ACS, Section 4.2.3. However, u cannot update the ciphertext containing the encrypted name of user v, which is stored at the LBS provider. That is because, u does not know the access policy of v, γ_v. Nevertheless, with the information added at the ACS, during querying processing, u will receive the ciphertext corresponding to the encrypted name of v, which u cannot decrypt but he knows it corresponds to v.

4.3.1 Extended two-layer symmetric encryption (2lSE). If a revoked user colludes with the LBS provider, we need the ACS to detect the attack and raise an alarm. To do so, we extend the *basic 2lSE* as follows: First, in the initialization phase, in addition to the steps of the *basic 2lSE*, Section 4.1.1, the key authority generates a pair of keys $(\mathsf{pk}_{ACS'}, \mathsf{sk}_{ACS'})$, and sends $\mathsf{sk}_{ACS'}$ to the *ACS*. The key authority keeps for itself the key $\mathsf{pk}_{ACS'}$. Second, in the registration phase, the key authority sends to each registering user v, in addition to the information sent with the *basic 2lSE*, Section 4.1.2, a ciphertext containing his identifier encrypted using two layers of encryption. The key for the inner layer is pk_{LBS} and the one for the outer layer is $\mathsf{pk}_{ACS'}$. That is, each user v receives, additionally, the ciphertext $\mathsf{Enc}(\mathsf{pk}_{ACS'}, (\mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS})))$. Third, in the access request phase, the tuple t that the grantor v sends to the ACS changes from $t = \langle r_{u_v}^{ACS}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS} \| r_{u_v}^{LBS}) \rangle$ to $t = \langle r_{u_v}^{ACS}, \mathsf{Enc}(\mathsf{pk}_{ACS'}, (\mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS}))), \mathsf{Enc}(\mathsf{pk}_{LBS}, r_{u_v}^{LBS}) \rangle$. Next, the ACS uses the key $\mathsf{sk}_{ACS'}$ to decrypt the element t_2, replaces t_2 with the decrypted information and adds t to the

set $Grantor_u$. Since v is the only user who knows the ciphertext $\mathsf{Enc}(\mathsf{pk}_{ACS'}, (\mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS})))$, the ACS knows that the access request has been registered by the actual grantor and not by a revoked user who has colluded with the LBS provider. In other words, even if a user u learns the identifier of one of his grantors v, in case of revocation, u cannot regain access because only v knows his ciphertext $c = \mathsf{Enc}(\mathsf{pk}_{ACS'}, (\mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS})))$ and only the key authority knows the key $\mathsf{pk}_{ACS'}$ to generate c. We assume that the decryption algorithm Dec, used by the ACS, indicates successful decryption. That is, the decryption $\mathsf{Dec}(\mathsf{sk}_{ACS'}, \mathsf{Enc}(\mathsf{k}', c))$, where c is a ciphertext, is called successful if $\mathsf{sk}_{ACS'} = \mathsf{k}'$. One can implement a successful decryption algorithm by concatenating the hash value $\mathsf{H}(c)$ to the encryption $\mathsf{Enc}(\mathsf{k}', c\|\mathsf{H}(c))$ and checking this relation in the decryption algorithm [13]. Fourth, in the query phase, the set of ciphertexts C that the ACS sends to the LBS provider is different from that of the *basic 2lSE*, Section 4.1.4. With the *basic 2lSE*, C is a set of ciphertexts, where each ciphertext contains an identifier concatenated with a random number. Here, C is a set of tuples, where each tuple contains two ciphertexts. Specifically, for each tuple t in the set $Grantor_u$, the ACS adds to C a tuple consisting of the elements t_2 and t_3, i.e., the encrypted identifier and the encrypted random number. The LBS provider has to decrypt both ciphertexts and proceed with the query processing as with the *basic 2lSE*.

4.3.2 Extended two-layer attribute-based encryption (2lABE). To solve the existing problems when a revoked user colludes with the LBS provider, we use the same technique as that of the *2lSE*, Section 4.3.1. That is, the ACS, during the access request phase, receives the following ciphertext: $\mathsf{Enc}(\mathsf{pk}_{ACS'}, (\mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS})))$, and verifies that the actual grantor has sent the received ciphertext. The query processing phase is similar to that of the *basic 2lABE*, Section 4.2.4.

5 SECRECY PROOFS

This section provides the secrecy analysis of our proposed approaches. The proofs are organized as follows: For each approach, we first prove that, under our adversary model defined in Section 2.2, the approach provides the secrecy guarantee $G_{position}$, then the secrecy guarantee $G_{distance}$, and finally the secrecy guarantee $G_{authorization}$.

5.1 Secrecy Proofs of the *2lSE* approach

For each of the proofs in this section, we first consider the case that a user colludes with the LBS provider and then the case that a user colludes with the ACS, during the first execution of the protocol. During the second execution of the protocol, the user is not allowed to collude anymore (twofold-composition strategy). The case where each party individually acts as an adversary is straightforward and therefore omitted. Given two entities A and B, we study the case where entity A colludes with B, and omit the case where B colludes with A. That is because, in the case of collusion of two entities A and B, we assume that either A or B has access to all information and functionality of A and B together.

Before starting with the proofs, let us recall the information that the users, LBS provider, and ACS have. This information will be used in the proofs. First, each user s knows: his identifiers id_s^{LBS}, id_s^{ACS}, the keys k_s, pk_H and pk_{ACS}, his encrypted identifier $\mathsf{Enc}(\mathsf{pk}_{ACS'}, (\mathsf{Enc}(\mathsf{pk}_{LBS}, id_s^{LBS})))$, and the parameters p, g, η_s. Second, the LBS provider stores: the user identifiers, the encrypted positions, the encrypted public numbers, the encrypted names together with the random numbers, and the secret key sk_{LBS}. Additionally, the LBS provider knows the relations existing between the information that it stores, e.g., the encrypted position that corresponds to a given identifier. Third, the ACS stores: the encrypted sets of grantors and grantees of all the users, and the keys sk_H, sk_{ACS} and $\mathsf{sk}_{ACS'}$. Additionally, the ACS knows the relations existing between the stored information.

Lemma 5.1. *The 2lSE approach guarantees that, in the presence of adversaries with the characteristics defined in our adversary model, Section 2.2, the physical positions of users are kept secret from any adversary.*

Proof. *Case 1: A user s colludes with the LBS provider.* During the first protocol execution, s has access to the information that he owns and the one stored at the LBS provider. However, none of the information that s has can be used to decrypt the encrypted positions. The decryption key sk_H is known only by the ACS, but according to our adversary model, s cannot collude with both entities at the same time. Furthermore, SHE, which is used to encrypt the physical positions of the users, is secure against indistinguishability chosen-plaintext attacks (IND-CPA), i.e., an adversary cannot learn any useful information from the encrypted data.

In addition to the information that s has learned, s can execute queries and get their respective answers. We now show that s cannot use this information to learn the physical positions of other users. During query processing, the LBS provider receives from the ACS a set containing the encrypted identifiers of the grantors of s. So, s knows that the set of identifiers used by the LBS provider to process his query corresponds to his grantors. The identifiers are random numbers, so s cannot link the identifiers with their owners, except if $|Grantor_s| = 1$. In this last case, s can learn the identifier of his unique grantor, namely u, id_u^{LBS}. However, id_u^{LBS} does not leak any information about the position of u or cannot be used to decrypt his encrypted position. Next, during query processing, s receives a set of encrypted names and public numbers corresponding to the grantors of s that fulfill the query condition. None of this information can be used to decrypt the encrypted positions. Then, s cannot learn the position of other users during the first protocol execution.

During the second protocol execution, since s is not allowed to collude anymore with any entity, s only has access to information that he owns, information that he has learned during the first protocol execution, and the one that he will obtain during the second protocol execution. However, during the second protocol execution, s does not learn any new information. Then, as in the first execution, this information is not enough to decrypt and learn the physical positions of other users. *Case 2: A user s colludes with the ACS.* During the first protocol execution, s has access to the information that he owns and the one stored at the ACS. Although s knows the key sk_H to decrypt the encrypted positions, s does not have the ciphertext to decrypt them because they are stored at the LBS provider. Since, according to our adversary model, s cannot collude with both entities at the same time, s cannot learn the physical positions of other users.

In addition to the information that s has learned, s can execute queries and get their respective answers. We now show that s cannot use this information to learn the physical positions of other users. During query processing, the ACS receives from the LBS provider a set containing the encrypted names, encrypted public numbers, and encrypted distances corresponding to the grantors of s. s can decrypt all the received information. However, none of this information contains the encrypted positions or the position of any user.

Next, during the second protocol execution, since s is not allowed to collude anymore with any entity, s only has access to information that he owns, information that he has learned during the first protocol execution, and the one that he will obtain during the second protocol execution. Similar to the first execution, this information is not enough to get the encrypted positions, and so s cannot learn the physical positions of other users.

Consequently, the *2lSE* approach guarantees that, in the presence of adversaries with the power defined in our adversary model, the physical positions of users are kept secret from any adversary. □

Lemma 5.2. *Given a user u, the 2lSE approach guarantees that, in the presence of adversaries with the characteristics defined in our adversary model, Section 2.2, only entities they themselves have authorized can learn the distance between their physical position and the one of u.*

Proof. *Case 1: A user s colludes with the LBS provider.* During the first protocol execution, s has access to the information that he owns and the one stored at the LBS provider. Knowing all this information, as proved in Lemma 5.1, s cannot learn the physical positions of other users. Thus, s cannot use the positions to learn the distance between his position and the one of the users who have not authorized him to do so.

In addition to the information that s has learned, s can execute queries and get their respective answers. We now show that s cannot use this information to learn the distance between his position and the one of the users who have not authorized him to do so. As explained in the proof of Lemma 5.1, during query processing, if $|Grantor_s| = 1$, s can learn the identifier of his unique grantor, namely u, id_u^{LBS}. However, id_u^{LBS} does not leak any information about the physical positions or the distance between users. Additionally, by following the query processing protocol, s receives a set of encrypted names and public numbers corresponding to the grantors of s that fulfill the query condition. Since the information received belongs to the grantors of s, s is allowed to learn such information. Next, assume that the LBS provider deviates from the query processing protocol, as part of the collusion. That is, the LBS provider processes the query with different identifiers from the ones sent by the ACS. So, instead of using the information that belongs to the grantors of s, the LBS provider selects different identifiers and random numbers to give s information that he is not authorized to access. However, the identifiers are random numbers, and neither s nor the LBS provider knows the relationship between identifier and users. Next, if the LBS provider processes the query with identifiers selected at random, s receives as query result a set of encrypted names and public numbers. However, the names are encrypted using probabilistic encryption, and s does not know the key to decrypt them. Moreover, by using the received public numbers, and the parameters p and g, s cannot compute the decryption key because of the security offered by the DH assumption, which has been proven to be computationally infeasible in [6]. Since s cannot compute the key and probabilistic encryption is secure against IND-CPA, s cannot learn, in the first protocol execution, the distance between his position and the one of users that he is not authorized.

Next, during the second protocol execution, since s is not allowed to collude anymore with any entity, s only has access to information that he owns, information that he has learned during the first protocol execution, and the one that he will obtain during the second protocol execution. Assume that in the second protocol execution, u revokes access to s. Then s can try himself to add id_u^{LBS} to his set $Grantor_s$ following the steps of the access request phase. However, s needs to send the ciphertext $c = \mathsf{Enc}(\mathsf{pk}_{ACS'}, (\mathsf{Enc}(\mathsf{pk}_{LBS}, id_u^{LBS})))$ to the ACS. Since s does not know c or the key $\mathsf{pk}_{ACS'}$ to generate c, s cannot add id_u^{LBS} in his set $Grantor_s$. Therefore, in the case of revocation, s cannot regain access.

Case 2: A user s colludes with the ACS. During the first protocol execution, s has access to the information that he owns and the one stored at the ACS. Knowing all this information, as proved in Lemma 5.1, s cannot learn the physical positions of other users. Thus, s cannot use the positions to learn the distance between his position and that of the users who have not authorized him to do so.

In addition to the information that s has learned, s can execute queries and get their respective answers. We now show that s cannot use this information to learn the distance between his position and the one of the users who have not authorized him to do so. By following the query processing protocol, s receives a set of encrypted names and public numbers corresponding to the grantors of s that fulfill the query condition. Since the information received belongs to the grantors of s, s is allowed to learn that information. Next, assume that the ACS deviates from the query processing

protocol, as part of the collusion. That is, the ACS includes in the set $Grantor_s$, information from the other sets of grantors to give access to s to information that he is not authorized to access. So, s receives as query answer, a set of encrypted names and public numbers. Based on the same arguments as the ones presented in the Case 1, s cannot compute the decryption key and cannot learn any information from the encrypted names. Then, during the first protocol execution, s cannot learn the distance between his position and the one of the users who have not authorized him.

During the second protocol execution, since s is not allowed to collude anymore with any entity, s only has access to information that he owns, information that he has learned during the first execution, and the one that he will obtain during the second protocol execution. Different from the case 1, after query processing, s cannot learn any extra information, i.e., s has access to the same information as in the first protocol execution. Using the same arguments as in the first execution, s cannot learn the distance between his position and the one of the users who have not authorized him. So, in the case of revocation, by using the information that s knows, s cannot regain access.

Consequently, given a user u, the *2lSE* approach guarantees that, in the presence of adversaries with the power defined in our adversary model, only entities they themselves have authorized can learn the distance between their physical position and the one of u. □

Lemma 5.3. *Given two users u and v, the 2lSE approach guarantees that, in the presence of adversaries with the characteristics defined in our adversary model, Section 2.2, an adversary will not be able to learn whether u or v are allowed to learn the distance between them.*

Proof. *Case 1: A user s colludes with the LBS provider.* During the first protocol execution, s has access to the information that he owns and the one stored at the LBS provider. First, the user identifiers are random numbers, and by using all the known information, neither the LBS provider nor s can determine the link between users and identifiers. Second, the information that s knows does not contain any link between grantees and grantors. Then, s cannot determine whether two given users u and v are allowed to learn the distance between them.

In addition to the information that s has learned, s can execute queries and get their respective answers. We now show that s cannot use this information to learn whether two given users u and v are allowed to learn the distance between them. As explained in the proof of Lemma 5.1, during query processing, if $|Grantor_s| = 1$, s can learn the identifier of his unique grantor, namely u, id_u^{LBS}. However, given a set of queries, neither s nor the LBS provider knows who are the querying users. Therefore, if the identifier id_u^{LBS} is used during query processing, s cannot determine who are the grantees of u. Additionally, during query processing, s receives a set of encrypted names and public numbers corresponding to the grantors of s that fulfill the query condition. However, the received information does not reveal any data about the grantees and grantors of other users. Then, given two users u and v, during the first protocol execution, s is not able to learn whether u or v are allowed to learn the distance between them.

During the second protocol execution, since s is not allowed to collude anymore with any entity, s only has access to information that he owns, the one that he has learned during the first execution, and the one that he will obtain during the second protocol execution. Since the information that s gets in the second protocol execution is similar to the one of the first execution, this information is not enough to learn whether u or v are allowed to learn the distance between them.

Case 2: A user s colludes with the ACS. During the first protocol execution, s has access to the information that he owns and the one stored at the ACS. First, the set of grantors is encrypted using asymmetric encryption and the key pk_{LBS}. Since s does not know pk_{LBS} and asymmetric encryption is secure against IND-CPA, s cannot learn any useful information from the encrypted data. Second, the set of grantees are encrypted using probabilistic encryption and the key of its

owner. Since s knows only the key to decrypt his set of grantees, and probabilistic encryption is secure against IND-CPA, s cannot learn any useful information from the encrypted data.

In addition to the information that s has learned, s can execute queries and get their respective answers. Similar to the case 1, the information that s received during query processing does not reveal any data about the grantees and grantors of other users. Next, during the second protocol execution, since s is not allowed to collude anymore with any entity, s only has access to information that he owns, the one that he has learned during the first protocol execution, and the one that he will obtain during the second execution. Similar to the case 1, s has access to the same information as in the first protocol execution. Therefore, using the same arguments as in the first protocol execution, s cannot learn whether u or v are allowed to learn the distance between them.

Consequently, given two users u and v, the *2lSE* approach guarantees that, in the presence of adversaries with the power defined in our adversary model, an adversary will not be able to learn whether u or v are allowed to learn the distance between them. □

5.2 Secrecy Proofs of the *2lABE* approach

Lemma 5.4. *The 2lABE approach guarantees that, in the presence of adversaries with the characteristics defined in our adversary model, Section 2.2, the physical positions of users are kept secret from any adversary.*

Lemma 5.5. *Given a user u, the 2lABE approach guarantees that, in the presence of adversaries with the characteristics defined in our adversary model, Section 2.2, only entities they themselves have authorized can learn the distance between their physical position and the one of u.*

Lemma 5.6. *Given two users u and v, the 2lABE approach guarantees that, in the presence of adversaries with the characteristics defined in our adversary model, Section 2.2, an adversary will not be able to learn whether u or v are allowed to learn the distance between them.*

The proofs of Lemmas 5.4 - 5.6 are analogous to the proofs of Lemmas 5.1 - 5.3, respectively. That is because, although the *2lSE* and the *2lABE* approaches have some differences, the properties of the *2lSE* used for the proofs also hold for *2lABE*. First, both approaches differ in the encryption scheme used to encrypt the usernames. The *2lABE* uses CP-ABE. CP-ABE is secure against IND-CPA [5]. Then an adversary cannot learn any useful information from the encrypted names. Second, different from the *2lSE*, the *2lABE* does not use the DH protocol to share keys between users. Instead, each user receives, as part of the CP-ABE, a secret key for decryption. A user can decrypt a ciphertext c only if the attributes used to generate his secret key satisfies the access policy used to generate c. That is, a user can decrypt only usernames that belong to his grantors. Then users cannot decrypt ciphertexts that they are not authorized to.

6 TIME COMPLEXITY ANALYSIS

A complexity analysis is helpful to predict the behavior of the *2lSE* and *2lABE* approaches and to facilitate meaningful comparisons. An average complexity analysis depends on the internal behavior of the database, which is specific to the product and is not openly available. Furthermore, if there are changes in the system settings, the average analysis is void. So our complexity analysis is a worst case analysis. Here, we focus on the complexity of the query phase because this is the most frequently used phase in our scenario.

6.1 Time Complexity Analysis of the *2lSE* and *2lABE* approaches

Since we are performing a worst case complexity analysis, we consider that all the users in the set of grantors of the querying user u are located within the query range. That is because the

number of operations at the user-side depends on the number of grantors of u that are inside the query range. Furthermore, to perform the complexity analysis of our approaches, one needs to specify the complexity of the encryption/decryption process, which depends on the type of encryption/decryption algorithm used. In this sense, we select the following well-known algorithms from the literature: For symmetric and asymmetric encryption schemes, we use AES and RSA, respectively. For SHE, we use algorithms based on the learning with errors problem over rings, such as the ones presented in [11, 23]. For CP-ABE, we select the algorithm presented in [19].

Let A be the total number of authorized access requests, i.e., $A = \sum_{u \in U} |Grantor_u|$, $B(p_u)$ be the bit string representation of the physical position p_u, and $|B(p_u)|$ its length. Further, let TC_{user}, TC_{ACS}, and TC_{LBS} denote the time complexity at the user-side, the ACS and the LBS provider, respectively. Next, a ciphertext generated using SHE is represented as a matrix M. We use $|M|$ to denote the size of the matrix. Additionally, the initialization of the RSA algorithm requires to select at random two large primes. We use N to denote the product of the selected two primes. Next, note that p and η_u are the integer and the secret number of a given user u, respectively, which are part of the DH protocol used by the *2lSE* approach.

Lemma 6.1. *Given a range query Range* $= (u, d)$*, the time complexities of the 2lSE approach at the user-side, the ACS, and the LBS provider are:*

$$\begin{aligned} TC_{user} &= \mathcal{O}\big(|Grantor_u| \cdot \log(p)^2 \cdot \log(\eta_u)\big) \\ TC_{ACS} &= \mathcal{O}\big(|Grantor_u| \cdot (|B(p_u)| \cdot \log(|B(p_u)|) + \log(N)^3)\big) \\ TC_{LBS} &= \mathcal{O}\big(|Grantor_u| \cdot |M|^3\big) \end{aligned} \tag{1}$$

Proof. We start by analyzing the complexity of the encryption/decryption process of each of the encryption schemes used in the *2lSE* approach, i.e., symmetric, asymmetric encryption, and SHE. First, the complexity of symmetric encryption schemes, specifically AES, depends on the length of the message m to be encrypted, $|m|$, [26]. Then, the complexity of the encryption/decryption process is $\mathcal{O}(|m|)$. In our scenario, we use symmetric encryption to encrypt the usernames. We consider that the length of the string that represents a username has a constant size of 12 bytes. Therefore, the complexity of the encryption/decryption process using symmetric encryption reduces to $\mathcal{O}(1)$.

Second, the complexity of asymmetric encryption schemes, specifically RSA, is based on the complexity of modular exponentiation. That is because, using RSA, given a message m, the resulting ciphertext c is $c = m^e \pmod N$, and the decryption of c is $m = c^d \pmod N$, where e is the public key and d is the secret key. Given the integer numbers B, C, N, the complexity of the modular exponentiation $B^C \pmod N$ is $\mathcal{O}(\log(N)^2 \cdot \log(C))$. We consider that the exponent e in the RSA algorithm, i.e., the public key, is fixed, as specified in FIPS-186-3 [22]. So the encryption complexity using RSA is $\mathcal{O}(\log(N)^2)$. Next, using standard RSA assumptions, the exponent d in the RSA algorithm, i.e., the secret key, has size in bits close to that of N. Then, the decryption complexity using RSA is $\mathcal{O}(\log(N)^3)$.

Third, in SHE schemes [11, 23], the encryption/decryption process depends on modular multiplication and addition of vectors. Given a message m, using SHE, the complexity of encrypting/decrypting m is $\log(|B(m)|)$ per bit [8], assuming the use of the Montgomery multiplication, which is one of the fastest methods available for performing modular multiplication. Then, the encryption/decryption complexity of SHE schemes is $\mathcal{O}(|B(m)| \cdot \log(|B(m)|))$.

The following steps are required to compute a given range query with the *2lSE* approach.

(1) Encrypt using SHE the position of the querying user u. The user executes this step. The complexity of this step is $\mathcal{O}(|B(p_u)| \cdot \log(|B(p_u)|))$.

(2) Use the identifier of the querying user u, id_u^{ACS} to retrieve the set of grantors $Grantor_u$. The ACS executes this step. We assume the ACS uses B-tree indexing. Then, the complexity of this step is $\mathcal{O}(\log(|U|))$.
(3) Decrypt the set of encrypted identifiers and the set of encrypted random numbers sent by the ACS. Since both sets have a size of $|Grantor_u|$ and each element of these sets is encrypted using the asymmetric encryption, the LBS provider has to execute $2 \cdot |Grantor_u|$ asymmetric decryptions. The complexity of this step is $\mathcal{O}(|Grantor_u| \cdot \log(N)^3)$.
(4) For each decrypted identifier in step 3, retrieve the corresponding encrypted position and encrypted public value. The LBS provider does this step. We assume the LBS provider uses B-tree indexing. Then, the complexity of this step is $\mathcal{O}(|Grantor_u| \cdot \log(|U|))$.
(5) Compute the encrypted square distances between the querying user and the ones retrieved in step 4. The LBS provider executes this step. Computing the encrypted square distance requires three homomorphic additions and two multiplications. Since a ciphertext generated using SHE is represented as a matrix, adding and multiplying two ciphertexts imply addition and multiplication of two matrices, respectively, [31]. Then, the addition and multiplication processes have a complexity of $\mathcal{O}(|M|^2)$ and $\mathcal{O}(|M|^3)$, respectively. Since $\mathcal{O}(|M|^3)$ dominates $\mathcal{O}(|M|^2)$, the complexity of computing one encrypted square distance is $\mathcal{O}(|M|^3)$. The complexity of this step is $\mathcal{O}(|Grantor_u| \cdot |M|^3)$.
(6) Retrieve from the set of encrypted names stored at the LBS provider the encrypted names corresponding to each of the random numbers decrypted in step 3. The LBS provider executes this step. The complexity of this step is $\mathcal{O}(|Grantor_u| \cdot log(A))$.
(7) Decrypt the encrypted square distances, the second layer of encryption of the encrypted names, and the encrypted public numbers sent by the LBS provider. The ACS executes this step. The total number of ciphertexts that the ACS has to decrypt is $3 \cdot |Grantor_u|$. However, the square distances are encrypted using SHE, and the names and public numbers are encrypted using asymmetric encryption. Then, the ACS has to perform $|Grantor_u|$ SHE decyptions and $2 \cdot |Grantor_u|$ asymmetric decryptions. Then, the complexity of this step is $\mathcal{O}(|Grantor_u| \cdot (|B(p_u)| \cdot \log(|B(p_u)|) + \log(N)^3))$.
(8) Compute the shared keys and decrypt the encrypted names corresponding to users that fulfill the query condition. The user does this step. The total number of shared keys that u has to compute is $|Grantor_u|$. The shared keys are generated using the DH protocol, which requires modular exponentiation. Given a public number of a user v, Z_v, and the secret number of u, η_u, the complexity of the modular exponentiation ${Z_v}^{\eta_u} \pmod p$ is $\mathcal{O}(\log(p)^2 \cdot \log(\eta_u))$, where p is the parameter of the DH protocol. The number of decryptions performed are $|Grantor_u|$, where each decryption has a complexity of $\mathcal{O}(1)$. Then the complexity of this step is $\mathcal{O}(|Grantor_u| \cdot \log(p)^2 \cdot \log(\eta_u))$.

By considering the step with the highest complexity that is performed by each entity of the system, one can easily construct the terms of Equation 1. □

Before studying the complexity of the *2lABE*, we introduce further notation: Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic groups of the same order, where $\mathbb{G}$ and $\mathbb{G}_T$ are the groups selected during the initialization of the CP-ABE scheme. Further, let e be a bilinear map of the form $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. The encryption/decryption process using CP-ABE is based on bilenear mappings and operations in the groups $\mathbb{G}$ and $\mathbb{G}_T$. We use C_e and $C_{\mathbb{G}_T}$ to denote the complexity of computing e, and the complexity of performing an operation in the group $\mathbb{G}_T$, respectively.

Lemma 6.2. *Given a range query Range = (u, d), the time complexities of the 2lABE approach at the user-side, the ACS, and the LBS provider are:*

$$\begin{aligned} TC_{user} &= \mathcal{O}\big(|Grantor_u| \cdot (C_e + C_{\mathbb{G}_T})\big) \\ TC_{ACS} &= \mathcal{O}\big(|Grantor_u| \cdot (|B(p_u)| \cdot \log(|B(p_u)|) + \log(N)^3)\big) \\ TC_{LBS} &= \mathcal{O}\big(|Grantor_u| \cdot |M|^3\big) \end{aligned} \tag{2}$$

Proof. We start by analyzing the complexity of the encryption/decryption process of each of the encryption schemes used in the *2lABE* approach, i.e., asymmetric encryption, SHE, and CP-ABE. First, as shown in the proof of Lemma 6.1, the encryption and decryption complexity of asymmetric encryption schemes are $\mathcal{O}(\log(N)^2)$ and $\mathcal{O}(\log(N)^3)$, respectively, and given a message m, the encryption/decryption complexity of SHE is $\mathcal{O}(|B(m)| \cdot \log(|B(m)|))$. Third, the encryption complexity of CP-ABE schemes [19] depends on the size of the access policy and the cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$. The encryption costs using CP-ABE is $(n + 3) \cdot C_{\mathbb{G}} + 2 \cdot C_{\mathbb{G}_T}$, where n is the size of the access policy and $C_{\mathbb{G}}$ is the complexity of performing an operation in the group $\mathbb{G}$ [34]. In our scenario, the size of the access policy depend on the size of the set of grantees of each user. Then, the encryption complexity of CP-ABE is $\mathcal{O}(|Grantee_u| \cdot C_{\mathbb{G}} + C_{\mathbb{G}_T})$. The decryption cost using CP-ABE is $2 \cdot C_e + 2 \cdot C_{\mathbb{G}_T}$ [34]. Then, the decryption complexiy of CP-ABE is $\mathcal{O}(C_e + C_{\mathbb{G}_T})$.

The following steps are required to compute a given range query with the *2lABE* approach.

(1)-(3) Similar to the step 1-3 of the *2lSE*.

(4) Retrieve the encrypted position and encrypted name corresponding to each decrypted identifier in step 3. The LBS provider executes this step. The complexity of this step is $\mathcal{O}(|Grantor_u| \cdot \log(|U|)$.

(5) Similar to the step 5 of the *2lSE*.

(6) Decrypt the encrypted square distances and the second layer of encryption of the encrypted names sent by the LBS provider. The total number of ciphertexts that the ACS has to decrypt is $2 \cdot |Grantor_u|$, where the square distances are encrypted using SHE and the names are encrypted using asymmetric encryption. This step is done by the ACS. The complexity of this step is $\mathcal{O}(|Grantor_u| \cdot (|B(p_u)| \cdot \log(|B(p_u)|) + \log(N)^3))$.

(7) Decrypt the encrypted names corresponding to users that fulfill the query condition. The user executes this step. The number of decryptions performed are $|Grantor_u|$. Then the complexity of this step is $\mathcal{O}(|Grantor_u| \cdot (C_e + C_{\mathbb{G}_T}))$.

By considering the step with the highest complexity that is performed by each entity of the system, one can easily construct the terms of Equation 2. □

6.2 Discussion

From our worst case analysis, Lemmas 6.1 and 6.2, one can observe that both approaches differ only at one entity, namely the user-side. Their differences rely on the decryption process, which is related to the encryption scheme used. With the *2lSE*, the encryption/decryption complexity of symmetric schemes is $\mathcal{O}(1)$. However, the querying user has to perform a total of $|Grantor_u|$ exponentiations module p to calculate the shared key. This computation is not needed with the *2lABE*, since the user uses the same key to decrypt the received ciphertexts. However, the encryption/decryption complexity of CP-ABE schemes depends on the size of the access policy used to generate a given ciphertext and bilinear pairing operations, which are computationally expensive [34]. Next, because a complexity analysis considers only the dominating operations, a worst case analysis is not enough to determine the performance of an algorithm in practice. For instance, for the complexity at the LBS provider, one can observe that both approaches have the same time complexity. However, significant differences between both approaches can be found on the less dominant steps. With the

2lSE approach, the LBS provider has to perform $|Grantor_u|$ extra searches with complexity $\log(A)$, which will affect its performance. Therefore, we additionally perform experiments to validate our complexity analysis, evaluate how the differences between both approaches impact on their performance, and determine at the end which approach performs better in practice.

7 EXPERIMENTS

This section presents an experimental analysis of the performance of the *2lSE* and *2lABE* approaches.

7.1 Experiment Setup

7.1.1 Dataset and Query Sample. We use the Tokyo dataset [36] in our experiments. This dataset contains 573703 real check-ins, i.e., positions. We choose a sample of 1000 users at random. Next, we divide the users into ten equally classes. We generate authorized access requests such that all the users in each class have the same number of grantors. Specifically, we generate the following grantors sizes: 10, 25, 50, 100, 250, 500, 750, 1000, 2500, and 5000.

7.1.2 Encryption Algorithms. We use the following libraries for the implementation: Microsoft SEAL [29] for the homomorphic encryption and the Java Pairing-Based Cryptography Library together with the Cryptographic Packages javax.crypto and java.security for symmetric, asymmetric encryption and CP-ABE.

7.1.3 Evaluation Measures. For our evaluation, we considered the access request and query phases. All other phases are executed only once, at least with respect to one of the entities of the systems. In this sense, we consider six measures: the storage size, the access request time, the query processing time at the user-side, the query processing at the LBS provider, the query processing time at the ACS, and the total query processing time. Note that, the total query processing time is the sum of the query processing time at each of the entities.

7.2 Results

We now present our experiment results, which evaluate the performance of our schemes using the metrics defined in Section 7.1.3. Note that we do not consider the network-communication time.

Storage-Size: Figure 5(a) shows the total storage size occupied by each of the approaches. The blue and red colors represent the storage size at the ACS and LBS provider, respectively. The *2lABE* approach requires more storage capacity. However, the difference between the storage capacity of both approaches is minimal (2 percent in our scenario).

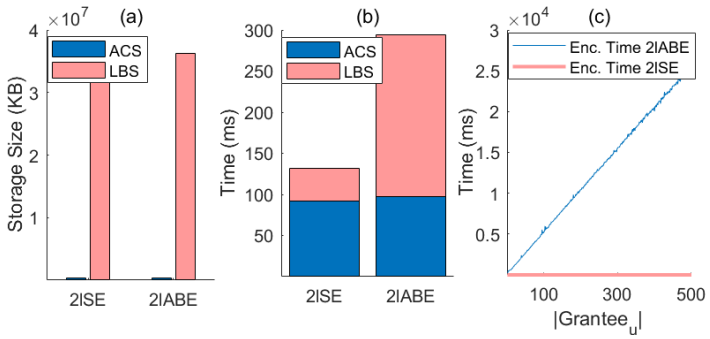

Fig. 5. Storage size, Access request time and Encryption time

Access request time: Given an authorized access request *accessReq*(u,v)=*true*, we evaluate the time required by the grantor v to (1) generate and add the information needed to the set $Grantor_u$ at the ACS, and (2) generate and store the information needed at the LBS provider, with each approach. Figure 5(b) shows the average access request time with both approaches for our sample. The blue and red colors indicate the time required by the grantors to generate and store the information at the ACS and LBS provider, respectively. Both approaches require the same amount of time to generate and store the corresponding information at the ACS (blue color). However, with the *2lABE* approach, the grantors require more time to generate and store the information at the LBS provider. That is because of the different encryption scheme used to encrypt the usernames which are stored at the LBS provider. One could think that our results are specific to the number of grantees that the user in our sample have. Next, we will show that our results apply to any number of grantees. As shown in the proof of Lemma 6.2, the encryption complexity using CP-ABE, which is used by the *2lABE* approach, depends, among others, on the number of grantees that a user has. Therefore, to analyze in depth, the effect of each encryption scheme, we select a user and increase his number of grantees, starting from 1 up to 500. Next, we measure the time required by the user to encrypt his username against the size of his set of grantees. Figure 5(c) shows the encryption time with both approaches. One can observe that even for a set of grantees with cardinality one, the *2lSE* approach performs better than the *2lABE*. Moreover, as expected from our complexity analysis, the encryption time with the *2lABE* grows linearly with the number of grantees, whereas with the *2lSE* the encryption time is constant. That is because, with the *2lSE*, each grantor generates a ciphertext for each of his grantees. Consequently, with the *2lSE*, the encryption time of each ciphertext is independent of the number of grantees that the grantor has. With the *2lABE*, although each grantor generates only one ciphertext which can be accessed by all of his grantees, the number of operations required to encrypt a ciphertext depends, among others, on the number of grantees, i.e., the encryption time grows linearly with the number of grantees.

Query time: Figures 6(a), 6(b), and 6(c) show the average query processing time for each of the ten classes at the ACS, the LBS provider, and the user, respectively. The query processing times of the LBS provider and ACS with the *2lSE* approach is greater than that of the ones with the *2lABE* approach. In contrast, the query processing time of the user with the *2lSE* approach is less than the one with the *2lABE*. This is explained as follows: First, with the *2lSE* approach, the LBS provider has to perform an additional search to recover the encrypted version of the name, which can be decrypted by the querying user. With the *2lABE*, there is only one encrypted version of the name for each user, which can be decrypted by all authorized users. Second, with the *2lSE* approach, the ACS has to decrypt for each user v in the query answer, apart from the second layer of the encrypted name of v and the encrypted square distance, the encrypted public number Z_{η_v}. With the *2lABE* approach, the ACS has to decrypt only the encrypted square distances and the second layer of the encrypted names. Third, with the *2lSE* approach, the user has to compute the shared key and decrypt the names which are encrypted using symmetric encryption, whereas with the *2lABE* the user has to decrypt the names which are encrypted using CP-ABE. The encryption/decryption process using CP-ABE is computationally more expensive than the one using symmetric encryption and even more expensive than computing the shared keys and decrypting. In fact, although, with the *2lABE*, the query processing time at two entities of the system, namely LBS provider and ACS, is less than the one with the *2lSE*, the performance of the *2lSE* on the user-side is that much high that it compensates, at the end, the advantages of the *2lABE*. As we can observe in Figure 6(d), the total average query processing time, i.e., the sum of the times required by the LBS provider, the ACS, and the user, with the *2lSE* approach is much less (approximately by a factor of 2) than the one with the *2lABE*.

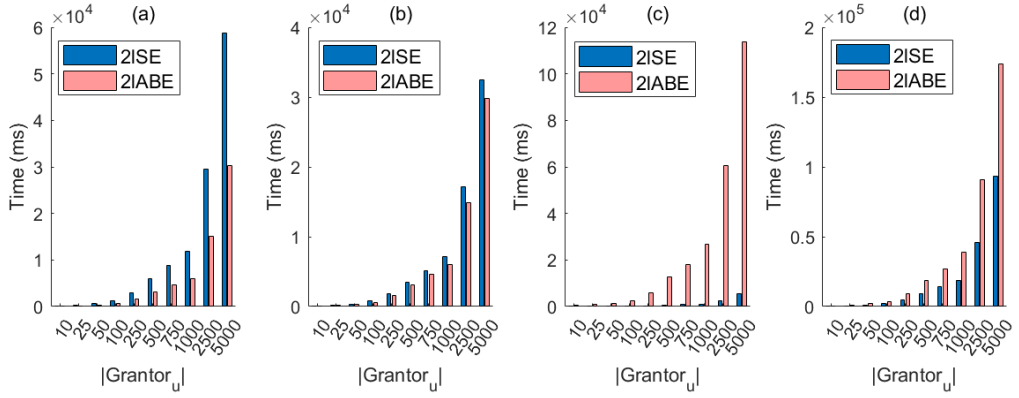

Fig. 6. Query Time

Discussion: One important difference between both approaches that affect query performance is the encryption scheme used to encrypt the usernames. In general, in our scenario, as our experimental results show, having multiple encrypted copies of a message (*2lSE*) is more efficient in terms of query performance. The additional search process needed with the *2lSE* approximately doubles the query processing time at the LBS provider side compare to the one with the *2lABE* approach. The opposite happens at the user-side, where the decryption time, with the *2lABE*, increases by more than twice compared to the one with the *2lSE*. However, the time contribution to the total query processing time of the LBS provider is less in comparison to that of the user. These differences are reflected in the total query processing time, where the *2lSE* approach is twice more efficient than the *2lABE*, which is impressive. Therefore, one can say that the 2 percent extra storage required by the *2lSE* pays off with better query performance. Our results not only apply to real online social networks like Orkut and LiveJournal, where the average number of connections of a person, i.e., the size of the set of grantors, is 223.99 and 520.04, respectively, [16], but also to unrealistic scenarios where the size of the set of grantors is 5000. These results are in line with our complexity analysis, and one may interpret them as an indication that our analysis also holds for the average case. Consequently, the *2lSE* approach is the most feasible option in our scenario.

8 RELATED WORK

We categorize exiting works aiming to preserve location secrecy in two groups: location secrecy in LBS and location secrecy in mSNs. The main difference between these two categories is that approaches in the first one do not consider access policies, i.e., users are allowed to access the location of any user in the system.

Location secrecy in LBS: Several techniques have been proposed in the literature to achieve location secrecy in LBS such as:

Mix zones: The fundamental idea of these approaches is to prevent an adversary from tracking long-term user movements [3]. A mix zone is a spatial region with a predetermined size, inside which users do not report their position or communicate with the LBS provider. These approaches focus on data anonymization and replace the user identity with a pseudonym. They offer anonymity guarantees by changing the pseudonyms of users inside a mix zone such that an adversary will not be able to link users that go inside the mix zone with those leaving it. Since these approaches do not hide the position of users outside the mix zones, one can perform any query on the plaintext data.

However, users inside the mix zones do not communicate with the LBS provider, and therefore, they cannot get any service. This lack of communication affects functionality.

Coordinates transformation: Approaches in this area aim to guarantee that an adversary does not learn the position of the users. These approaches consist of mapping the location of users to another space coordinate and addressing the query on the transformed space. In the approach presented in [12], users use a transformation function over their physical position before sending it to the LBS provider. The transformation function consists of shifts and rotations. Each user sets the parameters of the transformation function and distributes these parameters among the rest of users to allow them to recover the original physical position. This approach is subject to data reconstruction attacks [15]. To avoid this kind of attacks, the authors in [20] use agents to transform the physical positions of users. Agents are trusted third parties servers that act as middleware between the users and the LBS provider. Agents periodically change their transformation functions, which prevents the adversaries from analyzing the data. These approaches allow users to query for one specific user at each time, i.e., point queries. The secrecy of this approach relies on trusted third servers.

Cryptography: Approaches like [25, 38]keep the position of users secret from any adversary including the service provider. They use encryption techniques and hash functions to compute proximity between two users privately. The authors in [38] use, besides hash functions, location tags to prevent users from cheating their location, i.e., announcing an untruthful location. Location tags are pieces of information obtained from the network protocol like 802.11 frames in WiFi networks. In these approaches, users are allowed to query for the proximity of one specific user at each time, i.e., point queries.

Location secrecy in mSNs: Approaches in this area restrict access to the physical position of users based on access control policies. Each user defines a set of users who are authorized to read his physical position. In [28], the authors proposed a scheme to guarantee location secrecy, which uses encryption and coordinate transformation techniques. The LBS provider stores the transformed data. Querying requires that users distribute among their friends the transformation parameters and the corresponding decryption keys. This scheme focuses on dealing with point and nearest neighbor queries. Wei et al. [33] proposed an approach called Mobishare. The architecture of this approach consists of a trusted central tower, an untrusted LBS provider, and an untrusted Access Control Server (ACS). Mobishare uses dummy techniques to prevent the LBS provider and the ACS from learning the users identities and their physical positions. Before outsourcing the data to the LBS provider, Mobishare replaces users identities with pseudonyms and adds dummy pseudonyms together with dummy locations. Since the position of users is stored in plaintext, the LBS provider can compute any kind of query. Before sending the final query answer to the user, the ACS filters the data based on the access policies, and the central tower replaces pseudonyms with user identities. However, an adversary who can observe query executions will be able to identify real pseudonyms from dummy ones. Learning this information, an adversary could link a pseudonym with a user identity and therefore learn the position of the user. The adversary could also learn information about access policies, i.e., the set of users that have allowed a given user to access their position. To avoid these issues, the authors in [17] extended the previous approach by adding dummy queries and using a private set intersection protocol. The authors in [18] extended the latter approach by introducing a new architecture with multiple LBS providers. The authors aim to prevent an adversary from identifying queries coming from the same user, which can help an adversary to learn the users identities. To conclude, approaches in this area focus on point or nearest neighbor queries, or they rely on trusted central towers. Furthermore, their adversary model is weaker than ours, as explained in Section 2.2.

9 CONCLUSION

Location-based services are an important feature provided by mSNs. However, users are normally reluctant to share their physical position with others due to privacy reasons. In this paper, we have showed how to offer location-based services, in the example of range queries, in mSNs with a revocation feature while providing the secrecy guarantees $G_{position}$, $G_{distance}$, and $G_{authorization}$, to the users under collusion assumption. We introduced two approaches namely two-layer symmetric encryption, *2lSE*, and two-layer attribute-based encryption, *2lABE*. One of the main differences between the first and the second approach is that they use, among other encryption schemes, symmetric and attribute-based encryption, respectively. A complexity analysis of the query phase tells us that both approaches differ only at one entity, namely the user-side. Their differences rely on the decryption process, which is related to the encryption scheme used. We have further compared our approaches experimentally. Although with the *2lABE*, the key management is more straightforward than with the *2lSE*, and the LBS provider does not have to store multiple encrypted copies for each message, we have found that our former solution is on average twice more efficient in our scenario.

In the future, it will be interesting to study how to offer location-based services in mSNs with different kind access policies while providing secrecy guarantees. For instance, one can consider integrating mutual authorizations [32], where users grant access to their resources, i.e., physical positions, to users that allow them the same, or location constraints [4], where users restrict access to their resources based on the location of the accessing user.

ACKNOWLEDGMENTS

The first author thanks “Escuela Politécnica Nacional, Ecuador - Departamento de Informática y Ciencias de la Computación" for its support. This work was partially funded by the German Research Foundation (DFG) as part of the research Datenschutzkonforme Verwaltung relationaler Datenbestände.

REFERENCES

[1] Vijayalakshmi Atluri, Heechang Shin, and Jaideep Vaidya. 2008. Efficient security policy enforcement for the mobile environment. *Journal of Computer Security* 16, 4 (2008), 439–475.

[2] Yonatan Aumann and Yehuda Lindell. 2007. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography Conference*. Springer, 137–156.

[3] Alastair R Beresford and Frank Stajano. 2004. Mix zones: User privacy in location-aware services. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*. IEEE, 127–131.

[4] Elisa Bertino, Barbara Catania, Maria Luisa Damiani, and Paolo Perlasca. 2005. GEO-RBAC: A Spatially Aware RBAC. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*. New York, NY, USA, 9.

[5] John Bethencourt, Amit Sahai, and Brent Waters. 2007. Ciphertext-policy attribute-based encryption. In *IEEE Security and Privacy, 2007.*

[6] Dan Boneh. 1998. The decision diffie-hellman problem. In *International Algorithmic Number Theory Symposium*. Springer, 48–63.

[7] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 2001 IEEE International Conference on Cluster Computing*. IEEE, 136–145.

[8] Qi Cheng, Jun Zhang, and Jincheng Zhuang. 2016. LWE from Non-commutative Group Rings. *arXiv preprint arXiv:1612.06670* (2016).

[9] MacKenzie F Common. 2018. Facebook and Cambridge Analytica: let this be the high-water mark for impunity. *LSE Business Review* (2018).

[10] Howard Falk. 2011. Applications, architectures, and protocol design issues for mobile social networks: A survey. *Proc. IEEE* 99, 12 (2011), 2125–2129.

[11] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive* 2012 (2012), 144.

[12] Andreas Gutscher. 2006. Coordinate transformation-a solution for the privacy problem of location based services?. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 7–pp.

[13] Florian Hahn, Nicolas Loza, and Florian Kerschbaum. 2019. Joins Over Encrypted Data with Fine Granular Security. In *Data Engineering, 2019. ICDE 2019. IEEE 35th International Conference on*. IEEE, 674–685.

[14] Carmit Hazay and Yehuda Lindell. 2010. *Efficient secure two-party protocols: Techniques and constructions*. Springer.

[15] Hyeong-Il Kim, Seungtae Hong, and Jae-Woo Chang. 2016. Hilbert curve-based cryptographic transformation scheme for spatial query processing on outsourced private data. *Data & Knowledge Engineering* 104 (2016), 32–44.

[16] Silvio Lattanzi and Yaron Singer. 2015. The power of random neighbors in social networks. In *Proceedings of the 8th ACM WSDM Conference*.

[17] Jingwei Li, Jin Li, Xiaofeng Chen, Zheli Liu, and Chunfu Jia. 2014. {MobiShare}+: Security Improved System for Location Sharing in Mobile Online Social Networks. *J. Internet Serv. Inf. Secur.* 4, 1 (2014), 25–36.

[18] Jin Li, Hongyang Yan, Zheli Liu, Xiaofeng Chen, Xinyi Huang, and Duncan S Wong. 2017. Location-sharing systems with enhanced privacy in mobile online social networks. *IEEE Systems Journal* 11, 2 (2017), 439–448.

[19] Xiaohui Li, Dawu Gu, Yanli Ren, Ning Ding, and Kan Yuan. 2012. Efficient ciphertext-policy attribute based encryption with hidden policy. In *International Conference on Internet and Distributed Computing Systems*. Springer, 146–159.

[20] Dan Lin, Elisa Bertino, Reynold Cheng, and Sunil Prabhakar. 2008. Position transformation: a location privacy protection method for moving objects. In *Proceedings of the SIGSPATIAL ACM GIS 2008 International Workshop*. 62–71.

[21] Hong Liu and Yanbing Liu. 2014. Security assessment on block-Cat-map based permutation applied to image encryption scheme. *Optics & Laser Technology* 56 (2014), 313–316.

[22] G Locke and P Gallagher. 2009. Fips pub 186-3: Digital signature standard (dss). *Federal Information Processing Standards Publication* 3 (2009), 186–3.

[23] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1–23.

[24] Steve Moffat, Mohammad Hammoudeh, and Robert Hegarty. 2017. A survey on CP-ABE approaches to data security on mobile devices and its application to iot. In *Proceedings of the ICFNDS*. ACM, 34.

[25] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, Dan Boneh, et al. 2011. Location Privacy via Private Proximity Testing.. In *NDSS*, Vol. 11.

[26] Ghizlane Orhanou, Saïd El Hajji, and Youssef Bentaleb. 2011. EPS AES-based confidentiality and integrity algorithms: Complexity study. In *Multimedia Computing and Systems (ICMCS), 2011 International Conference on*. IEEE, 1–4.

[27] Raluca Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*. 85–100.

[28] Krishna PN Puttaswamy, Shiyuan Wang, Troy Steinbauer, Divyakant Agrawal, Amr El Abbadi, Christopher Kruegel, and Ben Y Zhao. 2014. Preserving location privacy in geosocial applications. *IEEE TMC* 13, 1 (2014), 159–173.

[29] SEAL 2018. Microsoft SEAL (release 3.1). https://github.com/Microsoft/SEAL. (Dec. 2018). Microsoft Research.

[30] Nan Shen, Jun Yang, Ke Yuan, Chuan Fu, and Chunfu Jia. 2016. An efficient and privacy-preserving location sharing mechanism. *Computer Standards & Interfaces* 44 (2016), 102–109.

[31] Thomas Shortell and Ali Shokoufandeh. 2015. Secure signal processing using fully homomorphic encryption. In *International Conference on Advanced Concepts for Intelligent Vision Systems*. Springer, 93–104.

[32] Gabriela Suntaxi, Aboubakr Achraf El Ghazi, and Klemens Böhm. 2019. Mutual Authorizations: Semantics and Integration Issues. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*. 213–218.

[33] Wei Wei, Fengyuan Xu, and Qun Li. 2012. Mobishare: Flexible privacy-preserving location sharing in mobile online social networks. In *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2616–2620.

[34] Runhua Xu, Yang Wang, and Bo Lang. 2013. A tree-based cp-abe scheme with hidden policy supporting secure data sharing in cloud computing. In *2013 International Conference on Advanced Cloud and Big Data*. IEEE, 51–57.

[35] Kefeng Xuan, Geng Zhao, David Taniar, and Bala Srinivasan. 2008. Continuous range search query processing in mobile navigation. In *Parallel and Distributed Systems ICPADS'08*. IEEE, 361–368.

[36] Dingqi Yang, Daqing Zhang, Vincent W Zheng, and Zhiyong Yu. 2015. Modeling user activity preference by leveraging user spatial temporal characteristics in LBSNs. *IEEE SMCS* 45, 1 (2015).

[37] Xun Yi, Russell Paulet, Elisa Bertino, and Vijay Varadharajan. 2014. Practical k nearest neighbor queries with location privacy. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*. IEEE, 640–651.

[38] Yao Zheng, Ming Li, Wenjing Lou, and Y Thomas Hou. 2012. Sharp: Private proximity test and secure handshake with cheat-proof location tags. In *European Symposium on Research in Computer Security*. Springer, 361–378.