SecReach: Secure Reachability Computation on Encrypted Location Check-in Data

Hanyu Quan[1,2], Boyang Wang[2], Iraklis Leontiadis[2], Ming Li[2], and Yuqing Zhang[1,3]

[1] School of Cyber Engineering, Xidian University, Xi'an, China
quanhanyu@gmail.com
[2] Department of Electrical and Computer Engineering,
The University of Arizona, Tucson, AZ, USA
{boyangwang, leontiad, lim}@email.arizona.edu
[3] University of Chinese Academy of Sciences, Beijing, China
zhangyq@ucas.ac.cn

Abstract. Reachability, which answers whether one person is reachable from another through a sequence of contacts within a period of time, is of great importance in many domains such as social behavior analysis. Recently, with the prevalence of various location-based services (LBSs), a great amount of spatiotemporal location check-in data is generated by individual GPS-equipped mobile devices and collected by LBS companies, which stimulates research on reachability queries in these location check-in datasets. Meanwhile, a growing trend is for LBS companies to use scalable and cost-effective clouds to collect, store, and analyze data, which makes it necessary to encrypt location check-in data before outsourcing due to privacy concerns. In this paper, for the first time, we propose a scheme, SecReach, to securely evaluate reachability queries on encrypted location check-in data by using somewhat homomorphic encryption (SWHE). We prove that our scheme is secure against a semi-honest cloud server. We also present a proof-of-concept implementation using the state-of-the-art SWHE library (i.e., HElib), which shows the efficiency and feasibility of our scheme.

Keywords: reachability, location privacy, homomorphic encryption

1 Introduction

Reachability, which answers whether a user is reachable from another through a sequence of contacts in a period of time, is of great importance in many domains, e.g., social behavior analysis, friend recommendations, public health monitoring, to name a few. Due to the prevalence of various location-based services (LBSs), such as Google Maps, Foursquare, Yelp, etc., a great amount of location check-in data is generated by individual GPS-equipped mobile devices and collected by these LBS companies. This stimulates research on reachability analysis in these location check-in datasets [19], [21]. For instance, if two people are in close proximity to each other, we can infer that they are socially connected or an

item (e.g., an infectious virus) could spread from one to another. Base on these inferences, companies or other authorized parties such as governments are able to build customized advertising systems, identify certain targets or trace epidemic contacts.

Meanwhile, with the increase in the volume of data (for example, there are millions of new check-ins in Foursquare each day), a growing trend is for LBS companies to use scalable and cost-effective cloud services to store and analyze data. For instance, both Foursquare and Yelp use Amazon S3 (Amazon Simple Storage Service) to store their data. Moreover, advanced cloud services (such as Amazon Kinesis Firehose) allow mobile LBS applications to send data directly to cloud stores (e.g., Amazon S3) from users' mobile devices, which enables LBS companies to scale location check-in data collection on clouds. However, outsourcing location check-in data to clouds poses the privacy concerns of users. Location check-ins are sensitive because they reveal private individual information including home addresses, interests, and state of health [20]. Furthermore, the anonymity of location check-in data is difficult to achieve. A recent research work [5] shows that in a dataset, where the locations are specified hourly, four spatiotemporal points are enough to uniquely identify most of the individuals. Given the sensitivity of location information, users, who are willing to make their locations available to LBS companies, may not fully trust third party clouds. On the other hand, LBS companies also do not want to reveal individually generated location check-in data to public clouds, due to legal and commercial reasons.

To prevent clouds from learning location check-in data, the most effective way is to use end-to-end data encryption. However, the analysis of the encrypted check-in data in clouds remains to be a very challenging problem. Specifically, in this paper, we study how to evaluate reachability queries on encrypted location check-in data. Theoretically, Fully Homomorphic Encryption (FHE) [7] allows an untrusted party to compute any functions on encrypted data, however, the state-of-the-art FHEs are far from being practical [16]. Somewhat Homomorphic Encryption (SWHE), which supports additions and a few multiplications, is more efficient than FHE. Unfortunately, it cannot be directly used for evaluating reachability queries, because the limited number of multiplications is not sufficient for comparisons in reachability queries. On the other hand, some recent methods have been proposed for similar queries on encrypted location data, such as trajectory similarity [12] and kNN [6], [17], in which they combine partially homomorphic encryption and secure two-party computation to implement comparisons on ciphertexts. However, this kind of method is ordinarily based on the system model of two cloud servers, which introduces extensive interactions.

In this paper, we propose a scheme for reachability queries evaluation on encrypted location check-in data. Our main contributions are as follows:

- To the best of our knowledge, this is the first work that studies reachability queries evaluation on encrypted location check-in data. We propose a method to compute 2-hop reachability using additions and a limited number of multiplications, instead of relying on comparisons which implies interactions. With the use of SWHE and Bloom filters, the evaluation of reachability

queries is non-interactive between a cloud server and a data analyzer. One of the key innovations in our scheme is a new method to determine whether an integer number is equal to a given integer k or whether it is in the range of $[0, k-1]$ in the ciphertext domain without decryption.
– We formally analyze the security of our scheme against a semi-honest server, and it is shown that our scheme does not leak any user locations, intermediate results or final reachability results to the server. We also present a proof-of-concept implementation, and experimental results show our scheme is feasible and efficient in practice.

The rest of this paper is organized as follows. Section 2 presents the problem statement and Section 3 introduces the preliminaries of our scheme. In Section 4 we describe the idea and details of our scheme. We analyze the security of our scheme in Section 5 and present a proof-of-concept implementation with a performance analysis in Section 6. Finally, Section 7 reviews the related work and Section 8 concludes the paper.

2 Problem Statement

In this paper we study the problem of evaluating reachability queries on encrypted location check-in data. In this section, we first introduce our system and adversarial model. Then, we state the data format of location check-ins and definition of reachability. Finally, we present our design objectives.

2.1 System and Adversarial Model

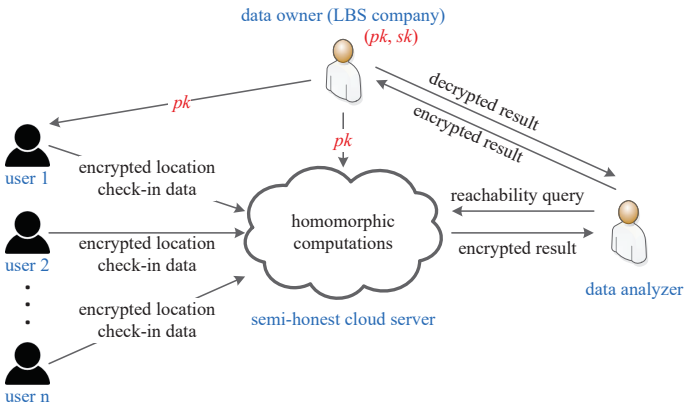


Fig. 1. The system model.

System Model Our system model, as shown in Fig. 1, consists of a set of *users*, a *data owner*, a *data analyzer*, and a *cloud server*. Specifically, each

user generates location check-in data on personal devices (e.g., smartphones or tablets), and uploads location check-ins to the cloud server over time. As a result, the cloud server stores and maintains a dataset containing a set of location check-ins, where each location check-in (also referred to as a *tuple*) is reported by one of the users at a certain time. Said differently, each tuple consists of three properties, including *who*, *where* and *when*. The dataset belongs to the data owner (e.g., an LBS company). The data analyzer is able to submit reachability queries to the cloud server to discover the reachability of a certain user to other users. Note that the data analyzer could also be the data owner itself. The cloud server should be able to perform the evaluation of reachability queries in a location check-in dataset, and return results to the data analyzer.

Adversarial Model We assume users trust the data owner, but not the cloud server. Specifically, we assume the cloud server is *semi-honest* (i.e., *honest-but-curious*), which means it follows protocols correctly but may try to learn the private location of each tuple in a location check-in dataset. To be more precise, given a location check-in, the cloud server tries to figure out *where* this location check-in is reported. Due to the privacy concerns of users, each location is encrypted with a public key of the data owner before uploading it to the cloud. Therefore, given a reachability query, the cloud server computes it on encrypted data, and returns encrypted result back to the data analyzer. The data analyzer asks the data owner to decrypt the result. Only the data owner can decrypt the result with its secret key. We assume the data analyzer and the cloud server do not collude. Previous examples of this type of system model can also be found in recent secure medical computation applications such as [3], [26].

2.2 Location Check-in Data, Proximity and Reachability

Location Check-in Data As discussed in [19], location check-in data is associated with both space dimension and time dimension as users move within a space over time. We define a location check-in data as a tuple $d = (u, l, t)$, where u is a user identity (i.e., who), l is the location of this user (i.e., where), and t is the time slot of generating this location check-in (i.e., when). Moreover, we leverage a square grid to index locations, as shown in Fig. 2. Specifically, a location $l = (x, y)$ is the center of a cell, and a user's location is reported as l on condition that its physical location is within this cell. Similar as square grid, other types of grids, such as hexagonal grid [15] can also be leveraged in location check-in data.

Proximity and Reachability Given a tuple $d = (u, l, t)$, where $l = (x, y)$, we define the *proximity range* of user u as its square neighborhood including nine cells, where the center of this square is (x, y) and the size of this square can be configured by changing the cell size. This square is in fact the Moore neighborhood of the cell (x, y) with a specific range, which is frequently used in geographic information system (GIS). If another user's location is inside the proximity range of user u in the same time slot t, we say that the two users are

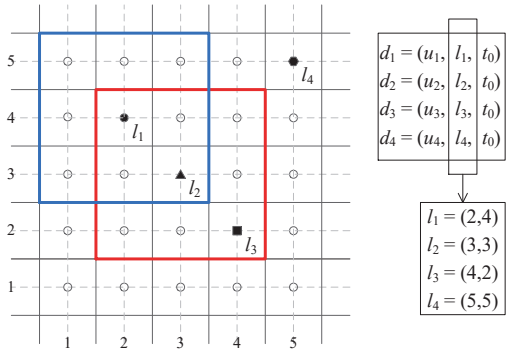


Fig. 2. An example of location check-in data generated by four users. The locations (i.e., *where*) are indexed using a grid. In this example, u_3 is reachable from u_1 through u_2 (i.e., there is a contact path $u_1 \rightarrow u_2 \rightarrow u_3$).

in *direct contact*. Correspondingly, two users can be in *indirect contact* if they have a common direct-contact user. A *contact path*, which is bidirectional, exists between two users if they are in contact (either direct or indirect). A reachability query of a user u_i to another user u_j tells whether there is a contact path between them in a given time interval T (including several time slots). In other words, it is a way to decide whether one user is reachable from another user within T.

Fig. 2 shows an example of location check-in data generated by four users in time slot t_0. In this example, for user u_3 and user u_1 in time interval $T = t_0$, u_3 is outside the proximity range of u_1, but it is inside the proximity range of u_2, who is in the proximity range of u_1. Thus, there is a contact path $u_1 \rightarrow u_2 \rightarrow u_3$ from u_1 to u_3, i.e., u_3 is reachable from u_1 through u_2. For user u_4 and user u_1, because u_4 is outside all the proximity ranges of the other three users, it is not reachable from any of those three users. More details about the definition of reachability can be found in [19].

2.3 Design Objectives

In this paper, we aim at designing a secure scheme to compute reachability queries under the preceding system and adversarial model. Our main design objectives include two aspects, data privacy and efficiency. Specifically,

- **Data Privacy.** The cloud server is not be able to reveal any of the locations in a location check-in dataset or the results of any reachability queries. In addition, the cloud server cannot reveal intermediate results (e.g., the proximity between users). The reason of such high level of privacy protection is that, if we leak either intermediate proximity results or final reachability results to the cloud server, the cloud server may be able to infer users' locations by encrypting locations of its choice and then conduct proximity tests with the locations in the dataset, and use triangulation attacks to learn users' locations with reasonable accuracy [11].

– **Efficiency.** Our scheme should efficiently carry out reachability queries on encrypted location check-ins and avoid interactions between the cloud server and data analyzer during the evaluation of reachability queries.

3 Preliminaries

3.1 Bloom Filters

A Bloom filter [1] is a space-efficient randomized data structure for membership testing (i.e., whether an element is in a set or not). More specifically, a Bloom filter is able to decide either an element is definitely not in a set or it is in a set with a very high probability. A (m, k)-Bloom filter $\mathsf{BF} = (b_1, \ldots, b_m)$ is essentially a binary vector of m bits, which are initially all set as 0s. There are k independent hash functions $h_1, \ldots, h_k$, where the hash values of each of those hash functions is within the range of $[1, m]$ and each value within this range maps a component index in a Bloom filter.

To add an element x to a Bloom filter BF, bits $\{b_{h_1(x)}, ..., b_{h_k(x)}\}$ are set as 1s. To query if an element y is in a set, we check whether $\{b_{h_1(y)}, ..., b_{h_k(y)}\}$ are all 1s. If not, then y is definitely not a member of the set; otherwise, y is in the set with a small false positive probability. We denote the above two algorithms as $\mathsf{BF.add}$ and $\mathsf{BF.query}$, respectively.

3.2 Somewhat Homomorphic Encryption

Somewhat homomorphic encryption (SWHE) is a fundamental building block of fully homomorphic encryption (FHE), which can be extended to FHE by utilizing bootstrapping [7]. Compared to FHE, SWHE can only carry out a limited number of multiplications, but it is much faster than FHE [14].

A typical public-key SWHE scheme consists of five algorithms, including $\mathsf{SWHE.KeyGen}$, $\mathsf{SWHE.Enc}$, $\mathsf{SWHE.Dec}$, $\mathsf{SWHE.Add}$ and $\mathsf{SWHE.Mul}$. Specifically, $\mathsf{SWHE.KeyGen}(1^\lambda)$ takes as input a security parameter and outputs a pair of public and secret keys (pk, sk). $c \leftarrow \mathsf{SWHE.Enc}(pk, m)$ and $m \leftarrow \mathsf{SWHE.Dec}(sk, c)$ are used for encryption and decryption, respectively, where m and c are a pair of plaintext/ciphertext. In addition, $\mathsf{SWHE.Add}$ and $\mathsf{SWHE.Mul}$ are used for homomorphically additions and multiplications, respectively. More concretely, $\mathsf{SWHE.Add}(c_1, c_2)$ takes as input two ciphertexts c_1 and c_2 and outputs a new ciphertext c_{add}, where c_1 is the ciphertext of m_1, c_2 is the ciphertext of m_2, and c_{add} is the ciphertext of $m_1 + m_2$. Similarly, $\mathsf{SWHE.Mul}(pk, c_1, c_2)$ outputs c_{mul}, which is the ciphertext of $m_1 m_2$.

4 SecReach: Secure Reachability Computation

As we discussed above, given two users (u_i, u_j) and a time interval T, a reachability query answers whether there is a contact path from u_i to u_j within time interval T. In this paper, we begin with the very fundamental problem, i.e.,

whether u_j is reachable from u_i by a contact path with two hops, which we call it a 2-hop reachability query. This type of queries can be easily seen in practice, e.g., finding a friend of a friend in social networks [23]. In the following, for ease of presentation, we first assume all the location check-ins are reported in one time slot t_0, and the query time interval is $T = t_0$. Then we will explain how to extend our scheme to support queries with different time slots.

4.1 Our Main Idea

Assume each user generates a location check-in in time slot t_0, and there are n users in total. Given a 2-hop reachability query of (u_i, u_j) in $T = t_0$, our main idea is to evaluate whether there is at least one user, e.g., u_k, is in direct contact with both u_i and u_j. If it is, then it implies that u_j is reachable from u_i within 2 hops, and there is at least one contact path, e.g., $u_i \rightarrow u_k \rightarrow u_j$; otherwise, u_i and u_j are not 2-hop reachable.

By following this logic, our method can be broken down in two steps. First, we compute the contacts between u_i and all the n users, and represent the results as an n-bit binary vector $\mathbf{v}_i$, which is referred to as the *contact vector* of user u_i. If user u_i and u_k $(1 \leq k \leq n)$ are in direct contact, the k-th position of contact vector $\mathbf{v}_i$ is set to 1 (i.e., $\mathbf{v}_i[k] = 1$); otherwise, it is set to 0. Similarly, we can compute a contact vector $\mathbf{v}_j$ for user u_j. In the second step of our approach, we compute an inner product of $\mathbf{v}_i$ and $\mathbf{v}_j$, which is represented as $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$. If this inner product is equal to or greater than 1, i.e., $\langle \mathbf{v}_i, \mathbf{v}_j \rangle \geq 1$, it indicates there is at least one user (e.g., u_k) in direct contact with both of the two users, and these two users are reachable within 2 hops. Note that, at least one same index is assigned as 1 in both of the two contact vectors $\mathbf{v}_i, \mathbf{v}_j$, e.g., $\mathbf{v}_i[k] = \mathbf{v}_j[k] = 1$.

In order to decide whether user u_i is in direct contact with a user u_k, or said differently, whether this user u_k is inside user u_i's proximity range, we describe each user's proximity as a set of locations that are close to it, and leverage a Bloom filter to represent this set. As a result, whether user u_i is in direct contact with user u_k is equivalent to say whether user u_k's location is a member of the Bloom filter of user u_i. The essential reason that we utilize membership testing other than computing and comparing distances of two locations, is because membership testing is more efficient on encrypted data. Specifically, we leverage inner product to conduct membership testing in Bloom filter, which can be efficiently computed on encrypted data using SWHE (with one homomorphic multiplication). Moreover, we present a new way to convert the result of this inner product to a binary number in ciphertext domain so as to build the contact vector described above.

4.2 The Details of Our Scheme

In the following, we describe the details of our scheme. Our scheme leverages a public-key somewhat homomorphic encryption scheme SWHE and we denote by $[\![x]\!]$ the ciphertext of x encrypted under SWHE.

Encrypt Locations on The User Side Given a location check-in tuple $d_i = (u_i, l_i, t_0)$ generated by user u_i in time slot t_0, user u_i encrypts its location l_i as follows:

1. Enumerate and add all the locations in its proximity range to an (m, k)-Bloom filter BF using $\mathsf{BF.add}$. Here we use an m-bit vector $\boldsymbol{\alpha}_i$ to represent BF and refer to it as the *proximity vector* of user u_i.
2. Create another m-bit vector $\boldsymbol{\beta}_i$, where all bits are initialized as 0s, and set the $h_j(l_i)$-th bit of $\boldsymbol{\beta}_i$ to 1, for $1 \leq j \leq k$, where h_j $(1 \leq j \leq k)$ are the k hash functions of BF. $\boldsymbol{\beta}_i$ is called the *location vector* of user u_i.
3. Encrypt $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i$ using $\mathsf{SWHE.Enc}$. More concretely,

$$[\![\boldsymbol{\alpha}_i]\!] \leftarrow (\mathsf{SWHE.Enc}(pk, \boldsymbol{\alpha}_i[1]), \ldots, \mathsf{SWHE.Enc}(pk, \boldsymbol{\alpha_i}[m])) \quad (1)$$

$$[\![\boldsymbol{\beta}_i]\!] \leftarrow (\mathsf{SWHE.Enc}(pk, \boldsymbol{\beta}_i[1]), \ldots, \mathsf{SWHE.Enc}(pk, \boldsymbol{\beta_i}[m])) \quad (2)$$

   Note that the above encryptions are bit-wise, where we encrypt each bit in vector $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i$ separately.

In the end, user u_i obtains an encrypted location check-in as $(u_i, [\![\boldsymbol{\alpha}_i]\!], [\![\boldsymbol{\beta}_i]\!], t_0)$, and sends it to the cloud server.

Evaluate 2-hop Reachability on Cloud Server Assume user u_i sends one encrypted location check-in $(u_i, [\![\boldsymbol{\alpha}_i]\!], [\![\boldsymbol{\beta}_i]\!], t_0)$, and there are n encrypted location check-ins in total from the n users. The cloud server stores these encrypted location check-ins in a data table with n rows and four columns as shown in Fig. 3. Note that the fourth column is empty before any computation.

user ID	proximity vector	location vector	contact vector
u_1	$[\![\boldsymbol{\alpha}_1]\!]$	$[\![\boldsymbol{\beta}_1]\!]$	$[\![\mathbf{v}_1]\!]$
u_2	$[\![\boldsymbol{\alpha}_2]\!]$	$[\![\boldsymbol{\beta}_2]\!]$	$[\![\mathbf{v}_2]\!]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
u_n	$[\![\boldsymbol{\alpha}_n]\!]$	$[\![\boldsymbol{\beta}_n]\!]$	$[\![\mathbf{v}_n]\!]$

Fig. 3. The data table maintained by the cloud server, and one table per time slot.

Given a 2-hop reachability query of (u_1, u_2) in time interval $T = t_0$, the cloud server first checks whether the contact vectors $[\![\mathbf{v}_1]\!]$ and $[\![\mathbf{v}_2]\!]$ have been computed and stored in the data table. If not, it computes $[\![\mathbf{v}_1]\!]$ and/or $[\![\mathbf{v}_2]\!]$ by Algorithm 1, and stores them in the data table. Specifically, in Algorithm 1, we present a novel approach, i.e., by homomorphically evaluating a function $f(x) = (k!)^{-1} \prod_{i=0}^{k-1}(x-i)$ on an encrypted integer x to check whether x is equal to k or whether $0 \leq x \leq k-1$, and represent the result as an encrypted binary

number in order to build the contact vector. Then, with $[\![\mathbf{v}_1]\!]$ and $[\![\mathbf{v}_2]\!]$, the cloud server computes $[\![\langle \mathbf{v}_1, \mathbf{v}_2 \rangle]\!]$ using SWHE.Add and SWHE.Mul and returns it to the data analyzer. $[\![\langle \mathbf{v}_1, \mathbf{v}_2 \rangle]\!]$ is the ciphertext of the inner product of vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. Note that the cloud server does not decrypt any encrypted data during this above evaluation, and all the computations are operated on encrypted data correctly based on the homomorphic properties of SWHE.

Algorithm 1 Compute contact vector $[\![\mathbf{v}_i]\!]$ of u_i by the cloud server

Input: $[\![\boldsymbol{\alpha}_i]\!]$ of u_i, $[\![\boldsymbol{\beta}_j]\!]$ of u_j $(1 \leq j \leq n)$
Output: The encrypted contact vector $[\![\mathbf{v}_i]\!]$ of u_i

1: **for** $j = 1$ to n **do**
2: **if** $i == j$ **then**
3: $[\![\mathbf{v}_i[j]]\!] \leftarrow \mathsf{SWHE.Enc}(pk, 1)$
4: **continue**
5: **else**
6: $[\![x]\!] = [\![\langle \boldsymbol{\alpha}_i, \boldsymbol{\beta}_j \rangle]\!]$ // using SWHE.Add and SWHE.Mul
7: $[\![\mathbf{v}_i[j]]\!] = [\![f(x)]\!]$ // $f(x) = (k!)^{-1} \prod_{i=0}^{k-1}(x - i)$ where $(k!)^{-1}$ is the inverse of $(k!)$ in message space of SWHE, using SWHE.Add and SWHE.Mul
8: // Note that here SWHE.Add and SWHE.Mul should take as input $[\![(k!)^{-1}]\!]$ and $[\![1]\!], [\![2]\!], \ldots, [\![(k-1)]\!]$, which can be pre-computed using SWHE.Enc by the cloud server.
9: **end if**
10: **end for**
11: **return** $[\![\mathbf{v}_i]\!] = ([\![\mathbf{v}_i[1]]\!], [\![\mathbf{v}_i[2]]\!], \ldots, [\![\mathbf{v}_i[n]]\!])$

Decrypt The Query Result The data analyzer sends $[\![\langle \mathbf{v}_1, \mathbf{v}_2 \rangle]\!]$ to the data owner. And the data owner decrypts it using its secret key sk. Specifically, the data owner computes

$$\langle \mathbf{v}_1, \mathbf{v}_2 \rangle \leftarrow \mathsf{SWHE.Dec}(sk, [\![\langle \mathbf{v}_1, \mathbf{v}_2 \rangle]\!]) \tag{3}$$

If $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle$ equals 0, the data owner returns 0 to the data analyzer, which means there is no 2-hop reachability from u_1 to u_2. If $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle$ is non-zero, the data owner returns 1 to the data analyzer, which means u_2 is reachable from u_1 within 2 hops. Note that the data analyzer could also be the data owner itself. In this case, the data owner simply decrypts $[\![\langle \mathbf{v}_1, \mathbf{v}_2 \rangle]\!]$ with its secret key.

Correctness In the plaintext domain, at line 6 in Algorithm 1, it essentially evaluates whether the location l_j of user u_j is in the Bloom filter BF generated by user u_i, where BF contains all the locations in u_i's proximity range. Obviously, if it is, x, i.e., the inner product of vector $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_j$, is equal to k, i.e., the number of hash functions used in Bloom filters; otherwise, $0 \leq x \leq k-1$. In other words, we use the inner product $\langle \boldsymbol{\alpha}_i, \boldsymbol{\beta}_j \rangle$ instead of BF.query to decide whether location l_j is in Bloom filter BF.

At line 7, $f(x)$ represents the binary number to indicate whether x is equal to k. Specifically, if $x = k$, $f(x) = 1$; if $x \in \{0, 1, \ldots, (k-1)\}$, $f(x) = 0$. We would like to emphasize that this mapping from x to $f(x)$ is important for the subsequent computations. Note that the calculation of the inner product of $\langle \boldsymbol{\alpha}_i, \boldsymbol{\beta}_j \rangle$ and the mapping from x to $f(x)$ together only contains additions and a limited number of multiplications, which can be implemented with SWHE.Add and SWHE.Mul on encrypted data by the cloud server.

Therefore, if user u_i and u_j are in direct contact, then $[\![\mathbf{v}_i[j]]\!] = [\![1]\!]$; otherwise, $[\![\mathbf{v}_i[j]]\!] = [\![0]\!]$. For the 2-hop reachability from u_1 and u_2, if there is a user u_3 in direct contact with both u_1 and u_2, then both $\mathbf{v}_1[3]$ and $\mathbf{v}_2[3]$ equal 1, which implies the inner product $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle$ will definitely not be zero (i.e., at least 1). Note that u_1 and u_2 could be in direct contact, in this case, w.l.o.g., u_3 could be either u_1 or u_2.

Process Reachability with Different Time Slots Our scheme can be extended to support reachability with different time slots. More specifically, given a 2-hop reachability query of (u_1, u_2) in time interval $T = [t_x, t_y]$, which includes $y - x + 1$ time slots, i.e., $t_x, t_{x+1}, \ldots, t_y$, if there is a contact path $u_1 \to u_3 \to u_2$, the second contact $u_3 \to u_2$ should happen in a time slot that is later than the time slot of the first contact $u_1 \to u_3$ happens. More precisely, assume $u_1 \to u_3$ occurs in time slot t_i while $u_3 \to u_2$ occurs in time slot t_j, then we have $x \le i \le j \le y$.

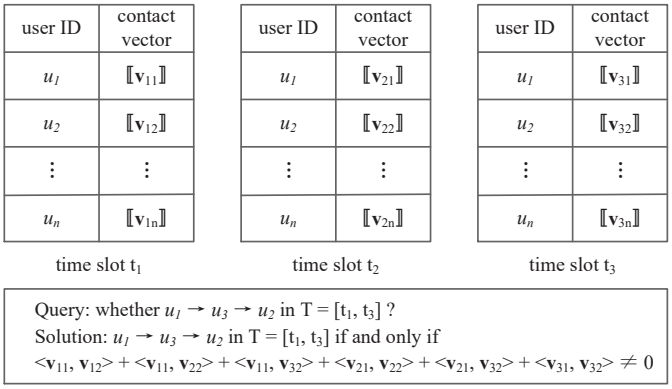


Fig. 4. The data tables maintained by the cloud server for 3 time slots (for the limitation of space we omit the second and third columns) and an example of reachability query about two users u_1 and u_2 within time interval $T = [t_1, t_3]$.

To evaluate this 2-hop reachability query, the cloud server should search all the possible combinations of t_i and t_j within $T = [t_x, t_y]$ to find out whether there is such a contact path $u_1 \to u_3 \to u_2$. Specifically, for each time slot, the cloud server maintains a data table as described above and returns the query

results as $[\![\sum_{i=x}^{y}(\sum_{j=i}^{y}(\langle \mathbf{v}_{i1}, \mathbf{v}_{j2}\rangle))]\!]$, where $\mathbf{v}_{i1}$ is the contact vector of u_1 in time slot t_i and $\mathbf{v}_{j2}$ is the contact vector of u_2 in time slot t_j. It is easy to prove that the 2-hop reachability exists if and only if $\sum_{i=x}^{y}(\sum_{j=i}^{y}(\langle \mathbf{v}_{i1}, \mathbf{v}_{j2}\rangle)) \geq 1$. Fig. 4 is an example for a query about $T = [t_1, t_3]$.

5 Security Analysis

In this section we analyze the security of our scheme against the semi-honest cloud server. Our scheme does not reveal any input locations, intermediate or final results to the semi-honest cloud, except the dataset size.

Formally, the location privacy guarantee of our scheme can be modeled under a standard CPA game for *multiple encryptions* as denoted in theorem 11.6 [9]. A challenger $\mathcal{C}$ chooses public and secret keys of a public encryption scheme (pk, sk). An adversary $\mathcal{A}$ submits two series of messages $\{m_0^i\}_{i=1}^n, \{m_1^i\}_{i=1}^n$, and $\mathcal{C}$ chooses $b \leftarrow \{0, 1\}$ uniformly at random and sends $\mathcal{A}$ encryptions of $\{m_b^i\}_{i=1}^n$. Finally $\mathcal{A}$ outputs its guess b'. Privacy is guaranteed if $\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$, where λ is the security parameter.

Theorem 1. *If public-key somewhat homomorphic encryption scheme SWHE is CPA-secure, then it also has indistinguishable multiple encryptions.*

In our scheme, given a location check-in data (u_i, l_i, t_i), the encryption of location l_i is $([\![\boldsymbol{\alpha}_i]\!], [\![\boldsymbol{\beta}_i]\!])$, where

$$[\![\boldsymbol{\alpha}_i]\!] = (\mathsf{SWHE.Enc}(pk, \boldsymbol{\alpha}_i[1]), \ldots, \mathsf{SWHE.Enc}(pk, \boldsymbol{\alpha}_i[m])) \tag{4}$$

$$[\![\boldsymbol{\beta}_i]\!] = (\mathsf{SWHE.Enc}(pk, \boldsymbol{\beta}_i[1]), \ldots, \mathsf{SWHE.Enc}(pk, \boldsymbol{\beta}_i[m])) \tag{5}$$

In other words, the encryption of a location in our scheme consists of *multiple encryptions* under SWHE. According to Theorem 1, we can conclude that, *given a CPA-secure somewhat homomorphic encryption scheme SWHE, the encryptions of locations in our scheme are indistinguishable under chosen plaintext attack.* Proof of Theorem 1 can be found in theorem 11.6 [9]. Also, the privacy of the intermediate results (e.g., the proximity vector $\mathbf{v}_i$) and the query results are protected under the security of the SWHE.

In the adversarial model, we assume that the data analyzer and the cloud server do not collude. Otherwise, the 2-hop reachability results are revealed and a semi-honest cloud server may be able to infer the users' locations by, for example, the triangulation attacks in [11]. But the impact of this attack should be much smaller since no direct proximity is known. Moreover, in practice, we can limit the risk of such collusion attack by letting the data owner restricting reachability queries to only authorized data analyzers.

6 Proof of Concept and Experimental Results

In this section, we present a proof-of-concept implementation of our scheme. We leverage the state-of-the-art SWHE scheme as the building block for our scheme,

and examine the performance of our design over synthetic location check-ins. Specifically, we implement our scheme using HElib [8], which is a C++ library that implements the BGV SWHE scheme [2].

Parameters and Encoding of Messages The parameters of our scheme consist of the parameters of Bloom filters and the parameters of HElib. In our scheme, each user adds nine locations to an empty Bloom filter to generate its proximity vector. We choose the number of hash functions as $k = 3$ and the length of Bloom filters as $m = 220$, such that the false positive probability of Bloom filters is around 0.001 (the probability is taken from [1]). As a result, the length of a proximity vector and a location vector is $|\boldsymbol{\alpha}| = |\boldsymbol{\beta}| = m = 220$.

Furthermore, given $k = 3$, our scheme needs $2k + 2 = 8$ depth of homomorphic multiplication ($k + 1$ for computing encrypted contact vector $[\![\mathbf{v}]\!]$ in Algorithm 1 and double $(k + 1)$ for computing $[\![\langle \mathbf{v}_i, \mathbf{v}_j \rangle]\!]$). Therefore, the depth of homomorphic multiplication of the SWHE should at least be eight. We set the parameters of HElib as shown in Table 1 to meet this requirement.

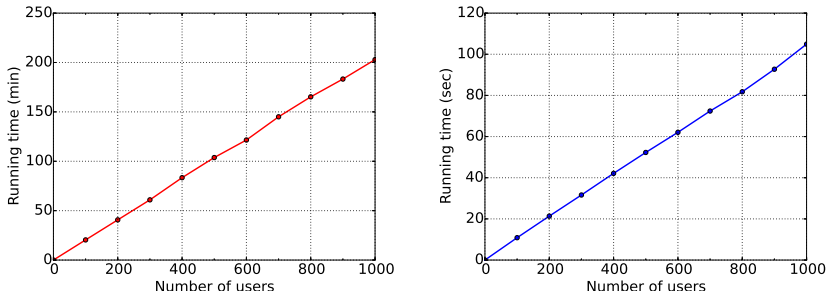
The underlying SWHE scheme (BGV) of HElib is based on the "ring learning with errors" (RLWE) problem, which means messages that can be encrypted with HElib are polynomials. We use *scalar encoding* to encode an integer by representing it as constant coefficient of a plaintext polynomial. Note that, if a constant coefficient increases beyond the plaintext base p', it will automatically be reduced by modulo p'. We choose $p' = 1009$, which means it can support up to one thousand users (e.g., in the worst case, if both user u_i and u_j are in contact with all the other $1000 - 1$ users in one time slot, the final result $\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle$ will be equal to 1000, which is less than p').

Table 1. The parameters of HElib in our implementation, where p' is the plaintext base, r' is the lifting, L' is the number of levels in the modulus chain, c' is the number of columns in the key-switching matrics, w' is the hamming weight of secret key, d' is the degree of the filed extension, and k' is the security parameter.

p'	r'	L'	c'	w'	d'	k'
1009	1	16	3	64	0	80

Experimental Results Our proof-of-concept implementation runs on a Ubuntu 14.04 virtual machine (VM) with 4 GB memory. The VM is hosted in a desktop PC with Inter(R) Core(TM) i7-4790 CPU @ 3.60GHz and 8 GB memory. We test the running time for generating one encrypted contact vector $[\![\mathbf{v}]\!]$, and evaluate the performance of answering 2-hop reachability query of two users within one time slot (i.e., computing $[\![\langle \mathbf{v}_i, \mathbf{v}_j \rangle]\!]$). The experimental results are illustrated in Fig. 5. Even with our un-optimized implementation, a 2-hop reachability query between two users can be evaluated on encrypted location data in approximately

100 seconds for a system contains 1,000 users. Although from Fig. 5(a), computing an encrypted contact vector seems to be time-consuming, we argue that this computation is only a one time operation. In other words, if a user has been queried before, the cloud server does not have to compute it again. Moreover, our implementation can be further optimized using more efficient message encoding methods. In addition, the implementation of SWHE (and FHE) on high performance computing platforms (e.g., GPU) is itself a question of interest (and has been studied recently in, e.g., [4], [10]), and the efficiency of our scheme can be significantly boosted up if we implement it with GPU. For instance, [10] implements a SWHE using GPU, which results in a speedup of 104x in homomorphic multiplication over the implementations with CPU.



(a) Compute encrypted contact vector (b) Evaluate 2-hop reachability

Fig. 5. The performance of our proof-of-concept implementation for one time slot

7 Related Work

To the best of our knowledge, current work does not tackle the problem of computing reachability on encrypted location check-in data. The previous works most relevant to ours are [27] and [18].

In [27],Yi et al. propose an optimized 2-hop labeling (which is an index of a graph), namely m-2-hop, for privacy-preserving reachability queries in a sparse graph. In their system model, there is a data owner who owns graph data and pre-computes an m-2-hop index offline. Then, the data owner encrypts the m-2-hop index and outsources it to the cloud server. A reachability query is processed on the encrypted m-2-hop index. Their solution is essentially for the reachability queries in static graph data (from which they can abstract an index first). However, in our work, we consider the reachability queries over individually generated location check-in data, which has both space and time dimension and cannot be represented as a static graph. Therefore, their work is not able to address our problem.

In [18], Shahabi et al. propose an extensible framework, PLACE, which consists of some building blocks including location proximity block, and enables privacy-preserving inference of social relationships from location check-in data. Specially, they state one of the use cases of PLACE is to analyze the reachability of two users. However, this work does not present any concrete designs.

In addition to the above two works, we briefly review two categories of studies on location privacy, private proximity testing (PPT) and searchable encryption for range search, which are also relevant to the topic of this paper.

Private Proximity Testing A PPT protocol enables a pair of users to test if they are within a certain distance of each other, but otherwise reveal no information about their locations to anyone. For example, in [15], Narayanan et al. present several PPT protocols by reducing the PPT problem to the problem of private equality testing (PET). However, PPT protocols are essentially secure two-party computation protocols, therefore, they are not compatible with our system model in which encrypted location check-in data and computations are centralized on an untrusted cloud server.

Searchable Encryption for Range Search Recently, a couple of searchable encryption schemes for range search, e.g., [25], [24], have been proposed. In [25], Wang et al. propose two searchable encryption schemes supporting circular range search on encrypted spatial data. They improve their work to support arbitrary geometric range search in [24]. However, in these searchable encryption schemes, a database server (e.g., a cloud server) will know search results, while our scheme does not leak those information.

Our work is also relevant to another problem called privacy-preserving location data publication. The works on this problem generally leverage non-cryptography anonymization techniques, such as k-anonymity [22]. Specially, in [13], Liu et al. propose the problem of reachability preserving anonymization (RPA), and design an RPA algorithm which supports computing reachability over anonymous graph. These anonymization techniques are generally very efficient, however, the security of these schemes cannot be proven formally and the results are not accurate because they have to modify the original data to achieve anonymization. Moreover, like [27], the method in [13] is designed for static graph, which cannot be used for the spatiotemporal location check-in data generated by individual users over time.

From a technical point of view, based on Wilson's Theorem $(p-1)! \equiv -1 \pmod{p}$ with p as a prime number greater than 2, Wang et al. [26] propose a function $g(x) = -\prod_{i=1}^{p-1}(i-x) \pmod{p}$ to check whether an integer x equals 0 or whether $1 \leq x \leq p-1$, which is similar to the $f(x)$ in Algorithm 1 in our design. Plausibly, we can also use $g(x)$ to check whether x is equal to k by checking whether $(k-x)$ equals 0. However, to compute "mod p" on encrypted data, they use *scalar encoding* and set p as the plaintext base (recall that in *scalar encoding*, the plaintext will automatically modulo p). In other words, their method is limited to applications with small message space, otherwise it will exceed the limitation of SWHE on the number of multiplications.

8 Conclusion and Future Work

In this paper we studied the problem of reachability queries on encrypted location check-in data. Specifically, we presented a scheme, namely SecReach, to support 2-hop reachability queries, which is based on a fresh approach of combining Somewhat Homomorphic Encryption and Bloom filters. Our scheme is provably secure and our experimental results demonstrate its practicality.

As part of future work, we are going to consider indexing location check-in data with more efficient data structures to improve efficiency (for instance, splitting the location space into smaller partitions including less users). Also, we plan to extend our scheme to enable multi-hop reachability queries, in which the length of a contact path is greater than 2, and implement our extension on high performance computing platforms (e.g., GPU).

Acknowledgments We would like to thank the anonymous reviewers for their valuable comments. This work was supported by the US NSF grant CNS-1218085, the 111 Project of China (No. B16037), the National Natural Science Foundation of China (No.61272481, No. 61572460), the National Key Research and Development Plan of China (No. 2016YFB0800703), and the China Scholarship Council.

References

1. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7), 422–426 (1970)
2. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 309–325. ACM (2012)
3. Cheon, J.H., Kim, M., Lauter, K.: Homomorphic computation of edit distance. In: International Conference on Financial Cryptography and Data Security. pp. 194–212. Springer (2015)
4. Dai, W., Sunar, B.: cuhe: A homomorphic encryption accelerator library. In: International Conference on Cryptography and Information Security in the Balkans. pp. 169–186. Springer (2015)
5. De Montjoye, Y.A., Hidalgo, C.A., Verleysen, M., Blondel, V.D.: Unique in the crowd: The privacy bounds of human mobility. Scientific reports 3 (2013)
6. Elmehdwi, Y., Samanthula, B.K., Jiang, W.: Secure k-nearest neighbor query over encrypted data in outsourced environments. In: 2014 IEEE 30th International Conference on Data Engineering. pp. 664–675. IEEE (2014)
7. Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University (2009)
8. Halevi, S., Shoup, V.: Algorithms in helib. In: Advances in Cryptology–CRYPTO 2014. pp. 554–571. Springer (2014)
9. Katz, J., Lindell, Y.: Introduction to modern cryptography (2nd edition). CRC press (2014)
10. Khedr, A., Gulak, G.: Securemed: Secure medical computation using gpu-accelerated homomorphic encryption scheme. Cryptology ePrint Archive, Report 2016/445 (2016)

11. Li, M., Zhu, H., Gao, Z., Chen, S., Yu, L., Hu, S., Ren, K.: All your location are belong to us: Breaking mobile social networks for automated user location tracking. In: Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing. pp. 43–52. ACM (2014)
12. Liu, A., Zhengy, K., Liz, L., Liu, G., Zhao, L., Zhou, X.: Efficient secure similarity computation on encrypted trajectory data. In: 2015 IEEE 31st International Conference on Data Engineering. pp. 66–77. IEEE (2015)
13. Liu, X., Wang, B., Yang, X.: Efficiently anonymizing social networks with reachability preservation. In: Proceedings of the 22nd ACM international conference on Conference on information & knowledge management. pp. 1613–1618. ACM (2013)
14. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM workshop on Cloud computing security workshop. pp. 113–124. ACM (2011)
15. Narayanan, A., Thiagarajan, N., Lakhani, M., Hamburg, M., Boneh, D.: Location privacy via private proximity testing. In: NDSS (2011)
16. Peikert, C.: A decade of lattice cryptography. Cryptology ePrint Archive, Report 2015/939 (2015), http://eprint.iacr.org/2015/939
17. Samanthula, B.K., Elmehdwi, Y., Jiang, W.: K-nearest neighbor classification over semantically secure encrypted relational data. IEEE transactions on Knowledge and data engineering 27(5), 1261–1273 (2015)
18. Shahabi, C., Fan, L., Nocera, L., Xiong, L., Li, M.: Privacy-preserving inference of social relationships from location data: a vision paper. In: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. p. 9. ACM (2015)
19. Shirani-Mehr, H., Banaei-Kashani, F., Shahabi, C.: Efficient reachability query evaluation in large spatiotemporal contact datasets. Proceedings of the VLDB Endowment 5(9), 848–859 (2012)
20. Shokri, R., Theodorakopoulos, G., Le Boudec, J.Y., Hubaux, J.P.: Quantifying location privacy. In: 2011 IEEE Symposium on Security and Privacy. pp. 247–262. IEEE (2011)
21. Strzheletska, E.V., Tsotras, V.J.: Ricc: Fast reachability query processing on large spatiotemporal datasets. In: International Symposium on Spatial and Temporal Databases. pp. 3–21. Springer (2015)
22. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10(05), 557–570 (2002)
23. Von Arb, M., Bader, M., Kuhn, M., Wattenhofer, R.: Veneta: Serverless friend-of-friend detection in mobile social networking. In: 2008 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications. pp. 184–189. IEEE (2008)
24. Wang, B., Li, M., Wang, H.: Geometric range search on encrypted spatial data. IEEE Transactions on Information Forensics and Security 11(4), 704–719 (2016)
25. Wang, B., Li, M., Wang, H., Li, H.: Circular range search on encrypted spatial data. In: Communications and Network Security (CNS), 2015 IEEE Conference on. pp. 182–190. IEEE (2015)
26. Wang, S., Zhang, Y., Dai, W., Lauter, K., Kim, M., Tang, Y., Xiong, H., Jiang, X.: Healer: Homomorphic computation of exact logistic regression for secure rare disease variants analysis in gwas. Bioinformatics 32(2), 211–218 (2016)
27. Yi, P., Fan, Z., Yin, S.: Privacy-preserving reachability query services for sparse graphs. In: Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on. pp. 32–35. IEEE (2014)