Geosocial Query with User-Controlled Privacy

Peizhao Hu
Department of Computer Science
Rochester Institute of Technology
New York, USA
ph@cs.rit.edu

Sherman S.M. Chow
Dept. of Information Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
sherman@ie.cuhk.edu.hk

Asma Aloufi
Department of Computer Science
Rochester Institute of Technology
New York, USA
ama9000@rit.edu

ABSTRACT

Geosocial applications collect (and record) users' precise location data to perform proximity computations, such as notifying a user or triggering a service when a friend is within geographic proximity. With the growing popularity of mobile devices that have sophisticated localization capability, it becomes more convenient and tempting to share location data. But the precise location data in plaintext not only exposes user's whereabouts but also mobility patterns that are sensitive and cannot be changed easily. This paper proposes cryptographic protocols on top of spatial cloaking to reduce the resolution of location and balance between data utility and privacy. Specifically, we interest in the setting that allows users to send periodic updates of precise coordinates and define privacy preferences to control the granularity of the location, both in an encrypted format. Our system supports three kinds of user queries – "Where is this user?", "Who is nearby?", and "How close is this user from another user?". Also, we develop a new algorithm to improve the multidimensional data access by reducing significant masking error. Our prototype and various performance evaluations on different platforms demonstrated that our system is practical.

ACM Reference format:
Peizhao Hu, Sherman S.M. Chow, and Asma Aloufi. 2017. Geosocial Query with User-Controlled Privacy. In *Proceedings of WiSec '17 , Boston, MA, USA, July 18-20, 2017,* 10 pages.
DOI: 10.1145/3098243.3098245

1 INTRODUCTION

A popular application of location-based services (LBS) is geosocial service, i.e., location-driven features over online social networks (e.g., locating a user's nearby friends within a geographic area, disseminating targeted recommendation within a city). Collecting whereabouts of users is crucial for enabling LBS. Most applications periodic collect updated GPS coordinates of users in plaintexts. To maintain social connection and enjoy the geosocial features, users are tempted to share their location with family members, friends, or even strangers in the vicinity and often disregard the potential privacy implications [3]. Fine-grained location data is highly sensitive [25]. Some geosocial networks even collect exact locations in real time [38]. Aggregating the collected coordinates reveals mobility trajectories and can infer where users live, work, shop, play, and much more. While some LBSs allow the user to be anonymous [38], these patterns make re-identification easier. Some even store a history of the location for each user. Data leaks can lead to severe loss of user privacy, especially when users cannot change their association with a location or routine easily.

In general, users are more comfortable with sharing coarse-grained location, such as simply the city or a larger geographic area, without fear of discovery [10, 32]. Users would demand the ability to *control the granularity* according to *whom to share with*, i.e., *relationships between users.* For example, parents can have access to the precise coordinates, but it becomes at the city level for colleagues. Some applications such as Facebook allow users to specify such a preference, yet, it is the server which controls the selective disclosure. That means the server learns both the most fine-grained location and the view on relationships of each user.

1.1 Spatial Cloaking

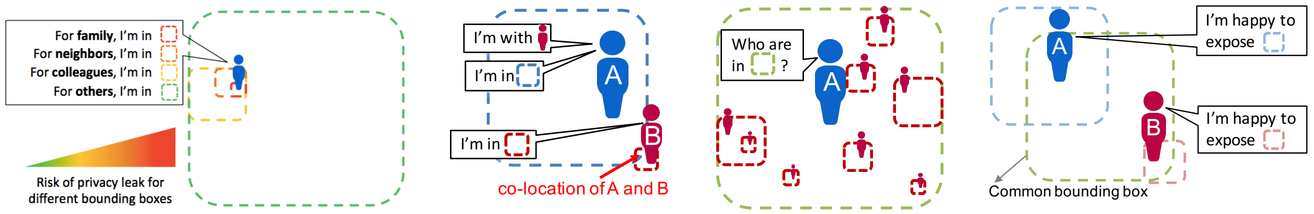
Spatial cloaking [20] reduces the resolution of location data by masking based on user preferences. It preserves locality while changing the level of detail. It produces an area that hides not only the user location but also the trajectories of movement within the masked area, as illustrated in Fig. 1 (a). The risk of being discovered and tracked reduces as we increase the size of the masking area [4].

If a user moves out from the masked area, a new masking area will be generated. For spatial cloaking, the masking areas are non-overlapping. The transition, in this case, incurs a privacy risk since it is for sure that the user just moves across two fixed areas. Specifically, the user must have passed through a point over the shared edge between the two masking areas.

Another potential privacy risk is the co-location problem [32, 38]. Fig. 1 (b) illustrates that in the general case for overlapping masking areas. Even for the specific case of spatial cloaking, if a user reports being co-located with another user, an adversary can infer that both are within the overlapped area which can be significantly smaller than the user specified masking area [32].

To avoid exposing user location through co-location problem, we investigate a *new spatial cloaking technique* that produces a geometric area which encloses the two users, as shown in Fig. 1 (d). This *common bounding box*, without disclosing user locations, can be used to compute the rough proximity between two users. The maximum distance between the two users will be the length of the diagonal of the bounding box. This proximity information can be useful for some social applications which provide services without requiring precise user location and distance measurement between users. In addition, a user can specify a geographic boundary and query the list of nearby friends, as shown in Fig. 1 (c). Ideally, we

WiSec '17 , Boston, MA, USA
© 2017 ACM. 978-1-4503-5084-6/17/07…$15.00
DOI: 10.1145/3098243.3098245



(a) Relationship dependent location masking (b) Risk of co-location (c) Nearby query (d) Proximity of users

Figure 1: Different scenarios for user controllable location privacy

would like to perform this nearby query without revealing locations of the parties involved and their relationships.

1.2 Hiding both Location and Preferences

We propose new algorithms for achieving user controllable privacy when sharing location data. We aim to support the following three kinds of queries: *(i) Bob may ask "where is Alice?" (ii) Alice may ask "which of my friends are nearby?" (iii) A social app may query "how close geographically are Alice and Bob?"* These queries are common in many geosocial applications, such as Google Latitude, Facebook Nearby Friends, Foursquare, Loopt, etc.

The research community has been studying privacy-preserving LBS mechanisms [25, 36, 38]. For example, *mix zones* in which user identities are anonymized and disassociated from the location of identification [15], and *statistical privacy* in which location data is obfuscated but statistical computation is still possible [11]. Many of these solutions require a trusted server to perform anonymization, obfuscation, or resolution reduction via spatial cloaking [17, 33].

We pose to ourselves three requirements: 1) Both the location data and the user preferences of data granularity are encrypted, such that we can reduce the trust requirement on the server to an *honest-but-curious* one. 2) The granularity of location data is controlled (by masking) according to the user preference before processing any query for ensuring security even when users collude. 3) The system can scale with a large number of users who share and periodically update their encrypted location data (and perhaps occasional updates of their encrypted preference). In particular, it does not require a user to send different copies of his/her location data for different parties. That is, every user will still periodically send precise location data to a service provider but in the form of encrypted coordinates; the server will perform computations to fulfill the three queries mentioned above and reply an encrypted result that a user can decrypt using a private key.

To preserve user privacy, we investigate how to use and extend homomorphic encryption (HE) to realize efficient spatial cloaking on encrypted location data. HE schemes offer primitive arithmetic operations on the encrypted data, such as homomorphic addition and multiplication. Fully homomorphic encryption (FHE) allows any computation over encrypted data. However, the state-of-the-art FHE solutions are still not practical enough for processing big data. Our proposed approach instead uses somewhat HE (SWHE) which is more efficient than FHE but with a limit on the number of consecutive multiplications that can be carried out on a ciphertext. For computing the location of a particular user (for answering "Where is Alice?"), over the encrypted user preference and location, our system just extends the very efficient ElGamal encryption scheme [12] which is multiplicatively homomorphic. We will also show to use block-ciphers, such as AES, on top of HE schemes to reduce communication overhead. We show how to support computation over data encrypted under different keys.

Apart from the technical contribution (of our spatial cloaking technique, twisting ElGamal encryption for supporting encryption of zeros without losing security, etc.), our experiments show that the proposed system is efficient across various mobile platforms.

2 PRELIMINARIES

2.1 Geo-Hashing with Z-Order Curve

Geo-hashing reduces the dimensionality of coordinates while preserving locality of points. If two points are close to each other, for most cases they are also close after the transformation. In this paper, we employ geo-hashing using *Z-order* curve as the space-filling curve [16]. It transforms the two-dimensional coordinates into an array of concatenated numbers. Each number corresponds to a particular level of detail at which there is a bounding box that encloses the point. Fig. 2 shows the first three levels of mapping in Microsoft's Bing Maps tile system. Many applications use geo-hashing to index satellite images for efficient retrieval.

When using Z-order curve, the indexing keys (the concatenated numbers) are represented in $\mathbb{Z}_4$, hence the name *quad-key*. To increase one level of detail, we divide a bounding box into four equal sub-boxes, with each assigned a new quad-key appended to the existing quad-key string (also see Fig. 2). Essentially, the longer the common prefix between the quad-keys of two points, the closer they are. Also, a longer quad-key provides a more precise reference to the original coordinates. The details of the transformation steps are described in [37]. Once the coordinates are transformed, another step in spatial cloaking is to reduce the granularity of location data by masking away some precisions in the quad-key.

Given the two-dimensional GPS coordinates (x, y) of a location in the WGS84 encoding, one can compute the quad-key $QK_d = \{q_1, .., q_d\}$ where d is the level of detail. As an example, the quad-key of $(x, y) = (43.584474, -77.675472)$ at $d = 5$ is $QK_5 = 03023$. We can also transform the quad-keys into binary-keys, like $BK_{10} = 0011001011$. Here we only show the quad-keys at a limited level. The maximum level of detail in the form of quad-keys is 22 (or 44 in binary-keys), which corresponds to the exact GPS coordinates.

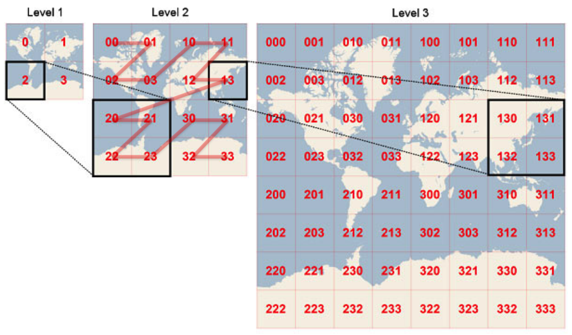


Figure 2: Z-order curve used in Microsoft's Bing Maps

2.2 SomeWhat Homomorphic Encryption

Our proposed system can be instantiated by most of the recent SWHE schemes that support plaintext space R_5 (further discussed below). There are a series of efficient lattice-based SWHE schemes that rely their security on the learning-with-error (LWE) problem or its extension as the ring-LWE problem [5–7, 13]. For self-containedness, we describe the core operations of Naehrig-Lauter-Vaikuntanathan [29] (hereinafter referred as NLV) as an example.

Given a positive integer d, we define $R = \mathbb{Z}[x]/(\Phi_d(x))$ as the ring of polynomials with integer coefficients modulo the d-th cyclotomic polynomial $\Phi_d(x) \in \mathbb{Z}[x]$, and we use $\Phi_d(x) = x^n + 1$ where $d = 2n$ is a power of 2. We use R as the underlying ring structure to define two finite rings: the plaintext space is $R_t = \mathbb{Z}_t[x]/(\Phi(x))$, where $\mathbb{Z}_t$ are integers modulo t, and the ciphertext space is $R_q = \mathbb{Z}_q[x]/(\Phi(x))$, where q is a prime and t is much smaller than q. We also use a Gaussian distribution χ_e on R which we use to introduce noise (error term) into the ciphertexts.

The NLV scheme is based on the ring-LWE assumption [27]: If we uniformly sample s and a_i from a ring $R_q = \mathbb{Z}_q[x]/(\Phi(x))$ and e_i from a Gaussian distribution χ_e, such that $b_i = a_i s + e_i$ for $i \in \mathbb{N}$, then b_i is *computationally* indistinguishable from elements that are uniformly sampled from R_q. In layman terms, we hide secret s covering it with a normal distribution of elements in R_q.

Key Generation. For a secret key $SK = s$, we sample its coefficients from a Gaussian distribution χ_k, denoted by $s \leftarrow \chi_k$, a random element $a_1 \in R_q$, and an error term $e \leftarrow \chi_e$. It is a relatively small private key [27]. To improve security, χ_k is different from χ_e in mean and/or standard deviation. We set the public key to be $PK = (a_0, a_1)$, where $a_0 = -(a_1 s + te)$ and t is the modulus of the plaintext space. s, a_0, a_1, and e are all elements of ring R_q.

Encryption. Given a plaintext $m \in R_t = \mathbb{Z}_t[x]/(\Phi(x))$ and $PK = (a_0, a_1)$, $Enc(m, PK) = (c_0, c_1) = (a_0 e_1 + te_2 + m, a_1 e_1 + te_3) \in (R_q)^2$, where $e_i, i = 1, 2, 3$ are noises sampled independently from the Gaussian distribution χ_e.

Decryption. While any fresh encryption will produce a ciphertext with two components $C = (c_0, c_1) \in (R_q)^2$, homomorphic multiplication (described below) will increase the number of elements in the ciphertext beyond two. Hence, we represent the ciphertext as $C = (c_0, \ldots, c_\xi) \in (R_q)^{\xi+1}$. The decryption function is defined as $Dec(C, SK) = \tilde{m} = \sum_{i=0}^{\xi} c_i s^i \in R_q$.

Homomorphic Operations. Given two ciphertexts $C = (c_0, \ldots, c_\xi)$ and $C' = (c'_0, \ldots, c'_\eta)$, the homomorphic addition is a straightforward component-wise addition. $C + C' = (c_0 + c'_0, \ldots, c_\xi + c'_\eta) \in (R_q)^{\max(\xi,\eta)+1}$, where we might need to pad the ciphertexts by 0's to match the length of the longer ciphertext.

Homomorphic multiplication is more difficult, because of the growth of elements, $C \otimes C' = (\hat{c}_0, \ldots, \hat{c}_{\xi+\eta})$, where $\hat{c}_i$ are appropriate convolutions defined in [7]. In a nutshell, homomorphic multiplication introduces terms with s^i, for $i > 1$. Take the case of multiplying two ciphertexts of length two: $C = (c_0, c_1)$ and $C' = (c'_0, c'_1)$. We want $Dec(C \otimes C', SK) = mm' + te_{mult}$ so that we get back $mm' \pmod{t}$ where m and m' are the corresponding messages, and e_{mult} is the error resulting from multiplying two ciphertexts. Working backward, we know that $m = c_0 + c_1 s \pmod{t}$ and $m' = c'_0 + c'_1 s \pmod{t}$, we thus have:

$$\begin{aligned} mm' + te_{mult} &= (c_0 + c_1 s)(c'_0 + c'_1 s) \pmod{t} \\ &= c_0 c'_0 + (c_0 c'_1 + c_1 c'_0)s + c_1 c'_1 s^2 \pmod{t}. \end{aligned}$$

Thus $C \otimes C' = (\hat{c}_0, \hat{c}_1, \hat{c}_2)$ where $\hat{c}_0 = c_0 c'_0$, $\hat{c}_1 = c_0 c'_1 + c_1 c'_0$, $\hat{c}_2 = c_1 c'_1$. A new term to be multiplied by s^2 is introduced. A "*relinearization*" technique can reduce the number of ciphertext terms [29].

3 RELATED WORK

Many studies aim to preserve location privacy in different senses [4]. Our approaches are closely related to those applying location transformation, cryptographic protocols, or a combination of both. Most work focuses on proximity test, i.e., whether the distance between two parties is less than a threshold, or retrieving information associated with a location, without revealing the location.

Spatial cloaking transforms an exact location to a cloaked area (e.g., rectangle or circle). Hashem and Kulik [20] proposed to compute the cloaked area by coordinating with nearby peers over wireless ad-hoc networks. This distributed approach does not require a trusted server, yet depends greatly on the availability of crowdsourcing participants within the close vicinity. Peng et al. [34] proposed a centralized transformation approach which requires a trusted anonymizer. Khoshgozaran and Shahabi [23] proposed the use of Hilbert curves (similar to Z-order curve) and a one-way trapdoor function to transform a location into a cloaked area containing a precomputed set of places-of-interest stored in a look-up table. We also adopt and extend spatial cloaking, but our use of probabilistic encryption ensures that the location data remains private. Hu et al. [22] have attempted to transform location points from WGS84 system to a 2-dimensional Cartesian coordinate system (UTM projection) and compute the Euclidean distance between two points. The UTM projection will introduce localization errors that increase as the two points become far apart. There are also other cloaking techniques such as semantic cloaking [2], where physical locations are abstracted to semantic locations.

Narayanan et al. [30] proposed to mask users' precise location data with overlapping hexagonal grids and developed a multiparty protocol to check whether two users are nearby. Nielsen et al. [31]

strength the security of the proximity test against an active adversary with zero-knowledge proofs [14]. Saldamli et al. [36] incorporated geometrical properties to reduce the number of encryption needed [30]. These three solutions essentially answer queries about whether two users are within the same fixed-sized cell.

Zhong et al. [41] proposed three protocols for proximity testing. The proximity was determined by computing the Euclidean distance which depends on grid-based location transformation. Also, their protocol uses an additively homomorphic variant of ElGamal which requires solving discrete logarithm for decryption and hence only supports a small range of coordinates. Khoshgozaran and Shahabi [24] propose a symmetric-key based mechanism for range and k nearest neighbor queries among a set of peers who have pre-shared a group key. In the same pre-shared secret key setting, Puttaswamy et al. [35] provide privacy-aware information retrieval from or near a coordinate, rather than a way for the user to share location data with different granularities. Mascetti et al. [28] also support proximity query with privacy preferences. Yet, it translates proximity query to a membership test of all nearby locations, while the privacy preferences are simply enforced by asking the user to report location at different granularities to different friends. In our solution, each user only reports one location, but to be masked differently according to a friend-specific preference. Instead of exhausting all possible locations, our proximity test is based on a direct computation.

Finally, homomorphic encryption is also used in other privacy-preserving applications enabling LBS. For example, Zhang et al. [40] leverage additively HE to realize wi-fi fingerprinting indoor localization, such that the requester users do not reveal their locations while the server does not need to reveal its fingerprints database.

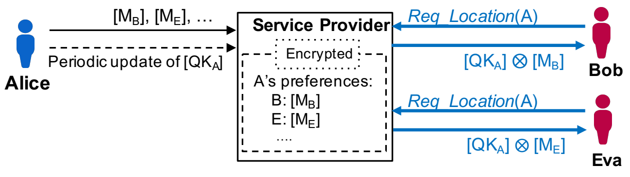
4 COMPUTING ON ENCRYPTED LOCATION

Now we present our geosocial query systems. For simplicity, we assume everything is encrypted by an SWHE scheme. The public key for SWHE is from the user who shares the location data and specifies privacy preference. The ciphertext encrypting m is denoted by $[m]$. In Section 6, we will extend it with a hybrid encryption approach and an additional key setup to allow computation across ciphertexts of different users, yet the user does not need to help in the decryption of the final query result.

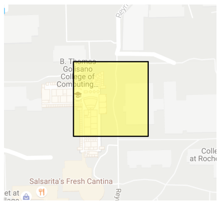
Consider three users, Alice (A) and her friends Bob (B) and Eva (E). All of them periodically transform their GPS coordinates (x, y) into the quad-key representation $QK = QK_{22} = (q_1, \cdots, q_{22})$; $q_i \in \mathbb{Z}_4$ using the Z-order curve geo-hashing technique and send the ciphertexts $[QK] = ([q_1], \cdots, [q_{22}])$ to the service provider (SP). In addition, every user generates a list of encrypted *protection bit-masks* for other users depending on their friendship, e.g., $[M_B] = ([m_1^B], \cdots, [m_d^B])$; $m_i \in \mathbb{Z}_2$ and $[M_E]$ for Bob and Eva respectively. These bit-masks are sent to SP and only updated when there is a change of preference. In this design, only users' current position is updated periodically.

4.1 Query 1: Where is Alice?

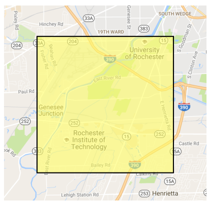
When Bob requests the location of Alice, SP homomorphically multiplies $[QK_A]$ with $[M_B]$ as illustrated in Fig. 3 (a). The bit-masking works fine except 0 in the masked results is ambiguous



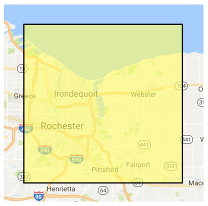
(a) Query 1: Where is Alice?



(b) Bob (building)



(c) Eva (suburb)



(d) Others (city)

Figure 3: System design and resulting masked views for different parties: Bob (d = 18), Eva (d = 12), and Others (d = 10)

since it can represent either a valid result or a forbidden retrieval due to the masking. As an example, $QK_A = 2300$ with $M_B = 1100$ and $M_E = 1111$ yields the same result. We address this problem by transferring each element of QK from $\mathbb{Z}_4$ into $\{1, 2, 3, 4\}$ before the encryption. The resulting QK will be converted back to $\mathbb{Z}_4$ after decryption on the client device.

This query requires a multiplicative depth of one. For higher efficiency, we also prototype our proposed algorithm using the standard ElGamal scheme over a group of prime order p to individually encrypt the location data. Yet, for the bit-mask, an ElGamal encryption of 0 results in a ciphertext 0, which fails to protect the privacy of the user preference. To circumvent this inconvenience, we employ a trick to encode 0 as a random number from $\mathbb{Z}_p$ but excluding everything that might lead to the multiplication result in $\{0, 1, 2, 3, 4\}$, i.e., $\mathbb{Z}_p \setminus \{0, 1, 2^{-1}, 3^{-1}, 4^{-1}, 2, 2 \cdot 3^{-1}, 3, 3 \cdot 2^{-1}, 3 \cdot 4^{-1}, 4, 4 \cdot 3^{-1}\}$.

After this encoding, a bit-mask element of 1 will preserve the location data while a random number will mask it. In the decrypted results, we remove any number that is not an element of $\{1, 2, 3, 4\}$ and produce a masked quad-key string that corresponds to a bounding box with the desired level of data granularity.

Before any additional query computation, the server applies the masking to preserve privacy even when the friends of the user form a coalition. Fig. 3 (b)-(d) show the resulting bounding boxes based on Alice's privacy preferences, who is at the GPS coordinates (43.08460614021896, −77.67964549827582). These bounding boxes not only hide users' positions but also their mobility patterns.

4.2 Query 2: Who is nearby?

Alice may want to retrieve a list of friends who are within a geographic area. To do this, she prepares a *query* bit-mask $[M_Q]$ according to her idea of proximity and sends it to SP as illustrated in Fig. 4. Note that this bit-mark is for querying instead of the aforementioned "encrypted location bit-masks" which are for protecting the level of granularity of the location to be shared.

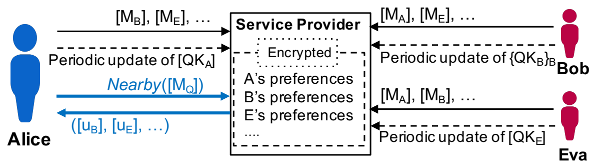


Figure 4: Query 2: Who is nearby?

The location should be protected by the protection bit-mask specifying the granularity level according to the friend-specific preference. Also, by only taking into account the locations up to the granularity level specified by the query bit-mask, the decision of proximity can already be made. With these, apart from applying the self-specified protection bit-mask on the location data as usual, SP also applies the protection bit-mask *of* the counterparty to the ciphertext of each other. Finally, the same query-bit mask is also applied on these post-processed and (further-)masked location of both Alice and her friends. Note that the bit masks are applied by the "AND" operations and hence they are commutative.

Slightly abusing the notation, suppose the ciphertexts after the above processing are denoted by $[QK'_A]$ for Alice and $[QK'_j]$ for the j-th friend of Alice. Then, SP performs a coordinate-wise subtraction $[QK'_A] - [QK'_j]$, where j iterates through all Alice's friends.

When both Alice and her j-th friend are at proximity according to the query (bit-mask) of Alice, the corresponding components up to the proximity level will be all zero. SP then rerandomizes the vector and homomorphically sums up the components of the resulting vector. These computations will produce $[u_j]$ an encryption of a single number that is either 0 implying a user is near Alice or a non-zero positive integer implying this user is not near to Alice.

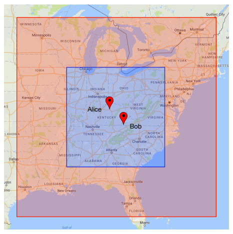
If two users have the same quad-key values after the maskings, they are within the same area. This is similar to the proximity testing in existing work [30, 31, 36]. SP then sends the individually computed result with Alice's friend list $([u_B], [u_E], \cdots)$ to Alice for decryption. Given this list of randomized results, even Alice will only learn whether a friend is close by or not, but nothing else.

4.3 Query 3: How close is Alice and Bob?

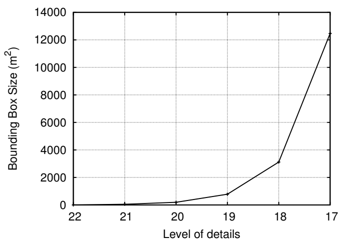
To answer proximity between two users, we propose an algorithm to compute a bounding box that encloses the two users. For brevity, our description below just operates on QK_A, which can always be masked and encrypted beforehand.

Observe that computing such a bounding box is the same as determining the common prefix (CP) from the corresponding two quad-keys (e.g., QK_A and QK_B). The highlighted square in Fig. 2 containing 130, 131, 132, 133 is an example. Since the maximum distance between any two points in the bounding box is the length of the diagonal, we know the upper bound of how far apart the two points are without giving away their exact positions. The amount of noise is controlled by the user-defined preferences.

We construct an arithmetic circuit to homomorphically compute the common prefix mask (*CPM*). See Alg. 1 for the pseudocode. Given two (encrypted and masked) quad-keys QK_A and QK_B, we produce the corresponding binary-key vectors BK_A and BK_B. For brevity, they are denoted by $\mathbb{A}$ and $\mathbb{B}$ below.



(a) Results analysis



(b) Ratio of bounding box size

Figure 5: Result and observation for Query 2

We apply a coordinate-wise XNOR to the two vectors. It returns 1 if the corresponding bit values are the same; otherwise, it returns 0. Lines 3-4 perform a prefix mask *purification* step in which bit value after the leftmost 0 is reset to 0. This process requires consecutive homomorphic multiplications which increase the multiplicative depth[1]. Relinearization is thus needed [29]. XNOR shares similar computation time profile with the existing (homomorphic) equality operator [8] $EQU(\mathbb{X}, \mathbb{Y}) = \wedge_{i=1}^{n}(1 \oplus x_i \oplus y_i)$ where $\mathbb{X} = (x_1, \cdots, x_n)$ and $\mathbb{Y} = (y_1, \cdots, y_n)$, $\wedge$ and $\oplus$ are bitwise *AND* and *XOR*. We believe our solution is relatively efficient.

Algorithm 1: Given the location data of users Alice and Bob, compute the Common Prefix Mask (CPM)

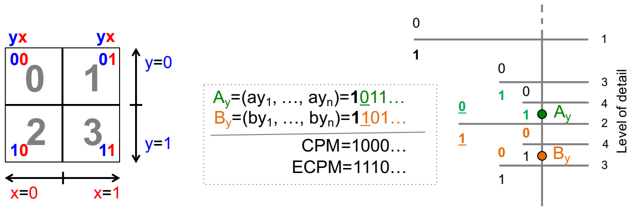
```
1 function CPM ($\mathbb{A}, \mathbb{B}$);
  Input  : Users' geo-hashed binary vectors $\mathbb{A} = (a_1, \cdots, a_n)$
           and $\mathbb{B} = (b_1, \cdots, b_n)$
  Output : the common prefix mask $\mathbb{M}$
2 $g_i = a_i b_i + \bar{a}_i \bar{b}_i$;      ▹ $G = XNOR(\mathbb{A}, \mathbb{B}) = (g_1, \cdots, g_n)$;
3 $m_1 = g_1$;                                ▹ $\mathbb{M} = (m_1, \cdots, m_n)$;
4 $m_i = m_{(i-1)} g_i$;
```

Once we obtain the common prefix mask $\mathbb{M}$, we can compute the common prefix by coordinate-wise homomorphic multiplications $\mathbb{A} \otimes \mathbb{M}$. Finally, the common prefix is used to compute the appropriate bounding box (as described in [37]), whose encryption will be returned to the requester.

Yet, this approach could generate unnecessarily large bounding boxes for points that are close but lay in different quadrants since geo-hashing requires each point to fall into one quadrant at each level. For example, computing common prefix for two neighboring points from regions 122 and 211 at level 3 in Fig. 2 will result in a bounding box of the whole world at level 1 because the two neighboring points lay in different quadrants at level 1. But it will be more appropriate to produce a bounding box containing regions 122, 211, 033, and 300. This is a common problem in solutions that are based on similar geo-hashing techniques [31].

We have tried some heuristics that reduce masking noise by one level, similar to some existing attempts [23] which hashing with different rotations in special cases. Fig. 5(a) shows two bounding boxes: the outer shaded box is the result of this approach, and

[1]Product of one homomorphic multiplication is used in another homomorphic multiplication.



(a) Coordinate (b) Running example

Figure 6: Example of exploiting the coordinates' properties

the inner shaded box is the result generated using a new set of heuristics. This corresponds to 1.20E+13m^2 reduction in area size. Even with the improved version, the area size is unnecessarily large. Fig. 5 (b) shows why reducing the masking noise is necessary as it grows exponentially with each decrease in the level of details. We ask ourselves the following question:

"*Can we further reduce the masking noise?*"

4.4 Improved Query 3

When the quad-keys are converted into binary-keys, each bit represents the position of a point on the corresponding axis. As shown in Fig. 6 (a), we can use one bit on the x-axis to determine if a point is on the left- or right-half of the box. Similarly, the bit value on y-axis determines whether a point is on the top- or bottom-half.

The above properties can be applied at any level of detail, which allows us to develop an algorithm that can substantially reduce the masking noise. Alg. 2, 3, and 4 describe the pseudocode. Fig. 6 (b) illustrates a running example of two binary-key vectors $\mathbb{A}_y = (ay_1, \cdots, ay_n) = 1011\cdots$ and $\mathbb{B}_y = (by_1, \cdots, by_n) = 1101\cdots$. These two vectors are the y-axis only binary vectors extracted from the two input binary-keys $\mathbb{A}$ and $\mathbb{B}$ when calling the *ECPM*($\mathbb{A}, \mathbb{B}$) function. We explain the operations acting on the y-axis vectors. The same operations will be applied separately to the x-axis vectors.

Upon receiving a request to compute a bounding box using *ECPM*, which splits the coordinates $\mathbb{A}$ and $\mathbb{B}$ into the corresponding x-axis and y-axis vectors. For each pair, we compute the pairwise common prefix mask *PCPM* in Alg. 3. In *PCPM*, we first compute the common prefix mask, *CPM*($\mathbb{P}, \mathbb{Q}$) for $\mathbb{P}$ and $\mathbb{Q}$ corresponding to $\mathbb{A}_y$ and $\mathbb{B}_y$ respectively (similar process applies to $\mathbb{A}_x$ and $\mathbb{B}_x$). We then find the leftmost position where the two vectors differ and store this information as $\mathbb{C}$ and $\mathbb{S}$. Suppose it is at the j position, we extract the bit value homomorphically at ay_j and by_j. Then, we use ay_j (in *EPM*) to check with the bit value at ay_k where $k = (j+1, \cdots, n)$. If $ay_k \neq ay_j$, we move to check ay_{k+1}; otherwise, we know $\mathbb{A}_y$ will only match up to level $(k-1)$. Same operations will be performed on $\mathbb{B}_y$ vector using the by_j bit value. Fig. 6 (b) illustrates that $\mathbb{A}_y$ and $\mathbb{B}_y$ is logically matched up to level 3 because $ay_4 \neq ay_j$, but $by_4 = by_j$. In contrast, if we only use the *CPM*($\mathbb{A}, \mathbb{B}$), then the two points are matched only up to level 1; because when $j = 2$, $ay_j = 0$, and $by_j = 1$. In essence, if two points are close, they should be close to each other when we increase the level of detail in both axes. We obtain the common prefix mask between $\mathbb{A}_y$ and $\mathbb{B}_y$ by a simple coordinate-wise multiplication, as shown in Line 6 of Alg. 2.

Algorithm 2: Given the location data of users Alice and Bob, compute the Extended Common Prefix Mask (ECPM)

```
1 function ECPM ($\mathbb{A}, \mathbb{B}$);
  Input  : Users' geo-hashed binary vectors $\mathbb{A} = (a_1, \cdots, a_n)$
           and $\mathbb{B} = (b_1, \cdots, b_n)$
  Output : the extended common prefix mask $\mathbb{M}$
2 $\mathbb{A}_x[i] = \mathbb{A}[2i] = (a_2, \cdots, a_n)$;
  $\mathbb{A}_y[i] = \mathbb{A}[2i-1] = (a_1, \cdots, a_{n-1})$;
3 $\mathbb{B}_x[i] = \mathbb{B}[2i] = (b_2, \cdots, b_n)$;
  $\mathbb{B}_y[i] = \mathbb{B}[2i-1] = (b_1, \cdots, b_{n-1})$;
4 $\mathbb{X} = PCPM(\mathbb{A}_x, \mathbb{B}_x)$;        ▷ $\mathbb{X} = (x_1, \cdots, x_n)$;
5 $\mathbb{Y} = PCPM(\mathbb{A}_y, \mathbb{B}_y)$;        ▷ $\mathbb{Y} = (y_1, \cdots, y_n)$;
6 $\mathbb{M} = \mathbb{X}\mathbb{Y}$;                    ▷ Bitwise
```

Algorithm 3: Given pairs of vectors for the x-axis ($\mathbb{A}_x, \mathbb{B}_x$) and y-axis ($\mathbb{A}_y, \mathbb{B}_y$), compute the Pairwise Common Prefix Mask (PCPM)

```
1 function PCPM ($\mathbb{P}, \mathbb{Q}$);
  Input  : Two vectors $(\mathbb{P}, \mathbb{Q}) = (\mathbb{A}_x, \mathbb{B}_x)$ or
           $(\mathbb{P}, \mathbb{Q}) = (\mathbb{A}_y, \mathbb{B}_y)$
  Output : the pairwise common prefix mask $\mathbb{W}$
2 $\mathbb{C} = CPM(\mathbb{P}, \mathbb{Q})$;        ▷ $\mathbb{C} = (c_1, \cdots, c_n)$;
3 $s_i = \bar{c}_{i-1} + \bar{c}_i$; ▷ $\mathbb{S} = (s_1, \cdots s_n)$ indicates the leftmost position
    where $\mathbb{P}$ and $\mathbb{Q}$ differ. $\bar{c}_i$ is complement of $c_i$;
4 $\mathbb{U} = EPM(\mathbb{P}, \mathbb{C}, \mathbb{S})$;        ▷ $\mathbb{U} = (u_1, \cdots, u_n)$;
5 $\mathbb{V} = EPM(\mathbb{Q}, \mathbb{C}, \mathbb{S})$;        ▷ $\mathbb{V} = (v_1, \cdots, v_n)$;
6 $\mathbb{W} = \mathbb{U}\mathbb{V}$;                    ▷ Bitwise
```

Algorithm 4: Given a vector from the pairs ($\mathbb{A}_x, \mathbb{B}_x$) or ($\mathbb{A}_y, \mathbb{B}_y$), generate an Extended Prefix Mask (EPM)

```
1 function EPM ($\mathbb{R}, \mathbb{C}, \mathbb{S}$);
  Input  : Three vectors: $\mathbb{R} = \mathbb{P}$ or $\mathbb{R} = \mathbb{Q}$, $\mathbb{C}$ and $\mathbb{S}$ are the
           same as in Alg. 3
  Output : An extended prefix mask $\mathbb{E}$ generated using the
           input vectors
2 $d = \oplus_{i=1}^{n}(s_i r_i)$;   ▷ Calculate the bit value in $\mathbb{R}$ at the leftmost
    position where $\mathbb{P}$ and $\mathbb{Q}$ differ
3 $g_i = d \oplus r_i; \forall i = 1, \cdots, n$;        ▷ $\mathbb{G} = (g_1, \cdots, g_n)$;
4 $h_i = g_i \bar{c}_i; \forall i = 1, \cdots, n$; ▷ $\mathbb{H} = (h_1, \cdots, h_n)$; $\bar{c}_i$ is complement
    of $c_i$
5 $t_i = c_i + s_i + h_i; \forall i = 1, \cdots, n$;    ▷ $\mathbb{T} = (t_1, \cdots, t_n)$; $c_i \in \mathbb{C}$;
    $s_i \in \mathbb{S}$;
6 $e_1 = t_1$;                                ▷ $\mathbb{E} = (e_1, \cdots, e_n)$;
7 $e_i = e_{i-1} t_i; \forall i = 2, \cdots, n$;
```

To elaborate, given two vectors $\mathbb{A}$ and $\mathbb{B}$, Alg. 2 describes the matching operations in which vectors on x- and y-axis are extracted accordingly (in lines 2 and 3) for the PCPM algorithm. Alg. 4 describes the operations performed on vectors $\mathbb{A}_y$ or $\mathbb{B}_y$ using the inputs of common prefix between them, $\mathbb{C}$, and a selector mask $\mathbb{S}$ generated in Alg. 3. In Alg. 4, we first compute the corresponding bit values at the leftmost position j where vectors $\mathbb{P}$ and $\mathbb{Q}$ differ.

We achieve the subsequent operations using a concept of a binary multiplexer. Finally, we merge all masks and run a prefix mask *purification* step (also used in line 6 of Alg. 1) to reset the bit values after leftmost 0 to 0. We develop homomorphic functions for all these operations. The results are validated using a plaintext version.

The result from this query is a bounding box that encloses both the requesting and responding users for a third-party (e.g., geosocial app). As illustrated in Fig. 8 (d), a third-party will not learn the exact location of either user within the resulting bounding box. As intermediate results for individual users, smaller bounding boxes (illustrated in Fig. 8 (b-c)) will not reveal the exact location of users due to the masking process.

5 EVALUATION AND DISCUSSION

5.1 Implementation and Evaluation Platforms

We instantiate our proposed framework using NLV [29] scheme. We use C++ with the support of polynomial operations from the Number Theory Library (NTL), which depends on the GNU Multiple Precision Arithmetic Library (GMP). We verified the correctness of our implementation through extensive validations, and compared our performance results with the data reported by Naehrig et al [29].

We conducted our performance evaluations on four platforms: Raspberry Pi model B+ (ARM1176JZF-S, 700 MHz, 512 MB memory), ODROID-C2 (Cortex-A53, 2 GHz, 2 GB), MacBook Pro (Intel core i5, 2.6 GHz, 2 GB), and Amazon EC2 (Xeon E5-2670, 2.5 GHz, 1 GB). These platforms run Raspbian, Ubuntu 16.04, or Mac OSX El Capitan with the standard installation of packages. The specifications of these platforms represent different classes of devices; from low-end smartphones to the resource-rich cloud computing environments. The ODROID board is specifically designed for Android application.

5.2 Evaluation Results

For HE, parameter selection determines the correctness, security, and performance. For the prototype implementation, we used the parameter settings given in Table 1: t is the plaintext space modulus, n is the degree of the polynomial $\Phi(x)$, $\lceil \log_2 q \rceil$ is the bit-length of q, and L is the required level of homomorphic multiplications. We selected the parameters similar to the NLV paper [29].

Table 1: Parameter settings

Parameter	HE Test	Queries 1, 2	Queries 3, 4
t	2	5	2
n	64	1024	54
$\lceil \log_2 q \rceil$	128	38	1300
L	1	1	44

We repeat each experiment 100 times on various platforms and record the average computation time of each operation as well as the standard deviation of the mean (which was relatively small). In our applications, most of the homomorphic operations are to be executed in the cloud environment, so we mainly focused on the performance of Amazon EC2. We can expect better performance if we run our experiments on faster cloud service configurations, rather than the restricted service that was we used. The performance results of the other platforms also demonstrated that the proposed approaches are feasible on common mobile devices.

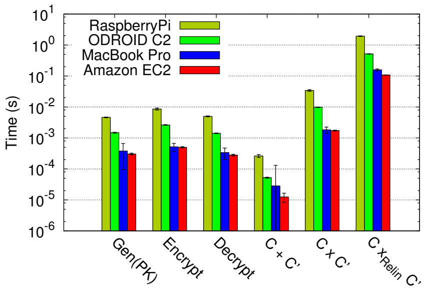
5.2.1 *Common homomorphic operations.*

Using the parameter settings in Table 1, Fig. 7 (a) shows the computation time of all homomorphic operations (addition, multiplication, and multiplication with relinearization) and public-key operations (key generation, encryption, and decryption). As expected, the computation time depends mainly on the CPU clock speed of the respective platform. For the same experiment, Raspberry Pi takes a significantly longer time than the other platforms. For example, one homomorphic multiplication took 34.12ms on the Pi, but it took 1.73ms on Amazon EC2. If consecutive multiplication is required, it took 107.68ms for Amazon EC2 to perform the multiplication and the relinearization step. Note also that the times are plotted in log-scale. Our results conform the expectation that homomorphic multiplication takes more time (longer if the relinearization step is included) compared to other HE operations [21].

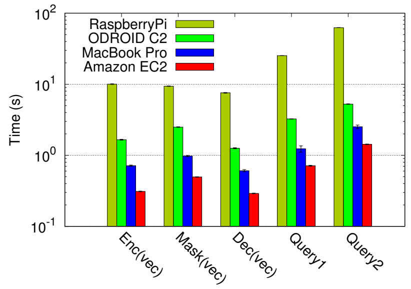
5.2.2 *Our proposed approaches.*

Fig. 7 (b) shows the feasibility study results of the spatial cloaking with HE schemes. Homomorphic encryption, masking, and decryption of a vector take approximately 0.7s on the MacBook Pro laptop and approximately 1.6s on the ODROID C2 board. These operations take about 10s on the RaspberryPi, which is reasonable. One interesting observation is that, different from the other two sets of experiments, Amazon EC2 takes longer than the MacBook Pro for all three operations. This is because spatial cloaking with HE involves sequential homomorphic multiplication of vector elements, and the MacBook Pro laptop results in faster computation time due to the high clock speed. However, once the complexity increases in each homomorphic operation the Amazon EC2 will outperform the laptop. For example, the total run time for Query 1 is 1.23s for the laptop and 0.71s for the EC2 instance. Similarly, the total run time for Query 2 is 2.51s and 1.43s respectively. We also implemented the ElGamal-based algorithm for performing Query 1, the total run time is around 76ms on the MacBook Pro.

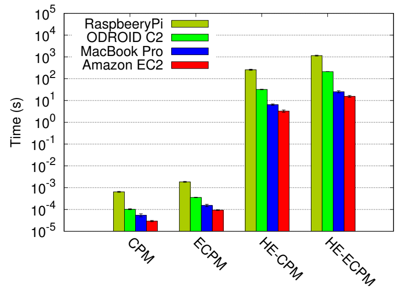
Fig. 7 (c) shows the computation time of different common prefix mask extraction processes (related to Query 3 and the improved Query 3). The computation time for the plaintext version (CPM and ECPM) serves as a reference to show the ratio of computation time increase. We can observe similar complexity increase in the ciphertext version using HE (HE-CPM and HE-ECPM). The HE-ECPM uses significantly more homomorphic operations, which increases the level of homomorphic multiplication with relinearization. However, the time increase is still reasonable. The best average computation time for the HE-ECPM approach is 15.52s, which is achieved in the Amazon EC2 environment. The HE-CPM approach on the same platform is 3.27s. The set of homomorphic operations on RaspberryPi took longer time. The Pi took 257.41s and 947.31s for computing HE-CPM and HE-ECPM respectively. Taking more than one minute to perform a common prefix mask may seem a bit too long, yet the users can control how their private data is used. We expect that many of the bit-wise operations could be significantly improved with parallel processing techniques. As in our current prototype demo system, the computation time for multiple requests will be a multiple of the individual HE operation's run time shown in Fig. 7. Exploring parallel processing architectures to speed up the computation will be a future work.



(a) Homomorphic operations

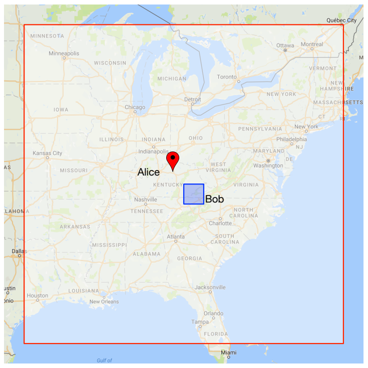


(b) Query 1 and Query 2

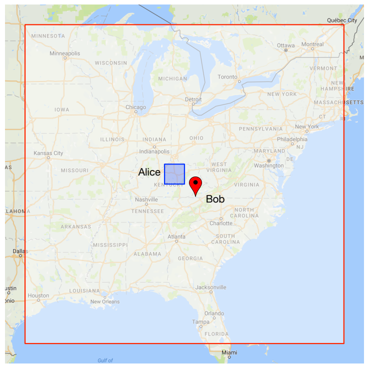


(c) Query 3 and Improved Query 3

Figure 7: Computation times measurement of different homomorphic operations: [Gen(PK)] public key generation; [Encrypt] encryption; [Decrypt] decryption; [$C + C'$] addition; [$C \otimes C'$] multiplication; [$C \otimes_{Relin} C'$] multiplication with relinearization; [Enc(vec)] encryption of the user location vector; [Mask(vec)] apply user defined mask on user location vector; [Dec(vec)] decryption of the user location vector; [CPM] common prefix mask (in plaintexts); [ECPM] extended common prefix mask (in plaintexts); [HE-CPM] common prefix mask with HE; [HE-ECPM] extended common prefix mask with HE.



(a) Alice's view



(b) Bob's view

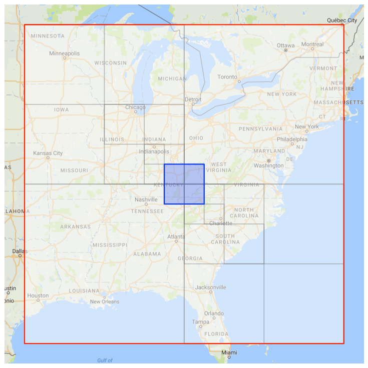

(c) Combined view

Figure 8: Bounding boxes produced by our demo system: these boxes are generated using location data of Alice and Bob; the outer bounding box is generated by the HE-CPM approach and is added for comparison.

Fig. 8 shows examples of bounding boxes generated by our demo system for different parties. In all three figures, the outer bounding box is generated by the HE-CPM approach and it is added for comparison. In Fig. 8 (b) and (c), users can see their current coordinate point as well as the cloaked area of the other user. These bounding boxes are generated by the HE-ECPM approach. Compared to the outer bounding box, the HE-ECPM approach produces bounding boxes which are two level smaller for the two example coordinates used in these experiments; this corresponds to 1.40E+13m^2 reduction in area size as shown in the figure. For these example coordinates, the reduction in area size in HE-ECPM is more significant than the HE-CPM with heuristic rules. If both users are happy for the cloud server to release their masked location, their bounding boxes can be shared with third-party applications. Alternatively, we can generate a combined view with slightly larger cloaking area, as shown in Fig. 8 (d).

6 EXTENSION TO A PRACTICAL SETTING

The description of our system assumes SWHE is used directly. However, due to ciphertext expansion, SWHE produces very large ciphertexts which are costly to transmit; especially, for the periodically updated location data. Moreover, since the data is encrypted in a public-key manner, each user (the "owner" of the location data) needs to help in the decryption of the final encrypted result.

We present an extension that adapts a block-cipher to reduce the overhead and leverage the non-colluding assumption between SP and users to support transformation of ciphertexts. Our goal is to free the data owners from decryption for every single encrypted result for querying their location data, by a one-time setup with each friend.

We will use AES to instantiate the block-cipher. Recall that $[\cdot]_A$ denotes an SWHE ciphertext for Alice. In the following, $\{\cdot\}_A$ denotes an AES ciphertext under the secret key of Alice.

6.1 Key Setup

When a user is first registered to the system, an AES key is generated and stored securely. When two users, say Alice and Bob, become friends, they set up a public key PK_{AB} jointly owned by them. This key can be set up, for example, by a distributed coin-flipping protocol to agree with the randomness, then using it to derive the private key of the underlying SWHE scheme. Now, both of them send this joint public key and an SWHE encryption of his or her own AES key under such a joint key. This will allow SP to convert user data into the HE domain.

6.2 Hybrid SWHE

We adopt a hybrid encryption approach [26, 29], such that the users only use block-cipher to encrypt their preferences and the periodically updated location data. Most of the more expensive public-key / homomorphic operations are left to the more powerful SP.

Recall that in the one-time setup, each user will send a list of HE encryption of the AES key for each friend and a list of AES encrypted bit-masks as privacy preferences for the friends. For example, Alice uploads $[k_A]_{AB}, \cdots$ and $\{M_B\}_A, \cdots$ to SP. In addition, each user periodically sends an AES encryption of the location data in the form of quad-key; that is, $\{QK_A\}_A$ for the case of Alice. This significantly improves the performance at the client slide and reduces the communication overhead.

In SP, the AES encrypted location data $\{QK_A\}_A$ is transformed into a ciphertext $[QK_A]_{AB}$ encrypted by SWHE using the *homAES-dec()* function [18, 19]. Note that all ciphertexts are either encrypted under corresponding symmetric keys for AES or public-keys for SWHE.

6.3 Computing on Data Encrypted under Multiple Keys

Upon receiving a user query, SP transforms the encrypted and masked location data from an encryption under an individual symmetric key to a public-key jointly owned between the two involving users. As an example, an AES ciphertext QK_{A_A} for Alice will be first transformed into $[QK_A]_{AB}$ which is an encryption under a joint key between Alice and Bob, then it can be easily masked (as described in Section 4.2 and becomes $[QK'_A]_{AB}$.

6.4 Key Revocation

Simply put, revocation can always be done by generating a new key and re-encrypting everything. When a user Alice wants to revoke her AES key, she can safely do that by generating a new key and encrypt new data using it. She will update her key by sending a new encrypted $[k_A]_{AB}$ to SP just like when she has newly acquired a friend. Without the new AES key, SP will not be able to perform the required homomorphic computations for the new location data. Hence, she can safely revoke the AES key at need. To renew and revoke a public-key, it can also be done in a similar manner.

6.5 Security Analysis

We adopt a *semi-honest* threat model. The server follows the protocol specifications but is curious to infer the locations or privacy preferences of users from the data collected through our system. Here, we consider these two possible coalitions.

When SP remains honest, our systems offer protection by masking user locations. As described in Section 4.1, SP first masks user location data before any query computation, with the data granularity is supplied by the user who owns the location data. Any user coalition can only learn the location up to the finest level that assigned to the "best" corrupted friend since only the friends of a user can get the decryption of the final query result returned by SP.

If SP deviates from the protocol, SP can skip the masking step and directly send the precise (encrypted) location data to the requesting user. In other words, the risk here is that the user loses the control of granularity. It is because we separated the location data from the granularity control. In other words, our system makes a trade-off for the uploading bandwidth requirement of the user and the efficiency of periodic updates.

If SP colludes with a corrupt user, any corrupt user who is trusted by the victim has the private key (corresponding to the public joint key) to recover the AES key of and hence the precise locations of the victim user. This is similar to the above threat. Again, under our efficiency constraint, there is not much security we can hope for since it is SP who enforces the masking and performs the computation. In other words, SP can skip the masking as we discussed, and not perform the computation but just return the (encrypted) result to the corrupt user. The corrupt user can for sure decrypt the result since that is part of the guarantee on the correct functionality. One can employ multi-key threshold homomorphic encryption [1] instead, yet it will involve the data owner to help in decryption for every query results.

We leave upgrading the security under such a malicious adversary model while keeping a similar level of efficiency as a future work. A possible approach is to resort to a non-colluding two-party computation model [9, 39].

7 CONCLUSIONS

We designed cryptographic algorithms based on homomorphic encryption for three major kinds of queries in a geosocial network. Users can specify privacy preference to control the data granularity when sharing the location data with other parties. Both the location data and the preferences are encrypted. Location privacy is protected by masking user location data according to the user preference. We also designed a new spatial cloaking algorithm that not only addresses the co-location problem but reduces error significantly. We implemented these algorithms based on recent advances in somewhat homomorphic encryption. We developed a demo system for proof-of-concept validation and conducted performance evaluations on mobile devices with different specifications.

ACKNOWLEDGEMENT

The authors are grateful for the comments of Cristina Nita-Rotaru, Russell W.F. Lai, Siyu Zhu, and other anonymous reviewers.

Sherman Chow is supported in part by General Research Fund (Grant No. 14201914) and the Early Career Award from Research Grants Council, Hong Kong.

REFERENCES

[1] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. 2012. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In *EUROCRYPT*.

Springer-Verlag.

[2] Omer Barak, Gabriella Cohen, and Eran Toch. 2016. Anonymizing mobility data using semantic cloaking. *Pervasive and Mobile Computing, Special Issue on Security and Privacy in Mobile Clouds* 28 (2016), 102–112.

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

[4] Claudio Bettini and Daniele Riboni. 2015. Privacy Protection in Pervasive Systems: State of the Art and Technical Challenges. *Pervasive and Mobile Computing* 17, Part B (2015), 159 – 174.

[5] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. 2013. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In *Cryptography and Coding*. LNCS, Vol. 8308. Springer Berlin Heidelberg, 45–64.

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) Fully Homomorphic Encryption without Bootstrapping. In *Innovations in Theoretical Computer Science (ITCS)*. 309–325.

[7] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *FOCS*. IEEE Computer Society, 97–106.

[8] Jung Hee Cheon, Miran Kim, and Kristin Lauter. 2015. Homomorphic Computation of Edit Distance. In *Workshop on Encrypted Computing and Applied Homomorphic Cryptography (WAHC)*. ACM, Isla Verde, Puerto Rico.

[9] Sherman S. M. Chow, Jie-Han Lee, and Lakshminarayanan Subramanian. 2009. Two-Party Computation Model for Privacy-Preserving Queries over Distributed Databases. In *Network and Distributed System Security Symposium (NDSS)*.

[10] Sunny Consolvo, Ian E. Smith, Tara Matthews, Anthony LaMarca, Jason Tabert, and Pauline Powledge. 2005. Location Disclosure to Social Relations: Why, when, & What People Want to Share. In *Human Factors in Computing Systems (CHI)*. ACM, New York, NY, USA, 81–90.

[11] Cynthia Dwork. 2006. Differential Privacy. In *ICALP (LNCS)*, Vol. 4052. Springer Verlag, Venice, Italy, 1–12.

[12] Taher El Gamal. 1985. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO*. Springer-Verlag New York, Inc., Santa Barbara, California, USA, 10–18.

[13] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. (2012).

[14] Uriel Feige, Amos Fiat, and Adi Shamir. 1988. Zero-Knowledge Proofs of Identity. *J. Cryptology* 1, 2 (June 1988), 77–94.

[15] Julien Freudiger, Maxim Raya, Márk Félegyházi, Panos Papadimitratos, and Jean-Pierre Hubaux. 2007. Mix-Zones for Location Privacy in Vehicular Networks. In *Proceeding of Win-ITS'07*. Vancouver, British Columbia.

[16] Volker Gaede and Oliver Günther. 1998. Multidimensional Access Methods. *ACM Comput. Surv.* 30, 2 (June 1998), 170–231.

[17] B Gedik, Kun-Lung Wu, P S Yu, and Ling Liu. 2006. Processing Moving Queries over Moving Objects using Motion-adaptive Indexes. *IEEE Transactions on Knowledge and Data Engineering* 18, 5 (2006), 651–668.

[18] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2015. Homomorphic Evaluation of the AES Circuit (Updated Implementation). In *Cryptography ePrint Archive*. LNCS, Vol. 7417. Springer Berlin Heidelberg, 850–867. Last Updated on 2015. Originally appeared in CRYPTO 2012.

[19] Shai Halevi and Victor Shoup. 2014. Algorithms in HElib. In *CRYPTO*. Springer, 554–571.

[20] Tanzima Hashem and Lars Kulik. 2011. "Don't trust anyone": Privacy Protection for Location-Based Services. *Pervasive & Mobile Computing* 7, 1 (2011), 44 – 59.

[21] Peizhao Hu, Tamalika Mukherjee, Alagu Valliappan, and Stanislaw Radziszowski. 2016. Evaluation of Homomorphic Primitives for Computations on Encrypted Data for CPS systems. In *IEEE CPS Week Smart City Security and Privacy Workshop (SCSP-W)*. Vienna, Austria.

[22] Peizhao Hu, Tamalika Mukherjee, Alagu Valliappan, and Stanislaw Radziszowski. 2016. Homomorphic Proximity Computation in Geosocial Networks. In *BigSecurity, an INFOCOM workshop*.

[23] Ali Khoshgozaran and Cyrus Shahabi. 2007. Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy. In *SSTD*. Springer-Verlag, 239–257.

[24] Ali Khoshgozaran and Cyrus Shahabi. 2009. Private Buddy Search: Enabling Private Spatial Queries in Social Networks. In *Social Intelligence and Networking (SIN), Computational Sci and Engg. (CSE) - Vol. 04*. IEEE Comp. Society, 166–173.

[25] John Krumm. 2009. A Survey of Computational Location Privacy. *Personal Ubiquitous Comput.* 13, 6 (Aug. 2009), 391–399.

[26] Tancrede Lepoint and Michael Naehrig. 2014. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *AfricaCrypt*. Springer, 318–335.

[27] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2013. On Ideal Lattices and Learning with Errors over Rings. *J. ACM* 60, 6, Article 43 (Nov. 2013), 35 pages.

[28] Sergio Mascetti, Dario Freni, Claudio Bettini, X. Sean Wang, and Sushil Jajodia. 2011. Privacy in Geo-social Networks: Proximity Notification with Untrusted Service Providers and Curious Buddies. *The VLDB Journal* 20, 4 (2011), 541–566.

[29] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. 2011. Can Homomorphic Encryption be Practical?. In *Cloud Comp. Sec. Ws. (CCSW)*. 113–124.

[30] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, and Dan Boneh. 2011. Location Privacy via Private Proximity Testing. In *Network and Distributed System Security Symposium (NDSS)*.

[31] Janus Dam Nielsen, Jakob Illeborg, and Michael Bladt Stausholm. 2012. Location Privacy via Actively Secure Private Proximity Testing. In *PerCom Workshop*. Lugano, Switzerland, 381–386.

[32] Alexandra-Mihaela Olteanu, Kévin Huguenin, Reza Shokri, and Jean-Pierre Hubaux. 2014. Quantifying the Effect of Co-location Information on Location Privacy. *Privacy Enhancing Technologies* 8555, Chapter 10 (2014), 184–203.

[33] Femi Olumofin, Piotr K Tysowski, Ian Goldberg, and Urs Hengartner. 2010. Achieving Efficient Query Privacy for Location Based Services. In *Privacy Enhancing Technologies Symposium (PETS)*. Springer-Verlag, 93–110.

[34] Tao Peng, Qin Liu, and Guojun Wang. 2013. Privacy Preserving for Location-Based Services Using Location Transformation. *CSS* 8300, Chap. 2 (2013), 14–28.

[35] Krishna P. N. Puttaswamy, Shiyuan Wang, Troy Steinbauer, Divyakant Agrawal, Amr El Abbadi, Christopher Kruegel, and Ben Y. Zhao:. 2014. Preserving Location Privacy in Geosocial Applications. *IEEE Trans. Mob. Comput.* (2014), 159–173.

[36] Gokay Saldamli, Richard Chow, Hongxia Jin, and Bart Knijnenburg. 2013. Private Proximity Testing with an Untrusted Server. In *ACM WiSec*. ACM, 113–118.

[37] Joe Schwartz. 2012. Bing Maps Tile System. https://msdn.microsoft.com/en-us/library/bb259689.aspx. (2012).

[38] Carmen Ruiz Vicente, Dario Freni, Claudio Bettini, and Christian S. Jensen. 2011. Location-Related Privacy in Geo-Social Networks. *IEEE Internet Computing* 15, 3 (2011), 20–27.

[39] Boyang Wang, Ming Li, Sherman S. M. Chow, and Hui Li. 2014. A tale of two clouds: Computing on data encrypted under multiple keys. In *IEEE Communications and Network Security (CNS)*. 337–345.

[40] Tao Zhang, Sherman S. M. Chow, Zhe Zhou, and Ming Li. 2016. Privacy-Preserving Wi-Fi Fingerprinting Indoor Localization. In *Advances in Information and Computer Security (IWSEC)*. 215–233.

[41] Ge Zhong, Ian Goldberg, and Urs Hengartner. 2007. Louis, Lester and Pierre - Three Protocols for Location Privacy. In *Privacy Enhancing Technologies*. 62–76.