Universal Location Referencing and Homomorphic Evaluation of Geospatial Query

Asma Aloufi, Peizhao Hu, Hang Liu, and Sherman S. M. Chow

Abstract—Location data is an important piece of contextual information in location-driven features for geosocial and pervasive computing applications. In this paper, we propose to geo-hash locations using space-filling curves, which are dimension reduction techniques that preserve locality. The proposed location referencing method is agnostic to specific maps or precoded location models and can effectively preserve users' location privacy based on user preferences. We employ post-quantum-secure encryption on location data and privacy preferences to minimize the risk of data leakage. We also design three algorithms to homomorphically compute geospatial queries on the encrypted location data without revealing either user locations or user preferences. One of the three proposed algorithms reduces the multiplicative depth by more than half; thus, significantly speeding up homomorphic computations. We then present a prototype of the proposed system and algorithms using a somewhat homomorphic encryption scheme and our optimization techniques. A systematic evaluation of the prototype demonstrates its utility in spatial cloaking.

Index Terms—Location privacy, Geohashing, Spatial cloaking, Homomorphic encryption

1 Introduction

Location data is an enabler in location-based services (LBS) and applications (apps), such as social network apps and dating apps [1], [2]. The position of a user or the user's proximity to nearby objects (or other users) is often used to support position and range queries. For example, ***Query 1:*** *Bob may ask "Where is Alice?"* ***Query 2:*** *Alice may ask "Which of my friends are nearby?"* ***Query 3:*** *A social app may inquire "how geographically close are Alice and Bob?"* Similar queries are also useful in Internet of Things (IoT) apps, where a user may wish to retrieve measurements from one or many sensing devices within the vicinity. To support these queries, periodic updates of GPS coordinates of users are usually shared with service providers.

Location data is often considered sensitive information, particularly in countries where General Data Protection Regulation (GDPR) is enforced. GPS coordinates can be used to profile an individual user and his/her lifestyle (e.g., inferring where other users live, work, shop, play, and much more from mobility trajectories) [3]. Generally, users do not mind sharing their coarse-grained location, e.g., the city or a bigger geographical region, rather than their exact location (such as zip code or sub-division where he/she is currently in) [4], [5]. Thus, users would generally rely on a trusted service provider to control the selective disclosure, such as *controlling the granularity of location data* based on *whom to share the location data with*, i.e., *relationships between users*. In other words, the service provider can learn both precise locations and relationships between users. This is potentially a source of data leakage [6]. In addition, recent high profile incidents (e.g., Facebook-Cambridge Analytica) also suggest that blindly trusting service providers to do the right thing may not be in the users' best interest.

Preserving the privacy of users and their location is a relatively well-studied area [3], [7]–[10] , although a number of challenges remain. For example, there have been solutions that rely on a trusted third-party. For example, in solutions based on *mix zones*, which anonymize and disassociate user identities from the location of identification [11], and solutions based on *statistical privacy*, which obfuscate location data but allow statistical computation [12]. A trusted server is required in many of these solutions to perform anonymization, obfuscation, or resolution reduction via *spatial cloaking* [13], [14]. There have also been attempts to design solutions without involving a trusted third-party (TTP) [15], [16]. However, such solutions often depend on precoded location models or specific encoding/decoding methods that are inflexible and not interoperable across geographical areas.

1.1 Contributions

In this paper, we propose a location referencing method, designed to be map-agnostic and provides a robust representation that supports efficient and privacy-preserving evaluation of position and range queries. Location data encoded by this new method allows users to define privacy preferences in the form of bit-masks for spatial cloaking. Then, we employ post-quantum-secure cryptographic techniques to protect the user's encoded location data and privacy preferences, while enabling secure queries on encrypted location with no the need for decryption. Finally, we design optimized algorithms to ensure the proposed system has practical performance. More specifically, our main contributions are three-fold.

First, we use a space-filling curve to geo-hash a GPS coordinate in the WGS84 form (latitude, longitude, and altitude) into a concatenation of numbers, which is a linear representation of the GPS coordinate. Space-filling curves [17] have been widely used as a dimension reduction technique when one wishes to transform high dimensional data into a linear representation, yet preserving the locality of points from the original representation. Hence, we should be able to geo-hash all coordinates on the earth surface and preserve the notion of "closeness" between points to support useful geospatial queries. This approach is a type of *dynamic location referencing*, which works better than *symbolic location referencing* (e.g., Room 3400 in Building 70) that is inflexible and requires a common definition that seldom exists between different

entities (e.g., due to the associated cost and efforts in creating and maintaining such a common reference) [18]. Also, our new geo-hashing method allows us to (i) perform spatial cloaking as very efficient string masking operations, and (ii) support the evaluation of geospatial queries as straightforward string matching operations. In addition, we systematically analyze the effectiveness of different space-filling curves for geo-hashing location data. Furthermore, we extend our system to geo-hash coordinate points from 3D space. This is the first method to support spatial cloaking with a cryptographic approach for location referencing in 3D space, at the time of this research.

Second, we design cryptographic algorithms to support the three position and range queries discussed earlier. These algorithms are constructed based on somewhat homomorphic encryption (SWHE), which allows the desired queries to be evaluated on encrypted location data without decryption in a semi-honest service provider. Specifically, we design an algorithm to evaluate the aforementioned Query 3 with a significant reduction in masking error. This improves the usefulness of our solution. This new algorithm can also be utilized in other applications, where Z-order curve is used as the dimension reduction technique. As location data is encrypted while it is being shared, any passive adversary will not learn the user's location and relationship other than what have been revealed by the protocol. We further present methods to make transmissions of encrypted location data efficient and to support computations to perform over ciphertexts encrypted by different key pairs.

Third, we design optimization and parallelization techniques to significantly accelerate the homomorphic computations; thus, ensuring that the proposed system is practical for real-world deployments. As an example, we design a new common prefix matching algorithm that can significantly reduce the multiplicative depth of the computation, which corresponds to smaller ciphertext size and faster computation over encrypted data due to the ability to select smaller parameters for the SWHE scheme. The proposed algorithm can also be adapted easily to other applications to perform common prefix matching over two encrypted vectors of numbers. This new algorithm reduces the computation time of Query 3, which is the most complex of the three example queries, by an order of magnitude of 7 as compared to our previous design [19].

1.2 Layout

The rest of the paper is organized as follows. Sec. 2 discusses geo-hashing using space-filling curves and the idea of spatial cloaking for privacy-preserving location referencing. This is followed by a discussion on the cryptographic scheme and its primitives in Sec. 3. We describe the system design and security assumption and present the details of our geosocial query system in Sec. 4. We evaluate and discuss the system performance in Sec. 5. Sec. 6 presents the extant literature. Finally, we conclude the paper in Sec. 7.

2 Geo-hashing and Spatial Cloaking

2.1 Geo-hashing using Space-Filling Curves

Space-filling curves [17] reduce the dimensionality of coordinates while preserving position of points. In most cases, the closeness of two close points is preserved after transformation. Because of this property, many map applications use space-filling curves to index satellite images for efficient retrieval, e.g., Microsoft Bing Map [20].

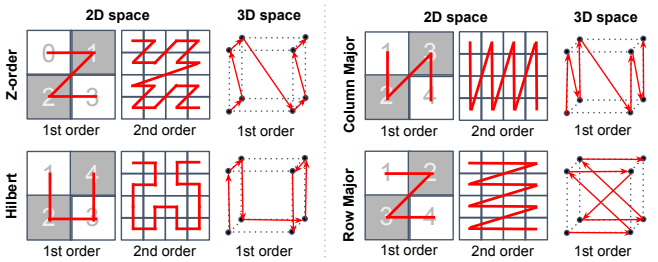


Fig. 1: Geo-hashing with different curves

In this paper, we analyze various space-filling curves, including *Z-order*, *Hilbert*, *Column Major*, and *Row Major* for location referencing. We call this technique geo-hashing. As illustrated in Fig. 1, these curves transform a point in the two- or three-dimensional space into a point on a line that goes through all sub-spaces or all points of a cube. This is done by encoding each sub-space using a number. Hence, the transformation converts a coordinate into a vector of concatenated values. Usually, space-filling curves label each space with an incremental number, except for Z-order curve which repeats between 0-3 in 2D (or 0-7 in 3D). Each number matches a specific level of detail at which there is a box (or a cube) that includes points within the geographical (or geospatial) region. This dimension reduction technique also allow more efficient storage of location data and more efficient computation of geospatial queries.

For clarity, we consider mostly the two-dimensional cases (ignoring attitude) in the rest of our discussions unless specified explicitly. In initial experiments, we found that the locality preserving property works in most cases, but in some cases they do not. As highlighted in the 1st order 2D space of all commonly used curves in Fig. 1, two points laying in the adjacent quadrants remain close to each other after transformation (even though their indexes are different), except for the Hilbert curve with a distance of 2. This problem becomes worse as we increase the order (or level of detail), as shown in Fig. 2 for the 2nd order representation, that is, Z-order curve has a distance of 3 whereas Hilbert curve has 6. We conducted further experiments to measure the Euclidean distance of every pair of neighboring spaces after transformation. Based on our results, Z-order and Hilbert curves outperform the other two curves for the purpose of geo-hashing, but each one has corner cases that make two adjacent points become far apart. Later, we propose an algorithm (namely ECPM for Query 3 in Sec. 4.3) to significantly reduce masking error.

In this paper, we use Z-order curve because it shares similar efficiency for geo-hashing as Hilbert curve, but it encodes coordinates using smaller numbers. Correspondingly, smaller numbers allow us to choose more efficient parameters for our encryption scheme making the proposed system practical. In Z-order curve, indexing keys (i.e., the concatenated values shown in Fig. 2) are represented in $\mathbb{Z}_4$, hence the name *QuadKey*. Correspondingly, we represent points from the 3D space in $\mathbb{Z}_8$ and give the name *OctaKey*. The level of detail cab be increased by dividing an area into four equal sub-areas, with each assigned a new QuadKey appended to the existing QuadKey string. Essentially, the longer the common prefix between the QuadKey of two points, the closer they are. Also, a longer QuadKey provides a more precise

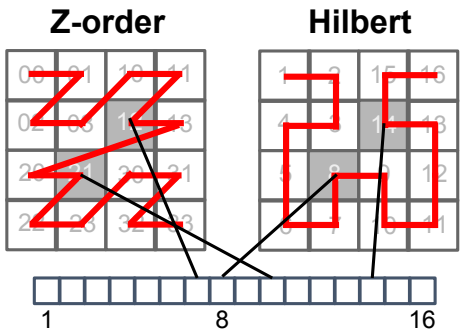


Fig. 2: Adjacent points may apart after transformation

reference to the original coordinates.

Given the two-dimensional GPS coordinates (x, y) of a location in the WGS84 encoding, the QuadKey can be computed as $QK_d = \{q_1, .., q_d\}$ where d is the level of detail [20]. In example, the QuadKey of $(x, y) = (43.084474, -77.675372)$ at $d = 5$ is $QK_5 = 03023$ (or an OctaKey of $OK_5 = 06147$ if we consider attitude). We can also transform the QuadKey into binary representation (hence BinKey), like $BK_5 = 0011001011$ in 2D and $BK_5 = 000110000101110$ in 3D. In this example, we demonstrate the QuadKey at a small level of detail. The maximum level for the QuadKey is 22 (same for OctaKey, but 44 in BinKey for 2D space and 66 in BinKey for 3D space), which corresponds to the precise GPS coordinates. Note, we will mostly use QuadKey and BinKey in the following discussions. We generate the OctaKey and BinKey of coordinates in 3D space only for empirical evaluation.

2.2 Spatial Cloaking

Our geo-hashing technique is based on space-filling curves also has privacy-preserving property for location data, which is similar to spatial cloaking [16] that hides the location data by masking according to user's preference. The result is an area that conceal both the user precise location and the movement trajectories in that masked area, as illustrated in Fig. 3 (a). Users should have the ability to define their location privacy preferences according to the trust level between them and their friends. As the masking area size increases, the risk of being discovered or tracked decreases [21].

When the user leaves a masked area, a new masking area is created, which is not overlapping with the previous one. This raises a privacy risk as the users transition between two fixed-size areas is learned. In other words, the user must have crossed a point on the edge shared between the two masking areas.

The problem of co-location [5], [7] is another possible privacy risk. Figure 3 (b) shows the risk of co-location in a general case of overlapping masking areas. An adversary can deduce, even with spatial cloaking, that two co-locating users are in the overlapped area, which could be much smaller than the user-defined masking area [5].

To prevent revealing the user location by exploiting the issue of co-locating, we study a *new technique for spatial cloaking* (Fig. 3 (d)) which generates a geometric area that includes the two users. Without revealing the locations of users, this *common box* can be utilized to compute two users' proximity. The box's diagonal length is the maximal distance between the two users. This rough proximity can be valuable for many social applications which provide services that do not require users exact location or exact distance between users. Furthermore, a user can define a geographic area and query for the list of nearby friends or devices as shown in Fig. 3 (c). In ideal cases, we want to perform the nearby query without exposing the exact locations of the involved users or their relationships.

2.3 Concealing User's Location and Preferences

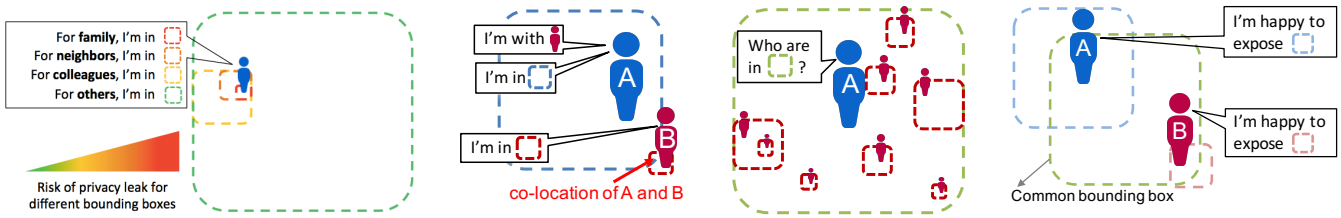
We pose to ourselves three requirements for our proposed system: 1) Both the location and the users preferred level of location granularity are encrypted, reducing the trust requirement on the server to a *semi-honest* one. 2) Before processing any query, the locations level of granularity is controlled (through masking) based on the privacy preference of the user to guarantee security even with colluding users. 3) Our system can be scaled to support a large number of users who send and periodically update their encrypted locations and may occasionally change their encrypted preference. Specifically, the user does not have to provide different versions of their locations for different users. In other words, each user periodically encrypts the precise location and sends it to a server provider. Then, the server provider computes on the encrypted location to answer the three stated queries and returns an encrypted result which the request decrypts using the corresponding secret key.

To achieve these requirements, we study how to accomplish practical spatial cloaking on encrypted location data with homomorphic encryption (HE). HE schemes support computations, such as homomorphic addition and multiplication, on the encrypted data. Our proposed method utilizes somewhat HE (SWHE) which has a limited multiplicative depth, i.e., bounded number of consecutive homomorphic multiplications performed on a ciphertext. Moreover, we address the use of block-ciphers, AES as an example, with the HE schemes to decrease the communication overhead and how to support computation on data encrypted under multiple keys. Due to space limitation, details are in the Appendix.

3 Cryptographic Scheme and Primitives

Our work uses the Brakerski-Gentry-Vaikuntanathan (BGV) SWHE scheme [22] which allows an arbitrary number of additions but *limited* consecutive multiplications. For a security level λ and a cyclotomic polynomial $\Phi(x) = x^\eta + 1$ where η is a power of 2, we define $R = \mathbb{Z}[x]/(\Phi(x))$ to be a polynomial ring of degree η with integer coefficient. Let q and t be two co-prime moduli where $q \gg t$, we define the ciphertext space $R_q = \mathbb{Z}_q[x]/(\Phi(x))$ and the plaintext space $R_t = \mathbb{Z}_t[x]/(\Phi(x))$. Let χ be a Gaussian error distribution over R_q.

The BGV scheme bases its security on the hardness of the ring learning with errors (RLWE) assumption [23]. Let s and A be uniformly sampled elements from R_q, and let e be an error term sampled from the Gaussian distribution χ, the RLWE assumption states that the pair of $(A, b_i = As + te)$ is *computationally* indistinguishable from any uniformly sampled pair of $(A, b_j) \in R_q^2$. Solving the RLWE problem has proved to be as hard as solving the shortest vector problem [24]. A variant of the RLWE problem shows that it is equivalently secure to sample s from a small distribution [24]. For parameters (λ, η, q, t), below describes the essential components of BGV.



(a) Relationship-based location masking (b) Co-location risk (c) Nearby query (d) Proximity of users

Fig. 3: various cases of user-controlled location privacy

KeyGen. Sample a small element $s \xleftarrow{\$} \chi_k$ such that secret key $sk = s$, and sample a small noise element $e \xleftarrow{\$} \chi_e$, where χ_k and χ_e are two Gaussian distributions over R_q. Also uniformly sample $A \xleftarrow{\$} R_q$ The public key is $pk = (A, As + te)$.

Encryption. Given a plaintext message $m \in R_t$, a public key $(A, As + te)$, and a uniformly sampled number $\gamma \xleftarrow{\$} R_t$, we encrypt the message m as $\texttt{Enc}(m, pk) = C = (c_0, c_1) \in R_q^2$ where $c_0 = \gamma A$ and $c_1 = \gamma(As + te) + m$.

Decryption. Given a ciphertext $C = (c_0, c_1) \in R_q^2$, we can decrypt using the secret key sk by computing $\texttt{Dec}(C, sk) = \tilde{m} = c_1 - c_0 s$. The decryption of a ciphertext in the BGV scheme is correct if and only if $(\tilde{m} \bmod t = m)$.

Homomorphic addition. Adding two ciphertexts $C = (c_0, c_1)$, $C' = (c'_0, c'_1) \in R_q^2$ results in $C_{\texttt{add}} = ((c_0 + c'_0), (s(c_0 + c'_0) + t(e + e') + (m + m')) \in R_q^2$.

Homomorphic multiplication. Given two ciphertexts $C, C' \in R_q^2$, their initial homomorphic multiplication yields $\tilde{C}_{\texttt{mult}} = C \cdot C' = (c_0, c_1) \cdot (c'_0, c'_1) = (c_0 \cdot c'_0, c_0 \cdot c'_1 + c'_0 \cdot c_1, c_1 \cdot c'_1) = (c_0, c_1, c_2) \in R_q^3$.

The additional component c_2 includes a quadratic element s^2 resulted from the multiplication $(c_0 \cdot s + t \cdot e + m)(c'_0 \cdot s + t \cdot e' + m')$. One thus needs to use *key switching* [22] after each homomorphic multiplication to reduce the dimension and yield a correct ciphertext that is decryptable by the secret key sk.

4 Blindfolded Querying on Location Data

In this section, we present our privacy-preserving query system. As illustrated in Fig. 4, the system consists of three main roles: system users, a service provider SP, and multiple decryption servers. First, if a user Alice A wants to add a new friend Bob B to the system, they go through a pairing phase and setup the necessary key-pair for AES and SWHE encryption, $k_{\texttt{A}}$, $k_{\texttt{B}}$ and $(pk_{\texttt{A}}, sk_{\texttt{A}})$, $(pk_{\texttt{B}}, sk_{\texttt{B}})$, respectively. Let assume Alice's friend Eva E is already in the system with key-pair $k_{\texttt{E}}$ and $(pk_{\texttt{E}}, sk_{\texttt{E}})$. Due to space limitation, details of hybrid and threshold encryption for communication efficiency and supporting multiple keys are discussed in Appendices A and C. 3We denote $[m]$ and $\langle m \rangle$ as a message encrypted under the SWHE scheme and the AES block cipher respectively.

After the key setup, all users use the Z-order curve geo-hashing technique to periodically convert their location (represented as GPS coordinates (x, y)) into QuadKeys $QK = QK_{22} = (q_1, \cdots, q_{22}); q_i \in \mathbb{Z}_4$. They send the AES encrypted ciphertexts $\langle QK \rangle = (\langle q_1 \rangle, \cdots, \langle q_{22} \rangle)$ to SP. Also, each user generates an encrypted *privacy bit-mask* for each of the other users according to their relationship. For example, Alice generates $\langle M_B \rangle = (\langle m_1^B \rangle, \cdots, \langle m_d^B \rangle); m_i \in \mathbb{Z}_2$ and $\langle M_E \rangle$ for Bob and Eva respectively. This list of bit-masks is shared with SP and only updated when the user changes the preference. On the other hand, users' current positions are periodically updated. All the ciphertexts are transformed by SP to the SWHE domain, as described in Appendix C, before processing a requested query. The processed encrypted result is sent to the user who decrypts it using the proposed decryption protocol described in Appendix B.

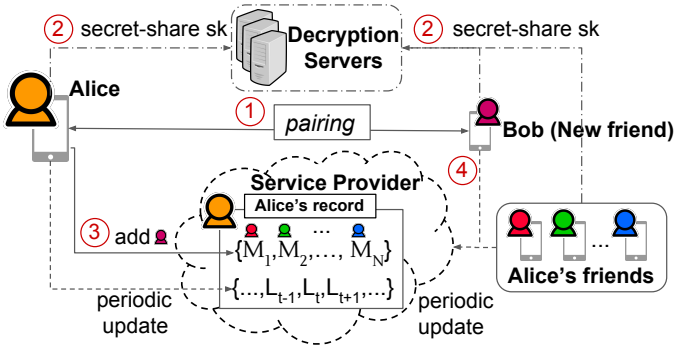
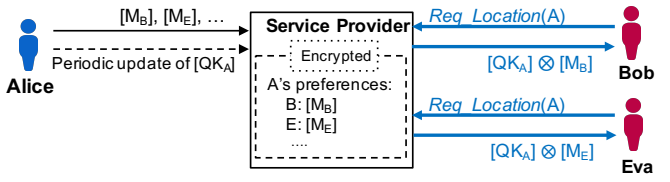
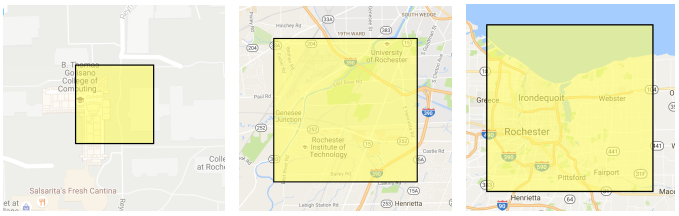


Fig. 4: Overview of system setup



(a) **Query 1**: Where is Alice?



(b) Bob (building) (c) Eva (district) (d) Others (city)

Fig. 5: Query 1 and resulting masked views for different parties: Bob ($d = 18$), Eva ($d = 12$), and Others ($d = 10$)

4.1 Query 1: Where is Alice?

Each time Bob queries on Alices whereabouts, SP masks the location $[QK_A]$ by homomorphically multiplying with $[M_B]$ as Fig. 5 (a) shows. There is an exception in the bit-masking result for 0 because it can be interpreted either as a correct result or as

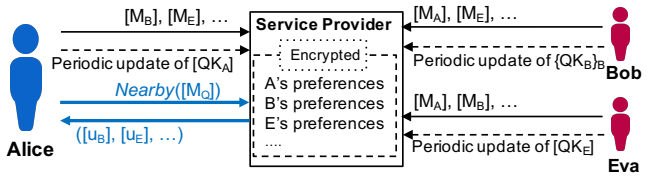


Fig. 6: Query 2: Who is nearby?

a masked result. For example, both bit masks $M_B = 1100$ and $M_E = 1111$ applied on $QK_A = 3100$ obtain the same result. To solve this issue, we increment each value in QK by one such that $q_i \in \{1, 2, 3, 4\}$ before the encryption. With this encoding, the element 1 in the bit-mask preserves the location data while the element 0 masks it.

After decrypting the result QK on the client device, we remove any 0s and convert back the elements to $\mathbb{Z}_4$. This produces a masked QuadKey string which produces a box with the preferred level of detail. The masking process in this query is efficient since it only requires one depth of homomorphic multiplication.

For any further location queries, SP applies this masking procedure first on the location to preserve the user's privacy even with colluding friends. Figures 5 (b)-(d) illustrate the produced boxes based on Alice's privacy preferences, whose precise GPS coordinates are $(43.084451, -77.680069)$. These boxes hide both locations and movement patterns of users.

4.2 Query 2: Who is nearby?

Alice might want to find who, from her friends, is currently within a geographic region. Based on her targeted proximity,she creates a *query* bit-mask $[M_Q]$ and sends it to SP as shown in Fig. 6. The query bit-mask $[M_Q]$ is different from the privacy bit mask defined in Query 1.

Location data is first masked by the privacy bit-mask defined by individual users and masked again by the query bit-mask. The two-round masking process requires a multiplicative depth of two. Hence, we can expect the computation time to be roughly twice of Query 1. With a slight abuse of notation, let the ciphertexts from above processing be denoted by $[QK'_A]$ for Alice and $[QK'_j]$ for her j-th friend in the list. Then, the SP computes an element-wise subtraction $[QK'_A] - [QK'_j]$.

If Alice and her j-th friend are close based on the provided query bit-mask, the output will be zero in all the elements corresponding to the the proximity level. Then, SP re-randomizes the resultant vector and homomorphically aggregates its elements afterwards. The output of these calculations will be a ciphertext $[u_j]$ which encrypts either 0 indicating the friend is close to Alice or a non-zero positive value indicating the friend is not nearby.

Two users are located in the same region if their masked QuadKey are the same. Related work [8], [25], [26] uses similar intuition to provide proximity testing. SP returns to Alice a list of encrypted results $([u_B], [u_E], \cdots)$ corresponding to her friends. Upon decryption, Alice will only learn from this randomized results list whether or not a friend is nearby, but nothing more.

4.3 Query 3: How close are Alice and Bob?

To determine the closeness of two users, we design an algorithm which generates a box enclosing them. Note that generating this box is equivalent to producing the common prefix (CP) from the two users' QuadKeys (e.g., QK_A and QK_B). First, we perform an element-wise XNOR on each bit in the two QuadKeys and produce a vector $\mathbb{V}$. Then, we perform a prefix mask *purification* function (PURIFY) in which bit value after the first leftmost 0 is reset to 0 to obtain the proper prefix mask $\mathbb{M}$.

Algorithm 1 shows the pseudocode and Fig. 7 shows an example of our new algorithm, which reduce the multiplicative depth L and leverage parallelization to speedup homomorphic computations. This new algorithm split the binary prefix mask $\mathbb{V}$ into k blocks of size $\lceil n/k \rceil$ and apply the purification step on all the blocks in parallel. Note that in this step, each block requires $(\lceil n/k \rceil - 1)$ consecutive multiplications. After the first round of the purification step, all blocks have been set according to what we need as outputs. But, we still need to either keep or reset all bits within each block depending on the last bit in the previous block. Intuitively, if the last bit of V_1 is 1, then we keep all the bits in V_2. If the last bit of V_2 is 0, then we reset all the bits in V_3 to 0. This procedure is achieved through homomorphic multiplications with the corresponding bits. The second step requires $(k-1)$ consecutive multiplications to set the bits. Hence, we derive the formula $L = (\lceil n/k \rceil + k - 2)$ to set the scheme's parameter for multiplicative depth.

Decreasing L, the multiplicative depth of the algorithm, is important to improve the run-time. Our study shows that there is a trade off between the multiplicative depth L and the total number of homomorphic multiplications performed when selecting different values of k. For example, setting $k = n$ decreases the number of multiplications to one per block, but the depth becomes $L = n - 1$. The number of blocks has to be carefully selected such that we balance between L and θ, which we derive the latter based on k. Specifically, $\theta = (k-1)(\lceil n/k \rceil - 1) + (\lceil n/k \rceil)(k-2) + 2(n - (\lceil n/k \rceil)(k-1)) - 1$, where $[(k-1)(\lceil n/k \rceil - 1) + (\lceil n/k \rceil)(k-2)]$ is the number of multiplications performed on fully-sized blocks in both steps of purification, and $[2(n - (\lceil n/k \rceil)(k-1)) - 1]$ is the number of multiplications in the last block that may not be of full size. This optimized algorithm allows us to perform the prefix matching procedure in parallel and achieve a significant speedup in the overall system.

Algorithm 1: Given a binary vector, transform it into a proper prefix mask.

```
function PURIFY (𝕍);
Input  : A vector 𝕍 = (v_1, ··· , v_n) as the output of
         XNOR(𝔸, 𝔹)
Output: A purified prefix mask 𝕄
V_i ∈ Partition(𝕍, k);                    ▷ 𝕍 = {V_1, ..V_k};
foreach V_i ∈ {V_1, ..V_k} do
    m_{i,1} = v_{i,1};
    foreach v_{i,j} ∈ V_i do
        m_{i,j} = m_{i,(j-1)} v_{i,j};          ▷ m_{i,j} ∈ 𝕄;
    end
end
foreach V_i ∈ {V_1, ..V_k} do
    foreach v_{i,j} ∈ V_i do
        m_{i,j} = m_{i-1,n/k} * m_{i,j};
    end
end
```

When we acquire the common prefix mask $\mathbb{M}$, we can

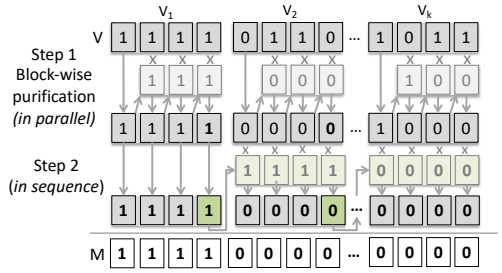
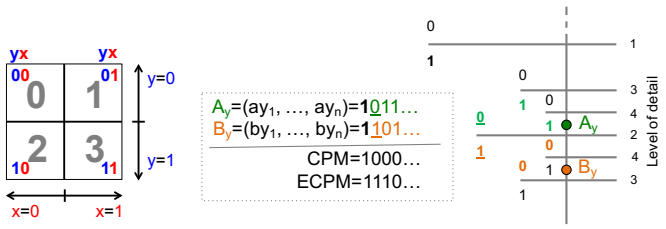


Fig. 7: Illustration of the PURIFY function



(a) Coordinate (b) Running example

Fig. 8: Utilizing the properties of coordinates.

calculate the common prefix by performing a component-wise homomorphic multiplication $\mathbb{A} \otimes \mathbb{M}$. Lastly, the encryption of the common prefix is returned to the requester, who use it to produce the appropriate box (as discussed in [20]). In this case, utilizing this method may produce unnecessarily large boxes for points that are close yet lay in differing quadrants since geo-hashing requires each point to fall into one quadrant at each level.

We tackle this problem by investigating the properties of the Z-order curve coordinate system. When the QuadKey is converted into BinKey, each bit represents the position of a point on the corresponding axis. As shown in Fig. 8 (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.

These properties is applicable at any level of detail, which we leverage to design an algorithm that substantially reduces the masking noise. The pseudocode is described in Alg. 2, 3, and 4. Fig. 8 (b) shows a running example of two BinKey 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 BinKey $\mathbb{A}$ and $\mathbb{B}$ when calling the $ECPM(\mathbb{A}, \mathbb{B})$ function. We explain the operations carried on the y-axis vectors. Same operations will be independently applied to the x-axis vectors.

When a request to compute a box enclosing Alice and Bob is received, the $ECPM$ splits the coordinates $\mathbb{A}$ and $\mathbb{B}$ into the corresponding x-axis and y-axis vectors. For each axis pair, the pairwise common prefix mask $PCPM$ is computed as 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 locate the leftmost position where the two vectors differ and store this information as $\mathbb{C}$ and $\mathbb{S}$. Let it be at the j-th 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 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. 8 (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 $by_j = 1$. Essentially, 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: Calculate the Extended Common Prefix Mask (ECPM) for users' (Alice and Bob) given location data.

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 $\mathbb{M}$
$\mathbb{A}_x[i] = \mathbb{A}[2i] = (a_2, \cdots, a_n)$;
$\mathbb{A}_y[i] = \mathbb{A}[2i-1] = (a_1, \cdots, a_{n-1})$;
$\mathbb{B}_x[i] = \mathbb{B}[2i] = (b_2, \cdots, b_n)$;
$\mathbb{B}_y[i] = \mathbb{B}[2i-1] = (b_1, \cdots, b_{n-1})$;
$\mathbb{X} = PCPM(\mathbb{A}_x, \mathbb{B}_x)$; ▷ $\mathbb{X} = (x_1, \cdots, x_n)$;
$\mathbb{Y} = PCPM(\mathbb{A}_y, \mathbb{B}_y)$; ▷ $\mathbb{Y} = (y_1, \cdots, y_n)$;
$\mathbb{M} = \mathbb{X}\mathbb{Y}$; ▷ Bitwise

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

Function PCPM$(\mathbb{P}, \mathbb{Q})$;
Input : The pair of 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 $\mathbb{W}$
$\mathbb{C} = \text{PURIFY}(\mathbb{P}, \mathbb{Q})$; ▷ $\mathbb{C} = (c_1, \cdots, c_n)$;
$s_i = \bar{c}_{i-1} + \bar{c}_i$; ▷ $\mathbb{S} = (s_1, \cdots s_n)$ denotes the leftmost position where $\mathbb{P}$ and $\mathbb{Q}$ differ. $\bar{c}_i$ is the complement of c_i;
$\mathbb{U} = \text{EPM}(\mathbb{P}, \mathbb{C}, \mathbb{S})$; ▷ $\mathbb{U} = (u_1, \cdots, u_n)$;
$\mathbb{V} = \text{EPM}(\mathbb{Q}, \mathbb{C}, \mathbb{S})$; ▷ $\mathbb{V} = (v_1, \cdots, v_n)$;
$\mathbb{W} = \mathbb{U}\mathbb{V}$; ▷ Bitwise

Algorithm 4: Generate the Extended Prefix Mask (EPM) for a given vector that is based on $(\mathbb{A}_x, \mathbb{B}_x)$ or $(\mathbb{A}_y, \mathbb{B}_y)$

function EPM $(\mathbb{R}, \mathbb{C}, \mathbb{S})$;
Input : The vector $\mathbb{R} = \mathbb{P}$ or $\mathbb{R} = \mathbb{Q}$ and the two vectors $\mathbb{C}$ and $\mathbb{S}$ described in Alg. 3
Output: The extended prefix mask $\mathbb{E}$
$d = \oplus_{i=1}^{n}(s_i r_i)$; ▷ Calculate the bit in $\mathbb{R}$ at the leftmost position where $\mathbb{P}$ and $\mathbb{Q}$ vary
$g_i = d \oplus r_i; \forall i = 1, \cdots, n$; ▷ $\mathbb{G} = (g_1, \cdots, g_n)$;
$h_i = g_i \bar{c}_i; \forall i = 1, \cdots, n$; ▷ $\mathbb{H} = (h_1, \cdots, h_n)$; $\bar{c}_i$ is complement of c_i
$t_i = c_i + s_i + h_i; \forall i = 1, \cdots, n$; ▷ $\mathbb{T} = (t_1, \cdots, t_n)$;
$c_i \in \mathbb{C}$; $s_i \in \mathbb{S}$;
$e_1 = t_1$; ▷ $\mathbb{E} = (e_1, \cdots, e_n)$;
$e_i = e_{i-1} t_i; \forall i = 2, \cdots, n$;

In more depth, the ECPM algorithm describes the operations of prefix matching by extracting the x- and y-axis vectors (in lines

2 and 3 of 4) from two given vectors $\mathbb{A}$ and $\mathbb{B}$ and performing the PCPM on them. Algorithm 4 describes the operations on vectors $\mathbb{A}_y$ or $\mathbb{B}_y$ using the common prefix inputs between them, $\mathbb{C}$, and a selector mask $\mathbb{S}$ generated in Alg. 3. The EPM algorithm starts with computing the corresponding bit values at the leftmost position j in which $\mathbb{P}$ and $\mathbb{Q}$ vectors are different. We accomplish the following operations by utilizing a binary multiplexer concept. Finally, we combine all masks and perform a prefix mask *purification* step (also used in line 6 of Alg. 1) to reset the bit values after leftmost 0 to 0.

The result of this query is a box enclosing the locations of the querying and responding users for a third-party (e.g., geosocial app). As shown in Fig. 10 (d), the precise locations of users remain unknown to the third-party since they are hidden within the resulting box. In fact, the smaller boxes in Fig. 10 (b-c) demonstrate that precise locations of individual users are still protected in intermediary operations due to the masking process.

5 Evaluation and Discussion

5.1 Evaluation Platform

We prototype our proposed framework in C++ and use the HElib [27], [28], which implements the BGV SWHE scheme [22]. We use the Number Theory Library (NTL), which depends on the GNU Multiple Precision Arithmetic Library (GMP), to support of polynomial operations. Parallelization is done with OpenMP. We validate the correctness of our execution through extensive testing.

We created a cloud instance on the CloudLab.us [29] for experiments. We run each of our experiments on a system with 10-core Intel E5-2640v4 at 2.40 GHz, and 64 GB RAM. We set the BGV scheme's security parameter λ to be 128 bits, which corresponds to the security of a scheme with 3072-bit asymmetric key (proposed for safeguarding genomic data [30]). The rest of the BGV scheme parameters were set according to each experiment's requirements.

5.2 Empirical Results

We repeat each experiment 100 times and record the average computation time of each operation as well as the standard deviation of the mean (which was relatively small).

We present our evaluation results in Fig. 9 in log-scale. The first set of results (Figs. 9 (a-c)) evaluates the location queries in 2D space, where the locations are encoded either as QuadKeys (represented as vectors of size 22 in $\mathbb{Z}_4$) or BinKeys of size 44. The second set of results (Figs. 9 (e-h)) evaluates the location queries in 3D space, where the locations are encoded either as OctaKeys (represented as vectors of size 22 in $\mathbb{Z}_8$) or 66-sized BinKeys.

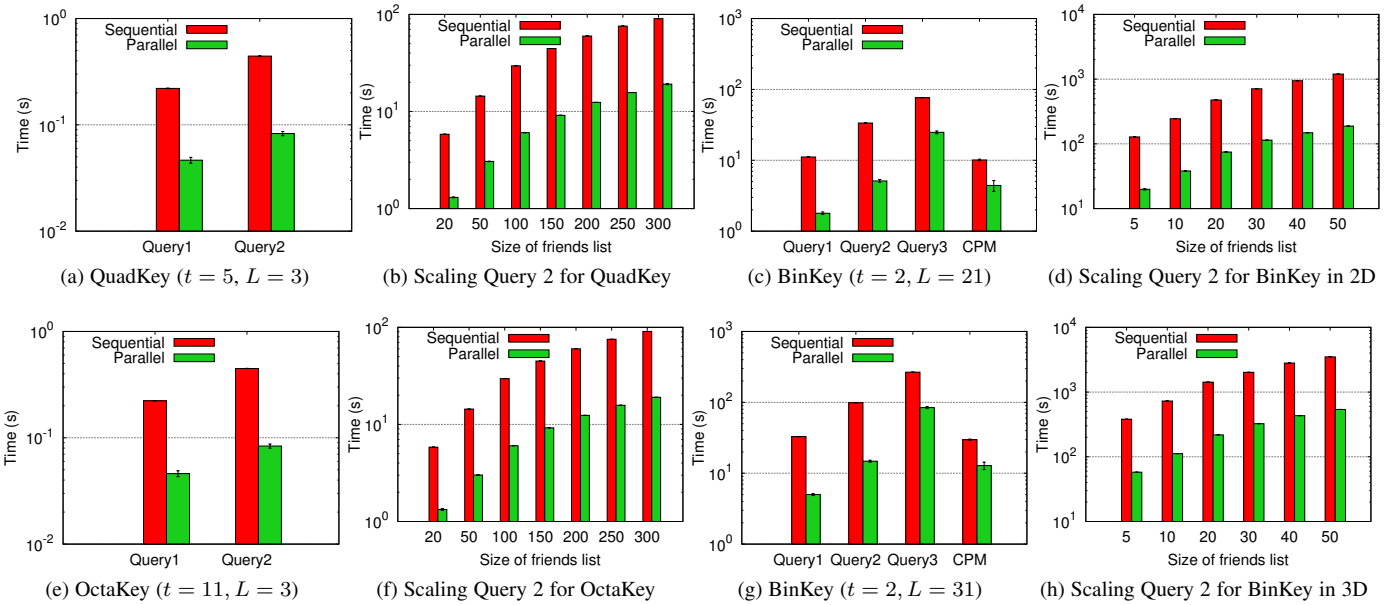
Figs. 9 (a) and (e) show the performance of the system when evaluating the commonly used position and range queries, Query 1 and Query 2. The two queries operate on 22 encrypted digits QuadKeys in Fig. 9 (a) and on 22 encrypted digits OctaKeys in Fig. 9 (e). We set the plaintext modulus of the BGV scheme according to the closest prime to the encoded geolocation, ($t = 5$) for QuadKey in $\mathbb{Z}_4$ and ($t = 11$) for OctaKey in $\mathbb{Z}_8$. The multiplicative depth is set to $L = 3$ for both experiments since two consecutive multiplications are needed when evaluating the second query, and an additional level for HE addition and subtraction. For 2D experiment, the run time of Query 1 is around 0.21 s, but with parallelization, it decreases to 0.04 s. When evaluating Query 2, it takes around 0.43 s which can be improved to 0.08 s with parallelism. Notice that the second query is nearly double the first query in performance due to applying two rounds of masking, one for the privacy bit-mask and the other of the query bit-mask, where each one of them performs homomorphic multiplication. Compared to its counterpart in the 2D space, the performance of 3D is almost the same. This is because both of them share the same scheme parameters except for the message space modular t, and correspondingly for a small difference in the ciphertext space modular q, which did not affect the performance

Figs. 9 (b) and (f) show the scaling of the system when performing Query 2 on QuadKey and OctaKey with different sizes for the user's friend list. We use the same parameter settings from the previous experiments. Likewise, the performance of the nearby query is similar in 2D and 3D as the number of the of ciphertexts is the same in both. Overall, the query run time increases linearly as the size of friends list increases. Our proposed system is scalable as it performs Query 2 under 20 s for 2D and 3D spaces on 300 friends, which is a reasonable number of friends in social networks.

Fig. 9 (c) shows the evaluation of the system if we include the third query, which computes the proximity between two friends. As mentioned in Sec 4.3, the third query requires the encrypted locations to be encoded as BinKey, which doubles the size of ciphertexts to 44 encrypted bits per location data and mask. We set the plaintext modular $t = 2$ because of binary inputs and $L = 21$ based on our derived formula discussed in Sec. 4.3. The performance of the first two queries is affected by the increase of both the number of ciphertexts (44 for BinKey compared to 22 for QuadKey) and their sizes, which increases to account for the larger multiplicative depth. The third query achieves around 24 s when performed in parallel, which is three times faster than when performed in sequential (which takes around 76 s). We further studied the performance and found that most of the computation run time is consumed by performing the common prefix matching CPM. Fig. 9 (g) shows the performance of the three proposed queries on BinKey. Due to increasing the dimension in the 3D experiment, we observe an decrease in performance. But, we compute the XNOR on binary using homomorphic addition instead of multiplication. Overall, the 2D performances are better than the one in 3D by about 3 orders of magnitude.

Figs. 9 (d) and (h) shows the scalabilty when performing the nearby query on BinKey. For each experiment, we use the same corresponding parameters used in Figs. 9 (c) and (g). The run time increases linearly with respect to the size of the friend list. In Fig. 9 (d), Query 2 finishes in 188 s in parallel, which is around 6.5 times faster than sequential. The estimated increase in computing time is around 4 s for every additional friend. In Fig. 9 (h), the parallel evaluation of a list of 50 friends takes about 500 s, i.e., around 10 s per friend.

Fig. 10 shows examples of different views with boxes generated by our demo system for different parties. In all three figures, the outer bounding box is generated by the CPM approach and it is added for comparison. In Fig. 10 (a) and (b), users can see their current coordinate point as well as the cloaked area of the other user. These boxes are generated by the ECPM algorithm. Compared to the outer bounding box, the ECPM approach produces 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 figures. If both users are happy to share their masked location with a



(a) QuadKey ($t = 5, L = 3$) (b) Scaling Query 2 for QuadKey (c) BinKey ($t = 2, L = 21$) (d) Scaling Query 2 for BinKey in 2D

(e) OctaKey ($t = 11, L = 3$) (f) Scaling Query 2 for OctaKey (g) BinKey ($t = 2, L = 31$) (h) Scaling Query 2 for BinKey in 3D

Fig. 9: Computation times for different experiments. (a-d) are for 2D space, whereas (e-h) are for 3D space.

(a) Alice's perspective (b) Bob's perspective (c) Others' perspective

Fig. 10: Perspectives of different parties produced by our demo system using the ECPM algorithm (for the inner box) on Alice's and Bob's locations. We add the outer box, which is computed by the CPM algorithm, for comparison.

third-party application such as Facebook for performing geosocial features, a union of their boxes will be generated and shared.

6 Related Work

One of location privacy techniques is spatial cloaking where a precise location is obfuscated by an area such as a rectangle or a circle. A solution proposed by Hashem and Kulik [16] utilizes nearby peers over a wireless ad-hoc networks to compute the cloaked area for user. No trusted server is needed for this distributed approach; however, it remains dependent on the presence of crowd-sourcing participants in close proximity. A centralized transformation approach that requires a trusted anonymizer is proposed by Peng et al. [31]. Hu et al. [32] proposed to compute the Euclidean distance between two location points after transforming them from WGS84 system to a 2-dimensional Cartesian coordinate system (UTM projection). The UTM projection introduces localization errors that increase as the two points become far apart. The work by Khoshgozaran and Shahabi [33] leverages Hilbert curves, which is similar to Z-order curve, and a one-way trapdoor function to convert a location into a cloaked area which contains a pre-processed set of places-of-interest stored in a look-up table. Our proposed solution also adopts spatial cloaking, but using a non-deterministic encryption guarantees that location data stay private. Other cloaking techniques involves semantic cloaking [34], which abstracts physical locations to semantic ones.

Users exact locations were masked with overlapping hexagonal grid system in Narayanan et al. [25], and MPC protocol was designed to privately inspect the closeness of two. Nielsen et al. [26] proposed a private proximity testing protocol with active security via zero-knowledge proofs [35]. Zhong et al. [36] proposed three protocols for proximity testing. Saldamli et al. [8] integrated geometrical properties to reduce the number of encryption needed [25]. These approaches address whether two users are within the same fixed-sized area in a secure manner.

Khoshgozaran and Shahabi [37] proposed k nearest neighbor and range queries based on a symmetric-key technique where a group key is pre-shared with a set of users. In a similar pre-shared key setting, Puttaswamy et al. [38] privately retrieve location from or near a coordinate instead of user sharing multiple location data with different granularities.

Mascetti et al. [39] proposed a proximity notification system in Geo-social networks. The proximity query is subjected to user's privacy preferences, but is computed as a membership test on all proximate locations. The privacy preferences are applied when user return location with different levels of granularity for different friends. In contrast, our solution only the user's location is returned once but differently masked based on the preference for each friend. Our proximity query is directly computed rather than trying all possible locations.

Finally, Damiani et al. [40] proposed PROBE framework, which obfuscates the user's location based on the geographic context and the privacy preferences defined in the user's profile. The privacy in their solution is controlled by the user's tolerance for defined sensitivity level for each semantic location (e.g., hospitals, banks, residence). This work requires precoded maps to link symbolic representations of locations to coordinates, otherwise spatial queries are limited to reasoning of containment relation.

7 Conclusions

We presented a new location referencing method that geo-hashes GPS coordinates using space-filling curves. The findings of our evaluation suggested that both Hilbert and Z-order curves outperform other curves. We also presented cryptographic algorithms using somewhat homomorphic encryption to support three position and range queries. One of these algorithms reduces masking noise significantly besides addressing the issue of co-location. We further designed optimization techniques that significantly reduce the multiplicative depth and achieve an improvement of an order of magnitude 7 as compared to our prior work. Since both periodically updated location data and privacy preferences are encrypted, the system protects user's location privacy from a semi-honest service provider. We also extended the proposed solution to support location referencing in 3D space. Finally, we developed and evaluated our prototype.

References

[1] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010, ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2009.06.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574119209000510.

[2] A. K. Dey, "Understanding and using context," *Personal Ubiquitous Comput.*, vol. 5, no. 1, pp. 4–7, Jan. 2001, ISSN: 1617-4909. DOI: 10.1007/s007790170019. [Online]. Available: http://dx.doi.org/10.1007/s007790170019.

[3] J. Krumm, "A survey of computational location privacy," *Personal Ubiquitous Comput.*, vol. 13, no. 6, pp. 391–399, Aug. 2009, ISSN: 1617-4909.

[4] S. Consolvo, I. E. Smith, T. Matthews, A. LaMarca, J. Tabert, and P. Powledge, "Location disclosure to social relations: Why, when, & what people want to share," in *CHI*, Portland, Oregon, USA: ACM, 2005, pp. 81–90, ISBN: 1-58113-998-5.

[5] A.-M. Olteanu, K. Huguenin, R. Shokri, and J.-P. Hubaux, "Quantifying the Effect of Co-location Information on Location Privacy," *Privacy Enhancing Technologies*, vol. 8555, no. Chapter 10, pp. 184–203, 2014.

[6] A. Newcomb, *Facebook data harvesting scandal widens to 87 million people*, https://www.nbcnews.com/tech/tech-news/facebook-data-harvesting-scandal-widens-87-million-people-n862771, Apr. 2018.

[7] C. R. Vicente, D. Freni, C. Bettini, and C. S. Jensen, "Location-related privacy in geo-social networks," *IEEE Internet Computing*, vol. 15, no. 3, pp. 20–27, 2011, ISSN: 1089-7801.

[8] G. Saldamli, R. Chow, H. Jin, and B. Knijnenburg, "Private Proximity Testing with an Untrusted Server," in *ACM WiSec*, ACM, Apr. 2013, pp. 113–118.

[9] T. V. A. Pham, I. I. Dacosta Petrocelli, G. F. M. Endignoux, J. R. Troncoso-Pastoriza, K. Huguenin, and J.-P. Hubaux, "Oride: A privacy-preserving yet accountable ride-hailing service," in *Proceedings of the 26th USENIX Security Symposium*, 2017.

[10] U. M. Aïvodji, K. Huguenin, M.-J. Huguet, and M.-O. Killijian, "Sride: A privacy-preserving ridesharing system," in *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ACM, 2018, pp. 40–50.

[11] J. Freudiger, M. Raya, M. Félegyházi, P. Papadimitratos, and J.-P. Hubaux, "Mix-Zones for Location Privacy in Vehicular Networks," in *Proceeding of Win-ITS'07*, Vancouver, British Columbia, Aug. 2007.

[12] C. Dwork, "Differential privacy," in *ICALP*, ser. LNCS, vol. 4052, Venice, Italy: Springer Verlag, Jul. 2006, pp. 1–12, ISBN: 3-540-35907-9.

[13] B. Gedik, K.-L. Wu, P. S. Yu, and L. Liu, "Processing Moving Queries over Moving Objects using Motion-adaptive Indexes," *IEEE TKDE*, vol. 18, no. 5, pp. 651–668, 2006.

[14] F. Olumofin, P. K. Tysowski, I. Goldberg, and U. Hengartner, "Achieving Efficient Query Privacy for Location Based Services," in *PETS*, Springer-Verlag, Jul. 2010, pp. 93–110.

[15] K. P. N. Puttaswamy, S. Wang, T. Steinbauer, D. Agrawal, A. E. Abbadi, C. Kruegel, and B. Y. Zhao, "Preserving location privacy in geosocial applications," *IEEE Transactions on Mobile Computing*, vol. 13, no. 1, pp. 159–173, Jan. 2014, ISSN: 1536-1233. DOI: 10.1109/TMC.2012.247.

[16] T. Hashem and L. Kulik, ""Don't trust anyone": Privacy Protection for Location-Based Services," *Pervasive & Mobile Computing*, vol. 7, no. 1, pp. 44–59, 2011, ISSN: 1574-1192.

[17] V. Gaede and O. Günther, "Multidimensional access methods," *ACM Comput. Surv.*, vol. 30, no. 2, pp. 170–231, Jun. 1998, ISSN: 0360-0300.

[18] P. Svensk and L. Wikström, *Georeferencing methods - a study on how to deal with georeferencing in the rosatte implementation platform*, Triona AB, Mar. 2013.

[19] P. Hu, S. S. Chow, and A. Aloufi, "Geosocial query with user-controlled privacy," in *Proceedings of WiSec*, ACM, 2017, pp. 163–172.

[20] J. Schwartz, *Bing maps tile system*, https://msdn.microsoft.com/en-us/library/bb259689.aspx, 2012.

[21] C. Bettini and D. Riboni, "Privacy Protection in Pervasive Systems: State of the Art and Technical Challenges," *Pervasive and Mobile Computing*, vol. 17, Part B, pp. 159–174, 2015, ISSN: 1574-1192.

[22] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS*, ACM, 2012, pp. 309–325.

[23] O. Regev, "The learning with errors problem (invited survey)," in *Proceeding of CCC'10*, Cambridge, MA, Jun. 2010, pp. 191–204.

[24] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *EUROCRYPT*, Springer, 2010, pp. 1–23.

[25] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh, "Location Privacy via Private Proximity Testing," in *NDSS*, Feb. 2011.

[26] J. D. Nielsen, J. Illeborg, and M. B. Stausholm, "Location Privacy via Actively Secure Private Proximity Testing," in *PerCom Workshop*, Lugano, Switzerland, Mar. 2012, pp. 381–386.

[27] S. Halevi and V. Shoup, "Algorithms in HElib," in *CRYPTO*, Springer, 2014, pp. 554–571.

[28] ——, *Faster homomorphic linear transformations in helib*, Cryptology ePrint Archive, Report 2018/244, 2018. [Online]. Available: https://eprint.iacr.org/2018/244.

[29] R. Ricci, E. Eide, and the CloudLab Team, "Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications," *;login:*, vol. 39, no. 6, pp. 36–38, Dec. 2014. [Online]. Available: https://www.usenix.org/publications/login/dec14/ricci.

[30] M. Backes, P. Berrang, M. Bieg, R. Eils, C. Herrmann, M. Humbert, and I. Lehmann, "Identifying personal DNA methylation profiles by genotype inference," in *IEEE S&P*, 2017, pp. 957–976.

[31] T. Peng, Q. Liu, and G. Wang, "Privacy Preserving for Location-Based Services Using Location Transformation," *Cyberspace Safety and Security, Vol. 8003 of LNCS*, no. Chap. 2, pp. 14–28, 2013.

[32] P. Hu, T. Mukherjee, A. Valliappan, and S. Radziszowski, "Homomorphic proximity computation in geosocial networks," in *BigSecurity, an INFOCOM workshop*, Apr. 2016.

[33] A. Khoshgozaran and C. Shahabi, "Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy," in *SSTD*, Springer-Verlag, 2007, pp. 239–257, ISBN: 978-3-540-73539-7.

[34] O. Barak, G. Cohen, and E. Toch, "Anonymizing mobility data using semantic cloaking," *Pervasive and Mobile Computing, Special Issue on Security and Privacy in Mobile Clouds*, vol. 28, pp. 102–112, 2016.

[35] U. Feige, A. Fiat, and A. Shamir, "Zero-Knowledge Proofs of Identity," *J. Cryptology*, vol. 1, no. 2, pp. 77–94, Jun. 1988.

[36] G. Zhong, I. Goldberg, and U. Hengartner, "Louis, Lester and Pierre - Three Protocols for Location Privacy," in *Privacy Enhancing Technologies*, 2007, pp. 62–76.

[37] A. Khoshgozaran and C. Shahabi, "Private buddy search: Enabling private spatial queries in social networks," in *Computational Sci and Engg.*, IEEE Comp. Society, 2009, pp. 166–173, ISBN: 978-0-7695-3823-5.

[38] K. P. N. Puttaswamy, S. Wang, T. Steinbauer, D. Agrawal, A. E. Abbadi, C. Kruegel, and B. Y. Zhao: "Preserving Location Privacy in Geosocial Applications," *IEEE Trans. Mob. Comput.*, pp. 159–173, 2014.

[39] S. Mascetti, D. Freni, C. Bettini, X. S. Wang, and S. Jajodia, "Privacy in geo-social networks: Proximity notification with untrusted service providers and curious buddies," *The VLDB Journal*, vol. 20, no. 4, pp. 541–566, 2011, ISSN: 1066-8888.

[40] M. L. Damiani, E. Bertino, and C. Silvestri, "Protecting location privacy against spatial inferences: The probe approach," in *Workshop on Security and Privacy in GIS and LBS*, 2009, pp. 32–41.

Appendix A
Threshold SWHE

For computation over data encrypted under multiple keys, we need the threshold extension [1] of the BGV scheme to produce a joint key. In a nutshell, we can leverage the additive homomorphism property to generate a joint key from keys owned by individual participants. Formally, given a set of public keys $pk_i = (A, As_i + te_i)$ where $i \in \mathbb{N}$ and the element $A \in R_q$ is shared, we can compute a joint public key $pk^* = (A, A\sum s_i + t\sum e_i)$. The underlying secret keys s_i are added together under the protection of the RLWE assumption. This threshold extension allows us to generate a joint key in advance for all homomorphic operations. But, a key setup phase is not always feasible.

Hence, we propose an alternative way to efficiently transform a ciphertext encrypted under different keys to an encryption under a joint key. Given a ciphertext $C_i = (c_0, c_1) = (\gamma A, \gamma(As_i + te_i) + m)$ encrypted under s_i for a message m, we can homomorphically add C_i with another ciphertext C_j that is the encryption of zero using s_j, assuming both γ and A are shared. For example, $C_i + C_j = (\gamma A, \gamma(A(s_i + s_j) + t(e_i + e_j)) + m)$. This produces a ciphertext C^* that is encrypted under the joint key pk^*.

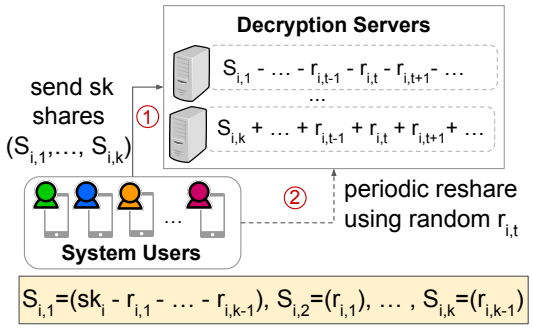


Fig. 11: Securing private keys with secret-sharing

Appendix B
Threshold Decryption Protocol

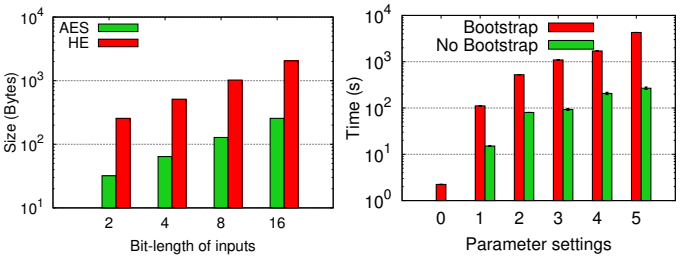
After processing a requested query, the service provider sends to the user the result that is encrypted under a joint public key pk^*, hence the user is not able to decrypt the result without their friends' secret keys. We need to transform the result to a ciphertext that is decryptable by the querier's secret key. This process reverses the additively homomorphism property by generating ciphertext C_j that removes s_j from s^*.

The decryption protocol hence requires either friends of the user to be online to generate C_j or delegate the decryption task to a decryption server who should have access to the secret keys s_j. In this paper, we propose the latter approach and improve the security of user's private keys by secret-sharing these keys among multiple *semi-honest* and *non-colluding* decryption servers [2], as illustrated in Fig. 11. We propose a *dishonest majority* model so that the system tolerates up to $N - 1$ servers being hacked. We also introduce a reshare protocol to change the representation of the private keys periodically. Then, we construct a multiparty computation protocol, similar to these [3], [4], to perform threshold-based decryption.

Key revocation is done simply by generating a new key pair, transforming the ciphertexts to be under the new public key, and secret-sharing the new secret key.

Appendix C
Hybrid Homomorphic Encryption

Despite the recent advancement, somewhat homomorphic encryption schemes in general produces ciphertexts that are large in size. Transmitting these homomorphically encrypted ciphertexts over the network introduces significant communication overhead, we



(a) Ciphertext size of AES and HE (b) Homomorphic AES decryption

Fig. 12: Performance for hybrid encryption

propose a hybrid encryption [5], [6], which couples an efficient block-cipher, such as AES, with SWHE. In a nutshell, this approach uses block-cipher to encrypt data on the client side and converts ciphertexts to the SWHE domain. More specifically, given an AES ciphertext $\texttt{Enc}_{\texttt{AES}}(m)$ of a message m and $\texttt{Enc}_{\texttt{BGV}}(k)$ of an AES key k, we can compute $\texttt{Enc}_{\texttt{BGV}}(m) = \texttt{homAESdec}(\texttt{Enc}_{\texttt{BGV}}(\texttt{Enc}_{\texttt{AES}}(\texttt{m})), \texttt{Enc}_{\texttt{BGV}}(\texttt{k}))$. This can be easily achieved by the `homAESdec` function supported in HElib [7].

To demonstrate the effectiveness of this approach, we conducted experiments to investigate the reduction in ciphertext size and computation time under different scenarios. Experiment results are presented in Fig. 12. Note, in Fig. 12(b), parameter 0 is for toy examples and parameter 5 is for huge realistic examples with higher security levels.

References

[1] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold FHE," in *EUROCRYPT*, Springer, 2012, pp. 483–501.

[2] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[3] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," *European transactions on Telecommunications*, vol. 8, no. 5, pp. 481–490, 1997.

[4] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," in *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 1998, pp. 1–16.

[5] T. Lepoint and M. Naehrig, "A comparison of the homomorphic encryption schemes FV and YASHE," in *AfricaCrypt*, Springer, 2014, pp. 318–335.

[6] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" In *ACM workshop on Cloud Computing Security Workshop*, 2011, pp. 113–124.

[7] S. Halevi and V. Shoup, "Algorithms in HElib," in *CRYPTO*, Springer, 2014, pp. 554–571.