Hindawi
Security and Communication Networks
Volume 2021, Article ID 5686506, 17 pages
https://doi.org/10.1155/2021/5686506

Research Article

Secure and Practical Group Nearest Neighbor Query for Location-Based Services in Cloud Computing

Jingjing Guo [1] **and Jiacong Sun** [2]

[1] *Jingjing Guo was with the State Key Laboratory of Integrated Service Networks (ISN), Xidian University, Shaanxi, China*
[2] *Jiacong Sun was with School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School, Shenzhen, China*

Correspondence should be addressed to Jiacong Sun; yuwangxpu@gmail.com

Received 12 April 2021; Accepted 20 August 2021; Published 25 September 2021

Academic Editor: AnMin Fu

Group nearest neighbor (GNN) query enables a group of location-based service (LBS) users to retrieve a point from point of interests (POIs) with the minimum aggregate distance to them. For resource constraints and privacy concerns, LBS provider outsources the encrypted POIs to a powerful cloud server. The encryption-and-outsourcing mechanism brings a challenge for the data utilization. However, as previous work from k–anonymity technique leaks all contents of POIs and returns an answer set with redundant communication cost, the LBS system cannot work properly with those privacy-preserving schemes. In this paper, we illustrate a secure group nearest neighbor query scheme, which is referred to as SecGNN. It supports the GNN query with $n\,(n \geq 3)$ LBS users and assures the data privacy and query privacy. Since SecGNN only achieves linear search complexity, an efficiency enhanced scheme (named SecGNN$^+$) is introduced by taking advantage of the KD-tree data structure. Specifically, we convert the GNN problem to the nearest neighbor problem for their centroid, which can be computed by anonymous veto network and Burmester–Desmedt conference key agreement protocols. Furthermore, the SecGNN$^+$ scheme is introduced from the KD-tree data structure and a designed tool, which supports the computation of inner products over ciphertexts. Finally, we run experiments on a real-database and a random database to evaluate the performance of our SecGNN and SecGNN$^+$ schemes. The experimental results show the high efficiency of our proposed schemes.

1. Introduction

With the prevalence of mobile phones and the rapid development of wireless, location-based service (LBS) provides the possibility for LBS users to proceed location queries according to their interests [1–3]. For instance, when locating in an unfamiliar place, LBS users can find the nearest restaurant through sending LBS queries to the Google map. A new employee also can find the shortest path to the company by sending a LBS query. Group nearest neighbor (GNN) query, a normal operation in the LBS system, requires LBS users to retrieve a meeting place from POIs with the minimum sum of distances to them.

Under the conditions of limited resources and frequent queries, the LBS provider always outsources heavy database of POIs to a powerful cloud server [4]. The LBS provider thus can enjoy the rich storage and computation resources, but database outsourcing raises privacy problems for the private sensitive information. In such a case, the LBS provider chooses the encryption-before-outsourcing mechanism, which avoids the potential leakage of private sensitive information. However, database encryption brings a challenging task for data utilization in the LBS system.

Hashem et al. [5] proposed the first privacy-preserving group nearest neighbor query scheme from k–anonymity technique. Users compute and send location regions instead of actual locations to the cloud server to assure the privacy of actual locations. Then, the cloud server searches the POIs over plaintexts and returns an answer set that contains the group nearest neighbor (namely, the meeting place) for

actual locations. To evaluate the real GNN from the answer set, the authors proposed a private filter technique to avoid the leakage of actual locations and the meeting place. With spatial technique, users in scheme [6] obfuscate actual locations in cloak regions and construct a single centroid region for the nearest neighbor query. In the second phase, users compute the meeting place from the received answer set by using a special secure multiparty computation. In above schemes, the cloud server returns a superset instead of the real answer, which increases the communication cost.

As an improvement, Wu et al. [7, 8] introduced another scheme by hybridizing actual locations with other $(k-1)$ dummy locations. Users put their actual locations in a specific position, which is shared among them, and send location sets to the cloud server. The cloud server searches and generates candidate results for locations on the same index in their location sets. Furthermore, it retrieves and returns the real GNN on ciphertext by leveraging generalized pallier encryption [9]. Finally, users decrypt the ciphertext and obtain the meeting place. Unfortunately, all solutions [5–8] adapt k–anonymity technique to assure the privacy of sensitive locations. It is noteworthy that k–anonymity only assures that an attacker can identify the actual location with an advantage at most $1/k$, but the content may be leaked. For instance, all candidate results in scheme [8] remain visible and an attacker can identify actual locations with advantage 1/32. The privacy-preserving method cannot satisfy the security requirements in practice.

Huang and Vishwanathan [13] proposed two GNN schemes, the centralized and distributed model, from cryptographic theory. The authors leverage garble circuit (GC) and oblivious transfer (OT) to construct a secure multiparty computation protocol. Then, users can jointly compute a meeting place from the secure multiparty computation framework. But the scheme [13] suffers a heavy computation and communication burden. Therefore, we aim to seek for a privacy-preserving GNN scheme based on cryptographic tools with whole security protection and high efficiency.

1.1. Our Contributions. In this paper, we design two privacy assurance group nearest neighbor query schemes, SecGNN scheme and SecGNN$^+$ scheme, for the LBS system in cloud computing. The designed SecGNN scheme preserves the data privacy and query privacy, while achieving linear complexity. To enhance query efficiency, we introduce the SecGNN$^+$ scheme based on the tree structure. The main contributions are demonstrated as follows.

(1) We convert the problem of finding group nearest neighbor to the problem of finding a point in POIs with minimum aggregate inner products. Thus, the construction of the secure GNN scheme is modeled as designing an inner product-preserving scheme with privacy preserving. Based on the asymmetric scalar-product-preserving encryption (ASPE) technique [15], we design a basic tool to compute a special inner product and then introduce a secure group nearest neighbor query ($n \geq 3$) scheme, which is referred to as the SecGNN scheme.

(2) Furthermore, we prove that finding group nearest neighbor for LBS users is equivalent to finding the nearest neighbor for their centroid, which can be secretly computed by using anonymous veto network protocol and Burmester–Desmedt conference key agreement protocol. Thus, the SecGNN$^+$ scheme is designed from the construction of nearest neighbor for the centroid and further improves the search efficiency of the SecGNN scheme. The proposed SecGNN$^+$ scheme achieves $O(n^{1-1/k} + m)$ search complexity, where n is the number of data items in database, k is the dimension of data items, and m is the size of answers.

(3) Moreover, we demonstrate the correctness analysis and security proof of both schemes. The security proof illustrates that our SecGNN and SecGNN$^+$ schemes achieve data privacy and query privacy.

(4) Finally, we evaluate the performance of SecGNN and SecGNN$^+$ schemes through experimental simulation on a real-database with 62,556 data items and a random database with 1,000,000 data items. The experimental results show that the SecGNN$^+$ scheme achieves practical search efficiency, about 0.2 s on millions' database.

1.2. Related Work. Group nearest neighbor query enables a group of LBS users to retrieve a point with the minimum aggregate distance. The authors [16] demonstrated multiple query method (MQM), single-point method (SPM), and minimum bounding method (MBM) for processing GNN queries. Moving a step forward, Papadias et al. [17] generalized the GNN query to minimize the maximum or minimum distance. Because of heavy storage and computation burden, the resource-constrained LBS provider always outsources the database to the cloud server possessing the powerful storage and computation resources. Since the cloud server is dishonest and may steal sensitive information, privacy-preserving methods always be chosen to assure data security. As shown in Table 1, numerous studies [5, 6, 13, 18, 19] have been done to protect the privacy of private sensitive location information from the dishonest cloud server and outside attackers.

Hashem et al. [5] introduced the first privacy-preserving GNN scheme by leveraging k-anonymity technique. They described the privacy assurance solution from three steps, sending the query request, searching the GNN, and finding the actual meeting place. That is, users $u_1, u_2, \ldots, u_n$ first send cloaked rectangles $R_1, R_2, \ldots, R_n$ (including locations $l_1, l_2, \ldots, l_n$) to the cloud server, respectively. After receiving the query rectangles $R_1, R_2, \ldots, R_n$, the cloud server proceeds the group nearest neighbor query for query rectangles over plaintexts and returns an answer set $A = \{p_1, p_2, \ldots, p_s\}$ that contains the GNN for users $u_1, u_2, \ldots, u_n$ with respect to the actual locations $l_1, l_2, \ldots, l_n$. The cloud server also returns the sum of maximum distance

Table 1: Comparison with existing works.

Group size	Schemes	Techniques	Leakage database	Result redundancy
$n = 1$	Bamba et al.'08 [10]	Cloak region	Yes	Yes
	Wang et al.'16 [11]	OPE + R-tree	No	No
	Guo et al.'20 [12]	Bloom filter +prefix encoding	No	No
$n > 1$	Hashem et al.'10 [5]	k-anonymity	Yes	Yes
	Huang et al.'10 [13]	MPC	No	Yes
	Ashouri et al.'15 [14]	Cloak region	Yes	Yes
	Wu et al.'21 [8]	Dummy + Paillier	Yes	No
	Our solution	ASPE + kd-tree	No	No

to query rectangles (namely, $d_{\max}(p) = \sum_{i=1}^{n} \text{Max Dist}(R_i, p)$) for each point p in the answer set A. Finally, LBS user u_j updates u_j by using the distance to his actual location l_j instead of $\text{Max Dist}(R_j, p)$. Specifically, LBS user u_j computes $d_{\max}(p) = \sum_{i=1}^{n} \text{Max Dist}(R_i, p) - \text{Max Dist}(R_j, p) + \text{Dist}(l_j, p)$. This process continues until all LBS users $u_1, u_2, \ldots, u_n$ have updated. The point p is the desired GNN if it has the minimum $d_{\max}$ in the answer set. However, the scheme [5] is expensive in communication cost and fails to assure the query privacy.

Subsequently, Ashouri-Talouki et al. [18] demonstrated an efficiency-enhanced group nearest neighbor query scheme. Users compute the minimum bounding rectangle (MBR) that contains all of them or the centroid from the anonymous veto network (AV-net) and Paillier encryption scheme [20]. Then, the cloud server searches the nearest neighbor (NN) over plaintexts for the MBR or centroid. Furthermore, it is worth that the authors take the NN for the centroid as an approximate GNN, and the scheme [18] fails to protect the location privacy of the meeting place. The authors design another privacy-preserving group nearest neighbor query scheme from cloaked-centroid protocols [6]. LBS users hide their precise locations in cloaked regions by generating rectangles that contain the exact locations, respectively. The cloaked region can be adjusted with the time and privacy requirements. LBS users publish their cloaked regions to the public bulletin board. After publishing, LBS users can compute the cloaked-centroid region and submit to the cloud server for a NN query. In the next phase, the cloud server searches the POIs and returns an answer set to them. LBS users compute the approximate meeting place by using AV-net protocol. But the cloud server returns a superset replacing the real answer, which increases the communication cost.

Wu et al. [7, 8] presented a privacy-preserving GNN scheme based on the k-anonymity method. LBS users use $(k-1)$ dummy locations to blur their actual locations and store the exact location in a specific position which is shared among them. A chosen LBS user proceeds the generalized Paillier encryption to generate the ciphertext for the indicator vector. The cloud server searches the group nearest neighbor over plaintexts and computes the encrypted answer by a private selection. Finally, LBS users decrypt the ciphertext and obtain the real meeting place. Unfortunately, all these solutions only protect the query privacy with an advantage at most $1/k$, where k is the size of dummy locations. Furthermore, it fails to protect the privacy of POIs since the cloud server searches the GNN or NN over plaintexts.

Some privacy-preserving GNN schemes [13, 21] based on cryptographic have been introduced. Huang and Vishwanathan [13] proposed a secure GNN scheme from secure garble circuit and oblivious transfer. In solution [13], LBS user Charlie constructs a garble circuit for the GNN and user David executes the computation task. Other users obtain ciphertexts from Charlie through oblivious transfer and send ciphertexts to David. Finally, David obtains the GNN by computing the garble circuit. LBS users in [21] jointly compute the aggregate distance for all POIs by using anonymous veto network and Burmester–Desmedt key establishment protocols. Thus, users can obtain the precise meeting place. However, both schemes [13, 21] bring high computation and communication costs for LBS users and are impractical in the real world.

1.3. Organization. The rest of this work is organized as follows. Some preliminaries and their definitions will be demonstrated in Section 2. In Section 3, we present the system model and privacy requirements of the proposed schemes. The introduced SecGNN scheme and its security analysis are illustrated in Section 4. We further describe the efficiency-enhanced SecGNN$^+$ scheme and its security analysis in Section 5. Finally, the performance evaluation and conclusion will be discussed in Sections 6 and 7.

2. Preliminaries

In this section, we introduce some preliminaries of KD-tree, anonymous veto network protocol, and Burmester–Desmedt conference key agreement protocol.

2.1. KD-Tree. KD-tree is a hierarchical data structure which is always used to improve the efficiency of the range query and nearest neighbor query. It has a $O(n)$ storage cost, $O(n \log n)$ construction time cost, and $O(n^{1-1/k} + m)$ query time cost, where n is the size of the database, k is the dimension of data record in the database, and m is the size of the answer set.

KD-tree is a binary tree structure in which every nonleaf node has two children. The children are split based on the x–coordinate or y–coordinate of their father node. From

root to leaf node, the splitting is done on x−coordinate, on y−coordinate for the children, and next on x−coordinate for the grandchildren, and so on. In the following, we describe the solution for KD-tree to solve the problem of the nearest neighbor query:

(1) The search process is done based on the splitting line (e.g., x−coordinate or y−coordinate of the farther node) and can quickly arrive at the leaf node. We return the leaf node value as a temporary nearest neighbor and mark down the road from root to this leaf node.

(2) We check whether the distance of its father node to query node is closer than the temporary nearest neighbor. If it is, we update the father node as a temporary nearest neighbor. Furthermore, we do the checking whether the distance from the query point to the splitting line is smaller than that to the temporary nearest neighbor. If it holds, we search the other child. The checking algorithm continues until backing at the root.

For instance, we illustrate an example of KD-tree in Figure 1. Given a query point $(2.1, 3.1)$, the search process is done to obtain the leaf node value $(3, 5)$ as the temporary nearest neighbor. When backing to the father node $(2, 3)$, it is easily seen that the father node is closer than the temporary nearest neighbor $(3, 5)$. We update father node $(2, 3)$ as the temporary nearest neighbor. Furthermore, we check whether the circle with center $(2.1, 3.1)$ and radius 0.14 (the distance from query point to temporary nearest neighbor $(2, 3)$) intersects $y = 3$ (the splitting line). Since it intersects with $y = 3$, we check whether point $(1, 1)$ is closer than temporary nearest neighbor $(2, 3)$. If it fails, we are back to check the father node $(4, 7)$ and obtain $(2, 3)$ as the nearest neighbor for query point $(2.1, 3.1)$.

2.2. Anonymous Veto Network Protocol. In anonymous veto network protocol and the following Burmester–Desmedt protocol, it is assumed that G is a finite cyclic group of prime order q in which the decisional Diffie–Hellman (DDH) problem is intractable. The generator in G is g, and all computations take place in G. There are n members in the group as $u_1, u_2, \ldots, u_n$, and they agree on (G, g).

Anonymous veto network protocol (AV-net) was introduced by Feng et al. [22, 23] to solve the problem of anonymous veto. The AV-net protocol contains two phases. In the first phase, users $u_1, u_2, \ldots, u_n$ choose random numbers $a_1, a_2, \ldots, a_n$ and publish the random ephemeral public keys $g^{a_1}, g^{a_2}, \ldots, g^{a_n}$, respectively. Then, each user u_i computes g^{b_i} by multiplying all the random ephemeral public keys before i and dividing all the random ephemeral public keys after i:

$$g^{b_i} = \frac{\prod_{j=1}^{i-1} g^{a_j}}{\prod_{j=i+1}^{n} g^{a_j}.} \quad (1)$$

In the next phase, users $u_1, u_2, \ldots, u_n$ compute and broadcast $g^{b_1 c_1}, g^{b_2 c_2}, \ldots, g^{b_n c_n}$, where $c_i = a_i$ if user u_i does not veto; otherwise, c_i is a random number $(i = 1, 2, \ldots, n)$. In such situation, if no one vetoes, we have $\Pi_{i=1}^{n} g^{b_i c_i} = \Pi_{i=1}^{n} g^{a_i b_i} = 1$ because of the vanishing property of AV-net exponents, $\sum_{i=1}^{n} a_i b_i = 0$. Otherwise, $\prod_{i=1}^{n} g^{b_i c_i} \neq 1$. Meanwhile, vetoing users remain anonymous.

2.3. Burmester–Desmedt Conference Key Agreement Protocol. Burmester–Desmedt (BD) protocol was developed by Burmester and Desmedt to establish a conference key [24, 25]. That is, users $u_1, u_2, \ldots, u_n$ aim to establish a conference key. The indices are taken in a cycle, namely, $u_{n+1} = u_1$ and $u_{n+2} = u_2$. As shown in Figure 2, BD-protocol includes two processes. In the first process, users $u_1, u_1, \ldots, u_n$ choose random numbers $e_1, e_2, \ldots, e_n$ and broadcast $g^{e_1}, g^{e_2}, \ldots, g^{e_n}$. In the second process, each user u_i publishes $r_i = (g^{e_{i+1}}/g^{e_{i-1}})^{e_i}$ and later computes the conference key by the following equation:

$$k_i = \left(g^{e_{i-1}}\right)^{ne_i} \cdot r_i^{n-1} \cdot r_{i+1}^{n-2} \cdots r_{i-2} \bmod q, i = 1, 2, \ldots, n. \quad (2)$$

Thus, users $u_1, u_1, \ldots, u_n$ have the same conference key:

$$k_1 = k_2 = \cdots = k_n = k = g^{e_1 e_2 + e_2 e_3 + \cdots + e_{n-1} e_n + e_n e_1} \bmod q. \quad (3)$$

Considering the intractability of the Diffie–Hellman problem in G, the conference k is only computable by group users and adversaries can find no information about it.

3. Problem Formulation

In this section, we first illustrate the system model and then describe the privacy requirements of the group nearest neighbor query schemes.

3.1. System Model. In this paper, we aim to design a secure and efficient group nearest neighbor query scheme. As shown in Figure 3, the system model of our scheme consists of three entities, cloud server, LBS provider, and LBS users. The cloud server has powerful storage and computation resources. It provides storage and computation services for LBS provider and further searches the encrypted database after receiving search tokens from LBS users. Since constrained resources and privacy concerns, the LBS provider (e.g., a company) owns the database and chooses the encryption-and-outsourcing mechanism to enjoy the powerful resources provided by the cloud server. According to their requirements, LBS users (e.g., employees) generate and submit search tokens to the cloud server. Subsequently, they obtain the encrypted results and decrypt them to have the final results. In this paper, we suppose that LBS users always submit a query request for a meeting place. That is, LBS users want to find a meeting place that has the minimum aggregate distance.

Furthermore, we assume that the cloud server is honest-but-curious. That means, the cloud server follows the protocol faithfully but intends to steal the information about the private sensitive database. We also assume that the LBS provider is honest and LBS users want to steal other actual locations but do not collude with others. The assumption is common in other papers [26, 27].

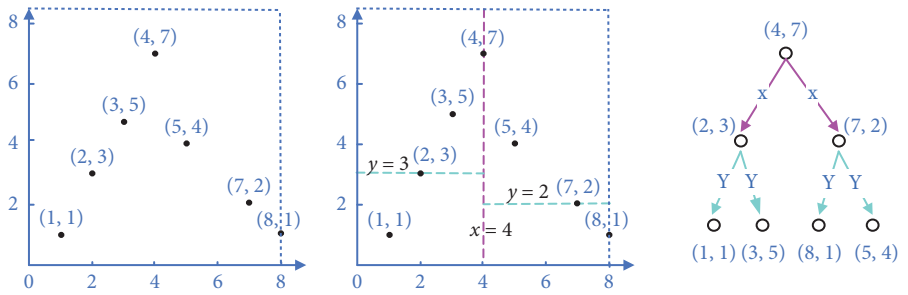

Figure 1: An example of KD-tree for the nearest neighbor query.

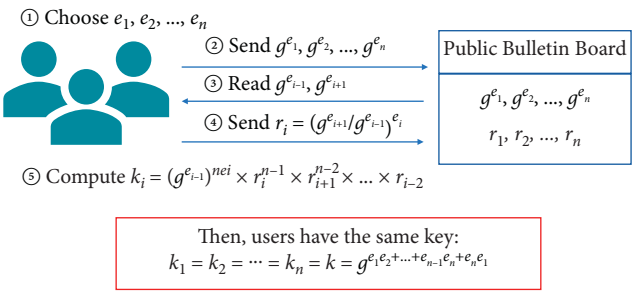

Figure 2: An instance of Burmester–Desmedt conference key agreement protocol.

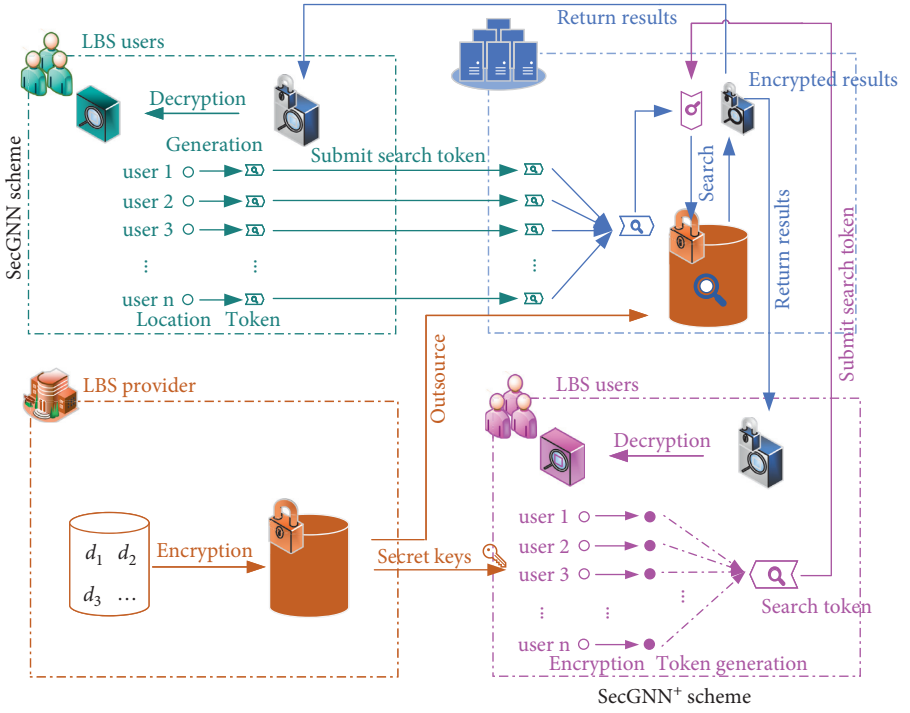

Figure 3: The system model of the group nearest neighbor query scheme.

Based on the assumption, we consider two threat models, known ciphertext model and known background model. In the known ciphertext model, only ciphertexts and encrypted index are known to the cloud server. In the known background model, the cloud server holds more information, such as the distribution of data records and queries. Both models are considered in some existing works [28, 29].

3.2. Privacy Requirements. Under above assumptions, the proposed scheme should protect the confidentiality of outsourced database and query data. The privacy requirements are illustrated in the following.

(1) Data privacy: the LBS provider owns the original database and builds an index to improve the search efficiency. Both data and index contains private sensitive information. Thus, a secure group nearest neighbor query scheme should ensure that outside attackers (i.e., cloud server) cannot infer the exact information from the encrypted database and index.

(2) Query privacy: the authorized LBS users submit GNN queries to the cloud server, and the latter executes search operations. However, the cloud server cannot infer specific contents of query requests from the search token and encrypted results.

3.3. Notations. For the convenience of describing SecGNN and SecGNN$^+$ schemes, we illustrate some notations in Table 2.

4. Secure Group Nearest Neighbor Query

In this section, we illustrate a secure group nearest neighbor query scheme called SecGNN in the LBS system. SecGNN enables LBS users to retrieve a meeting place from POIs that have the minimum aggregate distance. For a point $p = (s, t)$ in POIs, the sum of distances to LBS users is computed as follows:

$$\begin{aligned} d^2(p) &= (x_1 - s)^2 + (y_1 - t)^2 + (x_2 - s)^2 + (y_2 - t)^2 + \cdots + (x_n - s)^2 + (y_n - t)^2, \\ &= x_1^2 - 2sx_1 + s^2 + y_1^2 - 2ty_1 + t^2 + x_2^2 - 2sx_2 + s^2 + y_2^2 - 2ty_2 + t^2 + \cdots + x_n^2 - 2sx_n + s^2 + y_n^2 - 2ty_n + t^2, \\ &= x_1^2 + y_1^2 + x_2^2 + y_2^2 + \cdots + x_n^2 + y_n^2 + \langle\langle (s^2 + t^2, -2s, -2t), (1, x_1, y_1)\rangle \\ &\quad + \langle\langle (s^2 + t^2, -2s, -2t), (1, x_2, y_2)\rangle + \cdots + \langle\langle (s^2 + t^2, -2s, -2t), (1, x_n, y_n)\rangle, \end{aligned} \tag{4}$$

where (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) are the coordinates of LBS users $u_1, u_2, \ldots, u_n$. Assume that $\overline{p} = (s^2 + t^2, -2s, -2t)$, $\overline{tk}_1 = (1, x_1, y_1)$, $\overline{tk}_2 = (1, x_2, y_2)$, ..., and $\overline{tk}_n = (1, x_n, y_n)$. Since $x_1^2 + y_1^2 + x_2^2 + y_2^2 + \cdots + x_n^2 + y_n^2$ is a fixed number, $d(p)$ achieves the minimum value if the sum of inner products $\langle \overline{p}, \overline{tk}_1 \rangle$, $\langle \overline{p}, \overline{tk}_2 \rangle$, ..., and $\langle \overline{p}, \overline{tk}_n \rangle$ achieves the minimum value.

In the following, we adapt a privacy-preserving method to evaluate a point p that has the minimum aggregate inner products. Specifically, we design a basic tool, named CompInner, supporting the computation of inner products over ciphertexts and then introduce the BD-conference key agreement to generate a common random. Finally, we construct the SecGNN scheme from above protocols.

CompInner tool consists of five polynomial-time algorithms, Setup, Encrypt, GenToken, GenInner, and Decrypt. Firstly, the user runs the system setup algorithm (Setup) to generate the secret key and then encrypts data items in the encryption algorithm (Encrypt). Next, the token generation algorithm (GenToken) is run by the user to output a computation token for the query point, and inner products can be computed in the inner generation algorithm (GenInner). Finally, the user runs the decryption algorithm (Decrypt) to decrypt the encrypted results.

(1) Setup (λ): the user runs the system setup algorithm to generate two $d' \times d'$ invertible matrices M_1 and M_2, a d'-bit string S, and random values $w_1, w_2, \ldots, w_d (d = d' - 3)$. Thus, the secret key is

$$sk = (M_1, M_2, S, w_1, w_2, \ldots, w_d). \tag{5}$$

(2) Encrypt (sk, p): suppose that $p = (s, t)$, and the data encryption algorithm is run to do the following steps:

(1) Adding random numbers: for a point $p = (s, t)$, the user first computes $\overline{p} = (s^2 + t^2, -2s, -2t)$ and then extends it as $\widehat{p} = (s^2 + t^2, -2s, -2t, w_1, w_2, \ldots, w_d)$.

(2) Random splitting: split $\widehat{p}$ into two parts $\widehat{p}_a$ and $\widehat{p}_b$. For $i = 1, 2, \ldots, d'$,
If $S_i = 0$, $\widehat{p}_a[i] = \widehat{p}_b[i] = \widehat{p}[i]$.
If $S_i = 1$, $\widehat{p}_a[i]$ and $\widehat{p}_b[i]$ are random numbers such that $\widehat{p}_a[i] + \widehat{p}_b[i] = \widehat{p}[i]$.

(3) Data encryption: Compute the ciphertexts as $(p'_a = M_1^T \times \widehat{p}_a^T, p'_b = M_1^T \times \widehat{p}_b^T)$.

(3) GenToken(sk, l): suppose that $l = (x, y)$, and the token is generated as follows:

(1) Adding random numbers: for a query point $l = (x, y)$, the user first computes $\overline{l} = (r, rx, ry)$ and then extends it as $\widehat{l} = (r, rx, ry, r_1, r_2, \ldots, r_d)$, where $r > 0$ and r_1, r_2, ..., r_d are random numbers satisfying $r_d = -\sum_{i=1}^{d-1} r_i w_i / w_d$.

(2) Random splitting: split $\widehat{l}$ into two parts $\widehat{l}_a$ and $\widehat{l}_b$. For $i = 1, 2, \ldots, d'$,
If $S_i = 0$, $\widehat{l}_a[i]$ and $\widehat{l}_b[i]$ are random numbers such that $\widehat{l}_a[i] + \widehat{l}_b[i] = \widehat{l}[i]$.
If $S_i = 1$, $\widehat{l}_a[i] = \widehat{l}_b[i] = \widehat{l}[i]$.

(3) Data encryption: compute the ciphertexts as $(l'_a = M_1^{-1} \times \widehat{l}_a^T, l'_a = M_1^{-1} \times \widehat{l}_a^T)$.

Table 2: Key notations.

Notations	Description
S	a d-bit string
n	The number of users in the group
DB	The set of point-of-interests
DB*	The encrypted DB
m	The size of database DB
$p = (s,t)$	The coordinate of a point p in DB
$l = (x,y)$	The coordinate of an LBS user
$\{(x_i,y_i)\}_{i=1}^n$	The set of coordinates for LBS users
M_1, M_2	Invertible matrices
$w_1, w_2, \ldots, w_d$	Random numbers

(4) GenInner((p_a', p_b'), (l_a', l_b')): the inner generation algorithm computes and outputs ind $= \langle p_a', l_a' \rangle + \langle p_b', l_b' \rangle$.

(5) Decrypt (sk, (p_a', p_b')): this decryption algorithm first computes $\widehat{p}_a = (M_1^T)^{-1} \times p_a'$ and $\widehat{p}_b = (M_1^T)^{-1} \times p_b'$. For $i = 1, 2, \ldots, n$,

If $S_i = 0$, $\widehat{p}_a[i] = \widehat{p}_b[i] = \widehat{p}[i]$.
If $S_i = 1$, $\widehat{p}[i] = \widehat{p}_a[i] + \widehat{p}_b[i]$.

Finally, the user obtains $s = -\widehat{p}[2]/2$ and $t = -\widehat{p}[3]/2$.

Correctness: the correctness of CompInner tool is illustrated as follows. We have

$$\begin{aligned}
&(p_a')^T \times l_a' + (p_b')^T \times l_b', \\
&\quad = \widehat{p}_a \times M_1 \times M_1^{-1} \times \widehat{l}_a^T + \widehat{p}_b \times M_2 \times M_2^{-1} \times \widehat{l}_b^T \\
&\quad = \left\langle \widehat{p}_a, \widehat{l}_a \right\rangle + \left\langle \widehat{p}_b, \widehat{l}_b \right\rangle \\
&\quad = \left\langle \widehat{p}, \widehat{l} \right\rangle \\
&\quad = r\left(s^2 + t^2 - 2sx - 2ty\right) \\
&\quad = r\left\langle \left(s^2 + t^2, -2s, -2t\right), (1, x, y) \right\rangle.
\end{aligned} \tag{6}$$

4.1. The Main Scheme. The secure GNN scheme, named SecGNN, consists of six polynomial-time algorithms, Setup, EncDB, CompCen, GenToken, Query, and Decrypt. The details of these algorithms are described as follows.

(1) Setup (λ): on inputting a security parameter λ, it outputs a cyclic group G with order q and generator g. Furthermore, the LBS provider runs CompInner.Setup(λ) to output the secret key:

$$sk = \left(M_1, M_2, S, w_1, w_2, \ldots, w_d\right). \tag{7}$$

(2) EncDB (sk, p): suppose $p = (s,t)$ is a point in POIs' database DB. The algorithm EncDB is run by the LBS provider to encrypt the point as

$$c = \left(p_a', p_b'\right) = \textbf{CompInner Encrypt}\left(sk, p\right). \tag{8}$$

(3) ShareRan (G, g): LBS users proceed this algorithm to compute a random value that keeps secreted from outside attackers. In the first phase, LBS users $u_1, u_2, \ldots, u_n$ choose random numbers $e_1, e_2, \ldots, e_n \in \mathcal{Z}_q$ and publish $g^{e_1}, g^{e_2}, \ldots, g^{e_n}$ to other LBS users. In the second phase, users $u_1, u_2, \ldots, u_n$ compute and broadcast $r_1 = (g^{e_2}/g^{e_n})^{e_1}$, $r_2 = (g^{e_3}/g^{e_1})^{e_2}$, $\ldots$, $r_i = (g^{e_{j+1}}/g^{e_{j-1}})^{e_j}$, $\ldots$, and $r_n = (g^{e_1}/g^{e_{n-1}})^{e_n}$. Finally, LBS users compute and obtain

$$\begin{aligned}
k_1 &= \left(g^{e_n}\right)^{ne_1} \cdot r_1^{n-1} \cdot r_2^{n-2} \cdot \cdots \cdot r_{n-1} \bmod q, \\
k_2 &= \left(g^{e_1}\right)^{ne_2} \cdot r_2^{n-1} \cdot r_3^{n-2} \cdot \cdots \cdot r_n \bmod q, \\
&\cdots \\
k_i &= \left(g^{e_{i-1}}\right)^{ne_i} \cdot r_i^{n-1} \cdot r_{i+1}^{n-2} \cdot \cdots \cdot r_{i-2} \bmod q, \\
&\cdots \\
k_n &= \left(g^{e_{n-1}}\right)^{ne_n} \cdot r_n^{n-1} \cdot r_1^{n-2} \cdot r_{n-2} \bmod q.
\end{aligned} \tag{9}$$

It is easy to see that the values $k_1, k_2, \ldots, k_n$ satisfy $k_1 = k_2 = \cdots = k_n = g^{e_1e_2 + e_2e_3 + \cdots + e_ne_1} = k$. The value k is shared by all group users.

(4) GenToken ($sk, k, (x_1, y_1), \ldots, (x_n, y_n)$): after obtaining the secret value k, LBS users $u_1, u_2, \ldots, u_n$ compute $tk_1 = (l_a^1, l_b^1) =$ **Comp Inner.GenToken** $(sk, (x_1, y_1))$, $tk_1 = (l_a^2, l_b^2) =$ **CompInner.GenToken** $(sk, (x_2, y_2))$, $\cdots$, $tk_1 = (l_a^n, l_b^n) =$ **CompInner.GenToken** $(sk, (x_n, y_n))$, respectively. Note that the random number r in above algorithms is chosen as the shared value k.

(5) Query (DB*, $tk_1, tk_2, \ldots, tk_n$): let $p_1^* = (p_{1a}', p_{1b}')$ and $p_1^* = (p_{2a}', p_{2b}')$ are two encrypted points in DB^*. To determine whether the sum of inner products for query token to p_1^* is smaller than that to p_2^*, the cloud server first computes $tk_a = l_a^1 + l_a^2 + \cdots + l_a^n$ and $tk_b = l_b^1 + l_b^2 + \cdots + l_b^n$ and then checks whether

$$(p_{1a}')^T \times tk_a + (p_{1b}')^T \times tk_b - \left((p_{2a}')^T \times tk_a + (p_{2b}')^T \times tk_b\right) < 0. \tag{10}$$

If it holds, the aggregate inner products with p_1^* is smaller than those with p_2^*. Otherwise, the inner products with p_2^* is smaller. The cloud server can linearly search all points in encrypted database and obtains point p^* with the minimum aggregate inner products.

(6) Decrypt (sk, p^*): in the query phase, the cloud server finds with the minimum aggregate inner products. The LBS users run

$$p = (s,t) = \textbf{CompInner.Decrypt}\left(sk, \left(p_a, p_b\right)\right). \tag{11}$$

Finally, LBS users obtain the GNN location, namely, the meeting place.

Correctness: assume that p_1^* and p_2^* are two encrypted POIs in DB*. Note that

$$
\begin{aligned}
&(p_1^* - p_2^*) \cdot (tk_1 + tk_2 + \cdots + tk_n) \\
&= \left(M^T \overline{p}_1 - M^T \overline{p}_2\right) \cdot \left(kM^{-1}\overline{tk}_1 + kM^{-1}\overline{tk}_2 + \cdots + kM^{-1}\overline{tk}_n\right), \\
&= \left(M^T \overline{p}_1 - M^T \overline{p}_2\right)^T \left(kM^{-1}\overline{tk}_1 + kM^{-1}\overline{tk}_2 + \cdots + kM^{-1}\overline{tk}_n\right), \\
&= k(\overline{p}_1 - \overline{p}_2)^T \left(\overline{tk}_1 + \overline{tk}_2 + \cdots + \overline{tk}_n\right), \\
&= k\left(\left(\left\langle \overline{p}_1, \overline{tk}_1 \right\rangle + \left\langle \overline{p}_1, \overline{tk}_2 \right\rangle + \cdots + \left\langle \overline{p}_1, \overline{tk}_n \right\rangle\right)\right. \\
&\quad \left. - \left(\left\langle \overline{p}_2, \overline{tk}_1 \right\rangle + \left\langle \overline{p}_2, \overline{tk}_2 \right\rangle + \cdots + \left\langle \overline{p}_2, \overline{tk}_n \right\rangle\right)\right).
\end{aligned} \tag{12}
$$

Thus, if $(p_1^* - p_2^*) \cdot (tk_1 + tk_2 + \cdots + tk_n) < 0$, then the aggregate inner products for $\overline{p}_1$ is smaller than those for $\overline{p}_2$. That is, the aggregate distances for p_1 are smaller than those for p_2. In such a case, the cloud server can find the group nearest neighbor (GNN) through linearly searching the encrypted database.

4.2. Security Analysis. In this section, we first illustrate the security analysis of our CompInner tool and our SecGNN scheme. The security proofs are described in the following.

Theorem 1. *The proposed CompInner tool preserves data privacy and query privacy under the background model. Specifically, outside attackers could not learn the data items from the encrypted database and could not obtain the query point from the query token.*

Proof 1. Without the loss of generality, we will prove the theorem from data privacy and query privacy two parts. The proof for data privacy will be introduced in the following.

Suppose that outside attackers have the knowledge $H = \langle \mathrm{DB}^*, Q^* \rangle$ and its trace $Tr(H)$, where DB^* is the encrypted database and Q^* is the encrypted queries submitted by group users, and the trace contains access pattern $\alpha(H)$, search pattern $\sigma(H)$, path pattern $\delta(H)$, and identifiers $\mathrm{ID}(Q)$. Based on an encrypted data tuple $c = (p_a' = M_1^T \times \widehat{p}_a^T, p_b' = M_2^T \times \widehat{p}_b^T)$, an adversary should recover two invertible matrices M_1 and M_2 from p_a' and p_b' and further guess the bit string that are involved in the key if she wants to recover the original data from the ciphertexts. In such scenarios, an outside attacker $\mathscr{A}$ has to model $\widehat{p}_a$ and $\widehat{p}_b$ as two d'-dimensional random vectors. The equations for solving the unknowns are

$$
p_a' = M_1^T \times \widehat{p}_a^T, p_a' = M_1^T \times \widehat{p}_b^T. \tag{13}
$$

As we known, string S is a bits string with length d'; thus, the number of possible ways to split $\widehat{p}$ is $2^{d'}$, which is large enough to prevent an adversary from correctly guessing s. On the contrary, the advantage of an adversary correctly guesses s is negligible, and there are 2 d' unknowns in p_a' and p_b' and $2d^2$ unknowns in invertible matrices M_1 and M_2. Since the number of equations 2 d' are less than the number of unknowns 2 $d' + 2d^2$, an outside attacker $\mathscr{A}$ could not solve the equations for the unknowns.

For the query privacy, we pad $(d + 1)$ random numbers for the query point in the encrypt algorithm, which leads to the fact that a same point generates different $\widehat{p}$. Furthermore, we split n–dimensional $\widehat{p}$ into two random vectors $\widehat{q}_a$ and $\widehat{q}_b$, whose technique is same as in the encryption phase. Thus, query privacy is semantically secure.

In this case, we prove that our CompInner tool achieves data privacy and query privacy. □

Theorem 2. *The proposed SecGNN scheme preserves the data privacy and query privacy from outside attackers.*

Proof 2. In the SecGNN scheme, an outside attacker obtains the plaintext from the encrypted database. The encryption algorithm in SecGNN is the same with that in CompInner tool. Therefore, the data privacy can be assured in Theorem 1.

In the following, we proof the query privacy in two parts. First, we prove an outside attacker fails to learn the shared number key k (note that the leaks of random number k could not help an outside attacker to obtain actual locations. The shared number k only helps the LBS users to hide the query frequency). Learning k requires an outside attacker can disclose the AV-net mask. Specifically, an outside attacker computes $g^{e_i e_j}$ from g^{e_i} and g^{e_j} in AV-net protocol. Under the assumption of decisional Diffie–Hellman (DDH) problem, an outside attacker fails to learn knowledge of shared key k.

Furthermore, query tokens are produced in the CompInner.GenToken algorithm. Since CompInner achieves query privacy in Theorem 1, it is easy to see that actual locations can be protected from outside attackers.

Note that LBS users can obtain actual locations of other users in the SecGNN scheme. We will further improve the query privacy in the enhanced scheme SecGNN$^+$, namely, the actual location of a LBS user could not obtained by other LBS users until the rest of LBS users colluded. □

5. Efficiency-Enhanced Group Nearest Neighbor Query

In this section, we aim to construct an efficiency-enhanced group nearest neighbor query scheme (called SecGNN$^+$) that protects actual locations privacy from other LBS users and achieves faster than linear search complexity. The SecGNN$^+$ scheme is designed by the leveraging tree structure, such as KD-tree, R-tree, and quad-tree. In this work, we take KD-tree to describe the details of the proposed SecGNN$^+$ scheme.

Assume that (x_c, y_c) is the centroid of LBS users $u_1, u_2, \ldots, u_n$ and (s, t) is the coordinate of a point p in POIs. Therefore, the distance from the centroid to point p is

$$
\begin{aligned}
d(p)^2 &= (x_c - s)^2 + (y_c - t)^2, \\
&= x_c^2 + y_c^2 - 2sx_c - 2ty_c + s^2 + t^2 \\
&= x_c^2 + y_c^2 + \left\langle \left(s^2 + t^2, -2s, -2t\right), \left(1, x_c, y_c\right) \right\rangle.
\end{aligned} \tag{14}
$$

Since $x_c^2 + y_c^2$ is a fixed number, the NN for centroid (x_c, y_c) is to find a point p that has the minimum inner product $\langle (s^2 + t^2, -2s, -2t), (1, x_c, y_c) \rangle$. Considering GNN in Section 4, it requires to find a point p with the minimum aggregate inner products:

$$\begin{aligned}&\left\langle\left(s^2+t^2,-2s,-2t\right),(1,x_1,y_1)\right\rangle+\left\langle\left(s^2+t^2,-2s,-2t\right),\right.\\&\quad\left.\cdot(1,x_2,y_2)\right\rangle+\cdots+\left\langle\left(s^2+t^2,-2s,-2t\right),(1,x_n,y_n)\right\rangle,\\&=n\left\langle\left(s^2+t^2,-2s,-2t\right),\left(1,\left(x_1+x_2+\cdots+x_n\right)/n,\right.\right.\\&\quad\left.\left.\cdot\left(y_1+y_2+\cdots+y_n\right)/n\right),\\&=n\left\langle\left(s^2+t^2,-2s,-2t\right),(1,x_c,y_c)\right\rangle,\end{aligned}\tag{15}$$

where $x_c=(x_1+x_2+\cdots+x_n)/n$ and $x_c=(y_1+y_2+\cdots+y_n)/n$.

Therefore, the GNN problem requires to find a point p in POIs with the minimum inner product $\langle(s^2+t^2,-2s,-2t),(1,x_c,y_c)\rangle$. That is, the GNN problem can be converted as the NN problem for their centroid:

In the following, we describe the problem on how to use KD-tree to enhance the efficiency and support privacy preserving. The KD-tree for solving the NN problem requires comparison and computation operations, namely, the order comparison between query points and tree nodes should be proceeded, and distances from query points to tree nodes and splitting lines should also be computed.

From the root to leaf nodes, the search algorithm is done to check whether $x>s$ or $y>t$, where (x,y) is the coordinate of the query point q and (s,t) is the coordinate of the current tree node. The checking is done in each dimension. A function f on $x-$coordinate is proposed as

$$\begin{aligned}f&=(s-x)(s+\delta),\\&=s^2+(\delta-x)s-x\delta\\&=\left\langle\left(s^2,s,1\right),(1,\delta-x,-x\delta)\right\rangle,\end{aligned}\tag{16}$$

where δ is much bigger than $|s|$, and thus, $s+\delta$ is a positive number.

The square of distance from the query point to tree node is defined as

$$\begin{aligned}d^2&=(x-s)^2+(y-t)^2,\\&=x^2-2sx+s^2+y^2-2ty+t^2\\&=\left\langle\left(s^2,s,1\right),\left(1,-2x,x^2\right)\right\rangle+\left\langle\left(t^2,t,1\right),\left(1,-2y,y^2\right)\right\rangle.\end{aligned}\tag{17}$$

The distance from query point to splitting line (e.g., $x-$coordinate) is defined as

$$\begin{aligned}d_1^2&=(x-s)^2,\\&=x^2-2sx+s^2\\&=\left\langle\left(s^2,s,1\right),\left(1,-2x,x^2\right)\right\rangle.\end{aligned}\tag{18}$$

Thus, the order comparison and distance computation are converted to the inner products computation. As described in CompInner tool in the SecGNN scheme, we first introduce a tool called CompInner$^+$ to compute the above inner products. The basic idea of CompInner$^+$ is similar to CompInner. The differences occur at data encryption and token generation.

5.1. Basic Tool. The proposed CompInner$^+$ contains five polynomial-time algorithms, Setup, Encrypt, GenToken, GenInner, and Decrypt. The details are described as follows.

(1) Setup (λ): it is the same with the system setup algorithm in CompInner. Thus, the secret key is $sk=(M_1,M_2,S,w_1,w_2,\ldots,w_d)$.

(2) Encrypt (sk,p): suppose that $p=(s,t)$ is a data record, and this algorithm does the following steps.

(1) Adding random numbers: for a data $p=(s,t)$, the user computes $\overline{s}=(s^2,s,1),\overline{t}=(t^2,t,1)$ and then extends them as $\widehat{s}=(s^2,s,1,w_1,w_2,\ldots,w_d)$ and $\widehat{t}=(t^2,t,1,w_1,w_2,\ldots,w_d)$.

(2) Random splitting: split $\widehat{s},\widehat{t}$ into two parts, $\widehat{s}_a,\widehat{s}_b$ and $\widehat{t}_a,\widehat{t}_b$. For $i=1,2,\ldots,d'$,

If $S_i=0$, $\widehat{s}_a[i]=\widehat{s}_b[i]=\widehat{s}[i]$ and $\widehat{t}_a[i]=\widehat{t}_b[i]=\widehat{t}[i]$.

If $S_i=1,\widehat{s}_a[i],\widehat{s}_b[i]$ and $\widehat{t}_a[i],\widehat{t}_b[i]$ are random numbers such that $\widehat{s}_a[i]+\widehat{s}_b[i]=\widehat{s}[i]$ and $\widehat{t}_a[i]+\widehat{t}_b[i]=\widehat{t}[i]$.

(3) Data encryption: compute the ciphertexts as

$$\begin{aligned}c_1&=\left(s_a'=M_1^T\times\widehat{s}_a^T,s_b'=M_2^T\times\widehat{s}_b^T\right),\\c_2&=\left({}_a't=M_1^T\times\widehat{t}_a^T,t_b'=M_2^T\times\widehat{t}_b^T\right).\end{aligned}\tag{19}$$

Thus, the ciphertext for data point p is $c=(c_1,c_2)$.

(3) GenToken (sk,l): suppose that $l=(x,y)$, the token is generated as follows.

(1) Adding random numbers: for a data $l=(x,y)$, the user extends it as

$$\begin{aligned}\widehat{x}_1&=\left(1,\delta_1-x,-\delta_1x,r_{x1}^1,r_{x1}^2,\ldots,r_{x1}^d\right),\\\widehat{x}_2&=\left(1,-2x,x^2,r_{x2}^1,r_{x2}^2,\ldots,r_{x2}^d\right),\\\widehat{y}_1&=\left(1,\delta_2-y,-\delta_2y,r_{y1}^1,r_{y1}^2,\ldots,r_{y1}^d\right),\\\widehat{y}_2&=\left(1,-2y,y^2,r_{y2}^1,r_{y2}^2,\ldots,r_{y2}^d\right),\end{aligned}\tag{20}$$

where δ_1,δ_2, $r_{x1}^1,\ldots,r_{x1}^d$, $r_{x2}^1,\ldots,r_{x2}^d$, $r_{y1}^1,\ldots,r_{y1}^d$, and $r_{y2}^1,\ldots,r_{y2}^d$ are random numbers such that

$$\begin{aligned}r_{x1}^d&=-\sum_{i=1}^{d-1}\frac{r_{x1}^iw_i}{w_d},\\r_{x2}^d&=-\sum_{i=1}^{d-1}\frac{r_{x2}^iw_i}{w_d},\\r_{y1}^d&=-\sum_{i=1}^{d-1}\frac{r_{y1}^iw_i}{w_d},\\r_{y2}^d&=-\sum_{i=1}^{d-1}\frac{r_{y1}^iw_i}{w_d}.\end{aligned}\tag{21}$$

(2) Random splitting: split $\widehat{x}_1, \widehat{x}_2, \widehat{y}_1$, and $\widehat{y}_2$ into two parts $\widehat{x}_{1a}, \widehat{x}_{1b}, \widehat{x}_{2a}, \widehat{x}_{2b}, \widehat{y}_{1a}, \widehat{y}_{1b}$, and $\widehat{y}_{2a}, \widehat{y}_{2b}$. For $i = 1, 2, \ldots, d'$,

If $S_i = 0$, $\widehat{x}_{1a}[i]$, $\widehat{x}_{1b}[i]$, $\widehat{x}_{2a}[i]$, $\widehat{x}_{2b}[i]$, $\widehat{y}_{1a}[i]$, $\widehat{y}_{1b}[i]$, and $\widehat{y}_{2a}[i]$, $\widehat{y}_{2b}[i]$ are random numbers such that

$$\begin{aligned}
\widehat{x}_{1a}[i] + \widehat{x}_{1b}[i] &= \widehat{x}_1[i], \\
\widehat{x}_{2a}[i] + \widehat{x}_{2b}[i] &= \widehat{x}_2[i], \\
\widehat{y}_{1a}[i] + \widehat{y}_{1b}[i] &= \widehat{y}_1[i], \\
\widehat{y}_{2a}[i] + \widehat{y}_{2b}[i] &= \widehat{y}_2[i].
\end{aligned} \tag{22}$$

If $S_i = 1$,

$$\begin{aligned}
\widehat{x}_{1a}[i] = \widehat{x}_{1b}[i] &= \widehat{x}_1[i], \\
\widehat{x}_{2a}[i] = \widehat{x}_{2b}[i] &= \widehat{x}_2[i], \\
\widehat{y}_{1a}[i] = \widehat{y}_{1b}[i] &= \widehat{y}_1[i], \\
\widehat{y}_{2a}[i] = \widehat{y}_{2b}[i] &= \widehat{y}_2[i].
\end{aligned} \tag{23}$$

(3) Data encryption: compute the ciphertexts as

$$\begin{aligned}
tk_{11}^a &= M_1^{-1} \times \widehat{x}_{1a}, tk_{11}^b = M_1^{-1} \times \widehat{x}_{1b}, \\
tk_{12}^a &= M_1^{-1} \times \widehat{x}_{2a}, tk_{12}^b = M_1^{-1} \times \widehat{x}_{2b}, \\
tk_{21}^a &= M_1^{-1} \times \widehat{y}_{1a}, tk_{21}^b = M_1^{-1} \times \widehat{y}_{1b}, \\
tk_{22}^a &= M_1^{-1} \times \widehat{y}_{2a}, tk_{22}^b = M_1^{-1} \times \widehat{y}_{2b}.
\end{aligned} \tag{24}$$

Thus, the token generation algorithm outputs token as $tk = (tk_{11}, tk_{12}, tk_{21}, tk_{22})$, where $tk_{11} = (tk_{11}^a, tk_{11}^b)$, $tk_{12} = (tk_{12}^a, tk_{12}^b)$, $tk_{21} = (tk_{21}^a, tk_{21}^b)$, and $tk_{22} = (tk_{22}^a, tk_{22}^b)$.

(4) GenInner((p_a', p_b'), (l_a', l_b')): the inner generation algorithm computes and outputs

$$\text{ind} = \langle p_b', l_b' \rangle + \langle l_a', l_b' \rangle \tag{25}$$

(5) Decrypt($sk, c = (c_1, c_2)$): suppose that $c_1 = (s_a', s_b')$ and $c_2 = (t_a', t_b')$. The decryption algorithm first computes

$$\begin{aligned}
\widehat{s}_a &= \left(M_1^T\right)^{-1} \times s_a', \\
\widehat{s}_b &= \left(M_1^T\right)^{-1} \times s_b', \\
\widehat{t}_a &= \left(M_1^T\right)^{-1} \times t_a', \\
\widehat{t}_b &= \left(M_1^T\right)^{-1} \times t_b'.
\end{aligned} \tag{26}$$

For $i = 1, 2, \ldots, d'$,

If $S_i = 0$, $\widehat{s}[i] = \widehat{s}_a[i] = \widehat{s}_a[i]$ and $\widehat{t}[i] = \widehat{t}_a[i] = \widehat{t}_b[i]$
If $S_i = 1$, $\widehat{s}[i] = \widehat{s}_a[i] + \widehat{s}_b[i]$ and $\widehat{t}[i] = \widehat{t}_a[i] + \widehat{t}_b[i]$

Finally, the user obtains $s = \widehat{s}[2]$ and $t = \widehat{t}[2]$.

5.2. The Detailed Description. From above, we convert the problem of finding GNN for a group of users to the problem of finding NN for their centroid. In the following, we illustrate an efficiency-enhanced group nearest neighbor query scheme called SecGNN$^+$ based on KD-tree and CompInner$^+$ tool. The introduced SecGNN$^+$ scheme consists of seven polynomial-time algorithms, Setup, BuildTree, EncDB, CompCen, GenToken, Query, and Decrypt. The detailed description is illustrated as follows:

(1) Setup (1^λ): on inputting a security parameter λ, it outputs a cyclic group G with order q and generator g. Furthermore, this algorithm runs CompInner$^+$ tool and outputs a secret key:

$$sk = (M_1, M_2, S, w_1, w_2, \ldots, w_d). \tag{27}$$

(2) BuildTree (DB): on inputting the database DB of POIs, this algorithm proceeds the KD-tree algorithm and stores the data items in a KD-tree T.

(3) EncDB (sk, p): suppose that $p = (s, t)$ is a data record stored in KD-tree T. The LBS provider runs CompInner$^+$.Encrypt(sk, p) to compute the ciphertext as

$$c = (c_1, c_2) = \left((s_a', s_b'), (t_a', t_a')\right). \tag{28}$$

(4) CompCen ($l_1, l_2, \ldots, l_n$): suppose that $l_1 = (x_1, y_1)$, $l_1 = (x_2, y_2), \cdots, l_n = (x_n, y_n)$ are coordinates of LBS users $u_1, u_2, \ldots, u_n$. LBS users evaluate the centroid as follows.

(1) Each LBS user u_i chooses two random numbers a_i and $e_i \in \mathcal{Z}_q$ and publishes (g^{a_i}, g^{e_i}) to other LBS users. Then, u_i computes and further broadcasts $r_i = (g^{e_{i+1}}/g^{e_{i-1}})^{e_i}$.

(2) The user u_i computes

$$\begin{aligned}
g^{b_i} &= \frac{\Pi_{j=1}^{i-1} g^{a_j}}{\Pi_{j=i+1}^{n} g^{a_j}}, \\
k_i &= \left(g^{e_{i-1}}\right)^{ne_i} \cdot r_i^{n-1} \cdot r_{i+1}^{n-2} \cdot \cdots \cdot r_{i-2} \bmod q.
\end{aligned} \tag{29}$$

(3) User u_i computes and publishes $X_i = g^{a_i b_i} g^{e_{i-1} e_i} g^{x_i}$. Multiplying all X_i,

$$\begin{aligned}
\Pi_{i=1}^{n} X_i &= \Pi_{i=1}^{n} g^{a_i b_i} g^{e_{i-1} e_i} g^{x_i}, \\
&= \Pi_{i=1}^{n} g^{a_i b_i} \Pi_{i=1}^{n} g^{e_{i-1} e_i} \Pi_{i=1}^{n} g^{x_i} \\
&= k_i g^{\sum_{i=1}^{n} x_i} = k g^{\sum_{i=1}^{n} x_i}.
\end{aligned} \tag{30}$$

User u_i can get the value $g^{\sum_{i=1}^{n} x_i}$ through dividing by the secret key k. Since $\sum_{i=1}^{n} x_i$ is a small number, LBS users can compute the discrete logarithm by using the Pohlig–Hellman algorithm [30]. Considering the coordinates are six- or seven-decimal digits, they can be represented as a 32 bit data. Therefore, $\sum_{i=1}^{n} x_i$ is a small number and can be efficiently computed from $g^{\sum_{i=1}^{n} x_i}$.

(4) Finally, user u_i can compute the centroid as $x_c = \sum_{i=1}^{n} x_i/n$ and $y_c = \sum_{i=1}^{n} y_i/n$.

(5) GenToken $(sk, (x_c, y_c))$: after receiving the centroid (x_c, y_c), LBS user runs CompInner +.GenToken(sk, p) to compute the token as $tk = (tk_{11}, tk_{12}, tk_{21}, tk_{22})$, where $tk_{11} = (tk_{11}^a, tk_{11}^b)$, $tk_{12} = (tk_{12}^a, tk_{12}^b)$, $tk_{21} = (tk_{21}^a, tk_{21}^b)$, and $tk_{22} = (tk_{22}^a, tk_{22}^b)$.

(6) Query (T^*, tk): the query algorithm consists of two phases. In the first phase, the cloud server searches over the encrypted KD-tree T^* to find a leaf node whose corresponding region includes the query point. The cloud server computes $\text{ind} = \text{CompInner}^+.\text{GenInner}(c_1, tk_{11})$ (or $\text{ind} = \text{CompInner}^+.\text{GenInner}(c_2, tk_{21})$) and checks whether $\text{ind} > 0$. If it holds, the cloud server searches the left child; otherwise, it searches the right child. The cloud server writes down the route from root to the leaf node in a list L.

After arriving at the leaf node, the cloud server takes this leaf node as the temporary NN and proceeds the second phase to check whether the temporary NN is the NN. In the second phase, the cloud server checks the list L from leaf nodes back to the root. For each node, he computes

$$\begin{aligned} \text{ind}_1 &= \textbf{CompInner}^+.\textbf{GenInner}(c_1, tk_{12}), \\ \text{ind}_{\text{temp1}} &= \textbf{CompInner}^+.\textbf{GenInner}\left(c_{\text{temp1}}, tk_{12}\right), \\ \text{ind}_2 &= \textbf{CompInner}^+.\textbf{GenInner}(c_2, tk_{22}), \\ \text{ind}_{\text{temp2}} &= \textbf{CompInner}^+.\textbf{GenInner}\left(c_{\text{temp2}}, tk_{22}\right), \end{aligned} \tag{31}$$

and checks whether $\text{ind}_1 - \text{ind}_{\text{temp1}} + \text{ind}_2 - \text{ind}_{\text{temp2}} > 0$, where p_{temp} is the temporary NN and p is the current node in the list L. If it holds, the cloud server searches the father node; otherwise, the cloud server updates the current node as the temporary NN and further checks whether $\text{ind}_1 > 0$ (or $\text{ind}_2 > 0$). If it holds, the cloud server searches the other child of temporary NN. Otherwise, it backs to search the father node until to the root.

(1) Decrypt(p^*): the NN for the centroid is computed as p^* in the query algorithm. LBS users run CompInner$^+$.Decrypt(sk, c) to compute the plaintext. Finally, the LBS users can obtain the location, namely, the NN for the centroid and the GNN for the group of LBS users.

Correctness: during the SecGNN$^+$.Query phase, the cloud server evaluates $\text{ind} = \text{CompInner}^+.\text{GenInner}(c_1, tk_{11})$ (or $\text{ind} = \text{CompInner}^+.\text{GenInner}(c_2, tk_{21})$) and checks whether $\text{ind} > 0$:

$$\begin{aligned} \text{ind} &= \textbf{CompInner}^+.\textbf{GenInner}(c_1, tk_{11}), \\ &= \textbf{CompInner}^+.\textbf{GenInner}\left(\left(s_a', s_b'\right), \left(tk_{11}^a, tk_{11}^b\right)\right), \\ &= s_a' \cdot tk_{11}^a + s_b' \cdot tk_{11}^b, \\ &= \left(M_1^T \times \widehat{s}_a^T\right)^T \times M_1^{-1} \times \widehat{x}_{1a} + \left(M_2^T \times \widehat{s}_b^T\right)^T \times M_2^{-1} \times \widehat{x}_{1b}, \\ &= \widehat{s}_a \times \widehat{x}_{1a} + \widehat{s}_b \times \widehat{x}_{1b}, \\ &= s^2 + s(\delta_1 - x) - \delta_1 x, \\ &= (s - x)(s + \delta)_1. \end{aligned} \tag{32}$$

If $\text{ind} > 0$, then we have $(s - x)(s + \delta_1) > 0$, namely, $s > x$. In such a case, the cloud server searches the left child. For y−coordinate, we have the same conclusion. Therefore, the cloud server can arrive at the leaf node correctly:

$$\begin{aligned} \text{ind}_1 &= \textbf{CompInner}^+.\textbf{GenInner}(c_1, tk_{12}), \\ &= s^2 - 2sx + x^2, \\ &= (s - x)^2. \end{aligned} \tag{33}$$

We also compute $\text{ind}_{\text{temp1}} = (s - x_{\text{temp}})^2$, $\text{ind}_2 = (y - t)^2$, and $\text{ind}_{\text{temp2}} = (t - y_{\text{temp}})^2$. Thus, we have

$$\begin{aligned} \text{flag} &= \text{ind}_1 - \text{ind}_{\text{temp1}} + \text{ind}_2 - \text{ind}_{\text{temp2}}, \\ &= (s - x)^2 + (t - y)^2 - \left(s - x_{\text{temp}}\right)^2 - \left(t - y_{\text{temp}}\right)^2. \end{aligned} \tag{34}$$

If $\text{flag} > 0$, it is easily known that the current node is closer than the temporary NN. Thus, we update the current node as the new temporary NN. Furthermore, the cloud server checks whether to search the other child through checking $\text{ind}_1 > 0$ or $\text{ind}_2 > 0$. Therefore, the cloud server can search the NN for the query point over ciphertexts from the core idea of KD-tree.

5.3. Security Analysis. In this section, we illustrate the security analysis of the proposed CompInner$^+$ tool and SecGNN^+ scheme from data privacy and query privacy (e.g., actual location privacy of LBS users and centroid privacy).

Theorem 3. *The introduced CompInner$^+$ tool achieves data privacy and query privacy from outside attackers.*

Proof 3. The security analysis of the introduced tool is similar to CompInner, which is presented in Theorem 1. Thus, we skip the details of proof for the sake of clarity. □

Theorem 4. *The proposed SecGNN$^+$ scheme preserves the data privacy and the query privacy.*

Proof 4. In the following, we present the proof from data privacy against outside attackers and query privacy (actual locations and the centroid) against outside attackers and other LBS users.

First, we will introduce the data privacy of the proposed SecGNN^+ scheme. Data items are encrypted by using CompInner$^+$.Encrypt(sk, p). Since the data privacy can be assured in Theorem 3, the SecGNN^+ scheme preserves data privacy. We skip the details for the sake of clarity.

Furthermore, the centroid privacy from outside attackers will be presented as follows. In the CompCen algorithm, an outside attacker learns the knowledge about the secret key k from solving the DDH problem (as described in the proof of Theorem 2). Thus, the centroid privacy can be assured from outside attackers in the CompCen algorithm. In the GenToken algorithm, the centroid is encrypted as

$$tk = \textbf{CompInner}^+.\textbf{GenToken}(sk, l), \tag{35}$$

where $l = (x_c, y_c)$ is the centroid of group of users. Since CompInner$^+$ tool achieves query privacy, the centroid privacy can be assured in the GenToken algorithm.

Moreover, the introduced SecGNN$^+$ scheme preserves the privacy of actual locations from outside attackers and other LBS users. The centroid computation (CompCen) algorithm consists of AV-net and BD conference key agreement protocols. That is, the centroid privacy is based on these two schemes. An outside attacker fails to obtain the centroid because learning the location requires the attacker can disclose the AV-net mask and learn the knowledge of BD conference key. Specifically, an outside attacker needs to compute $g^{a_i b_i}$ from g^{a_i} and g^{b_i} in AV-net protocol and compute $g^{e_{i-1}e_i}$ from $g^{e_{i-1}}$ and g^{e_i} in BD conference key agreement, respectively. Under the assumption of decisional Diffie–Hellman (DDH) problem, an outside attacker fails to obtain the centroid. A LBS user u_i takes part in the attack game and wants to learn the knowledge of u_{i+1}'s actual location. Since LBS user u_i knows e_i and $g^{e_{i+1}}$, then he can compute $g^{e_i e_{i+1}}$. Based on the difficulty of the DDH problem, LBS user u_i fails to compute $g^{a_{i+1}b_{i+1}}$ from $g^{a_{i+1}}$ and $g^{b_{i+1}}$. Thus, LBS user u_i could not learn the knowledge about the actual location of LBS user u_{i+1}. □

6. Performance Evaluation

In this section, we first illustrate the performance of the basic tools (i.e., CompInner and CompInner$^+$) and then demonstrate that of the introduced SecGNN and SecGNN^+ schemes through experimental simulation. Specifically, we evaluate the performance with Python3 language on a machine with Intel(R) Core(TM) i7-9750H CPU processor running at 16 GB and 1 TB memory in Table 3. The matrix is provided by Numpy library, and the discrete logarithm is computed by the Pohlig–Hellman [30] algorithm. Throughout the experiment, we simulate all the LBS provider, LBS users, and the cloud server on the same machine. Furthermore, we set the extended length as $d' = 80$, that is, equivalent the security of 1024 bit RSA keys.

Database: in the experiment, we evaluate the performance on a real database and a random database, respectively. The real database is the Sequoia database (Sequoia database. http://chorochronos.datastories.org/?q=node/58) that contains 62,556 data items in California. The random database contains 1,000,000 data items which are randomly generated. The descriptions of the Sequoia database and random database are presented in Figure 4.

6.1. Performance of the Basic Tools. CompInner and CompInner$^+$ include system setup (Setup), data encryption (Encrypt), token generation (GenToken), inner computation (GenInner), and data decryption (Decrypt) algorithms. Figure 5 evaluates the performance of above algorithms with the increasing of expanded number d' (set $n = 1000$, namely, 1000 Setup, Encrypt, GenInner, GenToken, and Decrypt). Time costs of GenToken and Decrypt algorithms are mainly occupied by the computation of the inverse matrices for secret keys M_1 and M_2. If the inverse matrices M_1^{-1} and M_2^{-1} are precomputed, time costs of GenToken and Decrypt algorithms will present the similar performance with the encrypt algorithm in both tools. The detailed performance will be presented in the following experiments.

Table 3: Experimental simulation platform.

Operating system	Window 10
CPU	Intel(R) Xeon(R) i7-9750H, 2.6GHZ
Memory	16 GB RAM
Program language	Python 3.8

LBS provider runs CompInner.Setup(λ) and CompInner$^+$.Setup(λ) algorithms to produce the secret key sk, which contains two 80×80 invertible matrices M_1 and M_2, a 80 bit string S, and random numbers $w_1, w_2, \ldots, w_{77}$. To enhance the efficiency, we precompute the invertible matrices M_3 and M_4 for matrices M_1 and M_2. The system setup algorithm can be finished in 10 ms and thus is very efficient.

As illustrated in Figure 6, we evaluate the performance of CompInner tool and CompInner$^+$ tool over the Sequoia database and a random database. It is easily seen that all algorithms have a high efficiency in both tools. Since the GenInner in CompInner tool is the same with that in CompInner$^+$, time cost of the GenInner algorithm performs the same performance. Specifically, the GenInner algorithm has the best efficiency, about 0.5 s for 60,000 inner computations and only 7.6 s for 1,000,000 inner computations for CompInner and CompInner$^+$. The GenToken algorithm shows the worst time cost, 8.6 s for 60,000 tokens and 141.2 s for 1,000,000 tokens on CompInner tool and 32 s for 60,000 tokens and 513 s for 1,000,000 tokens on CompInner$^+$ tool. Furthermore, since GenInner is performed once for the data utilization, it is very efficient in the following group nearest neighbor query scheme, only 0.14 ms for one token in CompInner tool and 0.51 ms for one token in CompInner$^+$ tool. The encrypt algorithm is an one-time algorithm for database construction and is acceptable for both tools. In conclusion, both tools are very efficient for the data utilization in practice.

6.2. Performance of the Introduced Schemes. For the convenience of description, we adapt some notations in this section. We denote by PlainNN the linear search nearest neighbor on plaintexts, by PlainGNN the linear search group nearest neighbor on plaintexts, by KDNN the KD-tree used in the nearest neighbor query on plaintexts, by SecGNN the SecGNN scheme in Section 4, and by SecGNN$^+$ the efficiency enhanced scheme in Section 5.

The introduced SecGNN scheme contains six algorithms, system setup, database encryption, random share, token generation, database query, and result decryption, and the SecGNN$^+$ scheme consists of seven algorithms, system setup, KD-tree build, database encryption, centroid computation, token generation, database query, and result decryption. In the following, we will describe the performance of above algorithms, respectively.

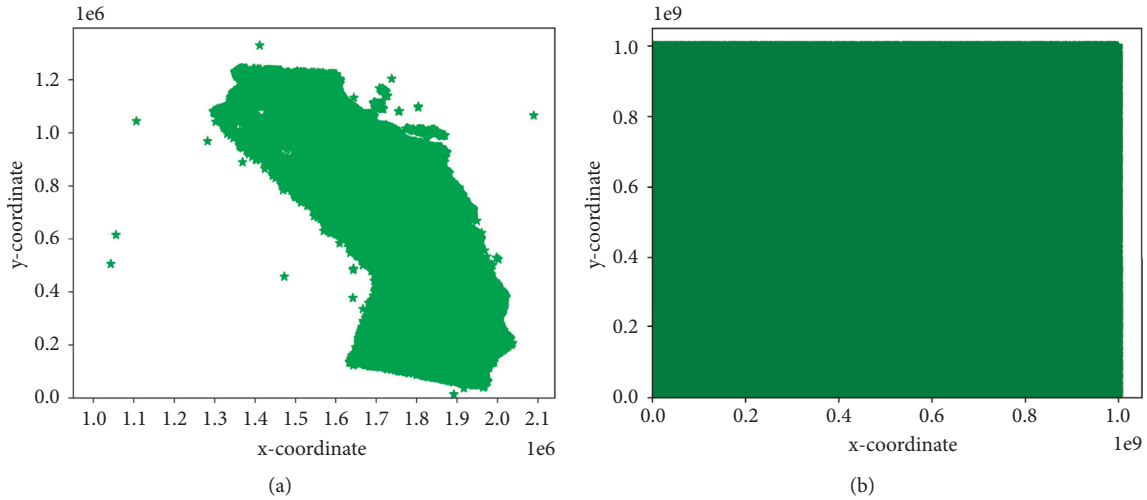

Figure 4: The normalization of the database. (a) Sequoia database. (b) Random database.

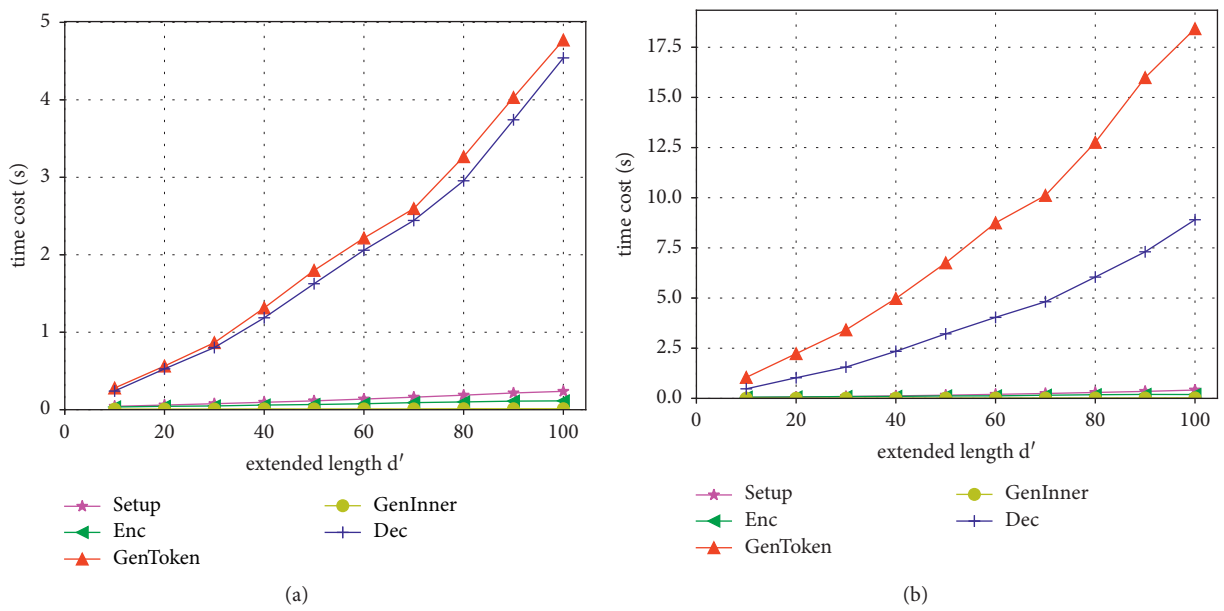

Figure 5: The performance of CompInner and CompInner$^+$ with the extended length d'. (a) CompInner tool. (b) CompInner$^+$ tool.

System setup: for the introduced SecGNN and SecGNN$^+$ schemes, the LBS provider runs the setup algorithm to output a cyclic group G with order q and generator g. Furthermore, the setup algorithm outputs a secret key sk, namely, two 80×80 invertible matrices M_1 and M_2, a 80 bit string S, and random numbers $w_1, w_2, \ldots, w_{77}$. For the convenience of computation, the invertible matrices of M_1 and M_2 are precomputed in this phase. The experimental simulation shows that the setup algorithm can be done in 5 ms.

Tree construction: the BuildTree algorithm is run by the LBS provider to store POIs in a KD-tree data structure. The performance is illustrated in Figure 7. Figure 7(a) illustrates the time cost increasing with the size of data items in the Sequoia database and Figure 7(b) presents that in a random database. Moreover, KDNN and SecGNN$^+$ present the same efficiency as the BuildTree algorithm and only requires the LBS provider to store POIs in a KD-tree structure.

Database encryption: the LBS provider runs the EncDB algorithm to assure the privacy of data items. Figure 8 presents the encryption efficiency of SecGNN and SecGNN$^+$ schemes. As illustrated in Figures 8(a) and 8(b), it is easily seen that both schemes achieve high efficiency, about 99.27 s for 1,000,000 data items on the SecGNN scheme and 182.4 s for 1,000,000 data items on the SecGNN$^+$ scheme. Furthermore, SecGNN achieves better encryption efficiency than the SecGNN$^+$ scheme. Because database encryption is one-time operation, it is acceptable for the SecGNN$^+$ scheme.

Shared random and centroid computation: LBS users run the ShareRan algorithm to share a random number in

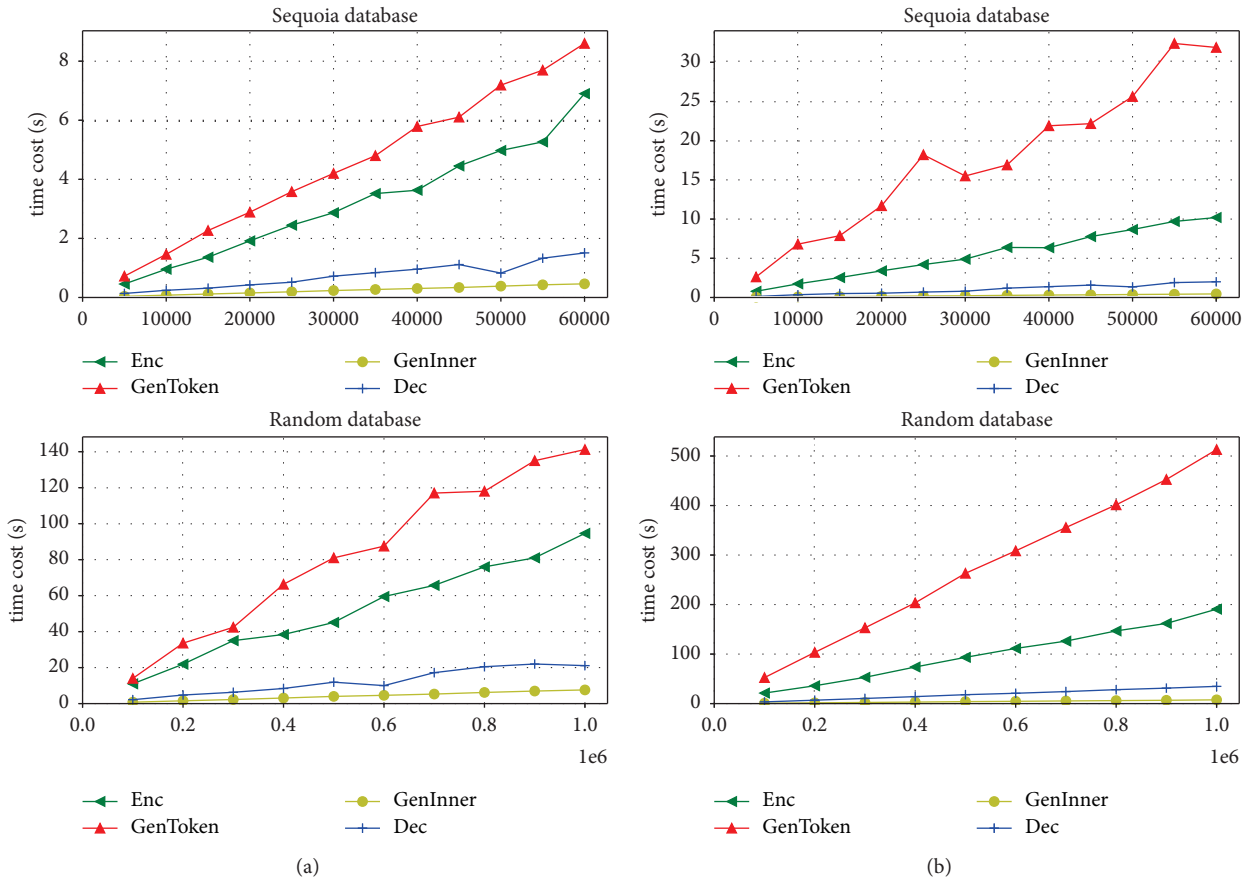

Figure 6: The performance of the CompInner tool and CompInner$^+$ tool increasing with the size of data items on the Sequoia database and a random database. (a) CompInner tool. (b) CompInner$^+$ tool.

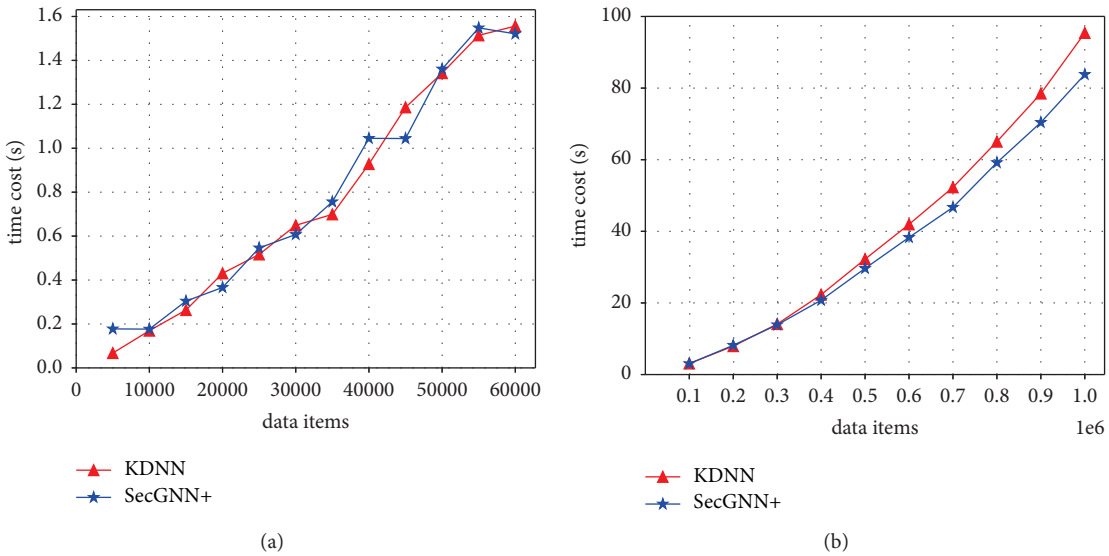

Figure 7: Time cost of the KD-tree construction. (a) Sequoia database. (b) Random database.

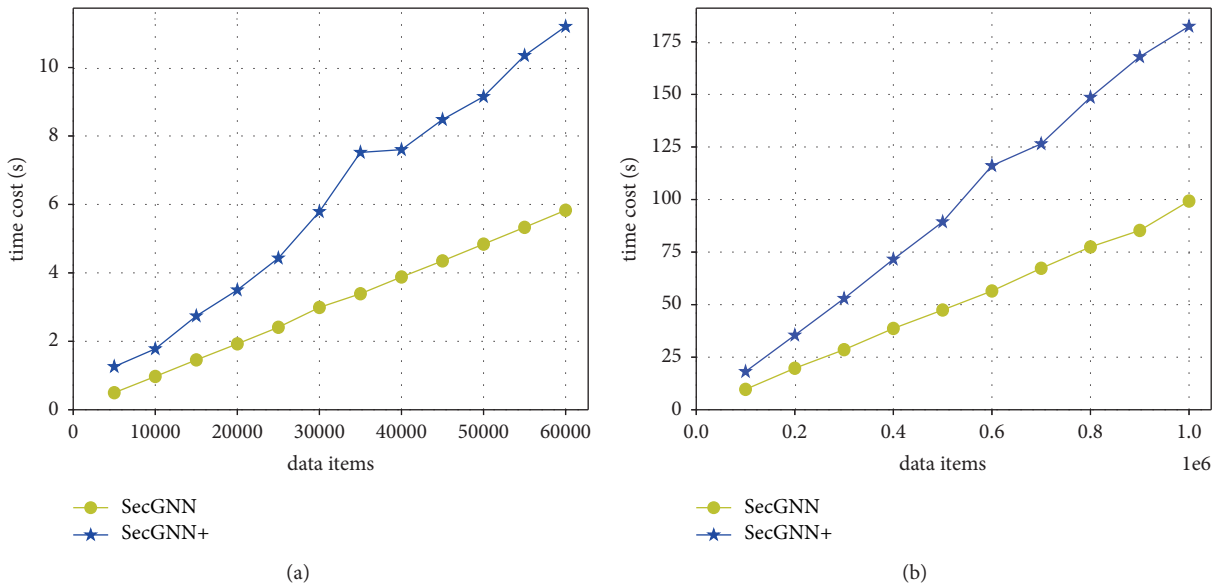

Figure 8: Time cost of the database encryption. (a) Sequoia database. (b) Random database.

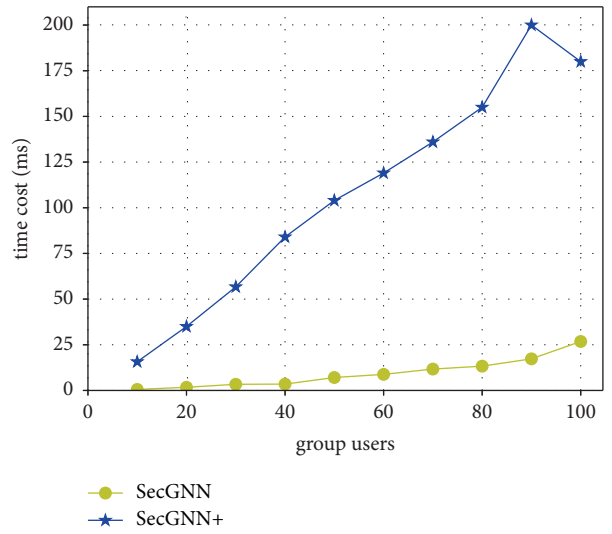

Figure 9: Time cost of centroid computation.

the SecGNN scheme and run the CompCen algorithm to compute the centroid for the following algorithms. LBS users compute the shared random number through BD-protocol and compute their centroid by using AV-net protocol and BD-protocol. Figure 9 presents the time cost increasing with the size of group users. The group size varies from 10 to 100. It costs about 25 ms to share a random number with a group of 100 LBS users in the SecGNN scheme and 180 ms to compute the centroid with 100 LBS users in the SecGNN$^+$ scheme. In the CompCen algorithm and the following algorithms, it is enough for one special LBS user to compute the centroid and proceed the group nearest neighbor query. Finally, the special LBS user broadcasts the meeting place using the conference key. As demonstrated in Figure 9, the centroid computation is very efficient, 18 ms for 10 users and 180 ms for 100 users.

Token generation: in the SecGNN and SecGNN$^+$ scheme, LBS users compute and generate the query token for the cloud server to search over the encrypted POIs. The cloud server directly searches the group nearest neighbor in PlainNN, PlainGNN, and KDNN, and therefore, they do not need to execute the GenToken algorithm. After receiving the shared random number in the SecGNN scheme, LBS users run the GenToken algorithm to compute the search token, respectively. The SecGNN$^+$ scheme requires a special LBS user to compute the search token and send it to the cloud server. As illustrated in Figure 6, the search token can be computed quickly, about 0.15 ms for SecGNN and 0.5 ms for the SecGNN$^+$ scheme.

Group nearest neighbor query: the cloud server proceeds the query algorithm to search the group nearest neighbor over encrypted POIs. In Figure 10, we illustrate the time cost

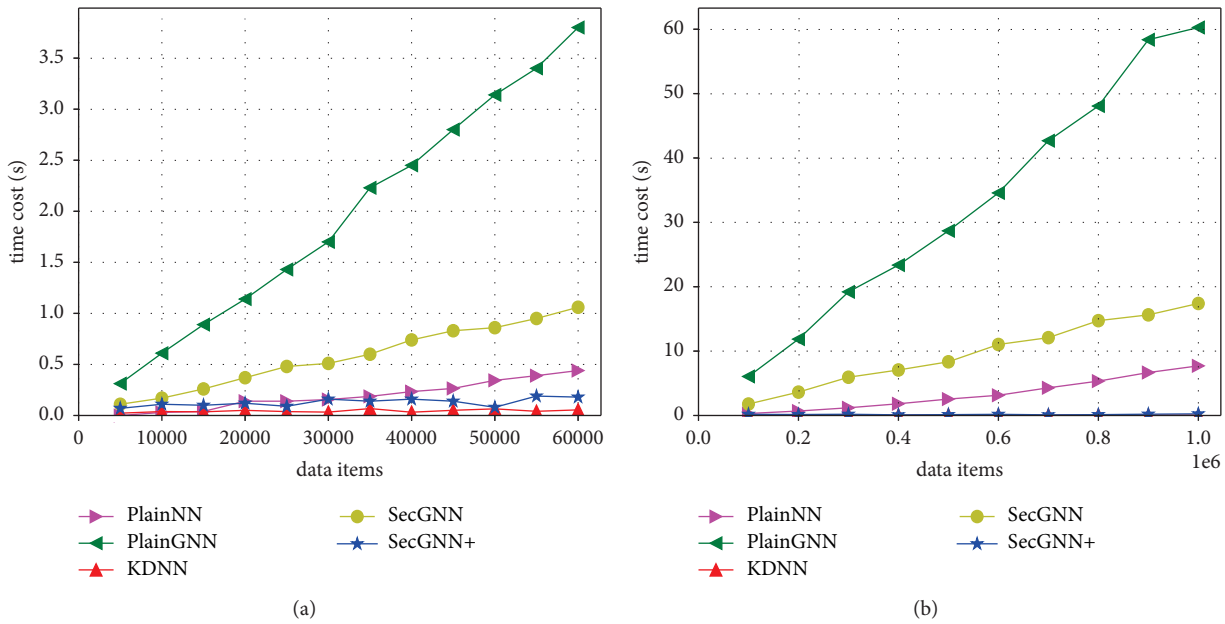

Figure 10: Time cost of the group nearest neighbor query. (a) Sequoia database. (b) Random database.

for PlainNN, PlainGNN, and KDNN on plaintexts and time cost for SecGNN and SecGNN$^+$ on encrypted POIs. Figure 10 presents that the PlainGNN scheme is a little slower than other introduced schemes in both the Sequoia database and random database. Since LBS users should compute the aggregate distance for each point in POIs to all group users ($n = 100$) and linearly search the entire POIs, time cost on PlainGNN has the worst efficiency. The KDNN scheme has the best efficiency for the group nearest neighbor query over above schemes. Time cost of SecGNN scales linearly with the data items. Thus, SecGNN is impractical with million data items. As illustrated in Figure 10, the SecGNN$^+$ scheme achieves practical search efficiency in data utilization on big data era, about 0.2 s on millions database.

Decryption: after receiving the query result, LBS users decrypt the results and finally receive the meeting place. The decrypt algorithm only contains two matrix multiplications. The multiplication is very fast and can be computed less than 2 ms.

7. Conclusions

In this work, we study the problem of the privacy-preserving group nearest neighbor query, namely, finding a meeting place with the minimum aggregate distance to a group of LBS users. The introduced SecGNN scheme supports group nearest neighbor ($n \geq 3$), while preserving the data privacy and query privacy from outside attackers. Unfortunately, it only achieves linear search complexity. To achieve high search efficiency, we design an efficiency-enhanced SecGNN$^+$ scheme by leveraging the KD-tree structure. More specifically, we convert the GNN problem to the NN problem for the centroid and then leverage AV-net protocol and BD-conference key agreement to compute the centroid. Furthermore, the KD-tree structure and ASPE algorithm are introduced to construct the efficiency-enhanced SecGNN$^+$ scheme. Note that the solution is compatible with other tree structures (e.g., R-tree and Quad-tree). Finally, we present the performance evaluation to show the high efficiency of our proposed schemes.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by National Natural Science Foundation of China (no. 61960206014) and National Crypto Development Foundation (no. MMJJ20180110).

References

[1] J. Zhou, Z. Cao, Q. Zhan, X. Dong, and K. Ren, "LPPA: lightweight privacy-preserving authentication from efficient multi-key secure outsourced computation for location-based services in vanets," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 420–434, 2020.

[2] Z. Hu, S. Liu, and K. Chen, "Privacy-preserving location-based services query scheme against quantum attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 5, pp. 972–983, 2020.

[3] F. Farouk, Y. Alkady, and R. Rizk, "Efficient privacy-preserving scheme for location based services in VANET system," *IEEE Access*, vol. 8, pp. 60101–60116, 2020.

[4] H. Hacigümüs, S. Mehrotra, and B. R. Iyer, "Providing database as a service," in *Proceedings of the 18th International*

Conference on Data Engineering, pp. 29–38, IEEE Computer Society, San Jose, CA, USA, March 2002.

[5] T. Hashem, L. Kulik, and R. Zhang, "Privacy preserving group nearest neighbor queries," in *Proceedings of theEDBT 2010, 13th International Conference on Extending Database Technology*, pp. 489–500, Lausanne, Switzerland, March 2010.

[6] M. Ashouri-Talouki, A. Baranni.-Dastjerdi, and A. Aydin Selçuk, "The cloaked-centroid protocol: location privacy protection for a group of users of location-based services," *Knowledge and Information Systems*, vol. 45, no. 3, pp. 589–615, 2015.

[7] Y. Wu, Ke Wang, Z. Zhang, W. Lin, H. Chen, and C. Li, "Privacy preserving group nearest neighbor search," in *Proceedings of the 21st International Conference on Extending Database Technology (EDTB)*, pp. 277–288, Vienna, Austria, March 2018.

[8] Y. Wu, Ke Wang, R. Guo et al., "Enhanced privacy preserving group nearest neighbor search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 39, p. 1, 2019.

[9] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of paillier's probabilistic public-key system," in *Proceedings of the Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC*, pp. 119–136, Cheju Island, Korea, February 2001.

[10] B. Bamba, L. Liu, P. . Pesti, and T. Wang, "Supporting anonymous location queries in mobile environments with privacygrid," in *Proceedings of the 17th International Conference on World Wide Web*, pp. 237–246, WWW 2008, Beijing, China, January 2008.

[11] B. Wang, Y. Hou, and M. Li, "Practical and secure nearest neighbor search on encrypted large-scale data," in *Proceedings of the IEEE International Conference on Computer Communications, INFOCOM*, pp. 1–9, San Francisco, CA, USA, July 2016.

[12] J. Guo and J. Sun, "Secure and efficient nearest neighbor query for an outsourced database," *IEEE Access*, vol. 8, pp. 83754–83764, 2020.

[13] Y. Huang and R. Vishwanathan, "Privacy preserving group nearest neighbour queries in location-based services using cryptographic techniques," in *Proceedings of the Global Communications Conference, GLOBECOM 2010*, pp. 6–10, Miami, FL, USA, December 2010.

[14] M. Ashouritalouki, A. Baranni-Dastjerdi, and A. Aydin Selçuk, "Preserving location privacy for a group of users," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 21, pp. 1857–1870, 2013.

[15] W. K Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pp. 139–152, Providence, RI, USA, July 2009.

[16] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *Proceedings of the 20th International Conference on Data Engineering (ICDE)*, pp. 301–312, Boston, MA, USA, April 2004.

[17] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui, "Aggregate nearest neighbor queries in spatial databases," *ACM Transactions on Database Systems*, vol. 30, no. 2, pp. 529–576, 2005.

[18] M. Ashouri-Talouki, A. Baraani-Dastjerdi, and A. AydınSelçukb, "A cryptographic approach for group location privacy," *Computer Communications*, vol. 35, no. 12, pp. 1527–1533, 2012.

[19] A. K. M. Mustafizur Rahman Khan, T. Hashem, E. Tanin, and L. Kulik, "Location oblivious privacy protection for group nearest neighbor queries," in *Procedings of the Geographic Information Science - 8th International Conference*, pp. 301–317, GIScience, Vienna, Austria, September 2014.

[20] P. Pascal and D. Pointcheval, "Efficient public-key cryptosystems provably secure against active adversaries," in *Advances in Cryptology - ASIACRYPT '99, International Conference on the Theory and Applications of Cryptology and Information Security, Singapore, November 14-18, 1999, Proceedings, Volume 1716 of Lecture Notes in Computer Science*, K.-Y. Lam, E. Okamoto, and C. Xing, Eds., pp. 165–179, Springer, New York, NY, USA, 1999.

[21] S. Azizi, M. Ashouri-Talouki, and M. Hamid, "Efficient privacy-preserving group-nearest-neighbor queries with the presence of active adversaries," *Wireless Networks*, vol. 25, no. 8, pp. 4799–4814, 2019.

[22] H. Feng and P. Zielinski, "A 2-round anonymous veto protocol," in *Proceedings of the Security Protocols, 14th International Workshop*, pp. 202–211, Cambridge, UK, March 2006.

[23] H. Feng and P. Zielinski, "The power of anonymous veto in public discussion," *Transaction Computer Science*, vol. 4, pp. 41–52, 2009.

[24] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system (extended abstract)," in *Proceedings of the Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques*, pp. 275–286, Perugia, Italy, May 1994.

[25] C. Boyd, A. Mathuria, and S. Douglas, "Protocols for authentication and key establishment," in *Information Security and Cryptography*, Springer, New York, NY, USA, 2nd edition, 2020.

[26] H. Zhu, F. Liu, and H. Li, "Efficient and privacy-preserving polygons spatial query framework for location-based services," *IEEE Internet Things J*, vol. 4, no. 2, pp. 536–545, 2017.

[27] M. Zeng, K. Zhang, J. Chen, and H. Qian, "P3GQ: a practical privacy-preserving generic location-based services query scheme," *Pervasive and Mobile Computing*, vol. 51, pp. 56–72, 2018.

[28] G. Xu, H. Li, Y. Dai, K. Yang, and X. Lin, "Enabling efficient and geometric range query with access control over encrypted spatial data," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 870–885, 2018.

[29] W. Yang, Y. Geng, Li Lu, X. Xie, and L. Huang, "Achieving secure and dynamic range queries over encrypted cloud data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 98, 2020.

[30] S. C. Pohlig and M. E. Hellman, "An improved algorithm for computing logarithms over gf(p) and its cryptographic significance," *IEEE Transactions on Information Theory*, vol. 24, no. 1, pp. 106–110, 1978.