Privacy-preserving k NN query processing algorithms via secure two-party computation over encrypted database in cloud computing

Hyeong-Jin Kim¹ · Hyunjo Lee¹ · Yong-Ki Kim² · Jae-Woo Chang¹

Accepted: 21 December 2021 / Published online: 17 January 2022
© The Author(s) 2022

Abstract

Since studies on privacy-preserving database outsourcing have been spotlighted in a cloud computing, databases need to be encrypted before being outsourced to the cloud. Therefore, a couple of privacy-preserving k NN query processing algorithms have been proposed over the encrypted database. However, the existing algorithms are either insecure or inefficient. Therefore, in this paper we propose a privacy-preserving k NN query processing algorithm via secure two-party computation on the encrypted database. Our algorithm preserves both data privacy and query privacy while hiding data access patterns. For this, we propose efficient and secure protocols based on Yao's garbled circuit. To achieve a high degree of efficiency in query processing, we also propose a parallel k NN query processing algorithm using encrypted random value pool. Through our performance analysis, we verify that our proposed algorithms outperform the existing ones in terms of a query processing cost.

Keywords Secure protocol · Privacy-preserving k NN query processing algorithm · Encrypted database · Database outsourcing · Cloud computing

✉ Jae-Woo Chang
jwchang@jbnu.ac.kr

Hyeong-Jin Kim
yeon_hui4@jbnu.ac.kr

Hyunjo Lee
o2near@jbnu.ac.kr

Yong-Ki Kim
braves0815@gmail.com

¹ Department of Computer Engineering, Chonbuk National University, Room 7401, 7th Engineering Building, Baekje-daero, Deokjin-gu, Jeonju-si, Jeollabuk-do 54896, Republic of Korea

² Department of IT Convergence System, Vision College of Jeonju, 235, Chun-jam-ro, Wansan-gu, Jeonju-si, Jeollabuk-do 55069, Republic of Korea

1 Introduction

Research on preserving data privacy in outsourced databases has been spotlighted with the development of a cloud computing. Since a data owner (DO) outsources his/her databases and allows a cloud to manage them, the DO can reduce the database management cost by using the cloud's resources with flexibility [1–3]. The cloud not only maintains the databases, but also provides an authorized user (AU) with querying services on the outsourced databases.

However, because the data are private assets of the DO and may include sensitive information such as financial records, they should be protected against adversaries including a cloud server. Therefore, the databases of the DO should be encrypted before being outsourced to the cloud. In addition, a user's query should be protected from the adversaries because the query may contain the private information of the user [4–10]. Therefore, a vital challenge in the cloud computing is to protect both data privacy and query privacy among the data owner, the users, and the cloud. However, during query processing, the cloud can derive sensitive information about the actual data items and users by observing data access patterns even if the data and the query are encrypted [11–17]. In addition, it is very challenging to process a query on the encrypted data without having to decrypt it.

Meanwhile, a k nearest neighbor (k NN) query, one of the most typical query types, has been widely used as a baseline technique in many fields, such as data mining and location-based services. The k NN query finds k neighbors that are closest to a given query. However, a k NN result is closely related to the interest and preference of a user. Therefore, researches on secure k NN query processing algorithms (S k NN) that preserve both the data privacy and the query privacy have been proposed [18–24]. However, the existing algorithms in [18, 19] are insecure because they are vulnerable to chosen- and known-plaintext attacks. In addition, the DO should be heavily involved in the query processing [19–21]. Furthermore, the algorithms in [18–21] do not protect data access patterns from the cloud. The algorithms in [22, 24] guarantee the confidentiality of both the outsourced databases and a user's query while hiding data access patterns. However, they suffer from high query processing cost.

To solve the problems, in the paper we propose a privacy-preserving k NN query processing algorithm via secure two-party computation on the encrypted database. Our algorithm preserves both data privacy and query privacy while hiding data access patterns. For this, we propose efficient and secure protocols based on Yao's garbled circuit [25] and a data packing technique. To enhance the performance of our k NN query processing algorithm, we also propose a parallel k NN query processing algorithm using improved secure protocols based on encrypted random value pool. To verify the security of our algorithm, we provide the formal security proofs of our privacy-preserving k NN query processing algorithms. Through the performance analysis, we verify that our proposed algorithms outperform the existing ones for both a syntactic dataset and a real dataset. Our contributions can be summarized as follows:

- We present a framework for outsourcing both encrypted databases and encrypted indexes.
- We propose new secure protocols (e.g., ESSED, GSCMP, GSPE) in order to preserve data privacy and query privacy while hiding data access patterns.
- We propose an encrypted random value pool to minimize the computational cost of secure protocols.
- We propose a new privacy-preserving parallel k NN query processing algorithm which can support efficient query processing.
- We also present an extensive experimental evaluation of our algorithms with various parameter settings.

The rest of the paper is organized as follows: Section 2 introduces background and related work. Section 3 presents system architecture and secure protocols. Section 4 proposes our privacy-preserving k NN query processing algorithm. Section 5 proposes our parallel k NN query processing algorithm. Section 6 shows the security proof of our privacy-preserving k NN algorithms under semi-honest model. Section 7 presents the performance analysis of our k NN query processing algorithms. Finally, Sect. 8 concludes this paper.

2 Background and related work

2.1 Background

Importance of hiding data access patterns The data access pattern is one of the most important factors for privacy preservation in cloud computing. If an attacker possesses the order of data accesses or their frequency, he/she can infer the original data by using data access patterns. Therefore, hiding data access patterns is as important as encrypting data. First, in location-based service (LBS), one of the well-known queries is to find a nearby point of interest (POI) with a current user's location. For data protection, POI data are indexed, encrypted, and outsourced using a spatial index structure. For query protection, a user's location is encrypted and used for query processing. Because the query and POI data are encrypted, the exact location is not exposed to an attacker. However, by observing accesses to an index structure, an attacker can obtain data access patterns. By using data access patterns, the attacker can know where a query issuer is located and when he/she is in a specific area. As a result, if a user continuously issues queries while moving, an attacker can obtain his/her personal information, such as his/her moving trajectory and preference.

Second, in a healthcare service, a data mining technique for classifying patients based on their health information and symptom is widely used. For classifying patients, the service finds the most similar disease by getting accesses to the previously generated disease classification table. Because the patient's health information is encrypted, an attacker cannot obtain sensitive information. However, an attacker can acquire data access patterns by repeatedly accessing the disease classification table with fake patients. As a result, by using the data access pattern, an attacker

can infer what kind of disease an actual patient has when the patient information is given. Therefore, hiding data access patterns is very essential for privacy preservation in cloud computing.

Paillier cryptosystem The Paillier cryptosystem [26] is an additive homomorphic and probabilistic asymmetric encryption scheme for public key cryptography. The public key pk for encryption is given by (N, g) , where N is a product of two large prime numbers p and q , and g is circular value in $Z_{N^2}^*$. Here, $Z_{N^2}^*$ denotes an integer domain ranged from zero to N^2 . The secret key sk for decryption is given by (p, q) . Let $E(\cdot)$ denote the encryption function and $D(\cdot)$ denote the decryption function. The Paillier cryptosystem has the following properties.

- (1) *Homomorphic addition* The product of two ciphertexts $E(m_1)$ and $E(m_2)$ results in the encryption of the sum c and m_2 (Eq. 1).

$$E(m_1 + m_2) = E(m_1) \times E(m_2) \bmod N^2 \quad (1)$$

- (2) *Homomorphic multiplication* The m_2 th power of ciphertext $E(m_1)$ results in the encryption of the product of m_1 and m_2 (Eq. 2).

$$E(m_1 \times m_2) = E(m_1)^{m_2} \bmod N^2 \quad (2)$$

- (3) *Semantic security* Encrypting the same plaintexts with the same public key results in distinct ciphertexts (Eq. 3).

$$m_1 = m_2 \not\Rightarrow E(m_1) = E(m_2) \quad (3)$$

Therefore, an adversary cannot infer any information about the plaintexts.

Yao's garbled circuit Yao's garbled circuits [25] allow two parties holding inputs x and y , respectively, to evaluate a function $f(x, y)$ without leaking any information about the inputs beyond what is implied by the function output. One party generates an encrypted version of a circuit to compute f . The other party obviously evaluates the output of the circuit without learning any intermediate values. Therefore, the Yao's garbled circuit provides a high security level. Another benefit of using the Yao's garbled circuit is that it can provide high efficiency if a function can be realized with a reasonably small circuit.

2.2 Related work

The typical k NN query processing schemes on encrypted databases are as follows. Wong et al. [18] processed a k NN query by devising an encryption scheme that supports distance comparison on the encrypted data. However, the scheme is vulnerable to chosen-plaintext attacks [27, 28] and cannot hide the data access pattern to the cloud. Yiu et al. [19] proposed a k NN query processing algorithm using the

R-tree index [29] encrypted by AES [30]. However, the scheme has a drawback that the most of the computation is performed at the user side rather than the cloud. In addition, data access pattern is not preserved as the user hierarchically requests the required R-tree nodes to the cloud. Hu et al. [20] proposed a k NN query processing algorithm by using the provably secure privacy homomorphism encryption method. However, the user is in charge of index traversal during the query processing. In addition, the scheme is known to be vulnerable to chosen-plaintext attacks and leaks the data access patterns. Zhu et al. [21] proposed a k NN query processing scheme by considering untrusted users. Because a user does not hold an encryption key, a data owner should encrypt the query. In addition, the cloud can know the identifiers of the query result that implies the leakage of the data access pattern.

Elmehdwi et al. [22] proposed the $SkNN_m$ scheme over the encrypted database. To the best of our knowledge, this is the first work that guarantees both the data privacy and the query privacy while hiding the data access pattern [14] at the same time. In addition, a data owner and a user do not participate in the query processing. However, the query processing cost of this scheme is extremely high because the scheme considers all of the encrypted data and makes use of secure protocols that take the encrypted binary representation of the data as inputs. Zhou et al. [23] proposed an asymmetric scalar-product-preserving encryption (ASPE) scheme based on Wong et al.'s work [18]. By using random asymmetric splitting with additional artificial dimensions, the scheme can resist known-plaintext attacks [28, 31]. In this scheme, the query issuers are fully trusted and the decryption key is partially revealed to the query issuers. However, the scheme cannot hide the data access pattern. Most recently, Kim et al. [24] proposed a k NN query processing scheme ($SkNN_p$) by using an encrypted index. The algorithm guarantees the confidentiality of both the data and the user query while hiding data access patterns. By filtering unnecessary data using a secure index mechanism, the algorithm provides better performance than $SkNN_m$. However, the algorithm still requires a high computation cost because it uses secure protocols that take the encrypted binary representation of the data as inputs.

3 System architecture and secure protocols

3.1 System architecture

The typical types of adversaries are semi-honest and malicious [32]. In this paper, we consider the cloud as insider adversaries who have more authorities than outsider attackers. In the semi-honest adversarial model, the cloud correctly follows the given protocol, but may try to obtain the additional information not allowed to them. In the malicious adversarial model, the cloud can deviate from the protocol specification. However, protocols against malicious adversaries are inefficient. Nonetheless, protocols associated with semi-honest adversaries are practical and can be used to design protocols against malicious adversaries. Therefore, according to earlier work [22, 24], we also adopt the semi-honest adversarial model. A secure protocol under the semi-honest adversarial model can be defined as follows.

Definition 1 *Secure protocol* Let $\prod_i(\pi)$ be an execution image of the protocol π at the C_i side and let a_i and b_i be the input and the output of the protocol π , respectively. Then, π is secure if $\prod_i(\pi)$ is computationally indistinguishable from the simulated image $\prod_i^s(\pi)$.

The system consists of four components: data owner (DO), authorized user (AU), and two clouds (C_A and C_B). The DO owns the original database (T) of n records [33–35]. A record t_i ($1 \leq i \leq n$) consists of m attributes, where m means the number of data dimensions, and the j th attribute value of t_i is denoted as $t_{i,j}$ ($1 \leq j \leq m$). The DO partitions T by using the kd -tree structure [36, 37] to provide the indexing on T . The reason why we use the kd -tree structure as an index structure is to hide data access patterns. Using a space filling curve (e.g., Hilbert curve) for partitioning data items into blocks (nodes) can guarantee data locality, but it cannot guarantee that data items are evenly distributed over blocks. As a result, an attacker may infer a specific block based on the number of data items stored in it, whereas using the kd -tree structure for partitioning data items into blocks makes an attacker unable to distinguish a block from each other. This is because data items are evenly distributed into blocks in the kd -tree structure even if data items are skewed. Meanwhile, while traversing the kd -tree structure in a hierarchical way, an attacker can know which block is relevant to the query, which results in the leakage of data access patterns. To tackle the problem, our algorithm accesses only the leaf nodes of the kd -tree during the query processing step, rather than traversing the tree structure in a hierarchical way.

Henceforth, a node refers to a leaf node. Let h denote the level of the constructed kd -tree and F be the fanout of each leaf node. A node is denoted by $node_z$ ($1 \leq z \leq 2^{h-1}$) where 2^{h-1} is the total number of leaf nodes. The region information of $node_z$ is represented as the lower bound $lb_{z,j}$ and the upper bound $ub_{z,j}$ ($1 \leq z \leq 2^{h-1}$, $1 \leq j \leq m$). Each node stores the identifiers (id) of data located inside the node region. To preserve the data privacy, the DO encrypts T attribute-wise using the public key (pk) of the Paillier cryptosystem [26] before outsourcing the database. Therefore, the DO generates $E(t_{i,j})$ for $1 \leq i \leq n$ and $1 \leq j \leq m$ by encrypting $t_{i,j}$. The DO also encrypts the region information of all kd -tree nodes to support efficient query processing. Specifically, lb and ub of each node are encrypted attribute-wise such that $E(lb_{z,j})$ and $E(ub_{z,j})$ are generated with $1 \leq z \leq 2^{h-1}$ and $1 \leq j \leq m$. We assume that C_A and C_B are non-colluding and semi-honest (or honest-but-curious) clouds. Thus, they correctly perform the given protocols and do not exchange unpermitted data. However, they may try to obtain additional information from the intermediate data while executing their own protocols. This assumption has been used in the related problem domains (for example, in [38]) even though it is not new as mentioned in the earlier works [22, 24, 39].

In this paper, we consider privacy-preserving kNN query processing which retrieves k nearest data items that are closest to the given query. To support kNN query processing over the encrypted database, a secure multi-party computation (SMC) is required for privacy-preserving kNN query processing algorithm [40]. A secure multi-party computation can be defined as follows.

Definition 2 *Secure multi-party computation* a given number of participants, $p_1, p_2, \dots, p_n (n \geq 2)$, each have private data, respectively, $d_1, d_2, \dots, d_n$. Participants want to compute the value of a public function on the private data: $F(d_1, d_2, \dots, d_n)$ while keeping their own inputs secret.

According to Definition 2, the proposed algorithm uses two clouds (e.g., C_A and C_B) because at least two parties are required for secure computation. Existing studies, such as Elmehdwi et al.'s work [22], Zhou et al.'s work [23], Wong et al.'s work [18] and Kim et al.'s work [24], also use two-party computation to support privacy-preserving k NN query processing. Thus, we do not consider a single-party computation model because the single-party computation model is vulnerable against semi-honest adversaries.

The DO outsources both the encrypted database and its encrypted index to the C_A with pk , while the DO sends sk to the C_B . The encrypted index includes the region information of each node in ciphertext and the ids of data located inside the node in plaintext. The DO also sends pk to AUs to enable them to encrypt a k NN query. When requesting a query, an AU first generates $E(q_j)$ by encrypting a query q attribute-wise for $1 \leq j \leq m$. C_A and C_B cooperatively process the query and return a query result to the AU without data leakage.

As an example, assume that an AU has sixteen data in two-dimensional space (x -axis and y -axis) as depicted in Fig. 1. The data items are partitioned into four nodes for a kd -tree: node₁, node₂, node₃, and node₄. To clarify the relationship between data items and nodes, we suppose that there is no data item on the boundary of a node. To outsource the database, the DO encrypts each data item and its region information attribute-wise. The i th data item d_i is represented as $\langle x_i, y_i \rangle$ in

Fig. 1 Data items and kd -tree in two-dimensional space

two-dimensional space. Therefore, the d_i can be encrypted to $\langle E(x_i), E(y_i) \rangle$ by using the Paillier cryptosystem. For example, $d_1 = \langle 2, 1 \rangle$ is encrypted as $E(d_1) = \langle E(2), E(1) \rangle$ and the encrypted index is shown in Fig. 1.

3.2 Enhanced secure protocols

Our k NN query processing algorithm is constructed using several secure protocols. We adopt four secure protocols from the literatures [22, 24, 39], such as secure multiplication (SM), secure bit-not (SBN), CoMPare-S (CMP-S), and secure minimum from set of n values (SMIN_n). All of the protocols except the SBN protocol use the SMC technique between C_A and C_B , while the SBN protocol can be solely executed at the C_A side. In addition, we propose three new secure protocols: enhanced secure squared Euclidean distance (ESSED), garbled circuit-based secure compare (GSCMP), and garbled circuit-based secure point enclosure (GSPE). For both GSCMP and GSPE, we use Yao's garbled circuits [25] that allow two parties holding inputs x and y , respectively, to evaluate a function $f(x, y)$ without leaking any information about the inputs beyond what is implied by the function output. One party generates an encrypted version of a circuit to compute f . The other party obviously evaluates the output of the circuit without learning any intermediate values. Therefore, the Yao's garbled circuit supports high security level. Another advantage of the Yao's garbled circuit is to provide high efficiency if a function can be realized with a reasonably small circuit [39]. Because our protocols do not take the encrypted binary representation of the data as inputs, contrary to the existing protocols [22, 24], they can provide a low computation cost.

ESSED protocol: Suppose that there are two m -dimensional vectors $\vec{X} = \{x_1, x_2, x_3, \dots, x_m\}$ and $\vec{Y} = \{y_1, y_2, y_3, \dots, y_m\}$. The goal of the ESSED (enhanced secure squared Euclidean distance) protocol is to securely compute $E\left(\left|\vec{X} - \vec{Y}\right|^2\right)$, where $\left|\vec{X} - \vec{Y}\right|$ denotes the Euclidean distance between $\vec{X}$ and $\vec{Y}$. Note that $\left|\vec{X} - \vec{Y}\right|^2 = \sum_{i=1}^m (x_i - y_i)^2$.

We utilize a data packing technique to enhance the efficiency of a secure protocol. Specifically, we pack λ number of σ -bit data instances to generate a packed value. The overall procedure of ESSED is as follows. First, C_A generates random numbers r_j for $1 \leq j \leq m$ and packs them to obtain R using Eq. (4).

$$R = \sum_{j=1}^m r_j \times 2^{\sigma(m-j)} \quad (4)$$

Then, C_A generates $E(R)$ by encrypting R . Second, C_A calculates $E(x_j - y_j)$ attribute-wise and packs these results to obtain $E(v)$ using Eq. (5). Then, C_A computes $E(v) = E(v) \times E(R)$ and sends $E(v)$ to C_B .

$$E(v) = \prod_{j=1}^m E(x_j - y_j)^{2^{\sigma(m-j)}} \quad (5)$$

Third, C_B acquires $[x_1 - y_1 + r_1 | \dots | x_m - y_m + r_m]$ by decrypting $E(v)$. C_B obtains $x_j - y_j + r_j$ for $1 \leq j \leq m$ by unpacking v through $v \times 2^{-\sigma(m-j)}$. C_B also calculates $(x_j - y_j + r_j)^2$ attribute-wise and stores their sum into d . C_B encrypts d and sends $E(d)$ to C_A . Finally, C_A obtains $E(|\vec{X} - \vec{Y}|^2)$ by eliminating randomized values using Eq. (6).

$$E(|\vec{X} - \vec{Y}|^2) = E(d) \times \prod_{j=1}^m (E(x_j - y_j)^{-2r_j} \times E(r_j^2)^{-1}) \quad (6)$$

Our ESSED outperforms the existing distance computation protocol, i.e., data packing-based secure squared Euclidean distance (DPSSSED) [39]. Table 1 shows the difference between the existing DPSSSED and our ESSED in terms of the number of encryptions. Our ESSED requires only one encryption on the C_B side, while the existing DPSSSED requires m times of encryptions. Therefore, our ESSED requires a total of two encryptions, whereas the existing DPSSSED requires a total of $m + 1$ encryptions. In addition, our ESSED calculates the randomized distance in plaintext on the C_B side, while the existing DPSSSED computes the sum of the squared Euclidean distances among all attributes over ciphertext on the C_A side. Therefore, the number of computations on encrypted data in our ESSED can be reduced greatly, compared with the existing DPSSSED.

GSCMP protocol: Suppose that $E(u)$ and $E(v)$ are the encrypted values of u and v , respectively. When $E(u)$ and $E(v)$ are given as inputs, GSCMP (garbled circuit-based secure CoMPare) protocol returns the result as follows.

$$\text{GSCMP}(E(u), E(v)) = \begin{cases} 1, & \text{if } u \leq v \\ 0, & \text{otherwise} \end{cases}$$

The main difference between GSCMP and CMP-S is that GSCMP receives encrypted data as inputs, while CMP-S receives the randomized plaintext. Furthermore, in the case of CMP-S, plaintext is returned as a result, whereas GSCMP encrypts the result of CMP-S and sends it to C_A . Through this, GSCMP can protect the data access patterns. The overall procedure of the GSCMP is as follows.

First, C_A generates two random numbers ru and rv and encrypts them. C_A computes $E(m_1) = E(u)^2 \times E(ru)$ and $E(m_2) = E(v)^2 \times E(1) \times E(rv)$. For the given input value u and v , Yao's garbled circuit returns one if $u < v$; returns zero if $u > v$; returns

Table 1 Comparison between the DPSSSED and our ESSED in terms of the number of encryptions

	DPSSSED protocol	ESSED protocol
# of encryption in C_A side	1 time	1 time
# of encryption in C_B side	m times (m is the data dimension)	1 time
A total number of encryptions	$m + 1$ times	2 times

a random value if $u=v$. To avoid returning a random value, our GSCMP protocol calculates $u'=2 \times u$ and $v'=2 \times v + 1$ in Eqs. (7) and (8) while maintaining the values of inequality. For example, when $u=v=3$, GSCMP calculates $u'=6$ and $v'=7$. Therefore, our GSCMP protocol avoids returning a random value when $u=v$.

$$E(m_1) = E(u)^2 \times E(ru) = E(2 \times u + ru) \quad (7)$$

$$E(m_2) = E(v)^2 \times E(1) \times E(rv) = E(2 \times v + rv + 1) \quad (8)$$

Second, C_A randomly chooses one functionality between $F_0: u \geq v$ and $F_1: u < v$. The selected functionality is oblivious to C_B . Then, C_A sends data to C_B , depending on the selected functionality. If $F_0: u \geq v$ is selected, C_A sends $\langle E(m_2), E(m_1) \rangle$ to C_B . If $F_1: u < v$ is selected, C_A sends $\langle E(m_1), E(m_2) \rangle$ to C_B .

Third, C_B obtains $\langle m_2, m_1 \rangle$ by decrypting $\langle E(m_2), E(m_1) \rangle$ if $F_0: u \geq v$ is selected. If $F_1: u < v$ is selected, C_B obtains $\langle m_1, m_2 \rangle$ by decrypting $\langle E(m_1), E(m_2) \rangle$.

Fourthly, C_A generates a garbled circuit consisting of two ADD circuits and one CMP circuit. Here, ADD circuit takes two integers u and v as input, and outputs $u+v$ while CMP circuit takes two integers u and v as input, and outputs 1 if $u < v$, zero otherwise. If $F_0: u \geq v$ is selected, C_A puts $-rv$ and $-ru$ into the first and second ADD gates, respectively. If $F_1: u < v$ is selected, C_A puts $-ru$ and $-rv$ into the first and second ADD gates.

Fifthly, if $F_0: u \geq v$ is selected, C_B puts m_2 and m_1 into the first and second ADD gates, respectively. If $F_1: u < v$ is selected, C_B puts m_1 and m_2 into the first and second ADD gates.

Sixthly, the first ADD gate adds two input values and puts the output $result_1$ into CMP gate. Similarly, the second ADD gate puts the output $result_2$ into CMP gate. Seventhly, CMP gate outputs $\alpha=1$ if $result_1 < result_2$ is true, $\alpha=0$ otherwise. The output of the CMP is returned to the C_B . Then, C_B encrypts α and sends $E(\alpha)$ to C_A . Since $E(\alpha)$ is an encrypted value, C_A cannot identify the data received from the C_B . If C_A receives α from C_B , C_A can know which data is relevant to the query, which can lead to the exposure of the data access patterns. Therefore, it is necessary that C_B sends $E(\alpha)$ rather than α to C_A .

Finally, when the selected functionality is $F_0: u \geq v$, C_A computes $E(\alpha) = \text{SBN}(E(\alpha))$ and returns the final $E(\alpha)$. If $E(\alpha)$ is $E(1)$, u is less than or equal to v .

GSPE protocol: Suppose that $E(p)$ is an encrypted value of a point p and $E(\text{range})$ is a set of the encrypted values containing the $E(\text{range}.lb_j)$ and the $E(\text{range}.ub_j)$ for $1 \leq j \leq m$ (m is the data dimension). When $E(p)$ and $E(\text{range})$ are given as inputs, GSPE (garbled circuit-based secure point enclosure) protocol returns the result as follows.

$$\text{GSPE}(E(p), E(\text{range})) = \begin{cases} 1, & \text{if } \text{range}.lb \leq p \leq \text{range}.ub \\ 0, & \text{otherwise} \end{cases}$$

If $E(p_j) \leq E(\text{range}.ub_j)$ and $E(p_j) \geq E(\text{range}.lb_j)$, the p is inside the *range*. To securely compare between a point and a range, the GSPE protocol needs to add random values for all data dimension. However, as the number of data dimensions increases, the number of data encryptions is increased. The GSPE protocol reduces the number of data encryptions by using a packing technique that transforms the m -dimensional data into one packed value.

The overall procedure of the GSPE is shown in Algorithm 1. First, C_A generates two random numbers ra_j and rb_j for $1 \leq j \leq 2m$ (line 1–2). C_A obtains PA and PB by initially packing ra_j and rb_j , respectively, by using Eq. (9) for $1 \leq j \leq 2m$ (line 3).

$$PA = \sum_{j=1}^{2m} ra_j \times 2^{\sigma(2m-j)}, \quad PB = \sum_{j=1}^{2m} rb_j \times 2^{\sigma(2m-j)} \quad (9)$$

Here, σ means the maximum length in bit to represent a data. Then, C_A generates $E(PA)$ and $E(PB)$ by encrypting PA and PB (line 4). Second, C_A computes $E(\mu_j) = E(p_j)^2$ and $E(\omega_j) = E(\text{range}.lb_j)^2$ for $1 \leq j \leq m$. C_A also computes $E(\delta_j) = E(p_j)^2 \times E(1)$ and $E(\tau_j) = E(\text{range}.ub_j)^2 \times E(1)$ for $1 \leq j \leq m$ (line 5–8). Third, C_A randomly chooses one of two functions, $F_0: u \geq v$ and $F_1: v > u$. Then, C_A performs encrypted data packing for $E(\mu_j)$, $E(\tau_j)$, $E(\omega_j)$ and $E(\delta_j)$, by using homomorphic multiplication and addition (Eqs. 1 and 2) depending on the selected function (lines 8–18). C_A sends $E(PA)$ and $E(PB)$ to C_B (line 19). Fourthly, C_B obtains PA and PB by decrypting $E(PA)$ and $E(PB)$ (line 20). C_B stores $x_j \leftarrow PA \times 2^{-\sigma(2m-j)}$ for $1 \leq j \leq 2m$ by unpacking PA, while C_B stores $y_j \leftarrow PB \times 2^{-\sigma(2m-j)}$ for $1 \leq j \leq 2m$ by unpacking PB (line 21–23). Here, x_j (or y_j) is one of the μ_j , τ_j , ω_j , and δ_j .

Fifthly, C_A generates two add gates and one compare gate (line 24). C_A puts $-ra_j$ and $-rb_j$ into the first and the second add gates, respectively, for $1 \leq j \leq 2m$ (line 25–26). C_B puts x_j and y_j into the first and the second add gate, respectively, for $1 \leq j \leq 2m$ (line 27). When $-ra_j$, $-rb_j$, x_j and y_j are given to the compare gate, the result of compare gate, $\alpha' = \langle \alpha'_1, \alpha'_2, \dots, \alpha'_{2m'} \rangle$, is returned to C_B (line 28–29). Sixthly, C_B encrypts α' and sends $E(\alpha')$ to C_A (line 30). Seventhly, C_A performs $E(\alpha'_j) = \text{SBN}(E(\alpha'_j))$ for $1 \leq j \leq 2m$ when the selected function is $F_0: u \geq v$ (line 32–33). Then, C_A computes $E(\text{result}) = \text{SMR}(E(\text{result}), E(\alpha'_j))$ for $1 \leq j \leq 2m$ where the initial value of $E(\text{result})$ is $E(1)$ (line 31, line 34). When all of $E(\alpha'_j)$ for $1 \leq j \leq 2m$ are $E(1)$, $E(\text{result})$ remains $E(1)$. Finally, GSPE returns $E(\text{result})$ (line 35).

Algorithm 1. GSPE(Garbled Circuit based Secure Point Enclosure) protocolInput : $E(range)$, $E(p)$ Output : $E(1)$ when $range$ includes point p , $E(0)$ otherwise C_A :

01. for $1 \leq j \leq 2m$
02. pick random value ra_j and rb_j
03. $PA \leftarrow \sum_{j=1}^{2m} ra_j \times 2^{\sigma(2m-j)}$, $PB \leftarrow \sum_{j=1}^{2m} rb_j \times 2^{\sigma(2m-j)}$
04. generate $E(PA)$ and $E(PB)$ by encrypting PA and PB , respectively
05. for $1 \leq j \leq m$
06. $E(u_j) \leftarrow E(p_j)^2$, $E(\tau_j) \leftarrow E(range.ub_j)^2 \times E(1)$
07. $E(\delta_j) \leftarrow E(p_j)^2 \times E(1)$, $E(\omega_j) \leftarrow E(range.lb_j)^2$
08. Randomly choose F_0 or F_1
09. for $1 \leq j \leq m$
10. if $F_0 : u \geq v$ is chosen then
11. $E(PA) \leftarrow E(PA) \times E(u_j)^{2^{\sigma(2m-j)}}$, $E(PB) \leftarrow E(PB) \times E(\tau_j)^{2^{\sigma(2m-j)}}$
12. else
13. $E(PA) \leftarrow E(PA) \times E(\tau_j)^{2^{\sigma(2m-j)}}$, $E(PB) \leftarrow E(PB) \times E(u_j)^{2^{\sigma(2m-j)}}$
14. for $m+1 \leq j \leq 2m$
15. if $F_0 : u \geq v$ is chosen then
16. $E(PA) \leftarrow E(PA) \times E(\omega_j)^{2^{\sigma(2m-j)}}$, $E(PB) \leftarrow E(PB) \times E(\delta_j)^{2^{\sigma(2m-j)}}$
17. else
18. $E(PA) \leftarrow E(PA) \times E(\delta_j)^{2^{\sigma(2m-j)}}$, $E(PB) \leftarrow E(PB) \times E(\omega_j)^{2^{\sigma(2m-j)}}$
19. send $E(PA)$ and $E(PB)$ to C_B

 C_B :

20. $PA = D(E(PA))$; $PB = D(E(PB))$
21. for $1 \leq j \leq 2m$
22. $x_j = PA \times 2^{-\sigma(2m-j)}$
23. $y_j = PB \times 2^{-\sigma(2m-j)}$

 C_A and C_B :

24. generate two add gate and one compare gate in Yao's garbled circuit
25. for $1 \leq j \leq 2m$
26. C_A put $-ra_j$ and $-rb_j$ into 1st ADD gate and 2nd ADD gate, respectively
27. C_B put x_j and y_j into 1st ADD gate and 2nd ADD gate, respectively
28. C_B put $-ra_j$, $-rb_j$, x_j and y_j into COMPARE gate
29. C_B assign the result of COMPARE gate to α_j'

 C_B :

30. send $E(\alpha')$ to C_A // $\alpha' = \langle \alpha_1', \alpha_2', \dots, \alpha_{2m}' \rangle$

 C_A :

31. $E(result) \leftarrow E(1)$
32. for $1 \leq j \leq 2m$
33. if $F_0 : u \geq v$ then $SBN(E(\alpha_j'))$
34. $E(result) \leftarrow SMR(E(result), E(\alpha_j'))$
35. return $E(result)$

End Algorithm

3.3 Secure protocols using encrypted random value pool

While processing a query in our secure system, C_B decrypts the received ciphertext. Thus, we need to prevent C_B from extracting meaningful information while executing secure protocols. For this, C_A generates a random value r from Z_N and encrypts r by using Paillier cryptosystem. Then, C_A adds the encrypted random value $E(r)$ to the encrypted plaintext $E(m)$ by computing $E(m+r) = E(m) \times E(r)$. Because $m \pm r$ is independent from m , C_B cannot obtain a meaningful information with decryption. However, in Paillier cryptosystem, the process of adding a random value to the ciphertext leads performance degradation because both encryption and decryption operations require higher computational cost than other encrypted operations. Therefore, we propose an encrypted random value pool to reduce the amount of computational cost for ciphertext generation. First, in a preprocessing phase, we generate random ciphertexts and store them to an encrypted random value pool. Second, while processing a query in C_A , a random ciphertext is selected from the encrypted random value pool whenever the secure protocol is called. Therefore, C_A not only prevents C_B from extracting meaningful information while processing a secure protocol, but also reduces a cost of generating encrypted random values. We use the encrypted random value pool to the SM protocol [22] and our GSCMP protocol. In SM and GSCMP, C_A generates two encrypted random values before sending the ciphertext to C_B . According to Table 2, we can reduce the number of encryption operations to 67% by using the encrypted random value pool.

Secure multiplication protocol using encrypted random value pool (SMR): Suppose that $E(u)$ and $E(v)$ are the encrypted values of u and v , respectively. When $E(u)$ and $E(v)$ are given as inputs, SMR protocol returns the result as follows.

$$\text{SMR}(E(u), E(v)) = E(u \times v)$$

SMR protocol is shown in Algorithm 2. When two encrypted values $E(u)$ and $E(v)$ are given as inputs, C_A selects two random values $E(r_a)$ and $E(r_b)$ from the encrypted random value pool (line 1). The rest of the SMR protocol is the same as the previous SM protocol (line 2–6).

Table 2 A comparison of amount in secure protocols

	SM protocol	Secure compare protocol
Elmehdwi et al. [22]	$3 \times E$	–
Kim et al. [24]	$3 \times E$	$\log_2 D \times E$
Proposed	$1 \times E$	$1 \times E$

E Encryption, D Data domain

Algorithm 2. SMR protocol	
Input	: $E(u), E(v)$
Output	: $E(u \times v)$
C_A : 01. pick random value $E(r_a)$ and $E(r_b)$ in random pool 02. $E(u') \leftarrow E(u) \times E(r_a)$; $E(v') \leftarrow E(v) \times E(r_b)$ 03. send $E(u'), E(v')$ to C_B C_B : 04. $h \leftarrow D(E(u')) \times D(E(v')) \bmod N$ $\therefore h = u \times v + u \times r_b v + v \times r_a u + r_a \times r_b$ 05. send $E(h)$ to C_A C_A : 06. $E(u \times v) \leftarrow E(h) \times E(u)^{N-r_b} \times E(v)^{N-r_a} \times E(r_a \times r_b)^{N-1}$ End Algorithm	

Garbled secure compare protocol using encrypted random value pool (GSCR): Suppose that $E(u)$ and $E(v)$ are the encrypted values of u and v , respectively. When $E(u)$ and $E(v)$ are given as inputs, GSCR protocol returns the result which is the same as our GSCMP as follows.

$$\text{GSCR}(E(u), E(v)) = \begin{cases} 1, & \text{if } u \leq v \\ 0, & \text{otherwise} \end{cases}$$

The difference between GSCR and GSCMP is to select a random ciphertext from the encrypted random value pool, instead of generating an encrypted arbitrary value.

4 KNN query processing algorithm

In this section, we present our k NN query processing algorithm (SkNN_G) that uses Yao's garbled circuit [25]. The algorithm consists of three phases: encrypted kd -tree search, k NN retrieval, and k NN result refinement.

4.1 Candidate node search phase

In the encrypted kd -tree search phase, the C_A securely extracts all of the data from a node containing a query point while hiding the data access patterns. The procedure of the encrypted kd -tree search phase is shown in Algorithm 3. First, C_A securely finds nodes that include a query by executing $E(\alpha_z) = \text{GSPE}(E(q), E(\text{node}_z))$ for $1 \leq z \leq \#_of_node$ where $\#_of_node$ means the total number of kd -tree leaf nodes (line 1–2). The result of GSPE for all nodes is stored in $E(\alpha) = \{E(\alpha_1), E(\alpha_2), \dots, E(\alpha_{\#_of_node})\}$. By utilizing GSPE, our k NN query processing algorithm can obtain better performance than the existing algorithms [22, 24] because we can avoid operations related to the SBD protocol that causes high computation overhead. Then, we perform the 8–24 lines of the index search algorithm in [24]. Second, C_A generates

$E(\alpha')$ by shuffling $E(\alpha)$ using a random shuffling function π and sends $E(\alpha')$ to C_B (line 3–4).

Algorithm 3. Encrypted kd -tree Search Phase	
Input	: $\langle E(data_1), E(data_2), \dots, E(data_n) \mid n = \#_of_data \rangle$, $\langle E(node_1), E(node_2), \dots, E(node_z) \mid z = \#_of_node \rangle$, $E(q)$
Output	: $\langle E(cand_1), E(cand_2), \dots, E(cand_{cnt}) \rangle$ // all candidates inside nodes being related to a query
C_A :	
01.	for $1 \leq i \leq num_{node}$ // $num_{node} = 2^{h-1}$ (h : level of the kd -tree)
02.	$E(\alpha_i) \leftarrow GSPE(E(q), E(node_i))$
03.	$E(\alpha') \leftarrow \pi(E(\alpha))$ // shuffle the order of array $E(\alpha)$
04.	send $E(\alpha')$ to C_B
C_B :	
05.	$\alpha' \leftarrow D(E(\alpha'))$
06.	$c \leftarrow$ the number of '1' in α'
07.	generate c number of Node Group(NG)
08.	for each NG
09.	assign a node with $\alpha'=1$ and $\#_of_node/c-1$ nodes with $\alpha'=0$
10.	$NG' \leftarrow$ shuffle the ids of nodes
11.	send NG' to C_A
C_A :	
12.	$cnt \leftarrow 0$ // cnt means the total number of data extracted from the encrypted leaf nodes of kd -tree
13.	$NG^* \leftarrow$ shuffle node ids using π^{-1} for each NG'
14.	for each NG^*
15.	for $1 \leq i \leq num$ // $num = \#$ of nodes in the selected NG^*
16.	$z = id$ of i^{th} nodes of NG^*
17.	for $1 \leq s \leq FanOut$ // $FanOut$: the number of data in each nodes
18.	for $1 \leq j \leq m$
19.	$E(t'_{i,j}) \leftarrow SM(node_z.data_{s,j}, E(\alpha_z))$
20.	$E(cant_{cnt+s,j}) \leftarrow E(cant_{cnt+s,j}) \times E(t'_{i,j})$
21.	$cnt = cnt + FanOut$
22.	send $E(cand)$
End Algorithm	

Third, C_B obtains α' by decrypting $E(\alpha')$ and counts the number of $\alpha'=1$ and stores it into c (line 5–6). Here, c means the number of nodes that the query is related to. Fourthly, C_B creates c number of node groups (NG) (line 7–11). C_B assigns to each NG a node with $\alpha'=1$ and $\#_of_node/c-1$ nodes with $\alpha'=0$. Then, C_B obtains NG' by randomly shuffling the *ids* of nodes in each NG and sends NG' to C_A . Fifthly, C_A obtains NG^* by shuffling the *ids* of nodes using π^{-1} in each NG' (line 12–13). Finally, C_A accesses one data in each node for each NG^* and performs $E(t'_{i,j}) = SMR(E(node_z.data_{s,j}), E(\alpha_z))$ where $1 \leq s \leq FanOut$ and $1 \leq j \leq m$ (line 14–22). Here, $E(\alpha_z)$ is the result of $GSPE$ corresponding to $node_z$. If a node has the less number of data than $FanOut$, it performs SMR by using $E(max)$, instead of using $E(node_z.data_{s,j})$. Here, max is the largest value in the domain. When C_A

accesses one datum from every node in a NG^* , C_A performs $E(cand_{cnt,j}) = \prod_{i=1}^{num} E(t'_{i,j})$ where num means the total number of nodes in the selected NG^* .

As a result, data items in the node related to the query are securely extracted without revealing the data access patterns [5, 14] because the searched nodes are not revealed. By repeating these steps, all of the data in the node are safely stored into the $E(cand_{i,j})$ for $1 \leq i \leq cnt$ and $1 \leq j \leq m$ where cnt means the total number of data extracted during the index search. Figure 2 shows an example of the candidate node search phase. The example uses the data items and their kd -tree structure that are represented in Fig. 1. First, C_A performs the GSPE between $E(q)$ and $E(Node_i.Range)$ for $1 \leq i \leq cnt$. C_A stores the GSPE result into $E(\alpha)$ and sends it to C_B . For example, in Fig. 2, C_A performs GSPE($\{<E(0), E(0)>, <E(4), E(5)>\}$, $<E(6), E(1)>$) between $E(Node_i.Range)$ and $E(q)$, and stores the GSPE

Fig. 2 The example of candidate node search phase

result, i.e., $E(0)$, into $E(\alpha_1)$ for Node_1 . Second, C_A shuffles the sequence of $\{ \langle \text{Node}_1, E(\alpha_1) \rangle, \langle \text{Node}_2, E(\alpha_2) \rangle, \dots, \langle \text{Node}_n, E(\alpha_n) \rangle \}$ and changes the shuffled node ids into new ids so as to hide the original node ids from C_B . To obtain the original sequence of node ids, C_A records their shuffled sequence. For example, in Fig. 2, the original sequence of $\{ \langle \text{Node}_1, E(0) \rangle, \langle \text{Node}_2, E(1) \rangle, \langle \text{Node}_3, E(0) \rangle, \langle \text{Node}_4, E(0) \rangle \}$ is shuffled to $\{ \langle \text{Node}_4, E(0) \rangle, \langle \text{Node}_1, E(0) \rangle, \langle \text{Node}_2, E(1) \rangle, \langle \text{Node}_3, E(0) \rangle \}$. Then, C_A changes Node_4 , Node_1 , Node_2 , and Node_3 into PN_1 , PN_2 , PN_3 , and PN_4 , respectively. As a result, the shuffled sequence is $\{ \langle \text{PN}_1, E(0) \rangle, \langle \text{PN}_2, E(0) \rangle, \langle \text{PN}_3, E(1) \rangle, \langle \text{PN}_4, E(0) \rangle \}$. Then, C_A sends the shuffled sequence to C_B . Third, C_B receives the shuffled sequence and decrypts it. For example, in Fig. 2, C_B receives $\{ \langle \text{PN}_1, E(0) \rangle, \langle \text{PN}_2, E(0) \rangle, \langle \text{PN}_3, E(1) \rangle, \langle \text{PN}_4, E(0) \rangle \}$ and obtains $\{ \langle \text{PN}_1, 0 \rangle, \langle \text{PN}_2, 0 \rangle, \langle \text{PN}_3, 1 \rangle, \langle \text{PN}_4, 0 \rangle \}$ by decrypting it. To generate node groups, i.e., NG s, C_B counts how many 1s are in the sequence. Each NG has one seed node whose α'_p equals 1 where $1 \leq p \leq \#_of_node$. Therefore, there exists the same number of NG s as the number of 1s in the sequence. Nodes whose α'_p equals 0, where $1 \leq p \leq \#_of_node$, are evenly assigned to the generated NG s. And C_B sends the generated NG s to C_A . For example, C_B counts 1s in the sequence of $\{ \langle \text{PN}_1, 0 \rangle, \langle \text{PN}_2, 0 \rangle, \langle \text{PN}_3, 1 \rangle, \langle \text{PN}_4, 0 \rangle \}$ and generates NG_1 with the seed PN_3 . The nodes $\langle \text{PN}_1, 0 \rangle, \langle \text{PN}_2, 0 \rangle$, and $\langle \text{PN}_4, 0 \rangle$ are assigned into NG_1 . C_B sends $NG_1 = \{ \text{PN}_3, \text{PN}_1, \text{PN}_2, \text{PN}_4 \}$ to C_A . Fourth, C_A obtains the original node ids from the received NG s by using the shuffled sequence of node ids. For example, in Fig. 2, C_A obtains $NG_1' = \{ \text{Node}_2, \text{Node}_4, \text{Node}_1, \text{Node}_3 \}$ as the original node ids by using both the received $NG_1 = \{ \text{PN}_3, \text{PN}_1, \text{PN}_2, \text{PN}_4 \}$ and the shuffled sequence of node ids. Fifth, C_A performs SMR protocol between $E(\alpha)$ and the encrypted data item in a node group, and makes a candidate set by summarizing all the result of the SMR. In Fig. 2, C_A performs $\text{SMR}(E(0), E(\text{Node}_1.\text{Data}))$, $\text{SMR}(E(1), E(\text{Node}_2.\text{Data}))$, $\text{SMR}(E(0), E(\text{Node}_3.\text{Data}))$, and $\text{SMR}(E(0), E(\text{Node}_4.\text{Data}))$. The result of the SMR are $E(0)$, $E(d_5)$, $E(0)$, $E(0)$ and C_A summarizes them and stores $E(d_5)$ into the candidate set. Sixth, for each NG , the SMR protocol and the summarization of results are performed as many times as the number of data items of each node. For example, in Fig. 2, C_A performs summarization of results four times. As a result, C_A obtains $\{ E(d_5), E(d_6), E(d_7), E(d_8) \}$ as the candidate set.

4.2 kNN retrieval phase

In the k NN retrieval phase, we retrieve the k nearest neighbors from the given query by partially utilizing the SkNN_m scheme [22]. However, we only consider $E(\text{cand}_i)$ for $1 \leq i \leq \text{cnt}$, which are extracted in the index search phase, whereas the SkNN_m considers all the encrypted data. In addition, we use our efficient secure protocols that require relatively low computation costs, instead of using the existing expensive secure protocols including SBD (secure bit decomposition) [22, 24]. The overall procedure of k NN retrieval algorithm is as follows: First, the algorithm calculates the distance between the encrypted data items and the encrypted query without data and query decryption. Second, the algorithm finds the minimum distance ($\text{dist}_{\min}$) among the calculated distances. It cannot know which data item has $\text{dist}_{\min}$ due to

the semantic security of the Paillier cryptosystem. Third, to obtain the encrypted data item with $\text{dist}_{\min}$, the algorithm performs the subtraction of $\text{dist}_{\min}$ with the calculated distance and the data item with $\text{dist}_{\min}$ has $E(\text{zero})$ as the result of subtraction. Here, $E(\text{zero})$ is the only value which is not changed by the homomorphic multiplication of the Paillier cryptosystem. Therefore, the algorithm can distinguish the nearest neighbor from others, while an attacker cannot determine which data item has the minimum distance. Fourth, in order to hide the original data items from the attacker, the algorithm performs the homomorphic multiplication of the result of subtraction with a random value. It also shuffles the sequence of the result of multiplication in order to hide data access pattern. Finally, by using our secure protocols, the algorithm finds the nearest neighbor and repeats the above process until k nearest neighbors are found.

The pseudocode of the k NN retrieval phase is shown in Algorithm 4. First, using ESSED, C_A securely calculates the squared Euclidean distances $E(\text{dist}_i)$ between a query and $E(\text{cand}_i)$ for $1 \leq i \leq \text{cnt}$ (line 1–2). Second, C_A performs SMIN_n to find the minimum value $E(\text{dist}_{\min})$ among $E(d_i)$ for $1 \leq i \leq \text{cnt}$ (line 3–4). Third, C_A calculates $E(\tau_i) = E(\text{dist}_{\min}) \times E(\text{dist}_i)^{N-1}$, for $1 \leq i \leq \text{cnt}$. C_A computes $E(\tau'_i) = E(\tau_i)^{r_i}$. C_A obtains $E(\beta)$ by shuffling $E(\tau')$ using a random shuffling function π and sends $E(\beta)$ to C_B (lines 4–9). For example, $E(\tau')$ is calculated as $\{E(0), E(-r)\}$ where r means a random number. The $E(0)$ corresponds to the $E(\text{dist}_{\min})$. Assuming that π shuffles data in reverse way, C_A sends $E(\beta) = \{E(-r), E(0)\}$ to C_B . Fourthly, after decrypting $E(\beta)$, C_B sets $E(U_i) = E(1)$ if $E(\beta_i) = 0$, otherwise $E(U_i) = E(0)$. C_B sends $E(U)$ to C_A (line 10–13). Fifthly, C_A obtains $E(V)$ by shuffling $E(U)$ using π^{-1} . Then, C_A performs SMR protocol by using $E(V_i)$ and $E(\text{cand}_{i,j})$ to obtain $E(V'_{i,j})$ (lines 14–17). Sixthly, by computing $E(n_{s,j}) = \prod_{i=1}^{\text{cnt}} E(V'_{i,j})$ for $1 \leq j \leq m$, C_A can securely extract the datum corresponding to the $E(\text{dist}_{\min})$ (line 18). Finally, for preventing the selected result from being selected in the later phase, C_A securely updates the distance of the selected result as $E(\text{max})$ by computing Eq. (10) (lines 19–22).

$$E(d_i) = \text{SMR}(E(V_i), E(\text{max})) \times \text{SMR}(\text{SBN}(E(V_i)), E(d_i)) \quad (10)$$

Because only the selected result has $E(V_i) = E(1)$, the $E(\text{dist}_i)$ corresponding to the datum selected in current round becomes $E(\text{max})$, while the other values remain the same. This procedure is repeated for k rounds to find the k NN result.

Algorithm 4. k NN Retrieval Phase	
Input	: $E(q)$, $\langle E(cand_1), E(cand_2), \dots, E(cand_{cnt}) \rangle$, k // k means the number of nearest neighbours for k NN
Output	: $\langle E(nn_1), E(nn_2), \dots, E(nn_k) \rangle$ // temporary k NN results
C_A :	
01.	for $1 \leq i \leq cnt$
02.	$E(dist_i) \leftarrow \text{ESSED}(E(q), E(cand_i))$
03.	for $1 \leq s \leq k$
04.	$dist_{min} \leftarrow \text{SMIN}_n(dist_1, \dots, dist_{cnt})$
05.	for $1 \leq i \leq cnt$
06.	$E(\tau_i) \leftarrow E(dist_{min}) \times E(dist_i)^{N-1}$
07.	$E(\tau'_i) \leftarrow E(\tau_i)^{r_i}$
08.	$E(\beta) \leftarrow \pi(\tau')$
09.	send $E(\beta)$ to C_B
C_B :	
10.	for $1 \leq i \leq cnt$
11.	if $D(\beta_i) = 0$ then $E(U_i) \leftarrow E(1)$
12.	else $E(U_i) \leftarrow E(0)$
13.	send $E(U)$ to C_A
C_A :	
14.	$E(V) \leftarrow \pi^{-1}(U)$
15.	for $1 \leq i \leq cnt$
16.	for $1 \leq j \leq m$
17.	$E(V'_{i,j}) \leftarrow \text{SMR}(E(V_i), E(cand_{i,j}))$
18.	$E(nn_{s,j}) \leftarrow \prod_{i=1}^{cnt} E(V'_{i,j})$
19.	for $1 \leq i \leq cnt$
20.	if $s < k$
21.	$E(dist_i) \leftarrow \text{SMR}(E(V_i), E(max)) \times \text{SMR}(\text{SBN}(E(V_i)), E(dist_i))$
22.	return $E(nn)$
End Algorithm	

Figure 3 shows an example of k NN retrieval phase. The example uses the data items and their kd-tree structure, as shown in Fig. 1. For simplicity and clarity, the shuffling function π is omitted. First, C_A calculates the distance by using the ESSED and stores the ESSED result into $E(dist_i)$ for $1 \leq i \leq cnt$ (Ⓐ). In Fig. 3, C_A performs $\text{ESSED}(E(d_5), \langle E(6), E(1) \rangle)$ for $E(d_5)$ and $E(q)$, and stores the ESSED result, i.e., $E(2)$, into $E(dist_5)$. Second, the minimum value is calculated by using the SMIN_n and stored the SMIN_n result into $E(dist_{min})$ (Ⓑ). In Fig. 3, C_A performs $\text{SMIN}_n(E(2), E(9), E(8), E(18))$ for obtaining the minimum distance among $E(dist_5)$, $E(dist_6)$, $E(dist_7)$, and $E(dist_8)$, and stores the SMIN_n result, i.e., $E(2)$, into $E(dist_{min})$. Third, in order to obtain the encrypted data item with the minimum distance to the given query, C_A performs $E(dist_{min} - dist_i)$ and stores the result into $E(\tau_i)$ for $1 \leq i \leq cnt$ (Ⓒ). When $dist_{min}$ is the same as $dist_i$, $E(0)$ is stored into $E(\tau_i)$. In Fig. 3, for $E(dist_5)$, C_A performs $E(2-2)$ and stores the result, i.e., $E(0)$, into $E(\tau_5)$. Fourth, in order to protect the value of $E(\tau_i)$ for $1 \leq i \leq cnt$, C_A performs the homomorphic multiplication of $E(\tau_i)$ by a random value (Ⓓ). In Fig. 3, for $E(\tau_6)$, C_A performs $E(-7 \times 3)$ with a random value = 3 and stores the result, i.e., $E(-21)$, into $E(\tau'_7)$. Fifth, C_B returns $E(1)$ as $E(V_i)$ if $E(\tau'_i)$ is $E(0)$ for $1 \leq i \leq cnt$; otherwise, it returns $E(0)$ as $E(V_i)$ (Ⓔ). Sixth, C_A obtains the nearest neighbor by performing the SMR between $E(d_i)$

Fig. 3 The example of kNN retrieval phase

and $E(V_i)$ for $1 \leq i \leq \text{cnt}$ and merging the result of the $\text{SMR}(\textcircled{6}-\textcircled{7})$. In Fig. 3, C_A performs $\text{SMR}(E(1), E(5))$, $\text{SMR}(E(0), E(6))$, $\text{SMR}(E(0), E(8))$ and $\text{SMR}(E(0), E(9))$ for x -axis and $\text{SMR}(E(1), E(2))$, $\text{SMR}(E(0), E(4))$, $\text{SMR}(E(0), E(3))$ and $\text{SMR}(E(0), E(4))$ for y -axis. C_A merges $E(5)$, $E(0)$, $E(0)$, and $E(0)$ for x -axis while merging $E(2)$, $E(0)$, $E(0)$, and $E(0)$ for y -axis. As a result, C_A obtains $\langle E(5), E(2) \rangle$ as the nearest neighbor. Seventh, by using Eq. (5), the C_A sets the distance of the found nearest neighbor to the maximum value so that C_A can avoid finding the found nearest neighbor again in the next round($\textcircled{8}$). In Fig. 3, C_A performs $\text{SMR}(E(1), E(\text{max})) \times \text{SMR}(E(0), E(2))$ for $E(d_5)$ and stores $E(\text{max})$ into $E(\text{dist}_5)$. Finally, C_A repeats the above process until k nearest neighbors are found($\textcircled{2}-\textcircled{7}$).

4.3 kNN result refinement phase

As mentioned in [24], the result of kNN retrieval phase may not be accurate because candidates are extracted from only one leaf node in index search phase. Therefore, the kNN result refinement is necessary to confirm the correctness of

the current query result. Specifically, assuming that the squared Euclidean distance between the k th closest result $E(nn_k)$ and the query is $dist_k$, the neighboring kd-tree nodes need to be searched to acquire data with the shorter distance than $dist_k$. For this reason, we use the concept of shortest point (sp) defined in [24]. The sp is a point in a given node whose distance is closest to a given point p as compared with the other points in the node. To find the sp in each node, we use the following properties described in [24]. (i) If both the lower bound (lb) and the upper bound (ub) of the node are lesser than the p , the ub becomes the sp of the region. (ii) If both the lb and the ub of the region are greater than p , the lb becomes the sp of the region. (iii) If p is between the lb and the ub of the region, p is the sp of the region. Since this property can be applied to each dimension, our k NN result refinement phase partially utilizes that of the existing algorithm [19, 21]. However, to reduce the computation cost, we do not use the existing expensive protocols, such as SBD, SSSED, SCMP, and SPE [22, 24].

Algorithm 5. k NN result refinement Phase	
Input	$\langle E(data_1), E(data_2), \dots, E(data_n) \mid n = \#_of_data \rangle,$ $\langle E(node_1), E(node_2), \dots, E(node_z) \mid z = \#_of_node \rangle, E(q), k$
Output	$\langle E(nn_1), E(nn_2), \dots, E(nn_k) \rangle$ // final k NN results
C_A : 01. $E(dist_k) = \text{ESSED}(E(q), E(nn_k))$ 02. for $\leq z \leq num_node$ 03. for $\leq j \leq m$ 04. $E(\psi_1) \leftarrow \text{GSCR}(E(q_j), E(node_z.lb_j))$ 05. $E(\psi_2) \leftarrow \text{GSCR}(E(q_j), E(node_z.ub_j))$ 06. $E(\psi_3) \leftarrow \text{SBXOR}(E(\psi_1), E(\psi_2))$ 07. $E(temp) \leftarrow \text{SMR}(E(\psi_1), E(node_z.lb_j))$ 08. $E(temp) \leftarrow E(temp) \times \text{SMR}(\text{SBN}(E(\psi_1)), E(node_z.ub_j))$ 09. $E(temp) \leftarrow \text{SMR}(E(temp), \text{SBN}(E(\psi_3)))$ 10. $E(sp_{z,j}) \leftarrow E(temp) \times \text{SMR}(E(\psi_3), E(q_j))$ 11. $E(spdist_z) \leftarrow \text{ESSED}(E(q), E(sp_z))$ 12. $E(temp) \leftarrow \text{SMR}(E(\alpha_z), E(max))$ 13. $E(spdist_z) \leftarrow E(temp) \times \text{SMR}(\text{SBN}(E(\alpha_z)), E(spdist_z))$ 14. $E(\alpha_z) \leftarrow \text{GSCR}(E(spdist_z), E(dist_k))$ 15. $E(cand') \leftarrow$ perform 3 ~ 22 lines of Algorithm 3 16. $E(cand') \leftarrow$ append $E(cand')$ to $E(nn)$ 17. $E(nn) \leftarrow$ perform Algorithm 4 18. for $1 \leq i \leq k$ 19. for $1 \leq j \leq m$ 20. pick up the random value $r_{i,j}$ 21. $E(\gamma_{i,j}) \leftarrow E(nn_{i,j}) \times E(r_{i,j})$ 22. send $E(\gamma)$ to C_B and $r_{i,j}$ to AU C_B : 23. for $1 \leq i \leq k$ 24. for $1 \leq j \leq m$ 25. $\gamma_{i,j} \leftarrow D(E(\gamma_{i,j}))$ 26. send γ to AU AU : 27. for $1 \leq i \leq k$ 28. for $1 \leq j \leq m$ 29. $nn_{i,j} \leftarrow \gamma_{i,j} - r_{i,j}$ End Algorithm	

The procedure of the k NN result refinement phase is shown in Algorithm 5. First, C_A computes $E(dist_k) = \text{ESSED}(E(q), E(nn_k))$ to obtain the squared Euclidean distance between the query and the k th closest result, which is returned from the k NN retrieval phase (line 1). Second, for each node, C_A performs GSCMP by using $E(q_j)$ and $E(node_z.lb_j)$ for $1 \leq z \leq num_{node}$ and $1 \leq j \leq m$ and stores the result in $E(\psi_1)$. C_A also performs GSCMP by using $E(q_j)$ and $E(node_z.ub_j)$ for $1 \leq z \leq num_{node}$ and $1 \leq j \leq m$ and stores the result into $E(\psi_2)$ (lines 2–5). When the value of $E(q_j)$ is equal to or less than the $E(lb_j)$ ($E(ub_j)$), the $E(\psi_1)$ ($E(\psi_2)$) has the value of $E(1)$. Then, C_A obtains $E(\psi_3)$ by carrying out $E(\psi_1) \times E(\psi_2) \times \text{SMR}(E(\psi_1), E(\psi_2))^{N-2}$ so as to acquire the result of bit-xor operation between $E(\psi_1)$ and $E(\psi_2)$ (line 6). Note that “-2” is equivalent to “ $N-2$ ” under Z_N . Third, C_A securely obtains the shortest point of each node, that is, $E(sp_{z,j})$, by computing $\text{SMR}(E(\psi_3), E(q_j) \times \text{SMR}(\text{SBN}(E(\psi_3)), f(E(lb_{z,j}), E(ub_{z,j}))))$ for $1 \leq z \leq num_{node}$ and $1 \leq j \leq m$. Here, $f(E(lb_j), E(ub_j))$ is obtained by computing $\text{SMR}(E(\psi_1), E(lb_{z,j})) \times \text{SMR}(\text{SBN}(E(\psi_1)), E(ub_{z,j}))$ for $1 \leq z \leq num_{node}$ and $1 \leq j \leq m$ (lines 7–10). Fourthly, C_A calculates $E(spdist_z)$, the squared Euclidean distances between the query and $E(sp_z)$ for $1 \leq z \leq num_{node}$ by using ESSED. In addition, C_A securely updates the $E(spdist_z)$ of the nodes, which are retrieved in index search phase, into $E(max)$ by computing $E(spdist_z) = \text{SMR}(E(\alpha_z), E(max)) \times \text{SMR}(\text{SBN}(E(\alpha_z)), E(spdist_z))$ (lines 11–13). Here, $E(\alpha_z)$ is the value returned by GSPE in index search phase. Then, C_A performs $E(\alpha_z) = \text{GSCMP}(E(spdist_z), E(dist_k))$ (line 14). If the $E(spdist_z)$ is less than $E(dist_k)$, the corresponding $node_z$ is assigned $E(\alpha) = E(1)$. The nodes with $E(\alpha) = E(1)$ need to be retrieved for k NN result refinement. The number of nodes to expand increases according to how many $E(\alpha_z)$ becomes $E(1)$. If the number of ‘1’ is c in $E(\alpha_z)$, c number of node groups are created in the C_B and C_A extract the data of each node group. Therefore, the number of cnt becomes $c \times \text{fanout}$.

Because the $E(spdist_z)$ of nodes being retrieved in the index search phase are $E(max)$, they are safely pruned. Fifthly, C_A securely extracts the data stored in the nodes with $E(\alpha) = E(1)$ by performing the index search using $E(\alpha)$ and appends them to $E(nn)$ (line 15–16). Then, C_A executes k NN search phase based on $E(nn)$ to obtain the final k NN result $E(result_i)$ for $1 \leq i \leq k$ (line 17). Therefore, the final result becomes $\{E(nn_1), E(nn_5)\}$ because the squared Euclidean distance of $E(nn_5)$ is $E(4)$. Sixthly, C_A returns the decrypted result to AU in cooperation with C_B to reduce the computation overhead at the AU side. To do this, C_A computes $E(\gamma_{i,j}) = E(result_i) \times E(r_{i,j})$ for $1 \leq i \leq k$ and $1 \leq j \leq m$ by using a random value $r_{i,j}$. Then, C_A sends $E(\gamma_{i,j})$ to C_B and $r_{i,j}$ to AU (lines 18–22). Then, C_B decrypts $E(\gamma_{i,j})$ and sends the decrypted value to AU (lines 23–26). Finally, AU obtains the actual k NN result by computing $\gamma_{i,j} \cdot r_{i,j}$ in plaintext (lines 27–29).

5 Parallel k NN query processing algorithm

5.1 Parallel encrypted kd -tree search phase

In the parallel encrypted kd -tree search phase, C_A simultaneously extracts all of the data from a node containing a query point. To expand encrypted kd -tree

search phase to parallel environment, we use a thread pool which stores tasks in a queue so that threads can process tasks in parallel. The procedure of the parallel encrypted kd -tree search phase is shown in Algorithm 6. First, C_A generates a queue-based thread pool (line 1). If a thread in the thread pool is available, it can process a task in FIFO manner. Second, C_A pushes the task, i.e., $GSPE(E(q), E(node_i))$, to the thread pool for $1 \leq i \leq \#_of_node$. A result of $GSPE$ protocol is stored in $E(\alpha) = \{E(\alpha_1), E(\alpha_2), \dots, E(\alpha_{\#_of_node})\}$ (lines 2–3). Third, C_A generates $E(\alpha')$ by shuffling $E(\alpha)$ using a random shuffling function π and sends $E(\alpha')$ to C_B (lines 4–5). Fourthly, C_B performs the same procedure in Algorithm 3 (line 6). Fifthly, C_A obtains NG^* by shuffling the ids of nodes using π^{-1} in each NG' (line 7). Finally, C_A accesses one datum in each node for each NG^* and pushes both $E(t'_{i,j}) = SMR(E(node_z.data_{s,j}), E(\alpha_z))$ and $E(cant_{cnt+s,j}) = E(cant_{cnt+s,j}) \times E(t'_{i,j})$ to the thread pool, where $1 \leq s \leq FanOut$ and $1 \leq j \leq m$ (lines 8–18).

Algorithm 6. Parallel Encrypted kd -tree Search Phase	
Input	$\langle E(data_1), E(data_2), \dots, E(data_n) \mid n = \#_of_data \rangle, \langle E(node_1), E(node_2), \dots, E(node_z) \mid z = \#_of_node \rangle, E(q)$
Output	$\langle E(cand_1), E(cand_2), \dots, E(cand_{cnt}) \rangle$ // all candidates inside nodes being related to a query
C_A :	
01.	generate <i>thread_pool</i> // create a thread and wait in the pool until a task is given
02.	for $1 \leq i \leq num_{node}$ // $num_{node} = 2^{h-1}$ (h : level of the kd -tree)
03.	call <i>thread_pool_push</i> ($GSPE(E(q), E(node_i)), E(\alpha_i)$) // assign the task to an available thread and return the result
04.	$E(\alpha') \leftarrow \pi(E(\alpha))$
05.	send $E(\alpha')$ to C_B
C_B :	
06.	perform line 5~11 in Algorithm 3
C_A :	
07.	perform line 12~13 in Algorithm 3
08.	for each NG^*
09.	for $1 \leq i \leq num$ (num : # of nodes in the selected NG^*)
10.	$z = \text{id of } i^{th} \text{ nodes of } NG^*$
11.	for $1 \leq s \leq FanOut$
12.	assign task T_s to threads in the thread pool
13.	for each T_s
14.	for $1 \leq j \leq m$
15.	$E(t'_{i,j}) \leftarrow SMR(node_z.data_{s,j}, E(\alpha_z))$
16.	$E(cant_{cnt+s,j}) \leftarrow E(cant_{cnt+s,j}) \times E(t'_{i,j})$
17.	$cnt = cnt + FanOut$
18.	send $E(cand)$
End Algorithm	

5.2 Parallel kNN retrieval phase

In the parallel k NN retrieval phase, we simultaneously retrieve the k closest data from the query by partially utilizing the $SkNN_m$ scheme [22]. We consider $E(cand_i)$ for $1 \leq i \leq cnt$, which are extracted in the parallel index search phase. The procedure of the parallel k NN retrieval phase is shown in Algorithm 7. First, using ESSED, C_A simultaneously calculates the squared Euclidean distances $E(d_i)$ between a query and $E(cand_i)$ for $1 \leq i \leq cnt$ (lines 1–2). Second, C_A performs $SMIN_n$ to find the minimum value $E(d_{min})$ among $E(d_i)$ for $1 \leq i \leq cnt$ (lines 3–4). Third, C_A simultaneously calculates both $E(\tau_i) = E(d_{min}) \times E(d_i)^{N-1}$ and $E(\tau'_i) = E(\tau_i)^{r_i}$ for $1 \leq i \leq cnt$ (line 5–7). C_A obtains $E(\beta)$ by shuffling $E(\tau')$ using a random shuffling function π and sends $E(\beta)$ to C_B (lines 8–9). Fourthly, after decrypting $E(\beta)$, C_B sets $E(U_i) = E(1)$ if $E(\beta_i) = 0$, otherwise $E(U_i) = E(0)$. C_B sends $E(U)$ to C_A (line 10). Fifthly, C_A obtains $E(V)$ by shuffling $E(U)$ using π^{-1} (line 11). Sixthly, instead of using SM protocol, C_A simultaneously performs SMR protocol with $E(v_i)$ and $E(cand_{i,j})$ to obtain $E(V'_{i,j})$ (lines 12–16). Seventhly, by computing $E(nn_{s,j}) = \prod_{i=1}^{cnt} E(V'_{i,j})$ for $1 \leq j \leq m$, C_A can simultaneously extract the datum corresponding to the $E(d_{min})$ (lines 17–18). Finally, C_A simultaneously updates the distance of the selected result as $E(max)$ by computing Eq. (11) (lines 19–24).

$$E(d_i) = \text{SMR}(E(V_i), E(max)) \times \text{SMR}(\text{SBN}(E(V_i)), E(d_i)) \quad (11)$$

Algorithm 7. Parallel k NN Retrieval Phase	
Input	$E(q), \langle E(cand_1), E(cand_2), \dots, E(cand_{cnt}) \rangle, k$ // k means the number of nearest neighbours for k NN
Output	$\langle E(nn_1), E(nn_2), \dots, E(nn_k) \rangle$ // temporary k NN results
C_A : 01. for $1 \leq i \leq cnt$ 02. call thread_pool_push(ESSED($E(q)$, $E(cand_i)$), $E(dist_i)$) 03. for $1 \leq s \leq k$ 04. $E(dist_{min}) \leftarrow \text{SMIN}_n(E(dist_1), \dots, E(dist_{cnt}))$ 05. for $1 \leq i \leq cnt$ 06. assign task T_i to threads in the thread pool 07. T_i is performed by using line 6~7 in Algorithm 4 08. $E(\beta) \leftarrow \pi(\tau')$ 09. send $E(\beta)$ to C_B C_B : 10. perform line 10~13 in Algorithm 4 C_A : 11. $E(V) \leftarrow \pi^{-1}(U)$ 12. for $1 \leq i \leq cnt$ 13. assign task T_i to threads in the thread pool 14. for each T_i 15. for $1 \leq j \leq m$ 16. $E(V'_{i,j}) \leftarrow \text{SMR}(E(V_i), E(cand_{i,j}))$ 17. for $1 \leq i \leq cnt$ 18. call thread_pool_push($\prod_{i=1}^{cnt} E(V'_{i,j})$, $E(nn_{s,j})$) 19. for $1 \leq i \leq cnt$ 20. assign task T_i to threads in the thread pool 21. for each T_i 22. if $s < k$ 23. $E(dist_i) \leftarrow \text{SMR}(E(V_i), E(max)) \times \text{SMR}(\text{SBN}(E(V_i)), E(dist_i))$ 24. return $E(nn)$ End Algorithm	

5.3 Parallel KNN result refinement phase

In the parallel k NN result refinement phase, C_A simultaneously checks whether results of k NN is correct or not. If not, C_A performs both index search phase and k NN retrieval phase again. The procedure of the parallel k NN result refinement phase is shown in Algorithm 8. First, C_A computes $E(dist_k) = \text{ESSED}(E(q), E(nn_k))$ to obtain the squared Euclidean distance between the query and the k th closest result which is returned from the k NN retrieval phase (line 1). Second, C_A simultaneously finds nodes that is closer than $disk_k$ by using both SMR and GSCR protocols (lines 2–16). Third, C_A performs 15–22 lines of Algorithm 5 (line 17). Fourthly, C_B decrypts $E(\gamma_{i,j})$ and sends the decrypted value to AU (lines 18–21). Finally, AU obtains the actual k NN result by computing $\gamma_{i,j} - r_{i,j}$ in plain-text (lines 22–24).

Algorithm 8. Parallel kNN result refinement Phase	
Input	: $\langle E(data_1), E(data_2), \dots, E(data_n) \mid n = \#_of_data \rangle,$ $\langle E(node_1), E(node_2), \dots, E(node_z) \mid z = \#_of_node \rangle, E(q), k$
Output	: $\langle E(nn_1), E(nn_2), \dots, E(nn_k) \rangle$ // final kNN results
<p>C_A:</p> <ol style="list-style-type: none"> 01. $E(dist_k) = ESSED(E(q), E(nn_k))$ 02. for $\leq z \leq num_{node}$ 03. assign task T_i to threads in the thread pool 04. for each T_i 05. for $1 \leq j \leq m$ 06. $E(\psi_1) \leftarrow GSCR(E(q_j), E(node_z.lb_j))$ 07. $E(\psi_2) \leftarrow GSCR(E(q_j), E(node_z.ub_j))$ 08. $E(\psi_3) \leftarrow SBXOR(E(\psi_1), E(\psi_2))$ 09. $E(tmp) \leftarrow SMR(E(\psi_1), E(node_z.lb_j))$ 10. $E(tmp) \leftarrow E(tmp) \times SMR(SBN(E(\psi_1)), E(node_z.ub_j))$ 11. $E(tmp) \leftarrow SMR(E(tmp), SBN(E(\psi_3)))$ 12. $E(sp_{z,j}) \leftarrow E(tmp) \times SMR(E(\psi_3), E(q_j))$ 13. $E(sp_{dist_z}) \leftarrow ESSED(E(q), E(sp_z))$ 14. $E(tmp) \leftarrow SMR(E(\alpha_z), E(max))$ 15. $E(sp_{dist_z}) \leftarrow E(tmp) \times SMR(SBN(E(\alpha_z)), E(sp_{dist_z}))$ 16. $E(\alpha_z) \leftarrow GSCR(E(sp_{dist_z}), E(dist_k))$ 17. perform 15~22 line in Algorithm 5 <p>C_B:</p> <ol style="list-style-type: none"> 18. for $1 \leq i \leq k$ 19. for $1 \leq j \leq m$ 20. $\gamma_{i,j} \leftarrow D(E(\gamma_{i,j}))$ 21. send γ to AU <p>AU:</p> <ol style="list-style-type: none"> 22. for $1 \leq i \leq k$ 23. for $1 \leq j \leq m$ 24. $nn_{i,j} \leftarrow \gamma_{i,j} - r_{i,j}$ <p>End Algorithm</p>	

6 Security proof under semi-honest model

As mentioned above, the proposed privacy-preserving kNN algorithm is implemented in a semi-honest attack model. Therefore, the security proof of the proposed privacy-preserving kNN algorithm is performed from the three viewpoints of C_A , C_B , and AU (Authorized User), which are the subjects of actions excluding data owners. In addition, the following lemmas are used in our security proof.

Lemma 1 *If a random element r is uniformly distributed on Z_N and independent from any variable $x \in Z_N$, then $r \pm x$ is also uniformly random and independent from x .*

Lemma 2 *The Paillier cryptosystem is semantically secure based on the composite residuosity class problem [26].*

Theorem 1 *The proposed privacy-preserving kNN algorithm is secure from the perspective of C_A under the semi-honest model.*

Proof C_A owns the cryptographic database and cryptographic index. However, since it does not own the decryption key, the encryption database and encryption index are not exposed. Data cannot be inferred even in a frequency-based attack because the same plaintext has different ciphertexts (Lemma 2). In addition, since all values that our secure protocol returns from C_B are encrypted values, no information is exposed from data received from C_B . Even though the query is received from the user, it cannot be inferred because the query is in a cryptographic state. $\square$

Theorem 2 *The proposed privacy-preserving kNN algorithm is secure from the perspective of C_B under the semi-honest model [32].*

Proof C_B decrypts the encrypted text received from C_A . Because C_A hides the original data by adding an arbitrary integer before it is passed to C_B , C_B cannot infer meaningful data from the decrypted plaintext (Lemma 1). $\square$

Theorem 3 *The proposed privacy-preserving kNN algorithm is secure from the perspective of AU under the semi-honest model.*

Proof The AU encrypts his/her query using the public key and sends the encrypted query to C_A . This can protect the user's preferences and personal information. Since the query results received from C_A and C_B do not include the information for the owner's data, it is impossible to infer the original data. $\square$

Theorem 4 *The proposed privacy-preserving kNN algorithm is secure even though c and cnt are exposed to C_A under the semi-honest model.*

Proof C_A can obtain both c and cnt in Algorithms 3 and 5. Here, c is the number of nodes relevant to the query and cnt equals to $c \times fanout$ (i.e., F). c initially equals to one in the candidate search phase and c is changed to be ranged from zero to a total number of leaf nodes in the result refinement phase. However, because the upper and lower bounds of all nodes are encrypted and the node *ids* are hidden through grouping and shuffling, it is impossible to know which node is related to the query. Therefore, even if c and cnt are exposed as plaintext to C_A , an attacker cannot know which nodes are related to the query, thus resulting in no additional information leakage. $\square$

Theorem 5 *The proposed privacy-preserving k NN algorithm is secure even though c is exposed to C_B under the semi-honest model.*

Proof C_B can obtain c in algorithms 3 and 5. Here, c denotes the number of nodes related to the query. Because C_B cannot know the $fanout(F)$ of the kd -tree, cnt is not disclosed to C_B . In addition, because the order of node ids is changed through shuffling, it is impossible to infer which node is related to the query. Therefore, even if c is exposed to C_B as plaintext, an attacker cannot know which nodes are related to the query, thus resulting in no additional information leakage. $\square$

According to Theorems 1, 2, 3, 4, and 5, the original data, an index, and a query are protected through the Paillier encryption system (Lemma 2), and when decrypted, the original data cannot be inferred through arbitrary data (Lemma 1). Through this, we prove that the proposed privacy-preserving k NN algorithm can guarantee data protection, query protection, and query result protection, while hiding data access patterns.

In addition, the proposed parallel k NN algorithm is implemented in a semi-honest attack model and its security proof is performed from the three viewpoints of C_A , C_B , and AU . Because the procedure of the proposed parallel k NN algorithm is the same as that of the proposed privacy-preserving k NN algorithm except for using multiple threads, the proposed parallel k NN algorithm is secure from the perspective of C_A , C_B , and AU under the semi-honest attack model. Therefore, we prove that the proposed parallel k NN algorithm can guarantee data protection, query protection, and query result protection, while hiding data access patterns [4, 5, 14].

7 Performance analysis

In this section, we compare the proposed privacy-preserving k NN algorithm ($SkNN_G$) with the existing algorithms, $SkNN_m$ [22] and $SkNN_I$ [24], which can hide data access patterns. We used the Paillier cryptosystem to encrypt a database for both schemes [22, 24]. We implemented our algorithm and the existing ones using C++. Experiments were performed on a Linux machine with an Intel Xeon E3-1220v3 4-Core 3.10 GHz and 32 GB RAM running Ubuntu 14.04.2. In addition, we compare the proposed parallel algorithm ($SkNN_{PG}$) with the parallel version of $SkNN_m$ ($SkNN_{pm}$) and that of $SkNN_I$ ($SkNN_{PI}$). Experiments for parallel algorithms were performed on a Linux machine with an Intel Xeon CPU E5-2630v4 2.20 GHz and 64 GB RAM running Ubuntu 14.04.2.

We conduct performance analysis using both a syntactic dataset and a real dataset. For a synthetic datasets, we randomly generated 30 k records with six attributes. For a real dataset, we make use of the Chess dataset available at

Table 3 Parameters used in our experiments

Parameter	Description
n	Number of records(data) 28,056 for real data, 5000 to 30,000, default = 30,000 for synthetic data
h	Tree depth of kd-tree (7 for synthetic data and 8 for real data)
k	Number of nearest neighbours for k NN (5, 10, 15, 20, default = 10)
m	Number of attributes in a record

Fig. 4 Performance with varying kd -tree depth for synthetic data

<http://archive.ics.uci.edu/ml/datasets> [41]. It consists of 28,056 records with six attributes. Parameters for our experiments are listed in Table 3. We use 512 bits for encryption key size (K) and set the default values of the required k as 10. The query was used by selecting a random integer from the range of data.

7.1 Performance using a synthetic dataset

Figure 4 shows the performances of both $SkNN_I$ and $SkNN_G$ in terms of the height of kd -tree(h). When the number of data items is fixed, $fanout(F)$ is decreased as h increases because the total number of leaf nodes is calculated by using h , i.e., 2^{h-1} . Therefore, it is important to choose the appropriate height(h) of kd -tree depending on the number of data items. If h is greatly high for the given number of data items, the number of leaf nodes to be searched increases and the cost for accessing leaf node increases in the candidate node search phase. On the contrary, if h is greatly low for the given number of data items, the number of data items to be search increases and the cost for calculating distances between data items and the query is increased in the k NN retrieval phase. Therefore, the existing algorithm ($SkNN_I$) and the proposed algorithm ($SkNN_G$) shows near-optimal performance in case of $h=7$. In particular, the performance of the existing algorithm is greatly affected by the height of the kd -tree(h) because the existing algorithm uses secure protocols based on an encrypted binary array, which requires high computation cost. However,

Fig. 5 Performance with varying n for synthetic data

Fig. 6 Performance with varying k for synthetic data

the proposed algorithm is relatively less affected by h than the existing algorithm because the proposed algorithm uses secure protocol based on garbled circuit, which requires low computation cost. As a result, we set h to 7 in our experiment.

Figures 5 and 6 show the performance of three algorithms in a single machine. With varying n , our $SkNNG$ shows 30.2 and 6.1 times better performance than $SkNN_m$ and $SkNN_I$, respectively. With varying k , our $SkNNG$ shows 33.2 and 4.9 times better performance than $SkNN_m$ and $SkNN_I$, respectively. As a result, our $SkNNG$ outperforms $SkNN_m$ because it can reduce the computation cost by pruning out unnecessary data with the kd -tree, contrary to considering all the data in $SkNN_m$. In addition, our $SkNNG$ outperforms $SkNN_I$ because our algorithm uses efficient secure protocols based on both Yao's garbled circuit and the data packing technique. First, if a function can be realized with a reasonably small circuit, Yao's garbled circuit provides a high degree of efficiency. Because our secure protocols, i.e., GSCMP and GSPE, do not take the encrypted binary representation of the data as inputs,

Fig. 7 Performance with varying # of threads for synthetic data

Fig. 8 Performance with varying n for synthetic data in parallel

contrary to the existing protocols used in [22, 24], our encrypted data is reasonably small. As a result, our $SkNN_G$ can provide a low computation cost by using GSCMP and GSPE. Second, our ESSED protocol requires only one encryption operation by using the data packing technique while other protocol (i.e., DESSED) needs m operations for data encryption. Moreover, ESSED calculates the randomized distance in plaintext, while other protocol computes the sum of the squared Euclidean distances among all attributes over ciphertext. Therefore, our $SkNN_G$ can greatly reduce a computation cost by using ESSED.

Figures 7, 8, and 9 show the performance of three parallel algorithms. In Fig. 7, when the number of threads is 2, 4, 6, 8 and 10, the query processing time of $SkNN_{PG}$ is 3309, 2009, 1572, 1136, and 994 s, respectively. The query processing time of $SkNN_{PG}$ is decreased according to the number of threads. In addition, our $SkNN_{PG}$ shows 12 and 7 times better performance on average than $SkNN_{pm}$ and $SkNN_{pi}$, respectively. In Fig. 8, when the number of data is 5 k, 10 k, 15 k, 20 k, 25 k, and 30 k, the query

Fig. 9 Performance with varying k for synthetic data in parallel

processing time of $SkNN_{PG}$ is 125, 241, 353, 462, 537, and 640 s, respectively. $SkNN_{PI}$ and $SkNN_{PG}$ show better performance than $SkNN_{pm}$ because they use the index-based data filtering technique. In Fig. 9, when the number of k is 5, 10, 15, and 20, the query processing time of $SkNN_{PG}$ is 586, 1173, 1773, and 2308 s, respectively. Our $SkNN_{PG}$ shows 10 and 5.2 times better performance on average than $SkNN_{pm}$ and $SkNN_{PI}$, respectively. Our $SkNN_{PG}$ outperforms $SkNN_{pm}$ and $SkNN_{PI}$ because it uses efficient secure protocols for parallel environment, i.e., SMR and GSCR.

Privacy-preserving kNN query processing algorithms generally use homomorphic encryption for providing data privacy and query privacy. Therefore, it is inevitable that they require high computational cost and their search time complexity is linear. As a result, the existing algorithms handle 10,000 data items in their performance evaluation [22, 24]. By following the existing algorithms, we conduct the performance evaluation of privacy-preserving kNN query processing algorithms when the number of data items is ranged from 5000 to 30,000. It is shown from our performance evaluation that the proposed algorithm ($SkNN_G$) has linear time complexity, but its slope is lower than the existing algorithms ($SkNN_m$, $SkNN_I$). This is because the proposed algorithm performs data filtering using a kd -tree structure and uses Yao's gabled circuit, which does not use the encrypted binary array.

In order that the proposed algorithm can handle a very large dataset (e.g., 1 million), we do the performance evaluation of the proposed algorithm when the number of data items is 300,000, 600,000, and 1,000,000 (1 million). But in our performance evaluation, we exclude the existing algorithms because they cannot work for a very large dataset due to both their extremely long execution time and the nonexistence of their parallel versions. We do the performance evaluation of the proposed algorithm when the number of data items is 300,000, 600,000, and 1,000,000 with two dimensions. Rather than six-dimensional data in Table 3, we use two-dimensional data because the size of main memory for our experiment is limited. Figure 10 shows the performance of the proposed parallel algorithm to show that it has the capability of dealing with a very large dataset (e.g., 1 million). The proposed algorithm requires 452, 834, and 1341 s when the number of data items is 300,000, 600,000, and 1,000,000,

Fig. 10 Performance with varying n for a very large dataset

Fig. 11 Performance with varying kd -tree depth for real data

respectively. It is shown from our experiment that the proposed algorithm is linear according to the number of data items. Thus, it is inferred from our observation that the proposed algorithm can handle a very large dataset with a linear time complexity.

7.2 Performance using a real dataset

According to Fig. 11, it is shown that the performances of both $SkNN_I$ and $SkNN_G$ are best when h is 7 and 8. So, we set h to 7 in our experiment. Figure 12 shows the performance of three algorithms in a single machine. With varying k , our $SkNN_G$ shows 22.3 and 5.9 times better performance than $SkNN_m$ and $SkNN_I$, respectively. As a result, our $SkNN_G$ outperforms $SkNN_m$ because it can reduce the computation cost by pruning out unnecessary data with the kd -tree, contrary to considering all the data in $SkNN_m$. In addition, our $SkNN_G$ outperforms $SkNN_I$ because our algorithm uses efficient secure protocols based on both Yao's garbled circuit and the data packing technique.

Figures 13 and 14 show the performance of three parallel algorithms. In Fig. 13, when the number of threads is 2, 4, 6, 8, and 10, the query processing time of $SkNN_{PG}$ is 1659, 977, 745, 624, and 536 s, respectively. The query processing time of $SkNN_{PG}$ is decreased according to the number of threads. In addition, our $SkNN_{PG}$

Fig. 12 Performance with varying k for real data

Fig. 13 Performance with varying # of threads for real data

Fig. 14 Performance with varying k for real data in parallel

shows 13.3 and 4.1 times better performance on average than $SkNN_{pm}$ and $SkNN_{PI}$, respectively. In Fig. 14, when the number of k is 5, 10, 15, and 20, the query processing time of $SkNN_{PG}$ is 277, 536, 765, and 1022 s, respectively. Our $SkNN_{PG}$ shows 12.1 and 3.7 times better performance on average than $SkNN_{pm}$ and $SkNN_{PI}$, respectively. Our $SkNN_{PG}$ outperforms $SkNN_{pm}$ and $SkNN_{PI}$ because it uses efficient secure protocols for parallel environment, i.e., SMR and GSCR.

7.3 Discussion

In this section, we not only clarify the differences between the existing privacy-preserving k NN query processing algorithms [22, 24] and our algorithm, but also highlight the advantage of our algorithm. In Table 4, we analyze the privacy-preserving k NN query processing algorithms, in terms of secure protocol, index structure, and random value pool.

Impact of secure protocol with low computational cost Secure protocols are very important for privacy-preserving query processing in cloud computing. We should make secure protocols more efficient because we target on the privacy-preserving query processing by using the Paillier cryptosystem, which consumes high computational cost. First, Elmehdwi et al.'s algorithm proposed secure protocols, such as SM, SBD, SMIN, $SMIN_n$ for k NN query processing. Elmehdwi et al.'s algorithm can protect data privacy, query privacy by using the Paillier cryptosystem. Also, it uses arithmetic operations to protect the original data and to hide data access patterns. However, the drawback of Elmehdwi et al.'s algorithm is excessively high computational cost because SBD, SMIN, and $SMIN_n$ protocols use an encrypted binary array as input value. For example, when we perform the SMIN protocol between $E(8)$ and $E(7)$, clouds transform an encrypted decimal value into an encrypted binary array: $E(8)_{(10)} = \{E(1), E(0), E(0), E(0)\}_{(2)}$, $E(7)_{(10)} = \{E(0), E(1), E(1), E(1)\}_{(2)}$. After that, clouds perform the SMIN protocol between $E(8)_{(10)} = \{E(1), E(0), E(0), E(0)\}_{(2)}$ and $E(7)_{(10)} = \{E(0), E(1), E(1), E(1)\}_{(2)}$. As a result, the SMIN requires high computational cost because it performs the encrypted operations as many times as the length

Table 4 Comparison of privacy-preserving k NN query processing algorithms

	Elmehdwi et al.'s work [22]	Kim et al.'s work [24]	The proposed algorithm
Secure protocol	SM, SMIN, $SMIN_n$	SCMP, SPE	GSCMP, GSPE
Index structure	No index	Kd-tree	Kd-tree
Encrypted random value pool	Not provision	Not provision	Provision

of data domain in bit. The SBD and $SMIN_n$ require high computational cost for the same reason of the SMIN. Second, Kim et al.'s algorithm proposed such secure protocols as SCMP and SPE, which are used for index search to find out k NN candidates. However, because both SCMP and SPE use encrypted binary array as input value, they require high computational cost for the same reason of the SMIN. Meanwhile, our algorithm proposed the GSCMP and the GSPE, which perform only one Paillier arithmetic operation. This is because they use an encrypted decimal value as input, rather than an encrypted binary array, due to utilizing Yao's garbled circuit. As a result, the GSCMP and the GSPE require low computational cost.

Impact of using index structure over encrypted database Because Elmehdwi et al.'s algorithm does not use index structure for data filtering, it should process all of the data items, which leads to performance degradation. To solve the problem, both Kim et al.'s algorithm and our algorithm use encrypted kd -tree as an index structure. As a result, both algorithms achieve performance enhancement by using kd -tree. It is shown from our experiment that our algorithm searches only 10% of all the data items on average, due to data filtering using kd -tree. Table 5 shows the comparison of privacy-preserving k NN algorithms in terms of the number of data items accessed.

Elmehdwi et al.'s algorithm requires $N \times k$ number of data items accessed in total, whereas both Kim et al.'s algorithm and our algorithm require $N/F + F \times k \times (c + 1)$ number of data items accessed in total. Here, N is the number of data items, h is the height of kd -tree, F is the *fanout* of the kd -tree, and c is the number of node group in the k NN result refinement phase. We show that both Kim et al.'s algorithm and our algorithm are better than Elmehdwi et al.'s algorithm, in terms of the number of data items accessed, by using Eq. (12). Here, we can ignore $1/F \times k$ because it is small enough. Because $F \times (c + 1)$ means the number of all the data items in the selected leaf nodes for the privacy-preserving k NN query processing, which is always the subset of N , we can say that Eq. (12) is true. By using the index structure over encrypted database, our algorithm and Kim et al.'s algorithm are shown to be better than Elmehdwi et al.'s algorithm.

Table 5 Comparison of privacy-preserving k NN algorithms in terms of the number of data items accessed

Query processing phase	Algorithms		
	Elmehdwi et al.'s algorithm [22]	Kim et al.'s algorithm [24]	The proposed algorithm
Candidate node search	–	N/F	N/F
k NN retrieval	$N(= 2^{h-1} \times F) \times k$	$F \times k$	$F \times k$
k NN result refinement	–	$F \times c \times k$	$F \times c \times k$
Total	$N(= 2^{h-1} \times F) \times k$	$N/F + F \times k \times (c + 1)$	$N/F + F \times k \times (c + 1)$

N =the number of data, h =height of kd -tree, F =*fanout*, c =the number of node group, k =the number of nearest neighbors

$$\begin{aligned}
 N \times k &\geq \frac{N}{F} + F \times k \times (c + 1) \\
 \Leftrightarrow k &\geq \frac{1}{F} + \frac{F \times k \times (c + 1)}{N} \\
 \Leftrightarrow 1 &\geq \frac{1}{F \times k} + \frac{F \times (c + 1)}{N} \\
 \approx 1 &\geq \frac{F \times (c + 1)}{N}
 \end{aligned} \tag{12}$$

Impact of encrypted random value pool for parallelism In our secure system, we use two-party computation for the parallel k NN query processing algorithm. Thus, we need to prevent C_B from extracting meaningful information while executing secure protocols. For this, C_A generates a random value r from Z_N and encrypts r by using the Paillier cryptosystem. Then, C_A adds the encrypted random value $E(r)$ to the encrypted plaintext $E(m)$ by computing $E(m + r) = E(m) \times E(r)$. Because $m \pm r$ is independent from m , C_B cannot obtain meaningful information with decryption. However, adding a random value to the ciphertext in the Paillier cryptosystem leads to performance degradation because both encryption and decryption operations require higher computation cost than other encrypted operations. As shown in Table 2, in the Secure Multiplication protocol, both Elmehdwi et al.'s work and Kim et al.'s work require three times of the encryption: 2 encryptions for random values at C_A and 1 encryption for the result of multiplication at C_B . Meanwhile, our algorithm requires only one encryption for the result of multiplication at C_B because it selects the encrypted random values from the random value pool without encrypting the random values at C_A . As shown in Table 2, in the secure compare protocol, Elmehdwi et al.'s work requires $\log_2 D$ times of encryption where D is a data domain. Kim et al.'s work requires three times of the encryption: 2 encryptions for random values at C_A and 1 encryption for the result of the comparison between two values at C_B . Meanwhile, our algorithm requires only one encryption for the result of comparison at C_B by using the random value pool. Therefore, our algorithm can reduce the amount of computation cost for encryption by using the encrypted random value pool.

8 Conclusion and future work

Due to the privacy issues, a database needs to be encrypted before being outsourced to the cloud. However, most of the existing k NN algorithms are insecure in that they disclose data access patterns during the query processing. To solve the problem, we proposed a new privacy-preserving k NN query processing algorithm via secure two-party computation. To achieve a high degree of efficiency in query processing, we also proposed a parallel k NN query processing algorithm using encrypted random value pool. Our algorithms can protect data, query and

data access patterns. In our performance analysis, our algorithms showed about 4–30 times better performance than the existing algorithms, in terms of a query processing cost. As a future work, we plan to expand our algorithms to support other types of queries, such as Top- k and k NN classification. In addition, to the best of our knowledge, the privacy-preserving k NN algorithms with homomorphic encryption are generally studied for low-dimensional data space because they require the high computational cost [22, 24]. For dealing with high-dimensional data space, we will extend our algorithm by using a data dimensionality reduction technique [42, 43].

Acknowledgements This work was partly supported by a National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIT) (No. 2019R1I1A3A01058375).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Oh D, Kim I, Kim K, Lee SM, Ro WW (2015) Highly secure mobile devices assisted with trusted cloud computing environments. *ETRI J* 37(2):348–358
2. Raja J, Ramakrishnan M (2020) Confidentiality-preserving based on attribute encryption using auditable access during encrypted records in cloud location. *J Supercomput* 76(8):6026–6039
3. Ahmad A, Ahmad M, Habib MA, Sarwar S, Chaudhry J, Latif MA, Shahid M (2019) Parallel query execution over encrypted data in database-as-a-service (DaaS). *J Supercomput* 75(4):2269–2288
4. Williams P, Sion R, Carbunar B (2008) Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In: *Proceedings of the 15th ACM Conference on Computer and Communications Security*, pp 139–148
5. Cui S, Belguith S, Zhang M, Asghar MR, Russello G (2018) Preserving access pattern privacy in sgx-assisted encrypted search. In: *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, pp 1–9
6. Mehmood A, Natgunanathan I, Xiang Y, Hua G, Guo S (2016) Protection of big data privacy. *IEEE Access* 4:1821–1834
7. Pingley A, Zhang N, Fu X, Choi HA, Subramaniam S, Zhao W (2011) Protection of query privacy for continuous location based services. In: *2011 Proceedings IEEE INFOCOM*. IEEE, pp 1710–1718
8. Eom CS, Lee C, Lee W, Leung C (2020) Effective privacy preserving data publishing by vectorization. *Inf Sci* 527:311–328
9. Kaiping X, Zhu B, Yang Q, Gai N, Wei D, Yu N (2020) InPPTD: a lightweight incentive-based privacy-preserving truth discovery for crowdsensing systems. *IEEE Internet Things J* 8(6):4305–4316
10. Kousika N, Premalatha K (2021) An improved privacy-preserving data mining technique using singular value decomposition with three-dimensional rotation data perturbation. *J Supercomput*:1–9
11. Carbunar B, Yu Y, Shi W, Pearce M, Vasudevan V (2010) Query privacy in wireless sensor networks. *ACM Trans Sens Netw (TOSN)* 6(2):1–34
12. Veugen T, Blom F, de Hoogh SJ, Erkin Z (2015) Secure comparison protocols in the semi-honest model. *IEEE J Sel Top Signal Process* 9(7):1217–1228

13. Youn TY, Jho NS, Chang KY (2018) Design of additive homomorphic encryption with multiple message spaces for secure and practical storage services over encrypted data. *J Supercomput* 74(8):3620–3638
14. Islam MS, Kuzu M, Kantarcioglu M (2012) Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In: *Ndss*, vol 20, p 12
15. Wu W, Xian M, Paramalli U, Lu B (2021) February). Efficient privacy-preserving frequent itemset query over semantically secure encrypted cloud database. In *World Wide Web* 24:607–629
16. Wu W, Paramalli U, Liu J, Xian M (2018) March). Privacy preserving k -nearest neighbor classification over encrypted database in outsourced cloud environments. In *World Wide Web* 22:101–123
17. Dai H, Ji Y, Yang G, Huang H, Yi X (2019) A privacy-preserving multi-keyword ranked search over encrypted data in hybrid clouds. In *IEEE Access* 8:4895–4907
18. Wong WK, Cheung DWL, Kao B, Mamoulis N (2009) Secure k NN computation on encrypted databases. In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pp 139–152
19. Yiu ML, Ghinita G, Jensen CS, Kalnis P (2010) Enabling search services on outsourced private spatial data. *VLDB J* 19(3):363–384
20. Hu H, Xu J, Ren C, Choi B (2011) Processing private queries over untrusted data cloud through privacy homomorphism. In: *2011 IEEE 27th International Conference on Data Engineering*. IEEE, pp 601–612
21. Zhu Y, Xu R, Takagi T (2013) Secure k -NN computation on encrypted cloud data without sharing key with query users. In: *Proceedings of the 2013 International Workshop on Security in Cloud Computing*, pp 55–60
22. Elmehdwi Y, Samanthula BK, Jiang W (2014) Secure k -nearest neighbor query over encrypted data in outsourced environments. In: *2014 IEEE 30th International Conference on Data Engineering*. IEEE, pp 664–675
23. Zhou L, Zhu Y, Castiglione A (2017) Efficient k -NN query over encrypted data in cloud with limited key-disclosure and offline data owner. *Comput Secur* 69:84–96
24. Kim HI, Kim HJ, Chang JW (2019) A secure k NN query processing algorithm using homomorphic encryption on outsourced database. *Data Knowl Eng* 123:101602
25. Yao ACC (1986) How to generate and exchange secrets. In: *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, pp 162–167
26. Paillier P (1999) Public-key cryptosystems based on composite degree residuosity classes. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Berlin, Heidelberg, pp 223–238
27. Camenisch J, Chandran N, Shoup V (2009) A public key encryption scheme secure against key dependent chosen plaintext and adaptive chosen ciphertext attacks. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Berlin, Heidelberg, pp. 351–368
28. Cambareri V, Mangia M, Pareschi F, Rovatti R, Setti G (2015) On known-plaintext attacks to a compressed sensing-based encryption: a quantitative analysis. *IEEE Trans Inf Forensics Secur* 10(10):2182–2195
29. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pp. 47–57
30. Daemen J, Rijmen V (1999) AES proposal: Rijndael
31. Yao B, Li F, Xiao X (2013) Secure nearest neighbor revisited. In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, pp 733–744
32. Hazay C, Lindell Y (2010) Efficient secure two-party protocols: Techniques and constructions. Springer
33. Tsai WT, Sun X, Balasooriya J (2010) Service-oriented cloud computing architecture. In: *2010 Seventh International Conference on Information Technology: New Generations*. IEEE, pp 684–689
34. Jadeja Y, Modi K (2012) Cloud computing-concepts, architecture and challenges. In: *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*. IEEE, pp 877–880
35. Bahrami M, Singhal M (2015) The role of cloud computing architecture in big data. In: *Information Granularity, Big Data, and Computational Intelligence*. Springer, Cham, pp 275–295
36. Beniley JL (1975) Multidimensional binary search trees used for associative searching. *ACM Commun* 18(9):509–517

37. Robinson JT (1981) The KDB-tree: a search structure for large multidimensional dynamic indexes. In: Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, pp 10–18
38. Bugiel S, Nürnberger S, Sadeghi AR, Schneider T (2011) Twin clouds: Secure cloud computing with low latency. In: IFIP International Conference on Communications and Multimedia Security. Springer, Berlin, Heidelberg, pp 32–44
39. Liu A, Zhengy K, Liz L, Liu G, Zhao L, Zhou X (2015) Efficient secure similarity computation on encrypted trajectory data. In: 2015 IEEE 31st International Conference on Data Engineering. IEEE, pp 66–77
40. Goldreich O (1998) Secure multi-party computation. Manuscript. Preliminary version 78
41. Michael Bain (2021) Chess (King-Rook vs. King) Data Set. <http://archive.ics.uci.edu/ml/datasets/Chess+%28King-Rook+vs.+King%29>. Accessed 21 April 2021
42. Ayesha S, Muhammad K, Talib R (2020) Overview and comparative study of dimensionality reduction techniques for high dimensional data. *Inf Fusion* 59:44–58
43. Reddy G, Reddy M, Lakshmana K, Kaluri R, Rajput DS, Srivastava G, Baker T (2020) Analysis of dimensionality reduction techniques on big data. *IEEE Access* 8:54776–54788

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.