electronics

MDPI

Article

Privacy-Preserving Top-*k* Query Processing Algorithms Using Efficient Secure Protocols over Encrypted Database in Cloud Computing Environment

Hyeong-Jin Kim [1], Yong-Ki Kim [2], Hyun-Jo Lee [1] and Jae-Woo Chang [1,*]

[1] Department of Computer Engineering, Jeonbuk National University, Jeonju-si 54896, Korea
[2] Department of IT Convergence System, Jeonju Vision College, Jeonju-si 55069, Korea
* Correspondence: jwchang@jbnu.ac.kr

Abstract: Recently, studies on secure database outsourcing have been highlighted for the cloud computing environment. A few secure Top-*k* query processing algorithms have been proposed in the encrypted database. However, the previous algorithms can support either security or efficiency. Therefore, we propose a new Top-*k* query processing algorithm using a homomorphic cryptosystem, which can support both security and efficiency. For security, we propose new secure and efficient protocols based on arithmetic operations. To obtain a high level of efficiency, we also propose a parallel Top-*k* query processing algorithm using an encrypted random value pool. Through our performance analysis, the proposed Top-*k* algorithms present about 1.5∼7.1 times better performance with regard to a query processing time, compared with the existing algorithms.

Keywords: secure protocol; privacy-preserving Top-*k* query processing algorithm; encrypted database; database outsourcing; cloud computing

check for updates

Citation: Kim, H.-J.; Kim, Y.-K.; Lee, H.-J.; Chang, J.-W. Privacy-Preserving Top-*k* Query Processing Algorithms Using Efficient Secure Protocols over Encrypted Database in Cloud Computing Environment. *Electronics* **2022**, *11*, 2870. https://doi.org/10.3390/electronics11182870

Academic Editors: Muath Obaidat, Thaier Hayajneh and Eman Hammad

Received: 9 August 2022
Accepted: 8 September 2022
Published: 11 September 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the increasing popularity of cloud computing, there has been growing interest in outsourcing databases. Cloud computing provides a service that allows internet-connected users to use virtual computing resources such as storage, computation, and network [1–5]. Generally, three requirements are considered for outsourcing databases in cloud computing. First, it is essential to provide data privacy because the data owner's database contains the sensitive information of clients [6,7]. Second, the query and the query result must be protected because these information can contain and infer the sensitive information of the query issuer [8,9]. Finally, data access patterns must be protected because an attacker can infer sensitive information by analyzing the data access patterns [10–12]. Therefore, secure query processing over an encrypted database has been researched to protect the original data, queries, and access patterns. Previous strategies modify plaintexts to their substituted data, and outsource them to a cloud [13–18]. However, these previous strategies cannot completely protect both data and queries because they are weak to various attacks. To tackle this problem, recent strategies encrypt the original data and outsource them to the cloud [19–25]. That is, a data owner encrypts the database before he/she outsources the original data to a cloud. The cloud can process the query that is issued from a user. The user can obtain the results from the cloud. Figure 1 shows an encrypted database outsourcing model.

Top-*k* query processing algorithms are used for various applications such as location-based service, e-commerce, and web search engine [26]. First, H-I. Kim et al.'s algorithm [27] proposed Top-*k* query processing algorithms based on the Paillier cryptosystem. However, while processing the Top-*k* query processing algorithms, the algorithm needs a high computational cost because it uses secure protocols based on binary operations. Second, H-J. Kim et al.'s algorithm [28] proposed Top-*k* query processing algorithms using Yao's

garbled circuit. By using Yao's garbled circuit, the algorithm uses a secure protocol to check whether or not a node includes a point without binary bit operations for ciphertext. However, since Yao's garbled circuit [29,30] is based on a hardware-based Boolean circuit, the algorithm still requires repetitive bit operations, thus causing a high computation cost. Both algorithms have a critical problem in that they need a high computational cost for Top-*k* query processing. To the best of our knowledge, there is no conventional Top-*k* query processing algorithm that is suitable for parallel processing. To tackle this problem, we propose a new Top-*k* query processing algorithm using new secure protocols based on arithmetic operations over encrypted database in cloud computing. In terms of efficiency, our algorithm uses both new secure protocols and a data filtering technique using *k*d-tree [31]. The proposed secure protocols optimize the procedure of comparison by using arithmetic operations, rather than employing binary bit operations, which require a high computation cost. Moreover, to improve the efficiency, we propose a parallel Top-*k* query processing algorithm using a thread pool. In addition, while hiding data access patterns, our algorithm support security for both original data and user query by using the Paillier cryptosystem [32] and secure two-party computation [33–35]. To prove the security of our algorithm, we provide the formal security proofs of both the proposed secure protocols and Top-*k* query processing algorithms. Through the performance analysis, we prove that our Top-*k* query processing algorithms outperform the existing ones. The contributions of our paper are as follows:

- We present an architecture for outsourcing both the encrypted data and index.
- We propose new secure protocols based on arithmetic operations (e.g., ASC, ASRO, and ASPE) to protect original data, user query, and access patterns.
- We propose a new Top-*k* query processing algorithm that can support both security and efficiency.
- We propose a new parallel Top-*k* query processing algorithm using a random value pool to improve the efficiency of Top-*k* query processing.
- We also present the comprehensive performance analysis of our algorithms with a synthetic and real dataset.

We organize the rest of the paper as follows. In Section 2, we describe the Paillier cryptosystem, adversarial attack model, and related work. In Section 3, we present the two-party computation structure and new secure protocols. In Section 4, we describe a a new Top-*k* query processing algorithm. In Section 5, we describe a parallel Top-*k* algorithm. In Section 6, we present the security proof of our Top-*k* algorithms. In Section 7, we describe the performance analysis of our Top-*k* algorithms. Finally, we conclude this paper.

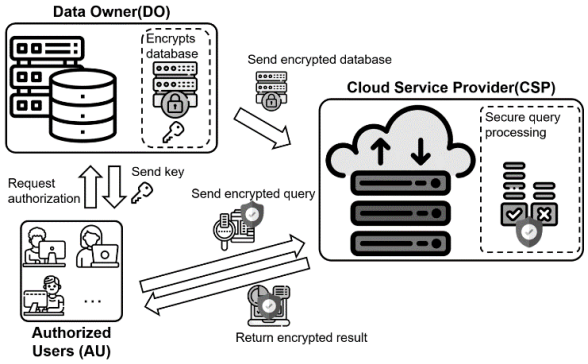


Figure 1. Encrypted database outsourcing model.

2. Background and Related Work

2.1. Background

Paillier cryptosystem.The Paillier cryptosystem [32] is an additive homomorphic and probabilistic asymmetric encryption scheme for public key cryptography. The public key

pk for encryption is represented by (N, g), where N is a multiplication of the large prime integer p and q, and g is a circular integer in $Z^*_{N^2}$. Here, $Z^*_{N^2}$ denotes an integer domain ranging from zero to N^2. The secret key *sk* for decryption is represented by (p, q). Let $E(\cdot)$ represent the encryption function and $D(\cdot)$ represent the decryption function. The Paillier cryptosystem includes the following characteristic.

1. Homomorphic addition: The multiplication of two ciphertexts $E(m_1)$ and $E(m_2)$ generates the ciphertext of the sum of their plaintexts m_1 and m_2 (Equation (1)).

$$E(m_1 + m_2) = E(m_1) \times E(m_2) \; mod \; N^2 \tag{1}$$

2. Homomorphic multiplication: The m_2th power of ciphertext $E(m_1)$ generates the ciphertext of the multiplication of m_1 and m_2 (Equation (2)).

$$E(m_1 + m_2) = E(m_1) \times E(m_2) \; mod \; N^2 \tag{2}$$

3. Semantic security: Encryptions of the same plaintexts generate different ciphertexts in the same public key (Equation (3)).

$$m_1 = m_2 \nRightarrow E(m_1) = E(m_2) \tag{3}$$

Adversarial attack model. In the outsourcing database environment, two attack models can be considered: a semi-honest attack model and a malicious attack model [36,37]. In the semi-honest (or honest-but-curious) attack model, the cloud performs its own protocol honestly, but attempts to obtain sensitive data about the data owner and the authorized user. To protect a semi-honest attack, original data should be encrypted before outsourcing. A malicious attack model tries to achieve the original data or disable service. To protect the original data under the malicious attack, a service provider concentrates on distinguishing attacks and restoring the damaged formal procedure. Since we concentrate on protecting sensitive data in an outsourced environment, we propose secure query processing under the semi-honest attack model. A secure protocol for the semi-honest attack model is defined as follows [36,37].

Definition 1. *Assuming α_i is the input parameters of cloud C_i, $\prod_i(\rho(\alpha))$ is the execution image of C_i for the protocol ρ. If the simulating execution image $\prod_{S_i}(\rho(\alpha))$ is indistinguishable from $\prod_i(\rho(\alpha))$, the protocol ρ is a secure protocol under the semi-honest attack model.*

2.2. Related Work

The existing privacy-preserving Top-*k* query processing algorithms are as follows. First, J. Vaidya et al. [38] proposed a privacy-preserving Top-*k* query processing algorithm using Fagin's scheme [39], in which the data are vertically partitioned. If each party reports the scored data being ordered based on the local score until there are at least *k* common data in the output of all of the parties, the union of the reported data includes Top-*k* results. The algorithm determines the actual Top-*k* results by identifying an approximate cutoff score separating the *k*-th data item from those below it. The algorithm has an advantage in that it does not reveal the score of an individual datum by using a secure comparison technique. However, in terms of query privacy, an attacker can infer the preference of a user because the cutoff score is easily estimated by using a binary search over the range of values. According to data privacy, because the algorithm does not encrypt the partitioned data, the attacker can obtain the original data under the semi-honest attack model. In addition, data access patterns are not protected because the identification of the Top-*k* result is disclosed.

Second, M. Burkhart et al. [40] proposed a Top-*k* query processing algorithm that utilizes hash tables and a secret sharing technique. The algorithm aggregates the distributed key-value data by using Shamir's secret sharing technique, and finds the *k* number of key-value pairs with the largest aggregated values as the Top-*k* result. To reduce computation time, the algorithm uses fixed-length hash tables for avoiding expensive key comparison

operations. However, the algorithm cannot guarantee the accurate result because the aggregated results are probabilistic, in that the algorithm performs a binary search for estimating the intermediate threshold separating the k-th item from the (k+1)-th item. Moreover, an attacker can infer the preference of a user because a threshold is easily estimated by using binary search over the range of values. The algorithm cannot conceal data access patterns because the index of a hash table related to the Top-k results is revealed.

Third, Y. Zheng et al. [41] proposed a privacy-preserving Top-k query processing algorithm over vertically distributed data sources. To support data privacy, the service provider (SP) divides the data owner's database into data sources ($DS = DS_1, DS_2, \ldots, DS_m$) for each attribute. The SP generates the encryption keys and determines the positive weight values $w_1, w_2, \ldots, w_m$ for the scoring function $f(x_1, x_2, \ldots, x_m) = \sum_{i=1}^{m} w_i x_i$, where x_i is a data record and $\sum_{i=1}^{m} w_i = 1$. Then, the SP distributes the encryption keys to each DS_i and sends the corresponding weight w_i to DS_i via a secure channel. For supporting the Top-k query, each DS_i outsources its data to the cloud. Because both x_i and w_i are private information, DS_i encrypts the product of x_i and w_i before outsourcing them to the cloud. However, in the semi-honest attack model, an attacker can achieve the sensitive data from data sources because a data owner transmits data records and weights to the data sources in plaintext. Therefore, the algorithm does not protect data, query, and access pattern.

Fourthly, H-I. Kim et al. [27] proposed a privacy-preserving Top-k query processing algorithm (STopk_I) grounded using both an encrypted kd-tree index and the Paillier cryptosystem [31]. The Paillier cryptosystem can provide homomorphic operation and protect a chosen-plaintext attack or a chosen-ciphertext one, so that the cloud can obtain the Top-k query result without decrypting both a user's query and original data. The algorithm also proposed a secure protocol based on binary bit operations that can access all leaf nodes without the exposure of data access patterns. The algorithm can protect the user's query, sensitive data, and data access patterns. However, because the secure protocol consists of binary bit operations for ciphertext, it is necessary to perform a binary bit operation as many times as the data size. The repetitive binary bit operations cause high computation cost.

Finally, H-J. Kim et al. [28] proposed a privacy-preserving Top-k query processing algorithm (STopk_G) using Yao's garbled circuit [29,30]. Yao's garbled circuit is a secure protocol that enables two-party secure computation in which two semi-honest parties can jointly calculate a function over private inputs by using a Boolean circuit. By using Yao's garbled circuit, the algorithm uses a secure protocol to check whether or not a node includes a point without binary bit operations for ciphertext. The algorithm can present more efficiently than STopk_I while providing the same level of privacy as STopk_I. However, since Yao's garbled circuit is based on a hardware-based Boolean circuit, the algorithm still requires repetitive bit operations, thus causing a high computation cost.

Table 1 summarize the existing studies based on the characteristics. We describe their comparison with regard to three major characteristics, i.e., hiding access patterns, computation overhead, and security risk. First, H-I. Kim et al.'s work [27] and H-J. Kim et al.'s work [28] can protect data access patterns, while J. Vaidya and C. Clifton's work [38], M. Burkhart and X. Dimitropoulos's work [40], and Y. Zheng et al.'s work [41] cannot protect it. Second, J. Vaidya and C. Clifton's work [38] and M. Burkhart and X. Dimitropoulos's work [40] require low computational overhead because they do not use any encryption scheme, while H-I. Kim et al.'s work [27] and H-J. Kim et al.'s work [28] need high computational overhead due to the use of the Paillier cryptosystem [32]. Finally, H-I. Kim et al.'s work [27] and H-J. Kim et al.'s work [28] have low risk with regard to security because they protect sensitive data, uses's query, and data access patterns, while J. Vaidya and C. Clifton's work [38], M. Burkhart and X. Dimitropoulos's work [40], and Y. Zheng et al.'s work [41] have high security risk because they only protect sensitive data.

Table 1. Comparison of the existing studies.

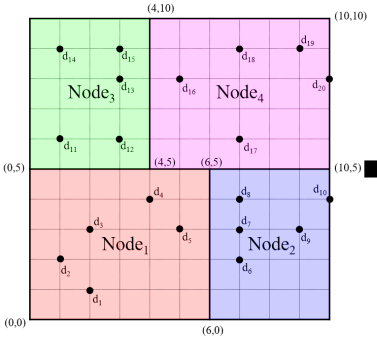
Features / Schemes	Data Privacy	Query Privacy	Hiding Data Access Pattern	Index	Computation Overhead	Encryption	User Involvement in Computation	Security Risk
J. Vaidya and C. Clifton's work [38]	Support	Not support	Not support	None	Low	None	Involved	High
M. Burkhart and X. Dimitropoulos's work [40]	Support	Not support	Not support	None	Low	None	Involved	High
Y. Zheng et al. [41]	Support	Not support	Not support	None	Moderate	AES	Involved	High
H-I. Kim et al. [27]	Support	Support	Support	Encrypted *k*d-tree	Very high	Paillier	Not involved	Low
H-J. Kim et al. [28]	Support	Support	Not support	Encrypted *k*d-tree	High	Paillier	Not involved	Low
Proposed	Support	Support	Support	Encrypted *k*d-tree	Low	Paillier	Not involved	Low

3. Overall System Architecture

3.1. System Architecture

Since we adopt the semi-honest attack model [36,37], we consider the cloud as insider adversaries. Under the semi-honest adversarial model, the cloud follows the procedure of protocol, but can attempt to achieve the additional information not allowed to the cloud. Therefore, earlier studies [13,16] are performed under the semi-honest attack model. Meanwhile, a secure protocol is a communication protocol that is an agreed sequence of actions performed by two or more communicating entities (e.g., cloud A and cloud B) in order to accomplish some mutually desirable goal [34,35]. It makes use of cryptographic techniques, allowing the communicating entities to achieve a security goal. For example, the entities encrypt the information or permutate the sequence of data by using the random value for data protection. A protocol can be proven to be secure under the semi-honest adversarial model by using Definition 1 in Section 2.

The system is organized according to four components: Cloud A(C_A), Cloud B(C_B), data owner (DO), and authorized user (AU). The DO possesses the original database (T) of a set of records [27]. A record t_i ($1 \leq i \leq n$) is composed of m columns, where m denotes the number of data dimensions, and the j-th columns of t_i is represented by $t_{i,j}$ ($1 \leq j \leq m$). The DO divides T by using the kd-tree [27] to provide the indexing on T. If we retrieve the tree structure in a hierarchical manner, the access pattern can be disclosed. Therefore, we only consider the leaf nodes of the kd-tree, and all the leaf nodes are retrieved once during the query processing step. Hence, a node denotes a leaf node. Let h denote the level of the constructed kd-tree, and F denotes 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 number of leaf nodes. The region data of $node_z$ is shown as the upper bound $ub_{z,j}$ and the lower bound $lb_{z,j}$ ($1 \leq z \leq 2^{h-1}$, $1 \leq j \leq m$). Each node retains the identifiers (id) of the 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 [32] before outsourcing the database to the cloud. Therefore, the DO make $E(t_{i,j})$ encrypt $t_{i,j}$ for $1 \leq i \leq n$ and $1 \leq j \leq m$. The DO also encrypts the region data of all nodes of the kd-tree, so as to support efficient query processing. Especially, ub and lb of each node are encrypted attribute-wise, such that $E(ub_{z,j})$ and $E(lb_{z,j})$ are created with $1 \leq z \leq 2^{h-1}$ and $1 \leq j \leq m$. Figure 2 presents the example of the encrypted kd-tree when the number of data items is 20 and the level of the kd-tree is 3. The encrypted kd-tree consists of four regions (leaf nodes), i.e., $Node_1$, $Node_2$, $Node_3$, and $Node_4$. For example, the lower bound and upper bound of $Node_1$ are <E(0), E(0)> and <E(6), E(5)>, respectively. Additionally, $Node_1$ includes five data items, i.e., E(d_1) = <E(2), E(1)>, E(d_2) = <E(1), E(2)>, E(d_3) = <E(2), E(3)>, E(d_4) = <E(4), E(4)>, and E(d_5) = <E(5), E(3)>.



Node.ID	E(Node Range)		Fanout	E(Data)
	E(Lower Bound)	E(Upper Bound)		
$Node_1$	E(0), E(0)	E(6), E(5)	5	E(d_1)=<E(2), E(1)>, … , E(d_5)=<E(5), E(3)>
$Node_2$	E(6), E(0)	E(10), E(5)	5	E(d_6)=<E(7), E(2)>, … , E(d_{10})=<E(10), E(4)>
$Node_3$	E(0), E(5)	E(4), E(10)	5	E(d_{11})=<E(1), E(6)>, … , E(d_{15})=<E(3), E(9)>
$Node_4$	E(4), E(5)	E(10), E(10)	5	E(d_{16})=<E(5), E(8)>, … , E(d_{20})=<E(10), E(8)>

Figure 2. Example of the encrypted kd-tree.

We consider that C_A and C_B are non-colluding and semi-honest clouds. Thus, they correctly follow the procedure of protocols. To support Top-k query processing over the encrypted database, a secure multi-party computation (SMC) is required between C_A and C_B. To establish the structure of SMC, the DO outsources both the encrypted data 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 data of each node in ciphertext and the ids of the data located inside the node, in plaintext. The *DO* also sends *pk* to *AU*s to authorize them to encrypt a Top-*k* query. When issuing a query, an *AU* first creates E(q_j) by encrypting a query q attribute-wise for $1 \leq j \leq m$. C_A and C_B process the query and return its result to the *AU*.

3.2. Secure Protocol

Since the proposed Top-*k* query processing algorithm is constructed using several secure protocols, we use four secure protocols from the literature [19,27,28], such as Secure Multiplication (SM) [19], Secure Bit-Not (SBN) [27], Secure Compare (CMP-S) [27,28], and Secure Minimum from Set of n values (SMS_n) [27,28]. In addition, we newly suggest secure protocols: Advanced Secure Compare (ASC), Advanced Secure Range Overlapping (ASRO), and Advanced Secure Point Enclosure (ASPE). By using arithmetic operations, the proposed secure protocols improve the existing comparison protocols that employ binary bit comparison [19] and garbled circuit comparison [29,30]. Therefore, the proposed secure protocols require a low computation cost.

Advanced Secure Compare (ASC) protocol. The ASC protocol compares two encryption values securely and returns whether the first value is greater than another or not. For two encryption values E(u) and E(v), the ASC protocol returns E(1) if $u \leq v$; otherwise it returns E(0). The procedure of the ASC protocol is described in Algorithm 1. First, C_A selects two random numbers and their encryption values, <r_a, E(r_a)>, <r_b, E(r_b)>, from the random value pool (line 1). Second, C_A computes $E(u') = E(r_a \times u + r_b) = E(u)^{r_a} \times E(r_b)$ and $E(v') = E(r_a \times v + r_b) = E(v)^{r_a} \times E(r_b)$ by using the Paillier cryptosystem (lines 2∼3). Third, C_A randomly selects one of two functions, F_0: $u \leq v$ and F_1: $u < v$. The chosen function cannot expose to C_B. Then, C_A sends data to C_B, according to the chosen function. If F_0: $u < v$ is chosen, C_A transmits <E(u'), E(v')> to C_B. If F_1: $u < v$ is chosen, C_A transmits <E(v'), E(u')> to C_B. Fourth, C_B receives a pair of encryption <E(α), E(β)> from C_A (line 9). C_B decrypts E(α) and E(β). Because C_B does not know the sequence of data received from C_A, C_B cannot determine whether <α, β> is <u',v'> or <v',u'>. If α is less than or equal to β, C_B sends E(γ) = E(1) to C_A. Otherwise, C_B sends E(γ) = E(0) to C_A (lines 10∼16). Finally, C_A receives E(γ) from C_B. If F_0: $u \leq v$ is selected, C_A returns E(γ). Otherwise, C_A returns SBN(E(γ)) because C_A sends <E(v'), E(u')>, which is the reverse order of the original pair <E(u'), E(v')> (lines 18∼21). The procedure of the ASC protocol is presented in Algorithm 1.

For example, C_A compares E(u) = E(5), E(v) = E(10) by using the ASC protocol. First, C_A picks the random value r_a = 3, r_b = 2 in the random value pool. C_A calculates E(u') = $E(u)^{r_a} \times E(r_b) = E(r_a \times u + r_b) = E(3 \times 5 + 2) = E(17), E(v') = E(v)^{r_a} \times E(r_b) = E(r_a \times v + r_b) = E(3 \times 10 + 2) = E(32)$. Second, C_A selects one of two functions, F_0 and F_1. If we assume that C_A chooses F_1, the sequence of data is <E(v'), E(u')>. Therefore, C_A transmits <E(32), E(17)> to C_B. Third, C_B sets <α, β> to <32, 17> by decrypting <E(32), E(17)>. Because α is greater than β, γ is set to 0. C_B encrypts γ and sends E(γ) = E(0) to C_A. Finally, C_A receives E(γ) = E(0) from C_B. Because C_A selects F_1, C_A returns SBN(E(γ)) = E(1).

Advanced Secure Range Overlapping (ASRO) protocol. The ASRO protocol determines whether two encrypted ranges are overlapped or not. A region is denoted by the lower bound (lb) and the upper bound (ub). Figure 3 presents the overlapping and the non-overlapping between two regions (i.e., $range_1$ and $range_2$) in the ASRO protocol. The $range_1$ and $range_2$ are overlapped if range$_2$.lb $\leq$ range$_1$.lb $\leq$ range$_2$.ub or range$_2$.lb $\leq$ range$_1$.ub $\leq$ range$_2$.ub for all dimensions. For two encrypted range values E($range_1$) and E($range_2$), the ASRO protocol returns E(1) if $range_1$ and $range_2$ are overlapped; otherwise it returns E(0).

Algorithm 1 ASC Protocol.

Input: $E(u), E(v)$
Output: if $u \leq v$, return E(1), otherwise, return E(0)

C_A:
01. pick two pairs <r_a, E(r_a)> and <r_b, E(r_b)> in random value pool
02. E(u') $\leftarrow E(u)^{r_a} \times E(r_b)$
03. E(v') $\leftarrow E(v)^{r_a} \times E(r_b)$
04. randomly selected F_0 or F_1
05. if F_0 is selected then
06. Transmit <E(u'), E(v')> to C_B
07. else if F_1 is selected then
08. Transmit <E(v'), E(u')> to C_B

C_B :
09. receive E(α), E(β) // C_B cannot know any information from C_A.
10. $\alpha \leftarrow$ D(E(α))
11. $\beta \leftarrow$ D(E(β))
12. if $\alpha \leq \beta$
13. E(γ) $\leftarrow$ E(1)
14. else
15. E(γ) $\leftarrow$ E(0)
16. Transmit $E(\gamma)$ to C_A

C_A :
17. receive $E(\gamma)$
18. if F_0 is selected, then
19. return E(γ)
20. else
21. return SBN(E(γ))

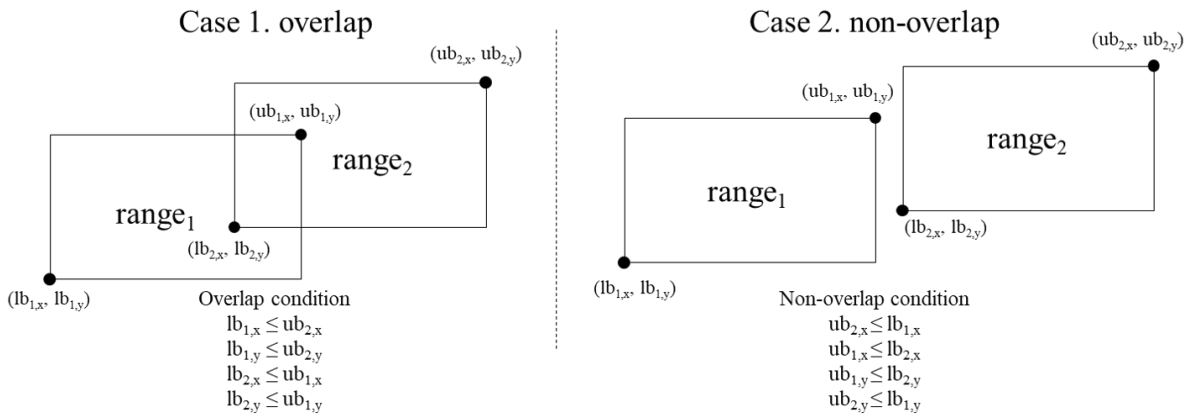


Figure 3. Overlapping and non-overlapping between two regions in the ASRO protocol.

First, C_A initializes E(α) as E(1) (line 1). Second, for all dimensions, C_A performs ASC(E($range_1.lb_i$), E($range_2.ub_i$)), where $1 \leq i \leq$ # of dimensions, and stores the result of the ASC protocol into E(β). The ASC protocol is used to check whether the lower bound of $range_1$ is less than the upper bound of $range_2$ for each dimension. Then, C_A performs SM(E(α), E(β)) [19] and stores its result into E(α) (lines 2∼4). Third, for all dimensions, C_A performs ASC(E($range_2.lb_i$), E($range_1.ub_i$)) and stores its result into E(γ). The ASC protocol is used to check whether the lower bound of $range_2$ is less than the upper bound of $range_1$ for each dimension. Then, C_A performs SM(E(α), E(γ)) and stores its result into E(α) (lines 5∼7). The SM protocol is used to determine whether the point is included into a range for all dimensions. Finally, C_A return E(α) as the result of the ASRO protocol. The procedure of the ASRO protocol is presented in Algorithm 2.

Algorithm 2 ASRO Protocol.

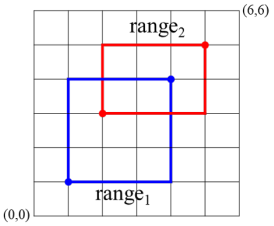
Input: E($range_1$), E($range_2$) (range consists of <$range.lb_i$, $range.ub_i$> $1 \leq i \leq m$, m is the number of dimension)

Output: if $range_1$ and $range_2$ is overlapped, return E(1), otherwise, return E(0)

C_A:

01. E(α) $\leftarrow$ E(1)
02. for $1 \leq i \leq m$
03. E(β) $\leftarrow$ ASC(E($range_1.lb_i$), E($range_2.ub_i$))
04. E(α) $\leftarrow$ SM(E(β), E(α))
05. for $1 \leq i \leq m$
06. E(β) $\leftarrow$ ASC(E($range_1.lb_i$), E($range_2.ub_i$))
07. E(α) $\leftarrow$ SM(E(β), E(α))
08. return E(α)

For example, in Figure 4, C_A checks the overlap between E($range_1$) and E($range_2$) by using the ASRO protocol. First, C_A sets E(α) as E(1). Second, for the x-axis, C_A calculates ASC(E($range_1.lb_x$), E($range_2.ub_x$)) = ASC(E(1), E(5)) = E(1) and stores its result into E(β). C_A calculates SM(E(α), E(β)) = SM(E(1), E(1)) = E(1) and stores it into E(α). For the y-axis, C_A calculates ASC(E($range_1.lb_y$), E($range_2.ub_y$)) = ASC(E(1), E(5)) = E(1) and stores it into E(β). C_A calculates SM(E(α), E(β)) = SM(E(1), E(1)) = E(1) and stores it into E(α). Third, for the x-axis, C_A calculates ASC(E($range_2.lb_x$), E($range_1.ub_x$)) = ASC(E(2), E(4)) = E(1) and stores it into E(γ). C_A calculates SM(E(α), E(γ)) = SM(E(1), E(1)) = E(1) and stores it in E(α). For the y-axis, C_A calculates ASC(E($range_2.lb_y$), E($range_1.ub_y$)) = ASC(E(3), E(4)) = E(1) and stores it into E(γ). C_A calculates SM(E(α), E(γ)) = SM(E(1), E(1)) = E(1) and stores it into E(α). Finally, C_A returns E(α) = E(1) as the result of the ASRO protocol.



	Lower bound		Upper bound	
	x-axis	y-axis	x-axis	y-axis
$range_1$	1	1	4	4
$range_2$	2	3	5	5

Figure 4. Overlapping between E($range_1$) and E($range_2$).

Advanced Secure Point Enclosure (ASPE) protocol. The ASPE protocol determines whether a point is included in the given range or not. A point p is included in a range if $range.lb \leq p \leq range.ub$ for all dimensions. When an encrypted point value E(p) and an encrypted range E(range) are given, the ASPE protocol returns E(1) if the range includes the point(p); otherwise, it returns E(0). First, C_A assigns E(1) to E(α) (line 1). Second, for each dimension i, C_A executes E(β) = ASC(E(p_i), E($range.ub_i$)) and E(α) = SM(E(α), E(β)) (lines 2∼4). The ASC protocol is used to check whether p is less than the upper bound of the range for each dimension. The SM protocol is used to determine whether the point is included into a range for all dimensions. Third, for each dimension i, C_A performs E(γ) = ASC(E($range.lb_j$), E(p_j)), and E(α) = SM(E(α),E(γ)) (lines 5∼7). Finally, C_A returns E(α) (line 8). We skip the example of the ASPE protocol because the process of the ASPE protocol is identical to that of the ASRO protocol, except that the input is a point, not a range. The procedure of the ASPE protocol is presented in Algorithm 3.

Algorithm 3 ASPE Protocol.

Input: E(p), E(range) (range consists of <$range.lb_i, range.ub_i$> $1 \leq i \leq m$, m is the number of dimension)
Output: if range includes p, return E(1), otherwise, return E(0)

C_A:
01. E(α) $\leftarrow$ E(1)
02. for $1 \leq i \leq m$
03. E(β) $\leftarrow$ ASC(E(p), E($range.ub_i$))
04. E(α) $\leftarrow$ SM(E(β), E(α))
05. E(γ) $\leftarrow$ ASC(E($range.lb_i$), E(p))
06. E(α) $\leftarrow$ SM(E(α), E(γ))
07. return E(α)

4. Privacy-Preserving Top-*k* Query Processing Algorithm

In this section, we suggest a privacy-preserving Top-k query processing algorithm that uses new secure protocols, such as the ASC, ASRO, and ASPE protocols mentioned in Section 3. The proposed Top-k query processing algorithm retrieves k data items that have the highest score for a scoring function over the encrypted database. The algorithm is organized in three phases; node data search phase, Top-k retrieval phase, and Top-k result refinement phase.

4.1. Node Data Search Phase

While hiding the data access patterns, C_A securely extracts all the data items from a node containing the highest score. The procedure of the node data search phase is presented in Algorithm 4. First, for the extraction of data items related to the query, C_A securely calculates a max point (mp) = <$mp_1, mp_2, \ldots, mp_m$>, where mp_j is the highest score for dimension j($1 \leq j \leq m$) (lines 1∼4). To calculate mp, C_A adds E($hint$) and E($coef_i$) for $1 \leq i \leq m$, where m is the number of dimensions. $hint$ means the largest absolute value among the coefficients. mp_i is determined according to the sign of the coefficients for the ith dimension. If $coef_i + hint$ is greater than hint, $coef_i$ is positive and so mp_i is set to E(max_i). Otherwise, mp_i is set to E(0). By finding the node including mp, C_A can extract the data items that have the possibility of the highest score.

Second, C_A finds a node including mp by executing E(δ_i) = ASPE(E(mp), E($node_i$)) for $1 \leq i \leq \#_of_nodes$, where $\#_of_nodes$ means the total number of the leaf nodes in the kd-tree (lines 5∼6). The result of the ASPE protocol for all leaf nodes, i.e., E(δ), is <E(δ_1), E(δ_2), …, E($\delta_{\#_of_node}$)>. Because our Top-k query processing algorithm utilizes the ASPE protocol, it can achieve a better performance than the existing algorithms [27,28]. In contrast, the existing algorithms cause a large computational overhead because they perform an iterative operation as many times as the bit length of data. Third, C_A make E(δ') by shuffling E(δ) using a random permutation function π. Then, C_A sends E(δ') to C_B (lines 7∼8).

Fourthly, by decrypting E(δ'), C_B counts how many δ'_j has 1 (i.e., c) (lines 9∼10). Fifthly, C_B generates c number of node groups (NG) (line 11). C_B assigns a node with $\delta'_j = 1$ into a different group among c node groups. C_B evenly assigns the remaining nodes with $\delta'_j = 0$ into c node groups (lines 12∼14). Then, C_B randomly shuffles the ids of the nodes in each node group and sends to C_A the shuffled NG, i.e., NG' (line 15).

Fifthly, C_A obtains NG^* by deshuffling NG' using π^{-1} (line 17∼18). Finally, for all the data items of each node for each NG^*, C_A performs E($cand_{w,t}$) = E($cand_{w,t}$) × SM(E($node_z.data_{s,t}$), E(δ_z)) where $1 \leq i \leq c$, $1 \leq j \leq \#_of_nodes$ in the selected NG^*_i, $1 \leq s \leq FanOut$ and $1 \leq t \leq m$ (lines 19∼25). Here, z means the id of the jth nodes of NG^*_i and w means $cnt + s$ where cnt is the number of data items extracted from encrypted leaf nodes of the kd-tree. E(δ_z) is the result of the ASPE protocol corresponding to $node_z$. By repeating these steps with an updated cnt, C_A returns <E($cand_1$), E($cand_2$), …, E($cand_{cnt}$)>. Consequently, all the data items in a node, including mp, are securely obtained without exposing the data access patterns [10,11].

Algorithm 4 Node data search phase.

Input: <E($node_1$), E($node_2$), ..., E($node_p$) | p = #_of_node >, E($coef$)=<E($coef_1$), E($coef_2$), ..., E($coef_m$) | $coef$ means coefficient of score function and m = #_of_dimension>, <E(max_1), E(max_2), ..., E(max_m) | max_i is the maximum data domain of i dimension >, E($hint$) // hint means the largest of the absolute values among coefficients

Output: <E($cand_1$), E($cand_2$), ..., E($cand_{cnt}$)> // all candidates inside nodes being related to a query

C_A:

01. for $1 \leq i \leq m$
02. E($coef_i + hint$) $\leftarrow$ E($coef_i$)$\times$E($hint$)
03. E(ϵ_i) $\leftarrow$ ASC(E($coef_i + hint$), E($hint$))
04. E(mp_i) $\leftarrow$ SM(E(ϵ_i), E(0)) $\times$ SM(SBN(E(ϵ_i)), E(max_i))
05. for $1 \leq i \leq p$ // $p = 2^{h-1}$ where h means the level of the kd-tree
06. E(δ_i) $\leftarrow$ ASPE(E(mp), E($node_i$))
07. E(δ') $\leftarrow$ π(E(δ)) // shuffle the order of array E(δ) = <E(δ_1), E(δ_2), ..., E(δ_p)>
08. send E(δ') to C_B

C_B:

09. $\delta' \leftarrow$ D(E(δ'))
10. $c \leftarrow$ the number of '1' in δ'
11. generate c number of Node Group such that NG = <NG_1, NG_2, ..., NG_c>
12. for $1 \leq i \leq c$
13. assign into NG_i a node with δ' = 1 and $\frac{\#_of_node}{c} - 1$ nodes with δ' = 0
14. $NG'_i \leftarrow$ shuffle the ids of nodes in NG_i
15. send NG' = <NG'_1, NG'_2, ..., NG'_c> to C_A

C_A:

16. $cnt \leftarrow 0$ // cnt is # of data items extracted from the encrypted leaf nodes of kd-tree
17. for $1 \leq i \leq c$
18. $NG^*_i \leftarrow$ shuffle node ids using π^{-1} for each NG'_i // NG^* = <NG^*_1, NG^*_2, ..., NG^*_c>
19. for $1 \leq i \leq c$
20. for $1 \leq j \leq num$ // num = #_of_nodes in the selected NG^*_i
21. z = id of jth nodes of NG^*_i
22. for $1 \leq s \leq FanOut$ //$FanOut$: # of data items in each nodes
23. for $1 \leq t \leq m$
24. if ($s == 1$)
25. E($cand_{w,t}$) $\leftarrow$ E(0) where $w = cnt + s$ // E($cand_w$) = <E($cand_{w,1}$), E($cand_{w,2}$), ..., E($cand_{w,m}$)>
26. E($cand_{w,t}$) $\leftarrow$ E($cand_{w,t}$)$\times$SM($node_z.data_{s,t}$, E(δ_z))
27. $cnt \leftarrow cnt + FanOut$
28. send <E($cand_1$), E($cand_2$), ..., E($cand_{cnt}$)>

Figure 5 presents an example of the node data search phase. The example reuses the same data items in Figure 2. First, C_A executes the ASPE protocol between E($Node_i.Range$) and E(mp) for $1 \leq i \leq$ #_of_node. C_A sets the result of the ASPE protocol as E(α) and transmits the result to C_B. In Figure 5, for $Node_1$, C_A performs the ASPE protocol between E($Node_1.Range$) = E(lb) = <E(0), E(0)>, E(ub) = <E(6), E(5)> and E(mp) = <E(10), E(10)>, and sets the result of the ASPE protocol, i.e., E(0), as E(α_1). C_A obtains the result of the ASPE protocol as <$Node_1$, E(0)>, <$Node_2$, E(0)>, <$Node_3$, E(0)>, <$Node_4$, E(1)>.

Second, C_A permutates the order of <$Node_1$, E(α_1)>, <$Node_2$, E(α_2)>, ..., <$Node_{\#_of_node}$, E($\alpha_{\#_of_node}$)> and assigns new ids based on the shuffled order of nodes, so that C_A conceals the original node ids from C_B. To recover the original node ids, C_A stores the pairs of <the original id, the new id>. For example, in Figure 5, the original order of <$Node_1$, E(0)>, <$Node_2$, E(0)>, <$Node_3$, E(0)>, <$Node_4$, E(1)> is permutated to <$Node_4$, E(1)>, <$Node_1$, E(0)>, <$Node_2$, E(0)>, <$Node_3$, E(0)>. Then, C_A converts $Node_4$, $Node_1$, $Node_2$, and $Node_3$ into

PN_1, PN_2, PN_3, and PN_4, respectively. The permutated order is <PN_1, E(1)>, <PN_2, E(0)>, <PN_3, E(0)>, <PN_4, E(0)>. Then, C_A sends the permutated order to C_B.

Third, C_B obtains the permutated order and decrypts it. In Figure 5, C_B obtains <PN_1, E(1)>, <PN_2, E(0)>, <PN_3, E(0)>, <PN_4, E(0)>, and gets <PN_1, 1>, <PN_2, 0>, <PN_3, 0>, <PN_4, 0> by decrypting it. To create node groups, C_B counts how many 1s are in the order. Each node group has one core node whose α'_p equals to 1, where $1 \leq p \leq \#_of_node$. Nodes whose α'_p equals to 0, where $1 \leq p \leq \#_of_node$, are uniformly designated in the node groups. Additionally, C_B transmits the node groups to C_A. For example, C_B counts how many 1s are in the order of <PN_1, 1>, <PN_2, 0>, <PN_3, 0>, <PN_4, 0>, and generates a node group with the core node (PN_1). The nodes <PN_3, 0>, <PN_2, 0>, and <PN_4, 0> are designated in the node group. C_B transmits the node group, i.e., PN_1, PN_3, PN_2, PN_4, to C_A.

Fourth, C_A recovers the original node ids by using the pairs of <the original id, the new id>. In Figure 5, using <$Node_1$, PN_2>, <$Node_2$, PN_3>, <$Node_3$, PN_4>, <$Node_4$, PN_1>; C_A gains $Node_4$, $Node_2$, $Node_1$, $Node_3$ as the original node ids. Fifth, C_A executes the SM protocol between the encrypted data item in a node group and E(α). C_A executes SM(E(1), E($Node_4.Data$)), SM(E(0), E($Node_2.Data$)), SM(E(0), E($Node_1.Data$)), and SM(E(0), E($Node_3.Data$)). The results of the SM protocol are E(d_{16}), E(0), E(0), E(0), and C_A stores E(d_{16}) in the candidate set by merging the results. Sixth, for each node group, the algorithm performs the steps 5∼7 as many times as the # of data items. In Figure 5, by merging the results, C_A obtains E(d_{16}), E(d_{17}), E(d_{18}), E(d_{19}), E(d_{20}).

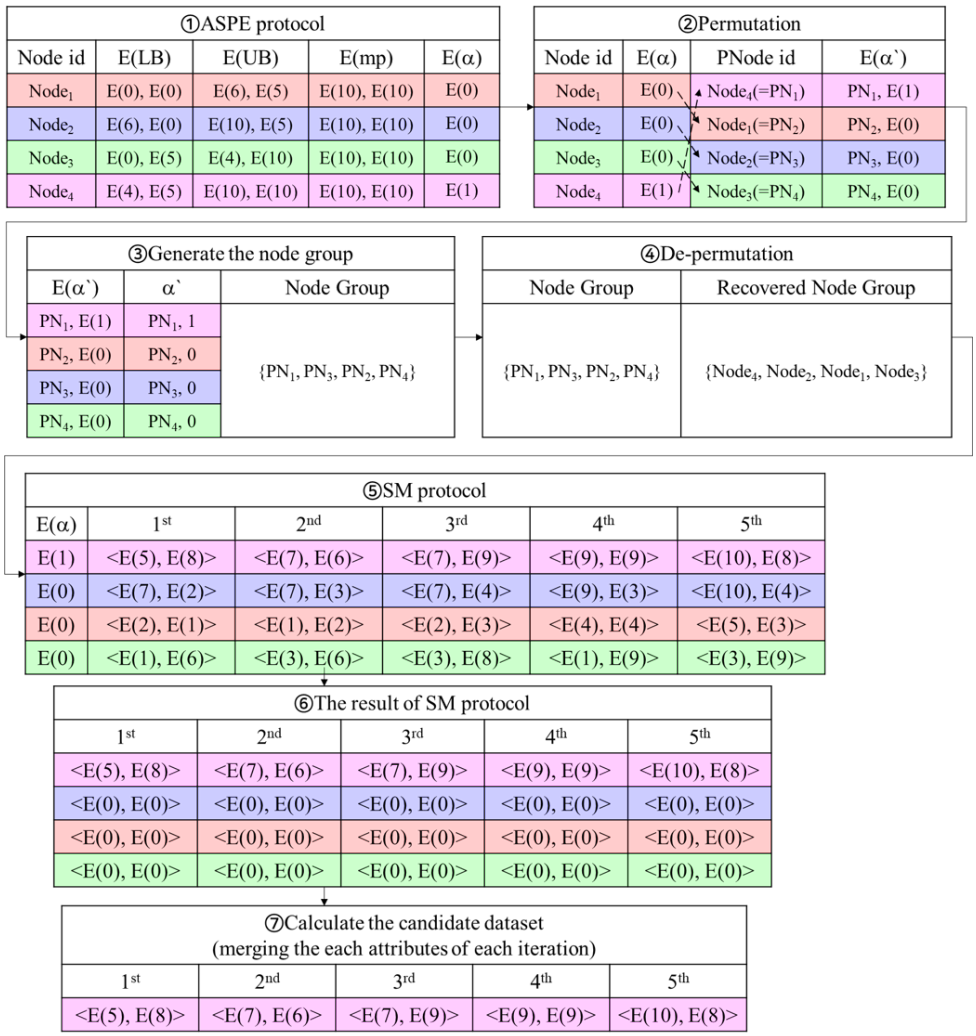


Figure 5. Example in node data search phase.

4.2. Top-k Retrieval Phase

From the candidates obtained in the previous phase, the algorithm retrieves k number of data items with the highest score. The procedure of the Top-k retrieval phase is as follows. First, C_A calculates the score by using a score function that is represented by the encrypted coefficients($query$). Second, C_A finds the maximum score, i.e., $score_{max}$, among the calculated score. An attacker cannot distinguish which data item has $score_{max}$ because of the Paillier cryptosystem [32]. Third, to mark the encrypted data item with $score_{max}$, C_A subtracts $score_{max}$ from the score. If the result of subtraction equals E(0), the data items have the maximum score. Because E(0) is unchanged by the homomorphic multiplication in the Paillier cryptosystem, the algorithm securely obtains Top-k. Fourth, the algorithm also executes the homomorphic multiplication of the result by a random value, so as to conceal the original data. To hide the data access patterns, the algorithm permutates the order of the result. Finally, the algorithm finds a data item with the highest score and replays the above process until k number of results are searched. The pseudo code of the Top-k retrieval phase is presented in Algorithm 5. First, using a score function (SF), C_A securely computes the score E($score_i$) between E($coef$) and E($cand_i$) for $1 \leq i \leq cnt$ (lines 1∼2). Second, C_A executes SMAX$_n$ to obtain the maximum value E($score_{max}$) among E($score_i$) for $1 \leq i \leq cnt$ (lines 3∼4). Third, C_A computes E(τ_i) = E($score_{max}$) × E($score_i$)$^{N-1}$, for $1 \leq i \leq cnt$. C_A calculates E(τ'_i) = $E(\tau_i)^{r_i}$, where r_i means a random value for $1 \leq i \leq cnt$. C_A obtains E(λ) by permutating E(τ') and transmits E(λ) to C_B (lines 5∼9). Fourthly, after decrypting E(λ), C_B stores E(U_i) = E(1) if E(λ_i) = 0 for $1 \leq i \leq cnt$. Otherwise, C_B sets E(U_i) = E(0). C_B sends E(U) = <E(U_1), E(U_2), …, E(U_{cnt})> to C_A (lines 10∼13). Fifthly, C_A obtains E(V) by shuffling E(U) using π^{-1}. Then, C_A does SM(E(V_i), E($cand_{i,j}$)) to obtain E($V'_{i,j}$) where $1 \leq i \leq cnt$ and $1 \leq j \leq m$ (lines 14∼17). Sixthly, C_A sets E($score_i$) with the highest score to E(0) by computing Equation (4) (lines 18∼19). Because the highest score is set to E(0) and the other scores are unchanged, the algorithm can obtain the Top-k result while preventing the same result from being selected more than twice.

$$E(score_i) \;=\; SM(E(V_i),\, E(0)) \;\times\; SM(SBN(E(V_i)),\, E(score_i)) \tag{4}$$

Finally, by calculating E($topk_{s,j}$) = $\prod_{i=1}^{cnt} E(V'_{i,j})$ for $1 \leq j \leq m$ and $1 \leq s \leq k$, C_A can securely extract the data item corresponding to the E($score_{max}$) (lines 20∼21). This procedure is replayed k times to search the Top-k result.

Figure 6 presents an example of the Top-k retrieval phase. The example reuses the data items in Figure 2. The permutated function π is bypassed. First, C_A computes the score by using a score function (SF), and stores the result in E($score_i$) for $1 \leq i \leq cnt$ (①). In Figure 6, C_A executes SF(E($cand_1$) = <E($cand_{1,x-axis}$), E($cand_{1,y-axis}$)>, E(coef) = <E($coef_1$), E($coef_2$)>) = SF(<E(5), E(8)>, <E(3), E(2)>) = E(31), and stores it into E($score_1$). Second, the highest score is calculated by using the SMAX$_n$. C_A stores SMAX$_n$(E($score_1$), E($score_2$), E($score_3$), E($score_4$), E($score_5$)) = SMAX$_n$(E(31), E(38), E(39), E(45), E(46)) = E(46) in E($score_{max}$) (②). Third, to gain the encrypted data item with the highest score, C_A stores E($score_{max}$-$score_i$) in E(τ_i) for $1 \leq i \leq cnt$ (③). If $score_{max}$ is the same as $score_i$, E(τ_i) is set to E(0). For E($score_5$), C_A stores E(46-46) = E(0) into E(τ_5). Fourth, C_A executes the homomorphic multiplication of E(τ_i) by a random value to prevent the leakage of sensitive data(④). For E(τ_1), when a random value = 2, C_A stores E(15×2) = E(30) into E(τ'_1). Fifth, for $1 \leq i \leq cnt$, if E(τ'_i) is E(0), C_B sets E(V_i) to E(1); otherwise, C_B sets E(V_i) to E(0) (⑤). Sixth, C_A obtains Top-k by doing the SM protocol between E($cand_i$) and E(V_i) for $1 \leq i \leq cnt$ and summing the result of the SM protocol up (⑥∼⑦). In Figure 6, for the x-axis, C_A performs SM(E(V_1 = 0), E($cand_{1,x-axis}$ = 5)), SM(E(V_2 = 0), E($cand_{2,x-axis}$ = 8)), SM(E(V_3 = 0), E($cand_{3,x-axis}$ = 7)), SM(E(V_4 = 0), E($cand_{4,x-axis}$ = 9)), and SM (E(V_5 = 1), E($cand_{5,x-axis}$ = 10)). For the y-axis, C_A performs SM(E(V_1 = 0), E($cand_{1,y-axis}$ = 8)), SM(E(V_2 = 0), E($cand_{2,y-axis}$ = 7)), SM(E(V_3 = 0), E($cand_{3,y-axis}$ = 9)), SM(E(V_4 = 0), E($cand_{4,y-axis}$ = 9)), and SM (E(V_5 = 1), E($cand_{5,y-axis}$ = 8)). C_A sums E(0), E(0), E(0), E(0), and E(10) for the x-axis while summing E(0), E(0), E(0), E(0), and E(8) for the y-axis. Thus, C_A gains <E(10), E(8)> as the Top-k

result. Seventh, using Equation (4), C_A stores the score of the searched Top-k result into E(0) so that C_A can avoid selecting the same Top-k data for the next time. In Figure 6, C_A stores SM(E(0), E(1))× SM(E(46), E(0)) = E(0) into E($score_5$). Finally, C_A replays the previous procedure until the Top-k data is searched (②∼⑧).

Algorithm 5 Top-k retrieval phase.

Input: <E($cand_1$), E($cand_2$), …, E($cand_{cnt}$) | cnt = #_of_$candidates$ >, E($coef$), k // $coef$ means coefficient of score function and k means the number of Top-k

Output: <E($topk_1$), E($topk_2$), …, E($topk_k$)> // temporary Top-k results

C_A:

01. for $1 \leq i \leq cnt$
02. E($score_i$) ← SF(E($coef$), E($cand_i$)) // SF is a score function for Top-k
03. for $1 \leq s \leq k$
04. E($score_{max}$) ← SMAX$_n$(E($score_1$), E($score_2$), …, E($score_{cnt}$))
05. for $1 \leq i \leq cnt$
06. E(τ_i) ← E($score_{max}$)×E($score_i$)$^{N-1}$
07. E(τ'_i) ← $E(\tau_i)^{r_i}$ // r_i means random value for i
08. E(λ) ← $\pi(\tau')$ // τ'=<τ'_1, τ'_2, …, τ'_{cnt}>
09. send E(λ)=<E(λ_1), E(λ_2), …, E(λ_{cnt})> to C_B

C_B:

10. for $1 \leq i \leq cnt$
11. if D(λ_i) = 0 then E(U_i) ← E(1)
12. else E(U_i) ← E(0)
13. send E(U)=<E(U_1), E(U_2), …, E(U_{cnt})> to C_A

C_A:

14. E(V) ← π^{-1}(U) // E(V)=<E(V_1), E(V_2), …, E(V_{cnt})>
15. for $1 \leq i \leq cnt$
16. for $1 \leq j \leq m$ // m means #_of_$dimension$
17. E($V'_{i,j}$) ← SM(E(V_i), E($cand_{i,j}$)) // E($cand_i$)=<E($cand_{i,1}$), E($cand_{i,2}$), …,E($cand_{i,m}$)>
18. if $s < k$
19. E($score_i$) ← SM(E(V_i), E(0))×SM(SBN(E(V_i)), E($score_i$))
20. for $1 \leq j \leq m$
21. E($topk_{s,j}$) ← $\prod_{i=1}^{cnt} E(V'_{i,j})$ // E($topk_s$)=<E($topk_{s,1}$), E($topk_{s,2}$), …, E($topk_{s,m}$)>
22. E($topk$) ← E($topk$) ∪ E($topk_s$)
23. Return E($topk$)=<E($topk_1$), E($topk_2$), …, E($topk_k$)>

4.3. Top-k Result Refinement Phase

The Top-k result refinement confirms the correctness of the current Top-k result. Especially, the neighboring nodes must be searched to obtain data items with the higher score than the criteria. Therefore, we calculate the max point of ith node (mp_node_i = <$mp_node_{i,1}$, $mp_node_{i,2}$, …, $mp_node_{i,m}$>, where $mp_node_{i,j}$ is the highest score for dimension j($1 \leq j \leq m$)), such that the mp_node_i is a point in the ith node whose score is the highest for the given query ($coef$). If the coefficient for the jth dimension is positive, C_A stores the upper bound value of the ith node into $mp_node_{i,j}$. Otherwise, C_A stores the lower bound value of the ith node into $mp_node_{i,j}$. If the score of mp_node_i is greater than the criteria, C_A extracts the data items from the ith node. Finally, after searching all the nodes, C_A recalculates the final Top-k result in the same way as Algorithm 5. The process of the Top-k result refinement phase is presented in Algorithm 6. First, C_A calculates E($criteria$) = SF(E($coef$), E($topk_k$)) to gain the minimum score between the Top-k result and the query (line 1). Second, for each node, C_A executes SM(E(ϵ_j), E($node_i.lb_j$))×SM(SBN(E(ϵ_j)), E($node_i.ub_j$)) for $1 \leq i \leq$ #_of_$nodes$ and $1 \leq j \leq m$, and stores the result into E($mp_node_{i,j}$) (lines 2∼4). E(ϵ_j) is the value computed by the ASC protocol for the jth dimension in the node data search phase. C_A securely obtains the max

point of the ith node, i.e., E(mp_node_i). Fourthly, C_A calculates the highest score between the query and E(mp_node_i) by using the score function (SF), i.e., E(hs_node_i) (line 5). To prevent the same node from being selected more than twice, C_A securely computes E(hs_node_i) = SM(E(δ_i), E(0))×SM(SBN(E(δ_i)), E(hs_node_i)), where E(δ_i) is the value returned by the ASPE protocol, and sets E(hs_node_i) to E(0), where the node has already been retrieved in the node data search phase (line 6). By performing E(δ_i) = ASC(E(criteria), E(hs_node_i)), C_A sets E(δ_i) to E(1) for the ith node if E(criteria) is less than E(hs_node_i). Otherwise, C_A sets E(δ_i) to E(0) (line 7). Fifthly, C_A securely obtains the data items stored in nodes with E(δ_i) = E(1) and generates E($cand'$) by merging them with E($topk$) (lines 8∼9). Then, C_A executes the Top-k retrieval phase based on E($cand'$) to calculate the final Top-k result E($topk_i^*$) for $1 \leq i \leq k$ (line 10). Sixthly, to hide the Top-k result from C_B, C_A calculates E($\gamma_{i,j}$) = E($topk_i^*$) × E($r_{i,j}$) for $1 \leq i \leq k$ and $1 \leq j \leq m$ using a random value $r_{i,j}$. Then, C_A transmits E($\gamma_{i,j}$) to C_B and $r_{i,j}$ to AU (lines 18∼22). Seventhly, C_B decrypts E($\gamma_{i,j}$) and transmits the decrypted value to AU (lines 23∼26). Finally, AU gains the plaintexts of the Top-k result by calculating $\gamma_{i,j}$-$r_{i,j}$ (lines 27∼29). As a result, AU can reduce the computation overhead without using decryption operations.

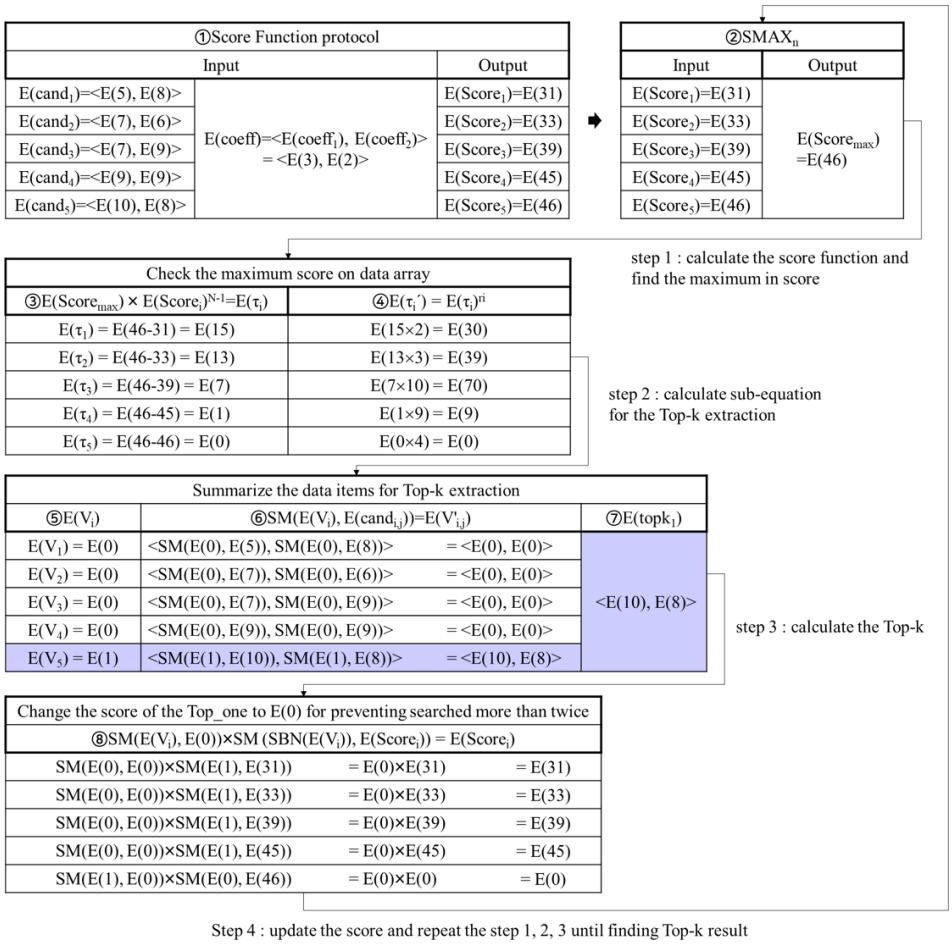


Figure 6. Example of Top-k retrieval phase.

Algorithm 6 Top-k result refinement phase.

Input: <E($node_1$), E($node_2$), …, E($node_p$) | p = #_of_$node$ >, <E($topk_1$), E($topk_2$), …, E($topk_k$)>, E($coef$), k // $coef$ means coefficient of score function and k means the number of Top-k

Output: $topk^*$=<$topk_1^*$, $topk_2^*$, …, $topk_k^*$> // final Top-k results

C_A:

01. E(criteria) = SF(E($coef$), E($topk_k$))
02. for $1 \leq i \leq p$
03. for $1 \leq j \leq m$
04. E($mp_node_{i,j}$)←SM(E(ϵ_j), E($node_i.lb_j$))×SM(SBN(E(ϵ_j)), E($node_i.ub_j$)) // E(δ_i) is value returned by ASPE protocol in line 6 of Algorithm 4
05. E(hs_node_i) ← SF(E(mp_node_i), E($coef$))
06. E(hs_node_i) ← SM(E(δ_i), E(0))× SM(SBN(E(δ_i)), E(hs_node_i)) // E(δ_i) is value returned by ASPE protocol in line 6 of Algorithm 4
07. E(δ_i) ← ASC(E(criteria), E(hs_node_i))
08. E($cand'$) ← perform 7 ∼ 27 lines of Algorithm 4 with <E($node_1$), E($node_2$), …, E($node_p$)>, E(δ) = <E(δ_1), E(δ_2), …, E(δ_p)>
09. E($cand'$) ← E($cand'$)∪<E($topk_1$), E($topk_2$), …, E($topk_k$)>
10. E($topk^*$) ← perform Algorithm 5 with E($cand'$), E($coef$) and k
11. for $1 \leq i \leq k$
12. for $1 \leq j \leq m$
13. pick up the random value $r_{i,j}$ // r_i=<$r_{i,1}$, $r_{i,2}$, …, $r_{i,m}$>
14. E($\gamma_{i,j}$) ← E($topk_{i,j}^*$)×E($r_{i,j}$) // E($topk_i^*$)=<E($topk_{i,1}^*$), E($topk_{i,2}^*$), …, E($topk_{i,m}^*$)> and E(γ_i)=<E($\gamma_{i,1}$), E($\gamma_{i,2}$), …,E($\gamma_{i,m}$)>
15. send <E(γ_1), E(γ_2), …, E(γ_k) > to C_B and <r_1, r_2, …, r_k> to AU // AU means authorized user

C_B:

16. for $1 \leq i \leq k$
17. for $1 \leq j \leq m$
18. $\gamma_{i,j}$ ← D(E($\gamma_{i,j}$)) // γ_i = <$\gamma_{i,1}$, $\gamma_{i,2}$, …,$\gamma_{i,m}$>
19. send <γ_1, γ_2, …, γ_k> to AU

AU:

20. for $1 \leq i \leq k$
21. for $1 \leq j \leq m$
22. $topk_{i,j}^*$ ← $\gamma_{i,j}$ - $r_{i,j}$ //$topk_i^*$ = <$topk_{i,1}^*$, $topk_{i,2}^*$, …, $topk_{i,m}^*$>

Figure 7 presents an example of the Top-k result refinement phase. First, C_A calculates the max point of the ith node (mp_node_i)(①). For $node_1$, C_A performs mp_node_1 = <E($mp_node_{1,x-axis}$), E($mp_node_{1,y-axis}$)> = <SM(E(0), E(0))×SM(E(1), E(6)), SM(E(0), E(0))×SM(E(1), E(5))> = <E(6), E(5)>. C_A calculates the same operation for $node_2$, $node_3$, and $node_4$. Second, C_A calculates the highest score in the ith node(hs_node_i) (②). For the highest score of $node_1$, C_A performs E(hs_node_1) = SM(E(3), E(6))×SM(E(2), E(5)) = E(28). C_A computes the highest scores for $node_2$, $node_3$, and $node_4$. Third, to prevent the same node from being selected more than twice, C_A sets E(hs_node_i) to E(0) because $node_i$ has been already searched in the node data search phase (③). For $node_4$, C_A performs E(hs_node_4) = SM(E(1), E(0))×SM(E(0), E(50)) = E(0). Fourth, for node expansion, C_A checks whether the ith node needs to be searched by performing the ASC protocol between E(criteria) and E(hs_node_i) for $1 \leq i \leq$ #_of_$node$(④). For $node_2$, C_A performs E(δ_2) = ASC(E(criteria), E(hs_node_2)) = ASC(E(38), E(40)) = E(1), where E(criteria) equals E(38). Fifth, C_A extracts all the data items in the expansion node($node_2$) by using the node data search phase (⑤). Finally, C_A obtains the final Top-k results by performing the Top-k retrieval phase in Figure 7 (⑥).

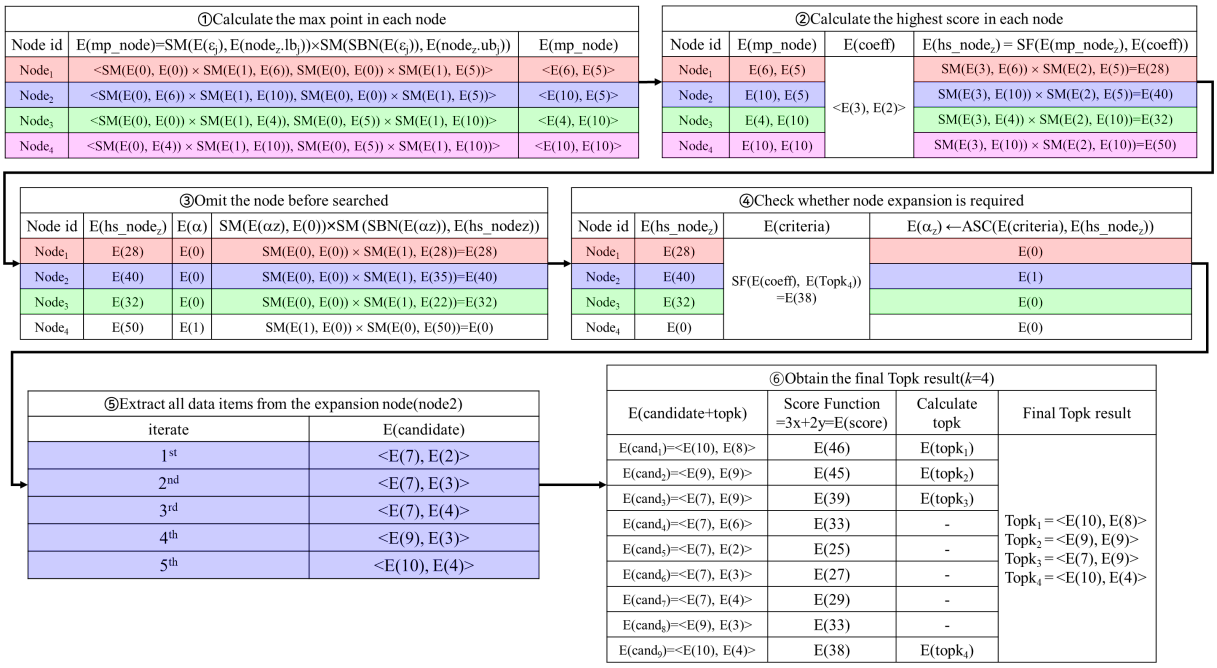


Figure 7. Example of Top-*k* result refinement phase.

5. Privacy-Preserving Parallel Top-*k* Query Processing Algorithm

In this section, we propose a privacy-preserving parallel Top-*k* query processing algorithm, which is the expansion of the proposed Top-*k* query processing algorithm mentioned in Section 4, so that it can be efficiently executed in a multi-core environment. The proposed parallel Top-*k* query processing algorithm consists of three phases; the parallel node data search phase, parallel Top-*k* retrieval phase, and parallel Top-*k* result refinement phase.

5.1. Parallel Node Data Search Phase

C_A obtains all the data items from a node having a max point(mp) in a parallel way. To expand the node data search phase in Section 4.1 to a multi-core environment, we utilize a thread pool where tasks can be executed simultaneously. The procedure of the parallel node data search phase is presented in Algorithm 7. First, C_A computes mp in the same way as Algorithm 4 (lines 1∼4). Second, C_A creates a thread pool based on queue (line 5). If a thread is able to process in the thread pool, C_A assigns a task to a thread in a First-In-First-Out manner. Third, for the ith node where $1 \leq i \leq \#_of_nodes$, C_A allocates to the thread pool the task of the ASPE protocol (Algorithm 3 in Section 3), i.e., Proc_ASPE(E(mp), E($node_i$), E(δ_i)). The result of the ASPE protocol is set in E(δ) = E(δ_1), E(δ_2), ..., E($\delta_{\#_of_node}$) (lines 6∼7). Fourthly, C_A creates E(δ') by permutating E(δ) using a random function π and transmits E(δ') to C_B (lines 8∼9). Fifthly, C_B performs the lines 9∼15 in Algorithm 4 to determine which node is selected (line 10). Sixthly, to recover the original node ids from the shuffled node ids, C_A obtains node groups (NG^*) with the original node ids by performing the lines 16∼18 in Algorithm 4 (line 11). Seventhly, C_A gets access to one data item in each node for each NG^* and allocates to the thread pool the task of Extract_candidate (lines 12∼17). The procedure of Extract_candidate securely obtains the candidate data items of the node, which includes mp. For this, C_A performs E($cand_{w,t}$)←E($cand_{w,t}$)×SM(E($node_z.data_{s,t}$), E(δ_z)), where $1 \leq t \leq m$, z = id of jth nodes of NG_i^*, $w = cnt + s$, and $1 \leq s \leq FanOut$. Here, $1 \leq i \leq c$, $1 \leq j \leq \#_of_nodes$ in the selected NG_i^*, and cnt = the number of data items extracted from the encrypted leaf nodes of kd-tree. Finally, C_A merges all candidate data items obtained in a parallel way (line 18).

Algorithm 7 Parallel node data search phase.

Input: <E($node_1$), E($node_2$), …, E($node_p$) | p = #_of_node >, E($coef$)=<E($coef_1$), E($coef_2$), …, E($coef_m$) | $coef$ means coefficient of score function and m = #_of_dimension>, <E(max_1), E(max_2), …, E(max_m) | max_i is the maximum data domain of i dimension >, E($hint$) // hint means the largest of the absolute values among coefficients

Output: <E($cand_1$), E($cand_2$), …, E($cand_{cnt}$)> // all candidates inside nodes being related to a query

C_A:

01. for $1 \leq i \leq m$
02. E($coef_i + hint$) ← E($coef_i$)×E($hint$)
03. E(ϵ_i) ← ASC(E($coef_i + hint$), E($hint$))
04. E(mp_i) ← SM(E(ϵ_i), E(0)) × SM(SBN(E(ϵ_i)), E(max_i))
05. generate thread_pool // create a thread and wait in the pool until a task is given
06. for $1 \leq i \leq p$
07. call thread_pool_push(Proc_ASPE(E(mp), E($node_i$), E(δ_i))) // assign the task to an available thread
08. E(δ') ← π(E(δ)) // shuffle the order of array E(δ) = <E(δ_1), E(δ_2), …, E(δ_p)>
09. send E(δ') to C_B

C_B:

10. perform line 9∼15 in Algorithm 4

C_A:

11. perform line 16∼18 in Algorithm 4
12. for $1 \leq i \leq c$
13. for $1 \leq j \leq num$ // num = #_of_nodes in the selected NG_i^*
14. z = id of jth nodes of NG_i^*
15. for $1 \leq s \leq FanOut$ //$FanOut$: the number of data items in each nodes
16. call thread_pool_push(Extract_candidate(E($node_z$), E(δ_z), m, s, cnt, E($cand_w$)))
17. $cnt \leftarrow cnt + FanOut$
18. return <E($cand_1$), E($cand_2$), …, E($cand_{cnt}$)>

procedure 1. PROC_ASPE(E(mp), E($node_i$))
Begin Procedure
01. E(δ_i) ← ASPE(E(mp), E($node_i$))
02. return E(δ_i)
End Procedure
end procedure

procedure 2. EXTRACT_CANDIDATE(E($node_z.data_s$) = <E($node_z.data_{s,1}$), E($node_z.data_{s,2}$), …, E($node_z.data_{s,m}$)>, E(δ_z))
Begin Procedure
01. for $1 \leq t \leq m$
02. if ($s == 1$)
03. E($cand_{w,t}$) ← E(0) where $w = cnt + s$ // E($cand_w$) = <E($cand_{w,1}$), E($cand_{w,2}$), …,E($cand_{w,m}$)>
04. E($cand_{w,t}$) ← E($cand_{w,t}$)×SM($node_z.data_{s,t}$, E(δ_z))
05. return E($cand_w$) = <E($cand_{w,1}$), E($cand_{w,2}$), …, E($cand_{w,m}$)>
End Procedure
end procedure

5.2. Parallel Top-k Retrieval Phase

C_A retrieves k number of data items with the highest score in a parallel way. The process of the parallel Top-k retrieval phase is presented in Algorithm 8. First, to calculate the scores of the candidates in a parallel way, C_A allocates to the thread pool the task of the score function, i.e., Proc_SF(E($coef$), E($cand_i$), E($score_i$)) for $1 \leq i \leq cnt$. The result of the Proc_SF is set to E($score$) = E($score_1$), E($score_2$), …, E($score_{cnt}$) (lines 1∼2). Second,

C_A executes SMAX$_n$ to search the maximum E($score_{max}$) among E($score_i$) for $1 \leq i \leq cnt$ (lines 3∼4). Third, to determine the data items with the highest score in a parallel way, C_A allocates to the thread pool the task of Mark_Top_one. The procedure of Mark_Top_one securely sets the data item with the highest score to E(0). For this, C_A performs E(τ'_i) ← (E($score_{max}$)×E($score_i$)$^{N-1}$)^r for $1 \leq i \leq cnt$, where r means a random number and N is created in the Paillier cryptosystem (lines 5∼6). Fourthly, C_A gains E(λ) by permutating E(τ') = < E(τ'_1), E(τ'_2), ..., E(τ'_{cnt})> using a random function π and sends E(λ) to C_B (lines 7∼8). Fifthly, after decrypting E(λ), C_B performs lines 10∼13 in Algorithm 4 to mark which data item contains the highest score (line 9). That is, C_B stores E(U_i) into E(1) if E(λ_i) = 0 for $1 \leq i \leq cnt$. Otherwise C_B sets E(U_i) to E(0). Sixthly, C_A obtains E(V) by recovering E(U) = <E(U_1), E(U_2), ..., E(U_{cnt})> using π^{-1}(line 10). Seventhly, to prune out the data items except the data item with the highest score (Top-one) in a parallel way, C_A allocates to the thread pool the task of Pruneout_From_Top2. The procedure of Pruneout_From_Top2 securely sets the data items except Top-one to E(0) by performing Prunout_From_Top2(E(V_i), E($cand_{i,j}$), E($score_i$), E($V'_{i,j}$)) for $1 \leq i \leq cnt$ and $1 \leq j \leq m$. The result of the Prunout_From_Top2 is stored in both E($score$) = E($score_1$), E($score_2$), ..., E($score_{cnt}$) and E(V'_i) = E($V'_{i,1}$), E($V'_{i,2}$), ..., E($V'_{i,m}$) (lines 11∼12). Eighthly, to obtain the Top-one in a parallel way, C_A allocates the task of Find_Top_one to the thread pool. The procedure of Find_Top_one securely merges all the data items. For this, Find_Top_one performs a secure additive operation, i.e., E($topk_{s,j}$) ← E($topk_{s,j}$)×E($V'_{i,j}$) for $1 \leq i \leq cnt$, $1 \leq j \leq m$, and $1 \leq s \leq \#_of_topk$. The result of the Find_Top_one is stored into E($topk_s$) = <E($topk_{s,1}$), E($topk_{s,2}$), ..., E($topk_{s,m}$)> (lines 13∼14). Finally, C_A obtains the Top-k result by merging all the Top_one for k rounds by repeating lines 4∼15 in Algorithm 8 (lines 15∼16).

5.3. Parallel Top-k Result Refinement Phase

C_A determines whether or not the Top-k result obtained in the parallel Top-k retrieval phase is sufficient or not, in a parallel way. If not sufficient, C_A performs both the parallel node data search phase and the parallel Top-k retrieval phase again. The process of the parallel Top-k result refinement phase is presented in Algorithm 9. First, C_A calculates E(criteria) = SF(E($coef$), E($topk_k$)) to gain the minimum score between the Top-k result and the query (line 1). Second, to determine the neighboring nodes that need to be searched to acquire data items with a higher score than E(criteria) in a parallel way, C_A allocates the task of check_node_expansion to the thread pool (lines 2∼5). To find the max point of each node, C_A performs E($mp_node_{i,j}$)←SM(E(ϵ_j), E($node_i.lb_j$))×SM(SBN(E(ϵ_j)), E($node_i.ub_j$)) for $1 \leq i \leq \#_of_nodes$ and $1 \leq j \leq \#_of_dimension$. Here, E($\epsilon_j$) is obtained by the ASC protocol in line 3 of Algorithm 6. If the coefficient for the jth dimension is positive, E(ϵ_j) is set to E(0). Otherwise, E(ϵ_j) is set to E(1). To calculate the highest score of each node, C_A performs E(hs_node_i) ← SF(E(mp_node_i), E($coef$)) for the ith node. To find nodes whose highest score is greater than the criteria, C_A performs E(δ_i) ← ASC(E(criteria), E(hs_node_i)) for the ith node. If E(δ_i) is E(1), the ith node needs to be expanded for the parallel Top-k result refinement. Third, C_A calculates the final Top-k result in ciphertext by including the additional Top-k result obtained by performing Algorithm 8 (line 6). Fourthly, to hide the Top-k result from C_B, C_A calculates E($\gamma_{i,j}$) = E($topk^*_i$) × E($r_{i,j}$) for $1 \leq i \leq k$ and $1 \leq j \leq m$, utilizing a random value $r_{i,j}$. Then, C_A transmits $E(\gamma_{i,j})$ to C_B and $r_{i,j}$ to AU (lines 7∼11). Fifthly, C_B decrypts E($\gamma_{i,j}$) and transmits them to AU (lines 12∼15). Finally, AU gains the plaintext of the Top-k result by calculating $\gamma_{i,j}$-$r_{i,j}$ (lines 16∼18). As a result, AU can reduce the computation overhead without using decryption operations.

Algorithm 8 Parallel Top-k retrieval phase.

Input: <E($cand_1$), E($cand_2$), …, E($cand_{cnt}$) | cnt = #_of_$candidates$ >, E($coef$), k // $coef$ means coefficient of score function and k means the number of Top-k

Output: <E($topk_1$), E($topk_2$), …, E($topk_k$)> // temporary Top-k results

C_A:

01. for $1 \leq i \leq cnt$
02. call thread_pool_push(Proc_SF(E($coef$), E($cand_i$), E($score_i$)))
03. for $1 \leq s \leq k$
04. E($score_{max}$) ← SMAX$_n$(E($score_1$), E($score_2$), …, E($score_{cnt}$))
05. for $1 \leq i \leq cnt$
06. call thread_pool_push(Mark_Top_one(E($score_{max}$), E($score_i$), E(τ'_i)))
07. E(λ) ← $\pi(\tau')$ // τ'=<τ'_1, τ'_2, …, τ'_{cnt}>
08. send E(λ)=<E(λ_1), E(λ_2), …, E(λ_{cnt})> to C_B

C_B:

09. perform line 10∼13 in Algorithm 5

C_A:

10. E(V) ← $\pi^{-1}(U)$ // E(V) = <E(V_1), E(V_2), …, E(V_{cnt})>
11. for $1 \leq i \leq cnt$
12. call thread_pool_push(Pruneout_From_Top2(E(V_i), E($cand_{i,j}$), E($score_i$), E($V'_{i,j}$)))
13. for $1 \leq i \leq cnt$
14. call thread_pool_push(Find_Top_one(E($V'_{i,j}$), cnt, E($topk_{s,j}$)))
15. E($topk$) ← E($topk$) ∪ E($topk_s$)
16. return E($topk$) = <E($topk_1$), E($topk_2$), …, E($topk_k$)>

procedure 3. PROC_SF(E($coef$), E($cand_i$))
Begin Procedure
01. E($score_i$) ← SF(E($coef$), E($cand_i$))
02. return E($score_i$)
End Procedure
end procedure

procedure 4. MARK_TOP_ONE(E($score_{max}$), E($score_i$))
Begin Procedure
01. E(τ_i) ← E($score_{max}$)×E($score_i$)$^{N-1}$
02. E(τ'_i) ← $E(\tau_i)^r$ // r means random value
03. return E(τ'_i)
End Procedure
end procedure

procedure 5. PRUNEOUT_FROM_TOP2(E(V_i), E($cand_{i,j}$), E($score_i$))
Begin Procedure
01. for $1 \leq j \leq m$ // m = #_of_dimension
02. E($V'_{i,j}$) ← SM(E(V_i), E($cand_{i,j}$))
03. if $s < k$ // k = #_of_top$k$ and s is from 1 to k
04. E($score_i$) ← SM(E(V_i), E(0))×SM(SBN(E(V_i)), E($score_i$))
05. return E($V'_{i,j}$), E($score_i$)
End Procedure
end procedure

procedure 6. FIND_TOP_ONE(E($V'_{i,j}$), cnt, E($topk_{s,j}$))
Begin Procedure
01. for $1 \leq j \leq m$ // cnt = #_of_candidates
02. E($topk_{s,j}$) ← E($topk_{s,j}$)×E($V'_{i,j}$) // s = index of loop from 1 to k, k = #_of_top$k$
03. return E($topk_{s,j}$)
End Procedure
end procedure

Algorithm 9 Parallel Top-k result refinement phase.

Input: <E($node_1$), E($node_2$), …, E($node_p$) | p = #_of_$node$ >, <E($topk_1$), E($topk_2$), …, E($topk_k$)>, E($coef$), k // $coef$ means coefficient of score function and k means the number of Top-k

Output: $topk^*$ = <$topk_1^*$, $topk_2^*$, …, $topk_k^*$> // final Top-k results

C_A:

01. E(criteria) = SF(E($coef$), E($topk_k$))
02. for $1 \leq i \leq p$
03. call thread_pool_push(check_node_expansion(E(ϵ), E($coef$), E(δ_i), E($node_i$), E(criteria), m), E(δ_i))
04. E($cand'$) ← perform 7 ∼ 27 lines of Algorithm 6 with <E($node_1$), E($node_2$), …, E($node_p$)>, E(δ) = <E(δ_1), E(δ_2), …, E(δ_p)>
05. E($cand'$) ← E($cand'$)∪<E($topk_1$), E($topk_2$), …, E($topk_k$)>
06. E($topk^*$) ← perform Algorithm 8 with E($cand'$), E($coef$) and k
07. for $1 \leq i \leq k$
08. for $1 \leq j \leq m$
09. pick up the random value $r_{i,j}$ // r_i = <$r_{i,1}$, $r_{i,2}$, …, $r_{i,m}$>
10. E($\gamma_{i,j}$) ← E($topk_{i,j}^*$)×E($r_{i,j}$) // E($topk_i^*$)=<E($topk_{i,1}^*$), E($topk_{i,2}^*$), …, E($topk_{i,m}^*$)> and E(γ_i) = <E($\gamma_{i,1}$), E($\gamma_{i,2}$), …,E($\gamma_{i,m}$)>
11. send <E(γ_1), E(γ_2), …, E(γ_k) > to C_B and <r_1, r_2, …, r_k> to AU

C_B:

12. for $1 \leq i \leq k$
13. for $1 \leq j \leq m$
14. $\gamma_{i,j}$ ← D(E($\gamma_{i,j}$)) // γ_i = <$\gamma_{i,1}$, $\gamma_{i,2}$, …,$\gamma_{i,m}$>
15. send <γ_1, γ_2, …, γ_k> to AU

AU:

16. for $1 \leq i \leq k$
17. for $1 \leq j \leq m$
18. $topk_{i,j}^*$ ← $\gamma_{i,j}$ - $r_{i,j}$ //$topk_i^*$ = <$topk_{i,1}^*$, $topk_{i,2}^*$, …, $topk_{i,m}^*$>

procedure 7. CHECK_NODE_EXPANSION(E(ϵ) = <E(ϵ_1),E(ϵ_2), …, E(ϵ_m)>, E($coef$), E(δ_i), E($node_i$), E(criteria), m = #_of_dimension)

Begin Procedure

01. for $1 \leq j \leq m$
02. E($mp_node_{i,j}$)←SM(E(ϵ_j), E($node_i.lb_j$))×SM(SBN(E(ϵ_j)), E($node_i.ub_j$)) // E(δ_i) is value returned by ASPE protocol in Algorithm 6
03. E(hs_node_i) ← SF(E(mp_node_i), E($coef$))
04. E(hs_node_i) ← SM(E(δ_i), E(0))× SM(SBN(E(δ_i)), E(hs_node_i)) // E(δ_i) is value returned by ASPE protocol in Algorithm 6
05. E(δ_i) ← ASC(E(criteria), E(hs_node_i))
06. return E(δ_i)

End Procedure

end procedure

6. Security Proof

6.1. Security Proof of the Secure Protocols

In this section, we present the security proof of the ASC and the ASPE protocols proposed in Section 3. To prove that the proposed protocols are secure under the semi-honest model, we present that the simulated images of the proposed protocols are computationally indistinct from their actual execution images. Security proof of the ASC protocol: We describe the security proof of the ASC protocol by analyzing the security of the execution images of C_A and C_B. First, the execution image on the C_B side, i.e., $\prod_{C_B}(ASC)$, is shown in Equation (5). Here, $E(v_1')$ and $E(v_2')$ are the encrypted data given from C_A (line 9 of Algorithm 1), v_1', and v_2', and are acquired by decrypting E(v_1') and E(v_2'). α is the result computed by the ASC protocol utilizing v_1' and v_2' on the C_B side.

$$C_B(ASC) = E(v_1'), E(v_2'), v_1', v_2', \alpha \tag{5}$$

For example, we assume that $CB_S(ASC) = E(s_1'), E(s_2'), s_1', s_2', s_3$ is the simulated execution image utilizing the ASC protocol on the C_B side. Here, E(s_1') and E(s_2') are the non-deterministic numbers chosen in Z_{N^2}, and s_1' and s_2' are the indistinct numbers that are added by random value. s_3' is the result of the ASC protocol utilizing s_1' and s_2' on the C_B side. Because the ASC protocol is executed based on the Paillier cryptosystem, it can support semantic security. Therefore, E(s_1') and E(s_2') are computationally indistinct from s_1' and s_2'. s_3' is indistinct from s_1' and s_2' because s_3' is computed by multiplying two indistinct numbers in C_A, s_1' and s_2'. Therefore, we say that $\prod_{C_B}(ASC)$ is computationally indistinct from $\prod_{C_BS}(ASC)$. Because C_B can check only the result (e.g., α) of the comparison between the non-deterministic numbers (e.g., v_1' and v_2'), C_B cannot obtain the original data while performing the ASC protocol. Furthermore, the execution image of C_A is $\prod_{C_A}(ASC) = \{E(\alpha)\}$, such that E($\alpha$) from C_B can be considered as the result of the ASC protocol. Assume that the simulated image of C_A is $\prod_{C_AS}(ASC) = \{E(s_4)\}$, where E($s_4$) is randomly made from Z_{N^2}. Thus, E(α) is computationally indistinct from E(s_4). Based on the above analysis, there is no information leakage, both at the C_A and C_B sides. Thus, we can conclude that the proposed ASC protocol is secure under the semi-honest adversarial model. Security proof of the ASPE protocol: We describe the security proof of the ASPE protocol by analyzing the security of the execution images of the C_A side and the C_B side. First, the execution image on the C_B side, i.e., $\prod_{C_B}(ASPE)$, is shown in Equation (6). Here, $\prod_{C_B}(ASC)$ means the execution image of the ASC protocol and $\prod_{C_B}(SM)$ means the execution image of the SM protocol.

$$\prod_{C_B}(ASPE) = \{\prod_{C_B}(ASC), \prod_{C_B}(SM)\} \tag{6}$$

In the security proof of the ASC protocol, we have already proven the security of the ASC protocol on the C_B side. Additionally, Y. Elmehdwi et al.'s work [19] proved the security of the SM protocol on the C_B side. Because the ASPE protocol is composed of the ASC protocol and the SM protocol, the ASPE protocol is secure on the C_B side, based on composition theory [42]. On the other hand, the execution image of C_A is $\prod_{C_A}(ASPE) = \{\prod_{C_A}(ASC), \prod_{C_A}(SM)\}$, which can be considered as the result of the ASC protocol and the SM protocol. In the security proof of the ASC protocol, we already proved the indistinguishability of the ASC protocol on the C_A side. Additionally, Y. Elmehdwi et al.'s work proved the security of the SM protocol on the C_A side [19]. Because the ASPE protocol consists of the ASC protocol and the SM protocol, the ASPE protocol is secure on the C_A side, based on composition theory [42]. Based on the above analysis, there is no information leakage, both at the C_A and the C_B side. Thus, we can conclude that the proposed ASPE protocol is secure under the semi-honest attack model.

6.2. Security Proof of the Proposed Top-k Query Processing Algorithm

We prove that the proposed Top-*k* query processing algorithm on the encrypted database is secure against the semi-honest adversarial model. The proposed Top-*k* query processing algorithm in the encrpyted database is composed of the node data search phase (Algorithm 4), the Top-*k* retrieval phase (Algorithm 5), and the Top-*k* result refinement phase (Algorithm 6). To prove that the proposed Top-*k* query processing algorithm is secure under the semi-honest adversarial model, security analysis is executed by each phase. First, because the node data search phase is composed of the ASPE protocol, which has been proven to be secure, Algorithm 4 is secure under the semi-honest adversarial model by composition theory [42]. Second, the Top-*k* retrieval phase is secure in the C_A side, because C_A performs the score function, and the SMAX$_n$ and SM protocols have been proven to be secure in previous studies [27,28]. Even if C_B decrypts the received data from C_A in the Top-*k* retrieval phase, C_B cannot obtain the original data. The reason for this that the data given from C_A is changed by raising the original data to the power of a random

integer and performing a permutation function. Thus, based on the composition theory [42], Algorithm 5 is secure under the semi-honest adversarial model. Lastly, the images made by the Top-k result refinement phase are the same as those made by Algorithms 4 and 5. Thus, the Top-k result refinement phase (Algorithm 6) is secure against the semi-honest adversarial model. Because all the phases of the proposed Top-k algorithm are secure, the proposed Top-k algorithm on the encrypted database is proven to be secure against the semi-honest adversarial model.

7. Performance Analysis

7.1. Performance Evaluation of the Proposed Top-k Query Processing Algorithm in a Single-Core Environment

In the performance analysis, we compare the proposed privacy-preserving Top-k query processing algorithm with H-I. Kim et. al.'s work [27] (STopk_I) and H-J. Kim et. al.'s work [28] (STopk_G). For performance analysis, three algorithms were programmed by using C++ language under CPU: an Intel(R) Xeon(R) E5-2630 v4 @ 2.20GHz, RAM: 64 GB (16 GB × 4AE) DDR3 UDIMM 1600 MHz and OS: Linux Ubuntu 18.04.2. We compare three algorithms regarding algorithm processing time by changing # of data, # of k, the level of the kd-tree, and the # of the data dimension. For performance experiments, we randomly make 100,000 data points with six dimensions. The domain of the data ranges from 0 to 2^{22}. Table 2 presents the parameters for performance evaluation in a single-core environment.

Table 2. Parameters for performance evaluation in a multi-core environment.

Parameter	Values	Default
Number of data items (n)	20 k, 40 k, 60 k, 80 k, 100 k	-
Level of kd-tree	7, 8, 9, 10, 11	10
Number of dimensions	2, 3, 4, 5, 6	6
Key size	512, 1024	-
Data domain (bit length)	2^{22}	2^{22}

Figure 8 presents the performances of STopk_I, STopk_G, and the proposed algorithm regarding the height of kd-tree(h). The number of data items(n) is computed as $F \times 2^{h-1}$, where 2^{h-1} is the number of leaf nodes in the kd-tree. Based on n, it is crucial to select the proper height(h) of the kd-tree. In Figure 8, three algorithms present the best performance when h = 10. The performance of STopk_I is greatly influenced by h because STopk_I utilizes secure protocols using encrypted binary operations, which need a high computational cost. Whereas, the proposed algorithm is relatively less influenced by h than STopk_I because it utilizes secure protocols using arithmetic operations that need low computational cost. Therefore, we set h to 10 in our experiment.

When the encryption key size is 1024 bit, Figure 9 presents the processing time with a varying number of data items. When the number of data items is 20 k, 40 k, 60 k, 80 k, and 100 k, the proposed algorithm requires 2383, 4453, 5287, 6836, and 8447 s, respectively. STopk_G requires 4082, 7653, 9156, 11,895, and 14,799 s, while STopk_I requires 23,124, 28,015, 32,280, 38,636, and 46,989 s when the number of data items is 20 k, 40 k, 60 k, 80 k, and 100 k. Therefore, the proposed algorithm presents 1.7 and 6.6 times better performance than STopk_G and STopk_I, respectively, because the proposed algorithm uses secure protocols based on arithmetic operations. STopk_G and STopk_I use secure protocols based on binary operations. The binary operations have the disadvantage of having to perform more iterations compared to the arithmetic operations.

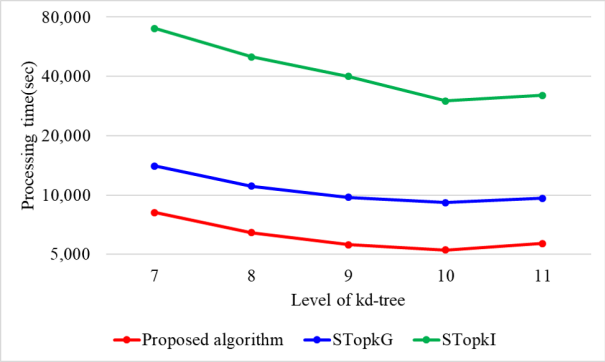


Figure 8. Processing time with varying levels of *k*d-tree.

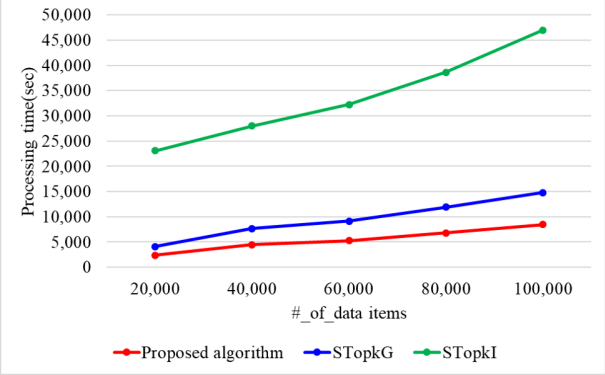


Figure 9. Processing time with a varying number of data items.

Figure 10 presents the processing time with a varying number of k when the encryption key size is 1024 bit and n is 60 k. When the number of k is 5, 10, 15, and 20, the proposed algorithm requires 4749, 5287, 7693, and 8856 s. STopk_G requires 8297, 9156, 13,232, and 15,043 s, while STopk_I requires 22,288, 32,380, 76,076, and 97,021 s. The proposed algorithm presents 1.7 and 7.9 times better performance than STopk_G and STopk_I, respectively, because the proposed algorithm uses secure protocols based on arithmetic operations. As k increases, the number of data items searched increases in the Top-k result refinement phase. Because the proposed algorithm utilizes the ASC protocol, it presents better performance than both STopk_I and STopk_G. STopk_G presents a better performance than STopk_I because it utilizes Yao's garbled circuit.

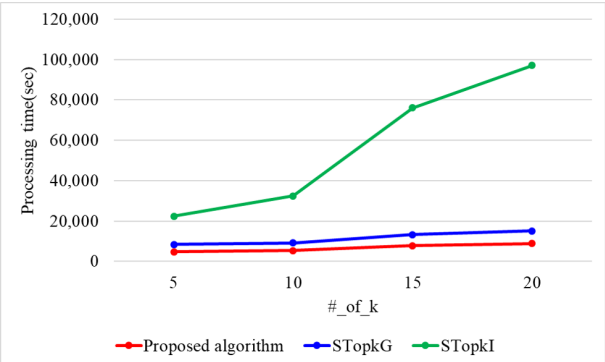


Figure 10. Processing time with a varying number of k.

When n is 100 k, Figure 11 presents the processing time with a varying key size. When the key size is 512 and 1024, the proposed algorithm requires 2041 and 8486 s, respectively. STopk_G requires 4367 and 14,847 s, while STopk_I requires 13,500 and 46,989 s. Because the proposed algorithm utilizes secure protocols using arithmetic operations, the proposed

algorithm outperforms STopk_G and STopk_I by 1.9 and 6.0 times. STopk_G and STopk_I use secure protocols based on binary operations, which have the disadvantage of having to perform more iterations compared to arithmetic operations.

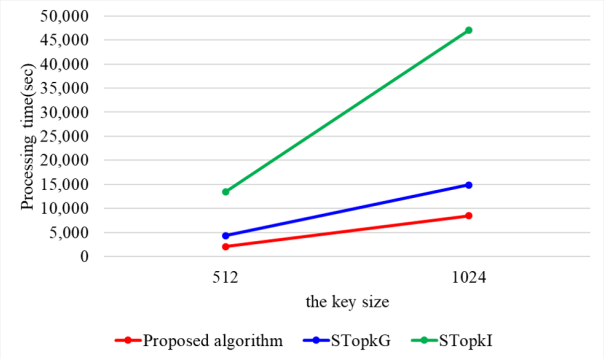


Figure 11. Processing time with a varying key size.

7.2. Performance Evaluation of the Proposed Top-k Query Processing Algorithm in a Multi-Core Environment

For performance evaluation in a multi-core environment, we compare the proposed parallel Top-k query processing algorithm with the parallel version of existing works. For this, we make parallel STopk_I (PSTopk_I) and STopk_G (PSTopk_G) so that H-I. Kim et. al.'s work [27] and H-J. Kim et. al.'s work [28] can operate in a multi-core environment, respectively. For performance analysis, three algorithms were programmed by using C++ language under CPU: Intel(R) Xeon(R) E5-2630 v4 @ 2.20 GHz, RAM: 64 GB (16 GB×4AE) DDR3 UDIMM 1600MHz and OS: Linux Ubuntu 18.04.2. We compare the proposed parallel algorithm with both PSTopk_G and PSTopk_I, regarding the algorithm processing time by changing the # of data(n), # of k, and # of threads. For our experiments, we randomly make 100,000 data points with six dimensions. The domain of the data ranges from 0 to 2^{22}. Table 3 describes the parameters for performance analysis in a multi-core environment.

Figure 12 presents the processing time with a varying number of threads when n is 100k and the number of k is 10. When # of threads is 2, 4, 6, 8, and 10, the proposed parallel algorithm requires 4451, 2547, 1871, 1552, and 1291 s, respectively. PSTopk_G requires 7621, 4304, 3154, 2575, and 2115 s while PSTopk_I requires 23,765, 13,293, 9520, 7671, and 6294 s when the number of threads is 2, 4, 6, 8, and 10. Therefore, the proposed parallel algorithm presents 1.6 and 5 times better performance than PSTopk_G and PSTopk_I, respectively. This is because the proposed parallel algorithm utilizes secure protocols using arithmetic operations. On the contrary, both PSTopk_G and PSTopk_I use a secure protocol based on binary operations, which have the disadvantage of performing more iterations compared to the arithmetic operations.

Table 3. Parameters for performance evaluation in multi-core environment.

Parameter	Values	Default
Number of data items (n)	20 k, 40 k, 60 k, 80 k, 100 k	-
Number of k	5, 10, 15, 20	10
Number of threads	2, 4, 6, 8, 10	10
Key size	1024	-
Data domain (bit length)	2^{22}	2^{22}

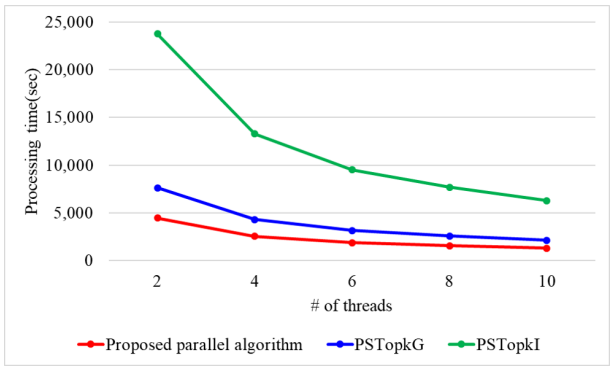


Figure 12. Processing time with a varying number of threads.

When the number of k is 10 and the number of threads is 10, Figure 13 presents the processing time with varying n. When n is 20 k, 40 k, 60 k, 80 k, and 100 k, the proposed algorithm requires 433, 762, 808, 1041, and 1301 s, respectively. SPTopk_G requires 642, 1192, 1337, 1711, and 2127 s, while PSTopk_I requires 3193, 3994, 4344, 5254, and 6272 s. Thus, the proposed parallel algorithm presents 1.5 and 5.5 times better performance than PSTopk_G and PSTopk_I, respectively. This is because the proposed algorithm utilizes secure protocols using arithmetic operations. On the contrary, PSTopk_G and PSTopk_I use a secure protocol based on binary operations that require more iterations.

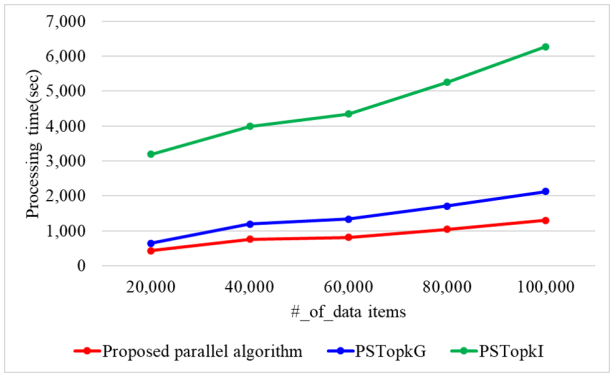


Figure 13. Processing time with a varying number of data items in 10 thread.

Figure 14 presents the processing time with a varying number of k when the number of threads is 10 and n is 60 k. When the number of k is 5, 10, 15, and 20, the proposed parallel algorithm requires 681, 845, 1401, and 1664 s. PSTopk_G requires 1165, 1392, 2148, and 2466 s, while PSTopk_I requires 3000, 4472, 10,235, and 13,042 s. Therefore, the proposed parallel algorithm presents 1.5 and 6.2 times better performance than PSTopk_G and PSTopk_I, respectively, As k increases, the number of data items searched increases in the Top-k result refinement phase. Because the proposed parallel algorithm utilizes the ASC protocol, it presents a better performance than both PSTopk_I and PSTopk_G. PSTopk_G presents a better performance than PSTopk_I because it uses Yao's garbled circuit.

Table 4 presents the parameters for our performance analysis using a real dataset. We utilize a chess dataset [43] made by a chess endgame database for a white king and rook against a black king. The chess dataset intends to categorize the optimal depth of win for white. We compare the proposed parallel algorithm with both PSTopk_G and PSTopk_I, regarding the query processing time by varying the number of k, the level of the kd-tree, and the number of threads.

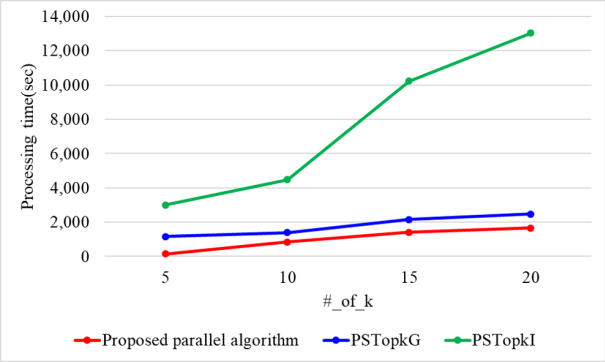


Figure 14. Processing time with a varying number of k in 10 thread.

Table 4. Parameters for performance evaluation for real dataset.

Parameter	Values	Default
Number of data items (n)	28,056	-
Level of kd-tree	5, 6, 7, 8, 9, 10, 11, 12	10
Number of dimensions	6	6
Number of threads	2, 4, 6, 8, 10	10
Number of k	5, 10, 15, 20	10
Key size	512	-
Data domain (bit length)	2^{12}	2^{12}

In order to select the optimal level of kd-tree, we conduct the performance evaluation of three algorithms regarding the level of kd-tree. Figure 15 presents the processing time with a varying level of kd-tree when the number of threads is 10 and the number of k is 10. When the level of kd-tree ranges from 5 to 12, the proposed parallel algorithm requires 639, 524, 425, 354, 305, 267, 289, and 312 s. PSTopk_G requires 1122, 916, 749, 631, 546, 453, 498, and 539 s, while PSTopk_I requires 4282, 3497, 2771, 2259, 1929, 1625, 1745, and 1931 s. Three algorithms present the best performance in the case of h = 10. Thus, for our experiment, we use h as 10.

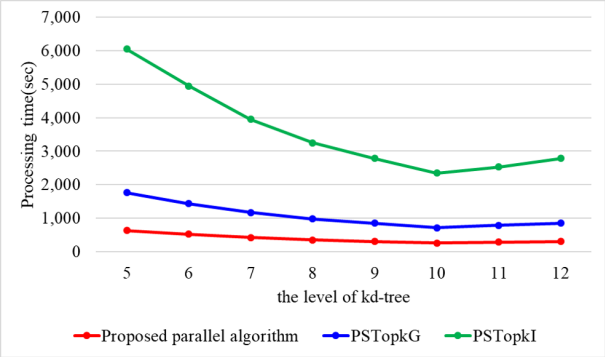


Figure 15. Processing time with a varying level of kd-tree in real dataset.

When the number of threads is 10, Figure 16 presents the processing time with a varying number of k. When the number of k is 5, 10, 15, and 20, the proposed parallel algorithm requires 147, 266, 407, and 505 s. PSTopk_G requires 316, 453, 769, and 933 s, while PSTopk_I requires 1028, 1625, 3003, and 3758 s. The proposed parallel algorithm outperforms PSTopk_G and PSTopk_I by 1.8 and 6.9 times. As k increases, the number of data items searched increases in the Top-k result refinement phase. Because the proposed parallel algorithm utilizes the ASC protocol using the arithmetic operation, it presents a better performance than both PSTopk_I and PSTopk_G. Because PSTopk_G uses Yao's garbled circuit, it presents a better performance than PSTopk_I.

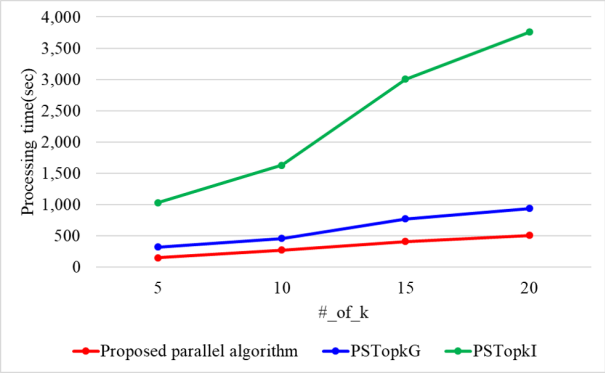


Figure 16. Processing time with a varying number of k in real dataset.

When the number of k is 10, Figure 17 presents the processing time with a varying number of threads. When the number of threads is 2, 4, 6, 8, and 10, the proposed parallel algorithm requires 718, 443, 349, 299, and 266 s. PSTopk_G requires 1280, 753, 597, 503, and 452 s, while PSTopk_I requires 6122, 3396, 2461, 1976, and 1629 s. Because the proposed parallel algorithm utilizes secure protocols using arithmetic operations, it outperforms PSTopk_G and PSTopk_I by 1.7 and 7.1 times. On the contrary, PSTopk_G and PSTopk_I use secure protocol based on binary operations which have to perform more iterations compared to the arithmetic operations.

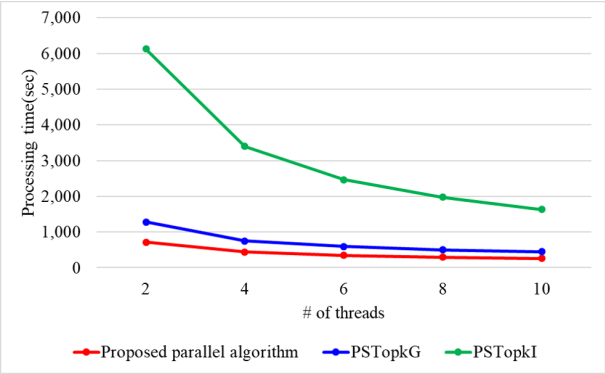


Figure 17. Processing time with a varying number of threads in real dataset.

8. Discussion

Impact of new secure protocols with low computation cost: A secure protocol is essential for secure query processing in a multi-party computation environment. We need to streamline the process of secure protocols because we aim for secure query processing by utilizing the Paillier cryptosystem, which uses a large amount of computing resources. First, H-I. Kim et al.'s work [27] used secure protocols, such as SCMP, SPE, SMAX, and SMAX$_n$ for Top-k query processing. By utilizing the Paillier cryptosystem, H-I. Kim et al.'s work [27] can hide sensitive data and user's query. It utilizes arithmetic operations to hide the sensitive data, and a permutation technique to hide data access patterns. However, the weakness of H-I. Kim et. al.'s work [27] is that it requires an extremely high computational cost because the SCMP, SPE, SMAX, and SMAX$_n$ protocols utilize binary operations. For example, when we execute the SMAX protocol between E(11) and E(15), clouds convert an encrypted decimal into an encrypted binary array: $E(11)_{(10)}$ = $\{E(0), E(1), E(0), E(1), E(1)\}_{(2)}$, $E(15)_{(10)}$ = $\{E(0), E(1), E(1), E(1), E(1)\}_{(2)}$. After that, clouds execute the SMAX protocol between $E(11)_{(10)}$ = $\{E(0), E(1), E(0), E(1), E(1)\}_{(2)}$ and $E(15)_{(10)}$ = $\{E(0), E(1), E(1), E(1), E(1)\}_{(2)}$. Consequently, the SMAX protocol needs a high computation cost since it executes binary operations as many times as its bit length. Due to the same reason, the SCMP, SPE, and SMAX$_n$ protocols require high computation costs. Second, H-J. Kim et al.'s work [44] and Y. Kim et al.'s work [45] proposed secure protocols such as GSCMP and GSPE, which are

utilized for index searching to perform *k*NN and *k*NN classification based on Yao's garbled circuit. However, since both GSCMP and GSPE utilize the binary array as input, they need high computation expense due to the same reason as the SMAX protocol. On the contrary, the proposed algorithm uses the new ASC and ASPE protocols, which calculate one Paillier arithmetic operation. The reason for this is that they utilize an encrypted decimal as input, rather than an binary array. Consequently, the new ASC and ASPE protocols need low computation cost.

Impact of a random value pool for parallel processing: In the proposed system architecture, we utilize multi-party computation for the parallel Top-*k* query processing algorithm. Thus, we must prevent C_B from the leakage of sensitive data while performing secure protocols. For this, C_A generates a random integer r from Z_N and encrypts r by utilizing the Paillier cryptosystem. Then, C_A performs the addition of both the encrypted random integer E(r) and the encrypted plaintext E(m) by computing E($m + r$) = E(m) × E(r). Because $m + r$ is independent from m, C_B cannot achieve sensitive data for decryption. However, calculating homomorphic addition and homomorphic multiplication to protect sensitive data results in poor performance, since modular and exponential operations in homomorphic addition and homomorphic multiplication require higher computing resources than other encrypted operations. In Table 5, regarding the Secure Multiplication protocol, H.-I. Kim et al.'s work [27] and H.-J. Kim et al.'s work [28] need three times the encryption: two encryptions for random integers at C_A and one encryption for the multiplication at C_B. On the contrary, the proposed algorithm needs one encryption for the multiplication at C_B because it picks up the random integers from the encrypted random value pool at C_A. With regard to the Secure Compare protocol, H-I. Kim et al.'s work [27] and H.-J. Kim et al.'s work [28] need $log_2 D$ times encryption, where D is a data domain. On the contrary, the proposed algorithm needs one encryption for the comparison at C_B utilizing the random value pool. Thus, the proposed algorithm can save the computational cost for encryption by utilizing the random value pool.

Table 5. Comparison of the number of encryption according to encrypted random value pool.

Protocol / Algorithm	SM Protocol	Secure Compare Protocol
H-I. Kim et al. [27]	$3 \times E$	$log_2 D \times E$
H-J. Kim et al. [28]	$3 \times E$	$log_2 D \times E$
Proposed algorithm	$1 \times E$	$1 \times E$

E = Encryption, D = Data domain.

Impact of time complexity: To prove that the proposed algorithm is more efficient than the existing algorithms, we analyze the time complexity of both the proposed algorithm and existing ones (H.-I. Kim et al. [27] and H.-J. Kim et al. [28]). Because the number of data, the number dimension, the number of bit length, and the number of Top-*k* affect the efficiency of the Top-*k* query processing algorithm, we describe the time complexity in terms of them. Table 6 shows the average time complexity and the worst time complexity for the proposed algorithm and the existing ones [27,28]. The worst time complexity occurs in the case of a query to expand other *k*d-tree nodes in the Top-*k* result refinement phase. The average time complexity is measured by considering all queries that require or do not require the expansion of other *k*d-tree nodes in the Top-*k* result refinement phase. In the average time complexity, the proposed algorithms show $m \times (l-1) + k \times (l-1) \times log_2 n$ lower time complexity than the existing algorithms. This is because the proposed algorithm uses the secure protocol based on decimal arithmetic operation, rather than using one based on binary operation. In the worst time complexity, the proposed algorithm shows $m \times (l-1) + k \times (l-1) \times log_2 n$ lower time complexity than the existing algorithms. The reason is the same as the average time complexity.

Table 6. Time complexity for proposed algorithm and existing algorithm.

Top-k Query Processing Algorithm	Average Time Complexity	Worst Time Complexity
H-I. Kim et al. [27]	$O(m \times l + n \times m + k \times (l \times log_2 n + n \times m))$	$O(m \times l + n \times m + k \times (l \times log_2 n + n \times m))$
H-J. Kim et al. [28]	$O(m \times l + n \times m + k \times (l \times log_2 n + n \times m))$	$O(m \times l + n \times m + k \times (l \times log_2 n + n \times m))$
Proposed algorithm	$O(m + n \times m + k \times (log_2 n + n \times m))$	$O(m + n \times m + k \times (log_2 n + n \times m))$

n = #_of_data, m = #_of_dimensions, l = #_of_bit length, k = #_of_Top-k.

9. Conclusions and Future Work

In this paper, we proposed a new Top-k query processing algorithm to support both security and efficiency in cloud computing. For security, we propose new secure and efficient protocols based on the arithmetic operations ASC, ASRO, and ASPE. To reduce the query processing time, we propose a privacy-preserving parallel Top-k query processing algorithm by utilizing a random value pool. In addition, we proved that the proposed algorithm is secure against the semi-honest adversarial model. From our performance analysis, the proposed algorithm outperformed the existing algorithms by 1.7∼6.6 times, since it uses new secure protocols with a streamlined process. In addition, when the number of threads ranges from 2 to 10, the proposed parallel algorithm outperforms the existing algorithms by 1.5∼7.1 times, since it uses not only new secure protocols, but also the random value pool. For future work, we plan to use the proposed secure protocols for various privacy-preserving data mining algorithms.

Author Contributions: Conceptualization, H.-J.K.; methodology, H.-J.K.; software, H.-J.K.; validation, H.-J.K., Y.-K.K., H.-J.L., J.-W.C.; formal analysis, H.-J.K., Y.-K.K., H.-J.L., J.-W.C.; investigation, H.-J.K., Y.-K.K., H.-J.L., J.-W.C.; resources, H.-J.K., Y.-K.K., H.-J.L., J.-W.C.; data curation, H.-J.K.; writing—original draft preparation, H.-J.K.; writing—review and editing, H.-J.K., Y.-K.K., H.-J.L., J.-W.C.; visualization, H.-J.K.; supervision, Y.-K.K., J.-W.C.; project administration, J.-W.C.; funding acquisition, J.-W.C., All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Korean government (MSIT) No. 2019R1I1A3A01058375. This paper was funded by Jeonbuk National University in 2020.

Acknowledgments: This work was supported by a National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. 2019R1I1A3A01058375). This paper was supported by research funds of Jeonbuk National University in 2020.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hayes, B. Cloud computing. *Commun. ACM* **2008**, *51*, 9–11. [CrossRef]
2. Qian, L.; Luo, Z.; Du, Y.; Guo, L. Cloud computing: An overview. In Proceedings of the IEEE International Conference on Cloud Computing, Bangalore, India, 21–25 September 2009; pp. 626–631.
3. Grolinger, K.; Higashino, W.A.; Tiwari, A.; Capretz, M.A. Data management in cloud environments: NoSQL and NewSQL data stores. *J. Cloud Comput. Adv. Syst. Appl.* **2013**, *2*, 1–24. [CrossRef]
4. Zhao, L.; Sakr, S.; Liu, A.; Bouguettaya, A. *Cloud Data Management*; Springer: Berlin/Heidelberg, Germany, 2014; 189p.
5. Agrawal, D.; Das, S.; Abbadi, A.E. Data management in the cloud: Challenges and opportunities. In *Synthesis Lectures on Data Management*; Springer: Nature/Cham, Switzerland, 2012; Volume 4, 138p.
6. Sun, Y.; Zhang, J.; Xiong, Y.; Zhu, G. Data security and privacy in cloud computing. *Int. J. Distrib. Sens. Netw.* **2014**, *10*, 190903. [CrossRef]
7. Sharma, Y.; Gupta, H.; Khatri, S.K. A security model for the enhancement of data privacy in cloud computing. In Proceedings of the 2019 Amity International Conference on Artificial Intelligence (AICAI), Dubai, United Arab Emirates, 4–6 February 2019; pp. 898–902.
8. Garigipati, N.; Krishna, R.V. A study on data security and query privacy in cloud. In Proceedings of the 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 23–25 April 2019; pp. 337–341.
9. Cao, N.; Yang, Z.; Wang, C.; Ren, K.; Lou, W. Privacy-preserving query over encrypted graph-structured data in cloud computing. In Proceedings of the 2011 31st International Conference on Distributed Computing Systems, Minneapolis, MN, USA, 20–24 June 2011; pp. 393–402.

10. Islam, M.S.; Kuzu, M.; Kantarcioglu, M. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In Proceedings of the 19th Annual Network and Distributed System Security Symposium, San Diego, CA, USA, 5–8 February 2012; p. 12.
11. Williams, P.; Sion, R.; Carbunar, B. 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, Alexandria, VI, USA, 27–31 August 2008; pp. 139–148.
12. Cui, S.; Belguith, S.; Zhang, M.; Asghar, M.R.; Russello, G. Preserving access pattern privacy in sgx-assisted encrypted search. In Proceedings of the 2018 27th International Conference on Computer Communication and Networks (ICCCN), Hangzhou, China, 30 July–2 August 2018; pp. 1–9.
13. Yiu, M.L.; Ghinita, G.; Jensen, C.S.; Kalnis, P. Enabling search services on outsourced private spatial data. *VLDB J.* **2010**, *19*, 363–384. [CrossRef]
14. Boldyreva, A.; Chenette, N.; Lee, Y.; Oneill, A. Order-preserving symmetric encryption. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, 26–30 April 2009; pp. 224–241.
15. Boldyreva, A.; Chenette, N.; O'Neill, A. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In Proceedings of the Annual Cryptology Conference, Santa Barbara, CA, USA, 14–18 August 2011; pp. 578–595.
16. Qi, Y.; Atallah, M.J. Efficient privacy-preserving k-nearest neighbor search. In Proceedings of the 28th International Conference on Distributed Computing Systems, Beijing, China, 17–20 June 2008; pp. 311–319.
17. Shaneck, M.; Kim, Y.; Kumar, V. Privacy preserving nearest neighbor search. In *Machine Learning in Cyber Trust*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 247–276.
18. Vaidya, J.; Clifton, C. Privacy-preserving top-*k* queries. In Proceedings of the 21st International Conference on Data Engineering (ICDE'05), Tokyo, Japan, 5–8 April 2005; pp. 545–546.
19. Elmehdwi, Y.; Samanthula, B.K.; Jiang, W. Secure k-nearest neighbor query over encrypted data in outsourced environments. In Proceedings of the 2014 IEEE 30th International Conference on Data Engineering, Chicago, IL, USA, 31 March–4 April 2014; pp. 664–675.
20. Kim, H.J.; Kim, H.I.; Chang, J.W. A privacy-preserving kNN classification algorithm using Yao's garbled circuit on cloud computing. In Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, HI, USA, 25–30 June 2017; pp. 766–769.
21. Zhou, L.; Zhu, Y.; Castiglione, A. Efficient k-NN query over encrypted data in cloud with limited key-disclosure and offline data owner. *Comput. Secur.* **2017**, *69*, 84–96. [CrossRef]
22. Kim, H.I.; Kim, H.J.; Chang, J.W. A secure kNN query processing algorithm using homomorphic encryption on outsourced database. *Data Knowl. Eng.* **2019**, *123*, 101602. [CrossRef]
23. Sun, X.; Wang, X.; Xia, Z.; Fu, Z.; Li, T. Dynamic multi-keyword top-*k* ranked search over encrypted cloud data. *Int. J. Secur. Its Appl.* **2014**, *8*, 319–332. [CrossRef]
24. Zhang, W.; Liu, S.; Xia, Z. A distributed privacy-preserving data aggregation scheme for smart grid with fine-grained access control. *J. Inf. Secur. Appl.* **2022**, *66*, 103118. [CrossRef]
25. Hozhabr, M.; Asghari, P.; Javadi, H.H.S. Dynamic secure multi-keyword ranked search over encrypted cloud data. *J. Inf. Secur. Appl.* **2021**, *61*, 102902. [CrossRef]
26. Ilyas, I.F.; Beskales, G.; Soliman, M.A. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.* **2008**, *40*, 1–58. [CrossRef]
27. Kim, H.I.; Kim, H.J.; Chang, J.W. A privacy-preserving top-*k* query processing algorithm in the cloud computing. In Proceedings of the International Conference on the Economics of Grids, Clouds, Systems, and Services, Athens, Greece, 20–22 September 2016; pp. 277–292.
28. Kim, H.J.; Chang, J.W. A new Top-*k* query processing algorithm to guarantee confidentiality of data and user queries on outsourced databases. *Int. J. Syst. Assur. Eng. Manag.* **2019**, *10*, 898–904. [CrossRef]
29. Yao, A.C.C. How to generate and exchange secrets. In Proceedings of the 27th Annual Symposium on Foundations of Computer Science, Toronto, ON, Canada, 27–28 October 1986; pp. 162–167.
30. Lindell, Y.; Pinkas, B. A proof of security of Yao's protocol for two-party computation. *J. Cryptol.* **2009**, *22*, 161–188. [CrossRef]
31. Bentley, J.L. Multidimensional binary search trees used for associative searching. *Commun. ACM* **1975**, *18*, 509–517. [CrossRef]
32. Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Prague, Czech Republic, 2–6 May 1999; pp. 223–238.
33. Lindell, Y. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*; IGI Global: Hershey, PA, USA, 2005; pp. 1005–1009.
34. Hazay, C.; Lindell, Y. *Efficient Secure Two-Party Protocols: Techniques and Constructions*; Springer Science & Business Media: Berlin, Germany, 2010.
35. Cramer, R.; Damgård, I.B. *Secure Multiparty Computation*; Cambridge University Press: Cambridige, UK, 2015.
36. Hazay, C.; Lindell, Y. A note on the relation between the definitions of security for semi-honest and malicious adversaries. Available online: https://eprint.iacr.org/2010/551.pdf (accessed on 7 September 2022).
37. Veugen, T.; Blom, F.; de Hoogh, S.J.; Erkin, Z. Secure comparison protocols in the semi-honest model. *IEEE J. Sel. Top. Signal Process.* **2015**, *9*, 1217–1228. [CrossRef]

38. Vaidya, J.; Clifton, C.W. Privacy-preserving kth element score over vertically partitioned data. *IEEE Trans. Knowl. Data Eng.* **2008**, *21*, 253–258. [CrossRef]
39. Fagin, R. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.* **1999**, *58*, 83–99. [CrossRef]
40. Burkhart, M.; Dimitropoulos, X. Fast privacy-preserving top-*k* queries using secret sharing. In Proceedings of 19th International Conference on Computer Communications and Networks, Zurich, Switzerland, 2–5 August 2010; pp. 1–7.
41. Zheng, Y.; Lu, R.; Yang, X.; Shao, J. Achieving efficient and privacy-preserving top-*k* query over vertically distributed data sources. In Proceedings of the ICC 2019-2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6.
42. Goldreich, O. *Foundations of Cryptography: Volume 2, Basic Applications*; Cambridge University Press: Cambridge, UK, 2009.
43. Chess (King-Rookvs.King) DataSet. Available online: http://archive.ics.uci.edu/ml/datasets/Chess+%28King-Rook+vs.+King%29 (accessed on 7 July 2022).
44. Kim, H.J.; Lee, H.; Kim, Y.K.; Chang, J.W. Privacy-preserving kNN query processing algorithms via secure two-party computation over encrypted database in cloud computing. *J. Supercomput.* **2022**, *78*, 9245–9284. [CrossRef]
45. Kim, Y.K.; Kim, H.J.; Lee, H.; Chang, J.W. Privacy-preserving parallel kNN classification algorithm using index-based filtering in cloud computing. *PLoS ONE* **2022**, *17*, e0267908. [CrossRef] [PubMed]