ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)

https://doi.org/10.17559/TV-20230415000536
Original scientific paper

# Enabling Access Control for Encrypted Multi-Dimensional Data in Cloud Computing through Range Search

Zhuolin MEI, Jing YU, Jinzhou HUANG*, Bin WU, Zhiqiang ZHAO, Caicai ZHANG, Zongda WU

**Abstract:** With the growing popularity of cloud computing, data owners are increasingly opting to outsource their data to cloud servers due to the numerous benefits it offers. However, this outsourcing raises concerns about data privacy since the data stored on remote cloud servers is not directly controlled by the owners. Encryption of the data is an effective approach to mitigate these privacy concerns. However, encrypted data lacks distinguishability, leading to limitations in supporting common operations such as range search and access control. In this research paper, we propose a method called RSAC (Range Search Supporting Access Control) for encrypted multi-dimensional data in cloud computing. Our method leverages policy design, bucket embedding, algorithm design, and Ciphertext Policy-Attribute Based Encryption (CPABE) to achieve its objectives. We present extensive experimental results that demonstrate the efficiency of our method and conduct a thorough security analysis to ensure its robustness. Our proposed RSAC method addresses the challenges of range search and access control over encrypted multi-dimensional data, thus contributing to enhancing privacy and security in cloud computing environments.

**Keywords:** access control; bucket embedding; ciphertext policy-attribute based encryption; cloud computing; policy design; range search

## 1 INTRODUCTION

With the rapid development of the Internet, businesses, organizations, and other institutions are collecting an increasing amount of data [1-3]. Driven by lower costs, higher reliability, better performance, and faster deployment, an increasing number of data owners are willing to outsource their data to cloud servers [4-6]. However, cloud servers cannot be fully trusted since they may be attacked, compromised by competitors of data owners, or infiltrated by curious insiders [7-9]. Therefore, the privacy of outsourced data has become a significant concern for data owners [4, 10-12]. Data encryption is considered one of the most effective methods to address this issue. Specifically, data owners can encrypt their data before outsourcing it to cloud servers to prevent information leakage. However, encrypted data are indistinguishable, which makes it difficult to support many common operations, such as range search and access control.

In recent years, numerous encryption methods [13-22] have been proposed to support common operations over ciphertexts. However, they still have limitations. Order-preserving encryption (OPE) was first proposed by Boldyreva et al. in 2009, and its security analysis was later improved in [14]. In OPE schemes [13, 14], data are encrypted in an ordered manner, allowing for efficient execution of range searches over ciphertexts. However, these schemes are not very secure because the plaintexts can be estimated precisely by analyzing the ordering information of the ciphertexts. To protect the ordering information of ciphertexts, some bucketization methods [16-18, 21] were proposed. Lee et al. [21] propose an ordered bucketization method in which all data in the same bucket are encrypted as a unit, and a partial order exists among different buckets. As a result, range searches can be executed efficiently, and the ordering information of plaintexts can also be protected to a great extent. Zheng et al. [23] propose a secure range search method over encrypted multi-dimensional data using matrix encryption. Recently, Yu Zhan et al. [24] propose an efficient multi-dimensional data order-preserving encryption (MDOPE) method that uses Bloom filters, prefix encoding, and one-way hash functions to create a secure index for all multi-dimensional data. However, these schemes [16-18, 21, 24] do not address the issue of access control over ciphertexts since all users share the same secret keys.

To enable access control over ciphertexts, numerous public key-based range search methods [15, 19, 20] have been proposed. In these methods, a user first sends a query request to the data owner, who verifies the user's identity. If the user is authorized, the data owner generates a search token and sends it to the user, who then sends the token to the cloud server to perform the range search. However, this procedure has some drawbacks: (i) the data owner must be online at all times to respond to users' search requests, and (ii) many cumbersome works must be performed by the data owner, such as verifying users' identities and distributing different search tokens to different users. Typically, both the data owner and users prefer that the powerful cloud servers handle all these works for them. Therefore, this range search procedure contradicts the original intention of adopting cloud servers.

After reviewing existing methods, we have found that while current range query methods are efficient, they do not provide access control during the query process because all users share the same key. This means that once the data owner sends the key to a user, that user can query all ciphertexts owned by the data owner. Furthermore, because all users share the same key, if one user's key is compromised, all ciphertexts owned by the data owner become vulnerable. Many publickey-based range search methods can support access control well, users must communicate with the data owner to obtain a search token before each query, which the data owner can restrict the user's query scope by distributing the appropriate search token to the user. This results in a cumbersome query process that requires multiple rounds of communication between users and the data owner. However, supporting both access control and range search is crucial in many scenarios. In the healthcare domain [25, 26], doctors need to access patients' medical records while protecting sensitive information, such as limiting access to information relevant to their medical field. Similarly, in the financial sector [27], bank employees need to access customer account information while protecting customer privacy, such as limiting access to specific transaction records.

Given these issues, we aim to establish a search method that allows users to perform range queries on ciphertexts only within their own privileges. In this paper, we propose a method called Range Search supporting Access Control over encrypted multi-dimensional data in cloud computing (RSAC). In RSAC, the data owner defines multiple buckets, and all multi-dimensional data within the same bucket are encrypted using a secure encryption method (such as AES [28]) and stored in the bucket. For each bucket, the data owner constructs a bucket policy that can be used to verify whether a queried range covers the ciphertexts in the bucket and whether a user has privilege to access the ciphertexts in the bucket. Using the bucket policy and the Ciphertext Policy-Attribute Based Encryption (CP-ABE) method, the data owner can build a secure index. After creating secure indexes for all the buckets and generating separate secret keys for different users according to their privileges, a user can generate search tokens for queried ranges using his/her secret key. The user can only search the ciphertexts that meet his/her privileges and queried ranges.

The contributions of this paper are listed as follows.

(1) We construct a bucket policy and secure indexes over encrypted multi-dimensional data.

(2) We propose an RSAC method that enables range search and access control over encrypted multi-dimensional data in the cloud server simultaneously.

(3) We conducted extensive experiments to evaluate the performance of RSAC and discuss its security. The experimental results demonstrate the exceptional performance of RSAC.

The remainder of this paper is structured as follows: Section 2 presents related work. Section 3 introduces the construction of our scheme. Section 4 presents the experimental results. Section 5 offers discussions. Finally, Section 6 presents the conclusions.

## 2 RELATED WORKS

Numerous methods have been proposed to support range search over encrypted data, including order-preserving encryption methods, bucketization methods, Bloom filter-based methods, and public key-based methods.

(1) In order-preserving encryption methods [13, 14], all data are encrypted in a specific order. For example, given two numeric data points $x_1$ and $x_2$, if $x_1 < x_2$, then there exists $E_{nc}(x_1) < E_{nc}(x_2)$ where $\mathrm{E}_{nc}(x_1)$ and $\mathrm{E}_{nc}(x_2)$ are the ciphertexts of $x_1$ and $x_2$, respectively. As a result, comparison operations between ciphertexts can be efficiently executed, enabling range search over ciphertexts. However, the comparison operation between ciphertexts risks revealing the ordering information, which can be exploited by attackers to precisely deduce plaintext values from ciphertexts. Thus, order-preserving encryption methods are not considered very secure. Tian et al. [29] introduce EAFS which enables range search over encrypted numerical databases while providing forward privacy. EAFS leverages OPE, pseudorandom function, and one-time pad to achieve efficiency, privacy, and accuracy. Chain-like search and embedded ciphertexts of OPE are also utilized in the scheme. However, EAFS does not consider the scenario of multi-users.

(2) In bucketization methods [16, 17], the data owner divides all data into multiple buckets. All data within a bucket are encrypted using a secure encryption method (e.g., AES [28]). If a search range intersects with a bucket, all ciphertexts within the bucket are returned as search results. To improve the search efficiency of bucketization methods [16, 17], Lee et al. [21] sorted all the buckets. By using these ordered buckets, range search can be efficiently performed. Wang et al. [18] constructed a secure index by embedding buckets under the leaf nodes of a normal R-tree and using asymmetric scalar product preserving encryption (ASPE) [30] to enhance the security of the normal R-tree. A user encrypts his/her search range and then performs range search over the secure index in a top-down manner. However, these methods share the same secret keys among users, making them unsuitable for access control over ciphertexts.

(3) Bloom filter-based methods [24, 31] have the potential to support efficient search, as the Bloom filter can be used to test whether an element is in a set by checking two bit strings. In [24], Li et al. used a Bloom filter, prefix encoding [32], and one-way hash functions to construct a secure index. First, a data owner constructs a tree index. Each internal node in the tree index contains several split data. The data owner encodes these split data into prefix codes and compute the hash values of these codes. Subsequently, the data owner inserts these hash values into Bloom filters to generate bit strings. During range search, a user computed the bit strings of the queried range using the same method. By comparing the bit strings of split data and queried range, the cloud server could efficiently perform range search over the index. However, as all users shared the same secret keys, Li's method did not support access control over ciphertexts.

(4) In public key-based methods [15, 19, 20], a data owner uses a public key to encrypt data and a secret key to generate search tokens. Users request search tokens from the data owner and then perform range searches on cloud servers. Shi et al. [19] proposed a method to support range search over encrypted multi-dimensional data. In Shi's method, a data owner constructs interval trees and encrypts each data using identities of nodes in these interval trees. Users request search tokens from the data owner. The data owner generates search tokens also using identities of nodes in these interval trees and distributes search tokens to users. If some identities in an encrypted data matched some identities in a search token, the encrypted data could be retrieved. Lu et al. [20] constructed a secure $\mathrm{B}^+$ tree index using the encryption method in [33] to encrypt a normal $\mathrm{B}^+$ tree. Although logarithmic range searches are supported, the adopted encryption method [33] is computationally expensive, making Lu's method inefficient. The main drawbacks of these schemes are that legal users need to request search tokens from the data owner before performing range searches, and the data owner needs to be continuously online to respond to users' requests, verify users' identities, and generate search tokens for them. Recently, Wu et al. [22] achieved range queries and verification over encrypted multi-dimensional data.

However, because the data owner distributes the same secret key to multiple users, their method does not support access control over ciphertexts.

## 3 CONSTRUCTION OF OUR METHOD

In this paragraph, we first introduce the CP-ABE method used in the underlying layer, then present the overall structure of our proposed method, and finally describe the detailed construction details of the method.

Definition 1. A Ciphertext Policy-Attribute Based Encryption (CP-ABE) method [34] consists of the following five algorithms.

$Setup(\lambda) \rightarrow (PK, MK)$ It takes a security parameter $\lambda$ as input. It outputs a public key *PK* and a master key *MK*.

$Encrypt(PK, M, P) \rightarrow CT$ . It takes the public key *PK*, a plaintext *M* and a policy *P* as input. It outputs the ciphertext *CT* of *M*.

$KeyGen(MK, S) \rightarrow SK$ . It takes the master key *MK* and an attribute set *S* as input. It outputs a secret key *SK*.

$Delegate(SK, \tilde{S}) \rightarrow \widetilde{SK}$ . It takes the private key *SK* and a set $\tilde{S} \subseteq S$ as input. It outputs a secret key $\widetilde{SK}$ for the attributes in $\tilde{S}$ .

$Decrypt(PK, CT, \widetilde{SK}) \rightarrow M$ . It takes the public key *PK*, a ciphertext *CT* (*P* is embedded in *CT*) and a secret key $\widetilde{SK}$ ( $\tilde{S}$ is embedded in $\widetilde{SK}$ ) as input. If attributes in $\tilde{S}$ satisfy *P*, then the algorithm decrypts *CT* and outputs *M*. Otherwise, the algorithm outputs $\perp$ .

In the above algorithms, $G_0$ and $G_1$ are two multiplicative cyclic groups of prime order *p*, *g* is the generator of $G_0$ and *e* is a bilinear map, i.e. $e: G_0 \times G_0 \rightarrow G_1$ . For all $u, v \in G_0$ and $a, b \in Z_p$ , there are $e(u^a, v^b) = e(u, v)^{ab}$ and $e(g, g) \neq 1$ .

Fig. 1 shows the overview of our method RSAC, which involves three types of entities: (i) an honest but curious cloud server that follows all designated protocols and procedures to fulfil its role as a service provider, but may be curious either due to personal interest or through a third-party compromise [35]; (ii) a data owner who securely outsources his/her data and services to the cloud server; and (iii) users who want to perform range searches over the outsourced data in the cloud server.

In our method RSAC, the data owner first encrypts the multi-dimensional data and stores them in several buckets. Then, the data owner constructs a bucket policy for each bucket based on user information (such as identity information, major, job title, etc.) and bucket information (e.g. bucket boundaries). Using the CP-ABE method and bucket policies, the data owner generates a secure index for each bucket. Finally, the data owner stores the buckets containing the encrypted data and secure indexes on the cloud server. The data owner assigns keys to different users based on their information (such as identity information, major, job title, etc.). Users use their obtained keys to generate search tokens for the queried range. Only when a secure index satisfies both the user query conditions and user privileges (i.e. the user information meets the access control conditions in the secure index), the ciphertexts in the bucket indexed by the secure index will be returned to the user as a query result. Specifically, the data owner defines buckets, denoted by $B_1, \ldots, B_n$, where *n* is the total number of buckets. All multi-dimensional data are partitioned into these buckets. Let $B = (a_1, b_1] \times \cdots \times (a_d, b_d]$ be one of these buckets, where $(a_i, b_i]$ is the range on the *i*th dimension, and *d* is the dimensionality. All multi-dimensional data in *B* are encrypted using a secure encryption method (e.g., AES [28]). The data owner then defines a bucket policy $p_B$ for *B*, which consists of a search policy $p_{sr}$ and an access control policy $p_{ac}$. The search policy $p_{sr}$ checks whether a queried range intersects with *B*, while the access control policy verifies whether a user has the privilege to access the cipher texts in *B*. The bucket policy checks whether a queried range intersects with and whether a user has the privilege to access the ciphertexts in simultaneously. Using the bucket policy $p_B$, the data owner builds a secure index $Idx_B$ for the encrypted multi-dimensional data. Once all multi-dimensional data is encrypted, and all the secure indexes $Idx_{B_1}, \ldots, Idx_{B_n}$ are built, the data owner outsources the encrypted multi-dimensional data and secure indexes $Idx_{B_1}, \ldots, Idx_{B_n}$ to the cloud server. The data owner then distributes secret keys to users. By using these secret keys, users can generate search tokens by themselves. Finally, after receiving these search tokens, the cloud server can perform range searches only on the ciphertexts that users have privileges to access and then return the retrieved ciphertexts as search results.

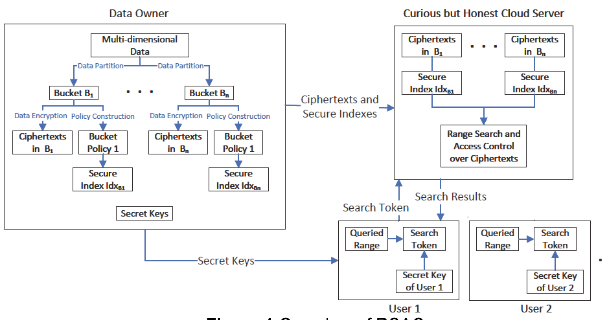

**Figure 1** Overview of RSAC

RSAC consists of five algorithms: System Setup, Policy Construction, Index Construction, Secret Key Generation, Search Token Generation, Range Search, and Access Control over Ciphertexts. The details of these algorithms are as follows.

System Setup. During the systemstep, the data owner encrypts all the multi-dimensional data based on its value and stores them in the corresponding buckets. The data owner takes the user's attributes (such as identity information, major, job title, etc.) and the bucket's boundary values as the attribute set for our RSAC method, and performs the CP-ABE method to generate system parameters (including the public key PK and the master key MK). The detailed process is as follows.

In RSAC, buckets can be flexibly defined by the data owner according to different situations. For example, if the multi-dimensional data is uniformly distributed, the data owner can define all buckets to be of the same size. If the data is unevenly distributed, the data owner can define buckets according to the density distribution of the multi-dimensional data. Specifically, the data owner can define more buckets in areas with high data density and fewer buckets in areas with low data density. As buckets are defined differently in various situations, details on how to define buckets are not considered in our method. However, there is a requirement which must be satisfied when defining these buckets. Suppose (i) all multi-dimensional data is covered by a multi-dimensional range *MR*, and (ii) $B_1$, …, $B_n$ are the buckets defined by the data owner (where *n* is the total number of all buckets). To ensure that any multi-dimensional data can be retrieved, it must satisfy the condition $B_1 \cup \cdots \cup B_n = MR$.

The data owner defines a set $S_{bou}$, which is the collection of all the boundary information of $B_1$, …, $B_n$. Suppose $B = (a_1, b_1] \times \cdots \times (a_d, b_d]$, where $B \in \{B_1, \ldots, B_n\}$. The collection of all the boundary information of *B* is $S_{bou(B)} = \{a_1\} \cup \{b_1\} \cup \cdots \cup \{a_d\} \cup \{b_d\}$. Thus, it is easy to know that the collection of all the boundary information of buckets is $S_{bou} = \bigcup_{B \in \{B_1, \ldots, B_n\}} S_{bou(B)}$. Then, the data owner defines a set $S_{att}$, which is the collection of all the attributes of users. To illustrate $S_{att}$ clearly, we give the following Example 1.

Example 1. $\langle StuID, Score, Age \rangle$ is a tuple in a database table. The tuple can be viewed as a multi-dimensional data. *B* is a bucket associated with the policy, $professor\ OR\ (student\ AND\ computerScience)$. This policy means that only professors or students of computer science can access the cipher texts in *B*. In this example, it is easy to know that the attribute set $S_{att}$ is $\{student, professor, computerScience, \ldots\}$.

The data owner runs the algorithm $Setup$ in CP-ABE [25, 34]. $Setup$ takes a security parameter $\lambda$ as input, and outputs a public key $PK = \{G_0, g, h = g^{\beta}, f = g^{1/\beta}, e(g,g)^{\alpha}\}$ and a master key $MK = \{\beta, g^{\alpha}\}$, where $G_0$ is a bilinear group, *p* is the prime order of $G_0$, *g* is the generator of $G_0$, $\alpha$ and $\beta$ are randomly chosen from $Z_p$. In the above parameters, $S_{att}$, $\{B_1, \ldots, B_n\}$ (as $\{B_1, \ldots, B_n\}$ is public, the set of boundary information $S_{bou}$ is also public) and *PK* are public, but *MK* is secretly kept by the data owner.

Policy Construction. This section introduces three types of policies: bucket policy, query policy, and access control policy. These polices are important parts when generating secure indexes. The bucket policy consists of a query policy and an access control policy. The query policy is used to support range queries, and the access control policy is used to determine if a user has the privilege to access the cipher texts in a bucket. Thus, the bucket policy can support both range queries and access control, and all three policies can be converted into tree structures.

In general, let $B = (a_1, b_1] \times \cdots \times (a_d, b_d]$ be a bucket. For the bucket *B*, the data owner defines a search policy $p_{sr}$ and an access control policy $p_{ac}$. The search policy $p_{sr}$ is used to check whether a queried range intersects with *B*. The access control policy $p_{ac}$ is used to check whether a user has the privilege to access the ciphertextsin *B*. After constructing $p_{sr}$ and $p_{ac}$, the data owner combines $p_{sr}$ and $p_{ac}$ together to construct a bucket policy $p_B$. Thus, the bucket policy $p_B$ has the properties of both $p_{sr}$ and $p_{ac}$. Namely, suppose *R* is a range queried by a user, the user can search and access the ciphertexts in $R \cap B$ if and only if the queried range *R* satisfies $p_B$ and the user's attributes also satisfy $p_B$. In the following paragraphs, we first briefly illustrate the tree structures of policies. Then, we construct search policy, access control policy and bucket policy respectively.

Search policy, access control policy and bucket policy can be transformed into tree structures. In these tree structures, each internal node is associated with a threshold gate. If an internal node is associated with the threshold gate $TG(t', t)$, it means that (i) the internal node has *t* children; (ii) $TG(t', t)$ returns true if and only if at least $t'$ children return true. It is easy to know: (i) OR can be represented by $TG(1, t)$; (ii) AND can be represented by $TG(t, t)$. In these tree structures, each leaf node is associated with an element. Specifically, the leaf nodes in the tree structure of a search policy are associated with the elements in $S_{bou}$, and the leaf nodes in the tree structure of an access control policy are associated with the elements in $S_{att}$. As a bucket policy consists of a search policy and an access control policy, some leaf nodes in the tree structure of a bucket policy are associated with the elements in $S_{bou}$ and the other leaf nodes are associated with the elements in $S_{att}$.

(1) Search policy and its tree structure. $B = [a_1, b_1] \times \cdots \times [a_d, b_d]$ denotes a bucket. $R = [a_1^R, b_1^R] \times \cdots \times [a_d^R, b_d^R]$ denotes a queried range. If *R* can be used to search the ciphertexts in *B*, there are $[a_1^R, b_1^R] \cap [a_1, b_1] \neq \varnothing$, …, $[a_d^R, b_d^R] \cap [a_d, b_d] \neq \varnothing$. Thus, the search policy $p_{sr}$ for *B* can be represented as

$$p_{sr} = \left([a_1^R, b_1^R] \cap [a_1, b_1] \neq \varnothing\right) AND \cdots AND \left([a_d^R, b_d^R] \cap [a_d, b_d] \neq \varnothing\right).$$

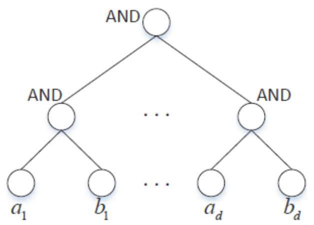

**Figure 2** Tree structure of search policy

According to the representation of $p_{sr}$, we can construct the tree structure $T_{p_{sr}}$ of $p_{sr}$. As shown in Fig. 2, $T_{p_{sr}}$ is the tree structure of $p_{sr}$. Each leaf node of $T_{p_{sr}}$ is associated with an element in $S_{bou(B)} = \{a_1\} \cup \{b_1\} \cup \cdots \cup \{a_d\} \cup \{b_d\}$ (the boundary information of the bucket *B*). Each internal node of $T_{p_{sr}}$ is associated with AND (the threshold gate is $TG(2,2)$) and the root node of $T_{p_{sr}}$ is also associated with AND (the threshold gate is $TG(d,d)$).

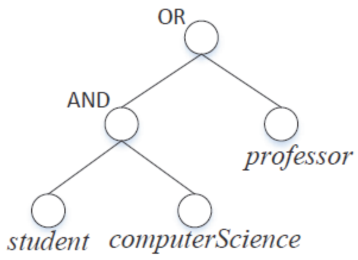

**Figure 3** Tree structure of access control policy

(2) Access control policy and its tree structure. For the bucket *B*, the data owner defines an access control policy $p_{ac}$. For ease of explanation, we use Example 1 to illustrate the access control policy $p_{ac}$. In Example 1, only professors or students of computer science have the privilege to access the cipher texts in *B*. Thus, the access control policy can be defined as $p_{ac} = professor\ OR\ (student\ AND\ computerScience)$. Fig. 3 shows the tree structure $T_{p_{ac}}$ of $p_{ac}$. Each leaf node of $T_{p_{ac}}$ is associated with an attribute in $S_{att} = \{professor, student, computerScience, \ldots\}$. An internal node is associated with AND (the threshold gate is $TG(2,2)$) and the root node is associated with OR (the threshold gate is $TG(1,2)$).

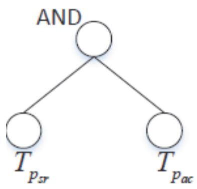

**Figure 4** Tree structure of bucket policy

(3) Bucket policy and its tree structure. For the bucket *B*, its bucket policy is denoted by $p_B$. To search and access the ciphertexts in *B*, a user's queried range should satisfy the search policy $p_{sr}$ and his/her privilege should satisfy the access control policy $p_{ac}$. Thus, the bucket policy of *B* can be defined as $p_B = p_{sr}$ AND $p_{ac}$. As $p_{sr}$ and $p_{ac}$ can be represented as tree structures $T_{p_{sr}}$ and $T_{p_{ac}}$ respectively, $p_B$ can be represented as a tree structure. Fig. 4 shows the tree structure $T_{p_B}$ of $p_B$. $T_{p_B}$ consists of two subtrees. One is $T_{p_{sr}}$ and the other is $T_{p_{ac}}$. These two subtrees are combined together by AND (the threshold gate is $TG(2,2)$), which indicates that a user can search and access the ciphertexts in *B* when his/her queried range satisfies the search policy $p_{sr}$ and privilege satisfies the access control policy $p_{ac}$.

**Index Generation.** This section introduces the method for generating secure indexes. Essentially, a secure index of a bucket is the ciphertext of its bucket ID. The encryption method we adopt is CP-ABE method. The data owner embeds the bucket policy into the ciphertext of the bucket ID, which enables the secure index maintains two characteristics of range search and access control in its bucket policy. Finally, this section introduces that the secure index hides the boundary information of the bucket in the secure index to ensure that the boundary information of the bucket is not leaked.

The data owner runs the algorithm *Encrypt* [34] to encrypt $ID_B$, where $ID_B$ is the identity of the bucket *B*. Then, the data owner uses the ciphertext of $ID_B$ to generate the secure index $Idx_B$ for *B*. Specifically, the algorithm *Encrypt* takes *PK*, $ID_B \,||\, 0^l$ and $T_{p_B}$ as input, and outputs the ciphertexts of $ID_B \,||\, 0^l$, where *PK* is the public key, $ID_B \,||\, 0^l$ denotes that $ID_B$ is appended by $0^l$ (*l* is an integer) and $T_{p_B}$ is the tree structure of the bucket policy $p_B$. Note that $ID_B \,||\, 0^l$ could help the cloud server to check whether a decryption is valid. If a decryption is a plain text appended by $0^l$, the decryption is valid. Otherwise, it is invalid.

In order to illustrate the secure index $Idx_B$ clearly, we briefly describe the algorithm *Encrypt* [34]. All the children of an internal node in $T_{p_B}$ are numbered from 1 to *num*. *index*(*x*) is a function which returns such a number of the node *x*. *parent*(*x*) is a function which returns the parent node of *x*. In $T_{p_B}$, each node *x* is associated with a polynomial $q_x$. Suppose *x* is associated with the threshold gate $T(t_{x'}, t_x)$, the degree of $q_x$ is set to $t_{x'} - 1$. For the root node $x_r$, *Encrypt* chooses a random number $s \in Z_p$ and sets $q_{x_r}(0) = s$. For any other node *x*, *Encrypt* sets $q_x(0) = q_{parent(x)}(index(x))$. *Encrypt* randomly chooses other points to completely define $q_{x_r}$ and $q_x$. Finally, *Encrypt* outputs the ciphertext $CT_B = \begin{bmatrix} T_{p_B}, \tilde{C} = (ID_B \,||\, 0^l) e(g,g)^{\alpha s}, C = h^s, \\ \forall x \in L : C_x = g^{q_x(0)}, C_{x'} = H(f(x))^{q_x(0)} \end{bmatrix}$, where *L* is the collection of leaf nodes in $T_{p_B}$, *f*(*x*) is the function which returns the element associated with the leaf node *x* and *H* is a hash function. Note that, in the algorithm *Encrypt*, all the elements associated with the leaf nodes of $T_{p_B}$ are in the form of plaintext. It is not secure when performing range searches. Thus, the algorithm *Encrypt*, should be slightly modified in our method RSAC.

In RSAC, $T_{p_{sr}}$ and $T_{p_{ac}}$ are two different sub-trees in $T_{p_B}$. All the leaf nodes of the sub-tree $T_{p_{ac}}$ are associated with the attributes in $S_{att}$. All the leaf nodes of the sub-tree $T_{p_{sr}}$ are associated with the boundary information in $S_{bou}$. As attributes do not leak information about the multi-dimensional data in *B*, these attributes in the leaf nodes of $T_{p_B}$ can be stored in the form of plaintext [34]. However, if the boundary information associated with the leaf nodes of $T_{p_B}$ are also stored in the form of plaintext, the range information of *B* will be leaked and a user's query purpose will also be leaked when his/her queried range *R* insects with *B*. To protect the boundary information in the leaf nodes, the algorithm *Encrypt* [34] should be slightly modified, and the algorithm *Decrypt* [34] should also be slightly modified (see the following paragraphs).

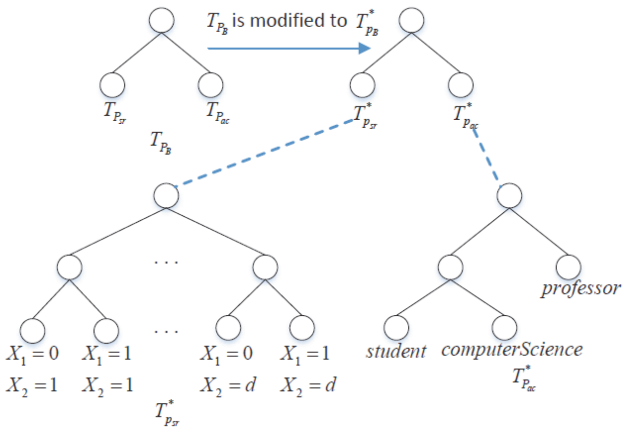

**Figure 5** The structure of $T^*_{PB}$

In the bucket $B=[a_1,b_1]\times\cdots\times[a_d,b_d]$, $a_i$ and $b_i$ ($i=1,\ldots,d$, $d$ is the dimensionality) are the boundary information on the *i*th dimension. In the sub-tree $T_{Psr}$ of $T_{P_B}$, if a leaf node is associated with $a_i$, then its sibling node is associated with $b_i$. For example, as shown in Fig. 2, a leaf node is associated with $a_1$ and its sibling node is associated with $b_1$. To hide the boundary information $a_i$ and $b_i$, the data owner replaces $a_i$ and $b_i$ with two symbols $X_1$ and $X_2$, where $X_1$ = 0, 1 and $X_2 = i$. If $X_1$ = 0 and $X_2 = i$, ($X_1$, $X_2$) denotes the minimal value $a_i$ in the range [$a_i$, $b_i$]. If $X_1$ = 1 and $X_2 = i$, ($X_1$, $X_2$) denotes the maximal value $b_i$ in the range [$a_i$, $b_i$]. For example, in Fig. 5, we can use ($X_1$ = 1, $X_2$ = 1) to denote $a_1$ and ($X_1$ = 1, $X_2$ = 1) to denote $b_1$. In this way, without knowing the exact values of $a_i$ and $b_i$, the search token and secure indexes can be quickly matched without leaking any boundary information of a bucket and next can be used to try to perform decryption. Thus, the data owner adds ($X_1$, $X_2$) to the leaf nodes in the sub-tree $T_{Psr}$ and deletes the boundary information $a_i$ and $b_i$ of leaf nodes in the sub-tree $T_{Psr}$. Additionally, as the threshold gate (AND or OR) of each internal node $x$ has already been embedded into the polynomial $q_x$, the data owner also deletes the threshold gate (AND or OR) associated with $x$ in the tree $T_{P_B}$. As shown in Fig. 5, the modified $T_{Psr}$ is denoted by $T^*_{Psr}$, the modified $T_{Pac}$ is denoted by $T^*_{Pac}$ and the modified $T_{P_B}$ is denoted by $T^*_{PB}$.

According to the ciphertext $CT_B$, the data owner constructs the secure index $Idx_B$ for $B$ as follows.

$$Idx_B=\left[\begin{array}{l} T^*_{PB}, \tilde{C}=(ID_B \,\|\, 0^l)e(g,g)^{\alpha s}, C=h^s, \\ \forall x\in L(T_{Psr}): \\ C_x=g^{q_x(0)}, C'_x=H(f(x))^{q_x(0)}, (X_1,X_2); \\ \forall x\in L(T_{Pac}): \\ C_x=g^{q_x(0)}, C'_x=H(f(x))^{q_x(0)}, att; \end{array}\right]$$

, where $f(x)$ returns the boundary information (or attribute) associated with the leaf node $x$, $L(T_{Psr})$ is the set of leaf nodes in the sub-tree $T_{Psr}$, $L(T_{Psr})$ is the set of leaf nodes in the sub-tree $T_{Pac}$, $X_1$ = 0, 1, $X_2$ = 1,…, $d$. ($d$ is the dimensionality) and $att$ is the attribute associated with $x\in L(T_{Pac})$.

Secret Key Generation. The following paragraphs describe how the data owner uses the CP-ABE method to generate keys for users.

For each legal user $u$, the data owner runs the algorithm $KeyGen(MK, S_{bou}\cup S_{att_u})$ to generate a secret key $SK_u$, where $MK$ is the master key, $S_{bou}$ is the set of all the boundary information of buckets and $S_{att_u}$ is the set of all the attributes of $u$. The algorithm *KeyGen* [34] runs four steps to generate $SK_u$: (i) *KeyGen* randomly chooses a number $\gamma$ from $Z_p$; (ii) For each boundary information $bou\in S_{bou}$, *KeyGen* randomly chooses a number $\gamma_{bou}$ from $Z_p$; (iii) For each attribute $att\in S_{att_u}$, *KeyGen* randomly chooses a number $\gamma_{att}$ from $Z_p$; (iv) *KeyGen* computes the secret key as. After the secret key $SK_u$ has been generated, the data owner distributes $SK_u$ to $u$.

$$SK_u=\left[\begin{array}{l} D=g^{(\alpha+\gamma)/\beta}, \\ \forall bou\in S_{bou}: \\ D_{bou}=g^{\gamma}\cdot H(bou)^{\gamma_{bou}}, D_{bou}'=g^{\gamma_{bou}}, \\ \forall att\in S_{att_u}: \\ D_{att}=g^{\gamma}\cdot H(att)^{\gamma_{att}}, D_{att}'=g^{\gamma_{att}} \end{array}\right]$$

Search Token Generation. After receiving the secret key $SK_u$ from the data owner, the user $u$ generates the search token for a queried range $R$ by executing the following two steps.

First, the user $u$ computes the minimal cover $MC_R$ of $R$ (see Definition 2) by using the buckets $B_1$, …, $B_n$. We give the following Example 2 to illustrate the minimal cover of a queried range.

Definition 2. Minimal Cover. Given a queried range $R$, its minimal cover is a set of some buckets in {$B_1$, …, $B_n$}. $MC_R$ is the minimal cover of $R$, if and only if (i) $R\subseteq\bigcup_{B\in MC_R} B$ and (ii) when $\forall MC\in\{MC\mid MC\subseteq MC_R\}$, there is $R\not\subset\bigcup_{B\in MC} B$.

Example 2. As shown in Fig. 6, these points in the coordinate system represent 2 - dimensional data. These data are partitioned into the buckets $B_1$, $B_2$, $B_3$ and $B_4$. $R$ is a queried range. The minimal cover $MC_R$ is {$B_1$, $B_2$}., because (i) $R\subseteq B_1\cup B_2$ and (2) $R\not\subset B_1$ and $R\not\subset B_2$.

Second, the user $u$ generates the search token $T_R$ for the queried range $R$. Suppose $B=[a_1,b_1]\times\ldots\times[a_d,b_d]$ is a bucket in $MC_R$). The set of the boundary information of $B$ is $S_{bou(B)}=\{a_1\}\cup\{b_1\}\cup\ldots\cup\{a_d\}\cup\{b_d\}$. The search token $T_R$ for the queried range $R$ is as follows.

$$T_R = \begin{bmatrix} D = g^{(\alpha+\gamma)/\beta}, \\ \forall B \in MC_R \quad and \quad \forall bou \in S_{bou(B)}: \\ D_{bou} = g^{\gamma} \cdot H(bou)^{\gamma b}, D_{bou}' = g^{\gamma bou}, X_1, X_2; \\ \forall att \in S_{att_u}: \\ D_{att} = g^{\gamma} \cdot H(att)^{\gamma att}, D_{att}' = g^{\gamma att}, att; \end{bmatrix},$$

where $X_1 = 0, 1$ and $X_2 = 1, \ldots, d$, ($d$ is the dimensionality), $att$ is the attribute associated with the leaf node $x \in L\left(T_{p_{ac}}\right)$.

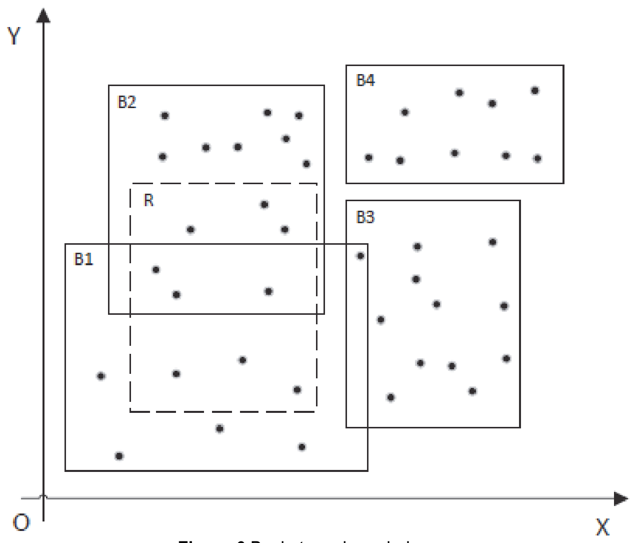

**Figure 6** Buckets and queried range

Note that the user $u$ may query different ranges and these queried ranges may have the same boundary information, e.g., $[3,6]\times[1,9]$ and $[2,6]\times[4,9]$ have the same boundary information 6 and 9. As all these search tokens are constructed according to the same secret key $SK_u$, the search tokens of different ranges may have the same secret key components. Thus, adversaries may analyze the secret key components across different search tokens to infer the boundary information of buckets. To prevent such attack, the user $u$ could run the delegate algorithm *Delegate* [34] to generate new secret key $SK_{u'}$, and then use $SK_{u'}$ to generate new search tokens. As new search tokens are totally different with old ones, adversaries can no longer infer the boundary information of buckets.

Range Search and Access Control over Ciphertexts. The cloud server matches the user's search token with the secure index. Once a match is found (i.e., the attributes in the search token match those in the secure index, and the boundary information $X_1$ and $X_2$ in the search token match those in the secure index), it tries to decrypt. If the result is the bucket ID, then all the ciphertext in the corresponding bucket is returned to the user as the query result. To realize range search and access control over ciphertexts, the decryption algorithm *Decrypt* [34] is adopted. *DecryptNode* is a sub-algorithm in *Decrypt*. We first illustrate *DecryptNode* and *Decrypt* in our method RSAC. Then, we represent the procedure of range search and access control over ciphertexts in the cloud server.

The cloud server runs $DecryptNode(Idx_B, T_R, x)$, where $Idx_B$ is the secure index of the bucket $B$, $T_R$ is the search token of the queried range $R$ and $x$ is a leaf node of $T_{PB}^{*}$ in $Idx_B$. As $X_1$ and $X_2$ (or $att$) are stored in the leaf node $x$ of $T_{PB}^{*}$ and also stored in the search token $T_R$, the cloud server can efficiently match (i) $(C_x, C_{x'})$ in $Idx_B$ with $\left(D_{ab}, D_{ab}'\right)$ in $T_R$ if ($X_1$, $X_2$) in $Idx_B$ equals to ($X_1$, $X_2$) in $T_R$, and (ii) $(C_x, C_{x'})$ in $Idx_B$ with $\left(D_{ab}, D_{ab}'\right)$ in $T_R$ if $att$ in $Idx_B$ equals to $att$ in $T_R$. Then, the cloud server computes $DecryptNode(Idx_B, T_R, x)$ as follows.

$$DecryptNode(Idx_B, T_R, x) =$$

$$= \frac{e(C_x, D_y)}{e(C_{x'}, D_{y'})} = \frac{e\left(h^{q_x(0)}, g^{\gamma} \cdot H(i)^{\gamma y}\right)}{e\left(H(i)^{q_x(0)}, g^{\gamma y}\right)} = e(g,g)^{\gamma q_x(0)},$$ where

(i) $D_y$ denotes $D_{bou}$ and $D_{y'}$ denotes $D_{bou}'$ when $x$ is a leaf node in the sub-tree $T_{p_{sr}}^{*}$ of $T_{PB}^{*}$; (ii) $D_y$ denotes $D_{att}$ and $D_{y'}$ denotes $D_{att}'$ when $x$ is a leaf node in the sub-tree $T_{p_{ac}}^{*}$ of $T_{PB}^{*}$. Otherwise, $DecryptNode(Idx_B, T_R, x)$ outputs $\perp$.

The cloud server runs $Decrypt(PK, Idx_B, T_R)$, where $PK$ is the public key, $Idx_B$ is the secure index and $T_R$ is the search token for the queried range $R$. The output of $DecryptNode(Idx_B, T_R, x)$ is denoted by $F_x$. $Decrypt(PK, Idx_B, T_R)$ calls $DecryptNode(Idx_B, T_R, x)$ in a top-down manner. For each child node $x'$ of $x$, $DecryptNode(Idx_B, T_R, x)$ calls $DecryptNode(Idx_B, T_R, x')$. The output of $DecryptNode(Idx_B, T_R, x')$ is denoted by $F_{x'}$. For illustration purposes, we suppose $x$ is associated with the threshold gate $T(t_{x'}, t_x)$. If $t_{x'}$ outputs of $F_{x'}$ are not $\perp$, the output of $DecryptNode(Idx_B, T_R, x)$ is $F_x = e(g,g)^{\gamma q_x(0)}$. Thus, when $x$ is the root node $x_r$ of $T_{PB}^{*}$, the output is $F_{x_r} = e(g,g)^{\gamma q_{x_r}(0)}$. Recall that $q_{x_r}(0) = s$. Thus, there is $F_{x_r} = e(g,g)^{\gamma s}$. The details about how the algorithm *Encrypt* calls the algorithm *DecryptNode* can be found in [34]. Finally, $Decrypt(PK, Idx_B, T_R)$ computes

$$\tilde{C} / \left(e(C,D) / e(g,g)^{\gamma s}\right) =$$

$$= \left(ID_B \,||\, 0^l\right) e(g,g)^{\alpha s} / \left(e\left(h^s, g^{(\alpha+\gamma)/\beta}\right) / e(g,g)^{\gamma s}\right) =$$

$$= ID_B \,||\, 0^l$$

and outputs the identity $ID_B$ of the bucket $B$.

The procedure of range search and access control over ciphertexts in the cloud server is as follows. After receiving a search token from a user $u$, the cloud server runs the above algorithm $Decrypt(PK, Idx_B, T_R)$ for all the secure indexes, where $Idx_B \in \left\{Idx_{B_1}, \ldots, Idx_{B_n}\right\}$. Then, all the identities of buckets which intersects with the queried range $R$ can be found out. The cloud server returns all the ciphertexts in these buckets to the user $u$ as the search results. Finally, $u$ decrypts these ciphertexts and filters out all the false positives.

## 4 RESULTS

In our experiments, we compare our method RSAC with the method MDOPE [24]. The method MDOPE is constructed based on Bloom filter, prefix encoding and network data structure. Our method RSAC is constructed based on policy design, bucket embedding and CP-ABE method [34]. The calculations about bilinear group in CP-ABE method [34] are implemented by using Java Pairing Based Cryptography Library 2.0.0 [36]. In order to compare RSAC and MDOPE fairly, we set the following experimental conditions: (i) As the method MDOPE does not support access control over ciphertexts, a user in MDOPE has to search all the ciphertexts to perform a range search. To enable a user to search all the ciphertexts in our method RSAC, the data owner constructs all the secure indexes by using the access control policy $p_{ac} = att_1\ and\ att_2$ ($att_1$ and $att_2$ are two attributes) and assigns these attributes $att_1$ and $att_2$ to the user. (ii) The experiments are conducted on five datasets with varying amounts of 2-dimensional data and different ranges, while maintaining a constant data density. The ranges of these datasets are (1,25) × (1,25), (1,50) × (1,50), (1,100) × (1,100), (1,200) × (1,200), and (1,400) × (1,400), respectively. To maintain the same data density, each range covers 100, 400, 1600, 6400, and 25600 2-dimensional data, respectively, which are randomly generated and evenly distributed within these ranges. In RSAC, each bucket contains 100 2-dimensional data, and in MDOPE, the number of split data in each node of the index is set to 1.

Our experiments are conducted on a Windows 10 computer with an AMD Ryzen 5 2500U CPU and 8 GB of RAM. During the experiments, we run each algorithm hundreds of times and calculate the average execution time for both RSAC and MDOPE.

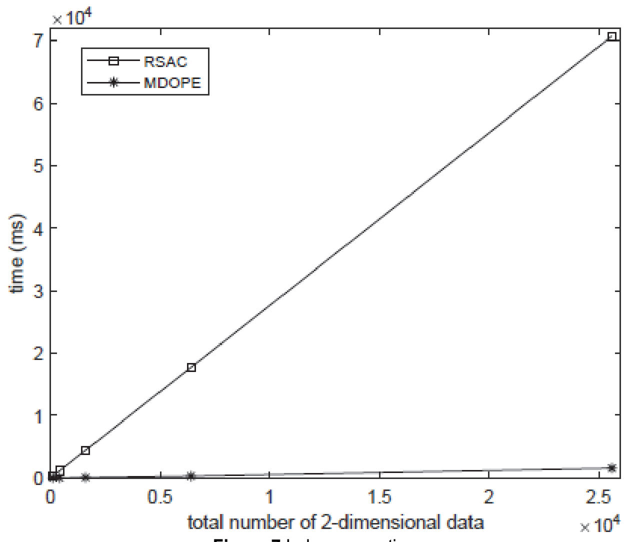

**Figure 7** Index generation

Index Generation Experimental Results: As shown in Fig. 7, the time required to generate indexes for both MDOPE and RSAC increases linearly with the amount of 2-dimensional data. Overall, MDOPE is more efficient than RSAC. For instance, generating indexes for 25600 2-dimensional data points takes RSAC approximately 1.15 minutes longer than MDOPE.

Analysis of the Results: In RSAC, index generation time increases linearly with the amount of 2-dimensional data due to (i) the need to build an index for each bucket, and (ii) the linear increase in the number of buckets with the amount of data. In MDOPE, index generation time increases with the amount of 2-dimensional data due to (i) the constant construction time of a node in the index, and (ii) the linear increase in the total number of nodes in the index with the amount of data. Unlike RSAC, MDOPE only converts the data in the index nodes to bit strings, calculates their hash and Bloom filter values, and this process is very efficient. In contrast, RSAC needs to perform calculations involving bilinear maps which are time-consuming. Consequently, MDOPE is more efficient than RSAC in generating indexes.

Note that the data owner first encrypts all the data and generates indexes. These indexes and encrypted data are then outsourced to a cloud server. Therefore, index generation can be considered a one-time setup process. We believe that the slightly lower efficiency of index generation in RSAC (about 1.15 minutes for 25600 2-dimensional data points) can be tolerated by the data owner.

Search Token Generation Experimental Results: As shown in Fig. 8, the time required to generate search tokens for RSAC increases linearly and slowly, while the time required for MDOPE increases exponentially. For example, when the amount of 2-dimensional data is 25600, RSAC takes only about 0.18 milliseconds on average to generate a search token, whereas MDOPE takes about 2 hours.

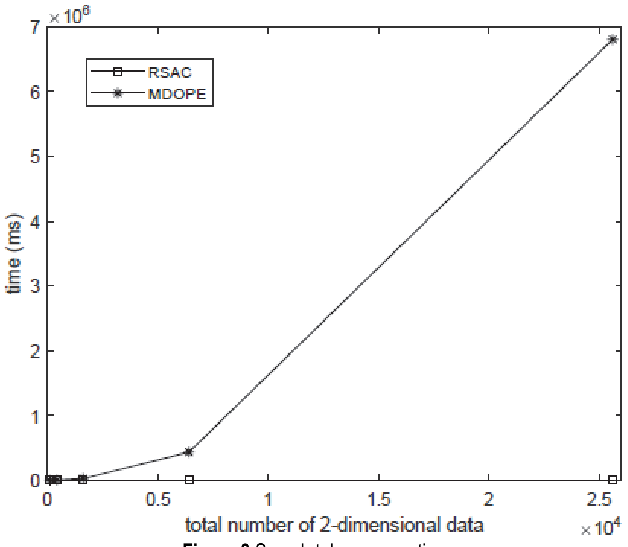

**Figure 8** Search token generation

Analysis of the Results: In our experiments, as the total number of 2-dimensional data points increases, the range covering all the data becomes larger while maintaining a constant data density. For example, when the total number is 100, the range is (1,25) × (1,25), and when the total number is 400, the range is (1,50) × (1,50). In MDOPE, when the range is (1,25) × (1,25), the data owner needs 12 bits (5 bits for encoding a split data in the range (1,25), 2 bits for supporting comparability, and 5 random bits appended at the end for security enhancement) to handle the split data in the index. Similarly, when the range is (1,50) × (1,50), the data owner needs 14 bits. To perform a range search over the queried range, a user must transform it into a bit string of length 12 bits or 14 bits and calculate the prefix codes for the values in the corresponding range. As the amount of 2-dimensional data points increases, the calculation workload for token generation in MDOPE increases exponentially, leading to an exponential increase in token generation time. Compared to MDOPE, our method RSAC is much more efficient because a user only needs to combine his/her secret key components to satisfy the queried range. As the size of the queried range and the range covering all the data increase, the user needs to do more work to generate search tokens, resulting in a linear and slow increase in token generation time for RSAC.

Range Search over Ciphertexts Experimental Results: As shown in Fig. 9, the time required for range search in both MDOPE and RSAC increases with the amount of 2-dimensional data. While both methods show similar trends in terms of increasing search time, MDOPE is more efficient than RSAC in the range search procedure. Fig. 10 shows the overall time required for MDOPE and RSAC from token generation to obtaining the search results. The results show that the overall time for RSAC is significantly lower than that for MDOPE.

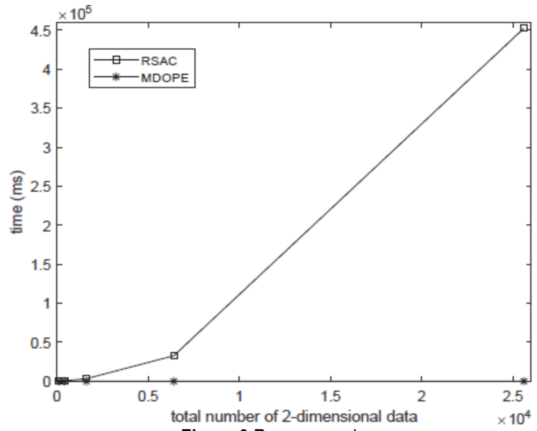

**Figure 9** Range search

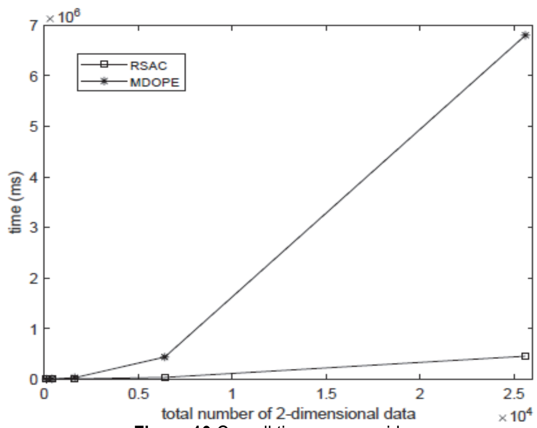

**Figure 10** Overall time on user side

Analysis of the Results: The search time of MDOPE shown in Fig. 9 can be attributed to two factors. First, the secure index in MDOPE is a tree structure, and during a range search, the algorithm only tests a few nodes in the index. However, when the amount of 2-dimensional data increases, the range covering all the data becomes larger, resulting in an increase in the total number of nodes in the index that need to be tested. This leads to an increase in search time. Second, the search algorithm in MDOPE uses a Bloom filter to test whether a queried range covers a 2-dimensional data, which is an efficient method for comparison. The search time of RSAC shown in Fig. 9 can also be explained by two factors. First, as the amount of 2-dimensional data increases, the total number of buckets increases linearly, and the average size of randomly generated queried ranges also increases. This leads to an increase in the average number of buckets in the minimal cover of queried ranges. As a user generates search tokens for all the buckets in the minimal cover and uses these tokens to perform range search over each index, the search time increases with the amount of 2-dimensional data. Second, the search algorithm in RSAC involves many calculations about bilinear maps, which are not very efficient.

Although RSAC is slower than MDOPE in the range search procedure, we would like to emphasize two important facts about our method. First, when a user performs a range search, the overall time required for search token generation and range search is more important than the time for range search alone. As shown in Fig. 10, the overall time for RSAC is significantly less than that for MDOPE, which means that RSAC can save more time for users overall. Second, in our experiments, we use a user attribute set $\{a_1, a_2\}$ of and embed all the indexes with the access control policy $p_{ac} = a_1$ and $a_2$. Therefore, the user has to test all the indexes in order to perform range searches. However, in real-world scenarios, a user usually does not have the privilege to search all the ciphertexts. Since the range search algorithm in RSAC can efficiently exclude the indexes that do not satisfy the user's privilege, the number of times that a user needs to perform range search in practical situations is much less than in our experiments.

## 5 DISCUSSION

In this section, first we analyse the security of the RSAC scheme under colluding attacks. Then, we introduce the Known Ciphertext Model [30, 37, 38] and provide a proof that our RSAC method is secure under this model.

As the adopted CP-ABE method [34] is collusion resistant, our method RSAC is also collusion-resistant. In our method, the secret keys of different users cannot be colluded to decrypt ciphertexts that are beyond their privileges. Moreover, users can generate new secret keys by running the delegate algorithm and use these new secret keys to generate search tokens for queried ranges. Since new and old secret keys cannot be colluded, the search tokens generated by these keys also cannot be used to decrypt ciphertexts that are beyond the privileges of the users.

Definition 3. Known Ciphertext Model: In the known ciphertext model, adversaries can only access the ciphertexts of multi-dimensional data, the identities of buckets, the secure indexes, the search tokens, and the retrieved ciphertexts. If an adversary records the search tokens and the retrieved ciphertexts, they can build up access patterns. Therefore, in the known ciphertext model, nothing beyond the access pattern should be leaked. To prove the security of our method, we adapt the definitions used in [38-40].

Definition 4. Search Pattern ($S_P$): Let $Q=\{R_1,\ldots,R_m\}$ be the set of $m$ consecutive queried ranges, $B_1,\ldots,B_n$ be the buckets defined by the data owner, $S_P$ be a $m\times m$ binary matrix s.t. $MC_{R_i}=MC_{R_j}$, then $S_P[i,j]=1$. Otherwise, $S_P[i,j]=0$ ($i,j=1,\ldots,m$, $MC_{R_i}$ and $MC_{R_j}$ are the minimal covers of $R_i$ and $R_j$ respectively).

Definition 5. Access Pattern ($A_P$): Let $ID_{B\in MC_{R_i}}$ be the set of identities of buckets in $MC_{R_i}$ and $T=\{T_{R_1},\ldots,T_{R_m}\}$ be the set of search tokens for $Q=\{R_1,\ldots,R_m\}$. Then, the access pattern for $Q$ is defined as $A_P=\left\{A_p\left(T_{R_1}\right)=ID_{B\in MC_{R_1}},\ldots,A_p\left(T_{R_m}\right)=ID_{B\in MC_{R_m}}\right\}$.

Definition 6. History ($H_m$): Let $Q=\{R_1,\ldots,R_m\}$ be the set of $m$ consecutive queried ranges and $C=\left\{C_{B\in MC_{R_1}},\ldots,C_{B\in MC_{R_m}}\right\}$ be the set of $m$ consecutive retrieved sets of ciphertexts ($C_{B\in MC_{R_i}}$ is the set of ciphertexts contained by the buckets in $MC_{R_i}$, where $i=1,\ldots,m$). Then, $H_m=(Q,C)$ is defined as a $m$-query history.

Definition 7. Trace ($\gamma$): Let $A_p(H_m)$ be the access pattern of $H_m$, $ID_{B\in MC_{R_i}}$ be the set of identities of buckets in $MC_{R_i}$, $C_{B\in MC_{R_i}}$ be the set of ciphertexts contained by the buckets in $MC_{R_i}$ ($i=1,\ldots,m$). Then, the trace of $H_m$ is defined as

$$\gamma(H_m)=\left\{A_p(H_m),\left\{ID_{B\in MC_{R_1}},\ldots,ID_{B\in MC_{R_m}}\right\},\left\{C_{B\in MC_{R_1}},\ldots,C_{B\in MC_{R_m}}\right\}\right\}.$$

Definition 8. View ($V$): Let $T=\{T_1,\ldots,T_m\}$ be the set of all the search tokens for $Q=\{R_1,\ldots,R_m\}$, $I=\left\{Idx_{B_1},\ldots,Idx_{B_n}\right\}$ be the set of all the secure indexes of buckets $B_1,\ldots,B_n$, $ID_{B\in MC_{R_i}}$ be the set of identities of buckets in $MC_{R_i}$, $C_{B\in MC_{R_i}}$ be the set of ciphertexts contained by the buckets in $MC_{R_i}$ ($i=1,\ldots,m$). Then, $V(H_m)=\left\{\begin{array}{l}T,I,\left\{ID_{B\in MC_{R_1}},\ldots,ID_{B\in MC_{R_m}}\right\},\\ \left\{C_{B\in MC_{R_1}},\ldots,C_{B\in MC_{R_m}}\right\}\end{array}\right\}$ is defined as the view of $H_m$. $V(H_m)$ is the information that is accessible to an adversary.

We use a simulator proof method that is similar to the one widely used in [38, 40]. The basic idea is that if two histories have the same trace, and the adversary cannot distinguish between them to learn any additional information about the indexes, search tokens, and encrypted multi-dimensional data beyond the search results and access pattern, then the method is considered secure [38].

Theorem 1. RSAC is secure in the known ciphertext model.

Proof. The notation $S$ denotes a simulator, which simulates a view $V^*(H_m)$. $V^*(H_m)$ should be indistinguishable from an adversary's view $V(H_m)=\left\{\begin{array}{l}T,I,\left\{ID_{B\in MC_{R_1}},\ldots,ID_{B\in MC_{R_m}}\right\},\\ \left\{C_{B\in MC_{R_1}},\ldots,C_{B\in MC_{R_m}}\right\}\end{array}\right\}$. To achieve this, $S$ does the followings:

(1) Identities of buckets are available in the trace. Thus, $S$ can copy these identities, i.e. $\left\{ID^*_{B\in MC_{R_1}}=ID_{B\in MC_{R_1}},\ldots,ID^*_{B\in MC_{R_m}}=ID_{B\in MC_{R_m}}\right\}$. As identities in the simulated view $V^*$ and the adversary's view $V$ are the same, they are computationally indistinguishable.

(2) $S$ generates $m$ empty sets $C^*_{B\in MC_{R_1}},\ldots,C^*_{B\in MC_{R_m}}$. $S$ chooses the first set $C_{B\in MC_{R_1}}$. For each ciphertext $c\in C_{B\in MC_{R_i}}$, $S$ generates a random value $c^*$, s.t. $|c^*|=|c|$. $S$ puts the random value $c^*$ in the set $C^*_{B\in MC_{R_1}}$. As multi dimensional data are encrypted by using a secure encryption scheme (e.g. AES [28]), the ciphertext $c$ is computationally indistinguishable from the random value $c^*$. Hence, $C^*_{B\in MC_{R_1}}$ and $C_{B\in MC_{R_1}}$ are computationally indistinguishable. If the ciphertext $c$ also exists in the set $C_{B\in MC_{R_1}}$ ($i=2,\ldots,m$), $S$ puts $c^*$ in the set $C^*_{B\in MC_{R_i}}$. By using this method, $S$ handles all the ciphertexts in $C_{B\in MC_{R_2}},\ldots,C_{B\in MC_{R_m}}$ and generates random values for the sets $C^*_{B\in MC_{R_2}},\ldots,C^*_{B\in MC_{R_m}}$ respectively. Finally, $S$ generates sets

$C^*_{B\in MC_{R_1}},\ldots,C^*_{B\in MC_{R_m}}$ which are computationally indistinguishable from $C_{B\in MC_{R_1}},\ldots,C_{B\in MC_{R_m}}$ respectively.

(3) Attribute set $S_{att}$ and boundary information set $S_{bou}$ are public. $S$ copies $S_{att}$ and $S_{bou}$, s.t. $S^*_{att}=S_{att}$ and $S^*_{bou}=S_{bou}$. $S$ randomly generates $n$ buckets $B^*_1,\ldots,B^*_n$ by using the boundary information in $S^*_{bou}$. Then, $S$ generates bucket policies for $B^*_1,\ldots,B^*_n$. Next, $S$ runs the algorithm *Encrypt* to generate secure indexes $I_{B^*_1},\ldots,I_{B^*_n}$ for $B^*_1,\ldots,B^*_n$ respectively. As the outputs of *Encrypt* are

computationally indistinguishable [34], $I_{B_i^*}$ and $I_{B_i}$ $(i=1,\dots,n)$ are computationally indistinguishable. Namely, $I^*=\left\{I_{B_1^*},\dots,I_{B_n^*}\right\}$ and $I=\left\{I_{B_1},\dots,I_{B_n}\right\}$ are computationally indistinguishable.

(4) *S* uses the following method to choose the queried range $R_i^*$ $(i=1,\dots,m)$ and then constructs the search token $T_i^*$ for $R_i^*$. *S* finds out all the buckets (these buckets are from $B_1^*,\dots,B_n^*$) whose identities are in $ID^*_{B\in MC_{R_i}}$. The set of these found buckets is denoted by $Set_{R_i^*}$. Then, *S* randomly chooses a range as $R_i^*$ s.t. $Set_{R_i^*}=MC_{R_i^*}$ $\left(MC_{R_i^*}\right.$ denotes the minimal cover of $\left.R_i^*\right)$. Namely, the minimal cover of $R_i^*$ is $Set_{R_i^*}$. Finally, for the queried range $R_i^*$, *S* runs the algorithm *KeyGen* to generate secret keys and then generates the search token $T_i^*$. As the secret keys are computationally indistinguishable [34], $T_i^*$ and $T_i$ are computationally indistinguishable. Thus, $T^*=\left\{T_1^*,\dots,T_m^*\right\}$ and $T=\left\{T_1,\dots,T_m\right\}$ are computationally indistinguishable.

Since each item in *V* and $V^*$ are computationally indistinguishable, we have the conclusion that RSAC satisfies the security definition presented in Theorem 1.

## 6 CONCLUSION

This paper proposes bucket policy, which supports range search and access control. Then we construct secure index by using bucket policy. Finally, based on the secure index, we propose a method named RSAC for performing range searches and access control over encrypted multi-dimensional data. Unlike Order Preserving Encryption, RSAC encrypts data using a secure encryption method and stores ciphertexts in each bucket as a single unit ensuring that the ordering information of ciphertexts is well protected. Additionally, RSAC considers access control by building a secure index for each bucket and distributing different secret keys to different users, allowing each user to only generate search tokens within his/her privilege. Unlike other public key-based encryption schemes, after the data owner distributes the secret key to a user, the user can generate search tokens on his/her own, which avoids the multi-round communication between the data owner and users to obtain search tokens. With RSAC, the data owner can set access privileges for his/her encrypted multi-dimensional data on the cloud server, and users can use their secret keys to generate search tokens and perform range searches within their own privileges. Because the underlying CP-ABE method used in RSAC is a public key encryption method, there are still some shortcomings in efficiency. Therefore, in future work, we hope to improve the underlying CP-ABE method and propose a more efficient scheme that supports access control and range queries over encrypted multi-dimensional data. Moreover, we also hope to study homomorphic setting of user privileges, specifically the revocation and the updating of privileges in our proposed RSAC scheme.

**Acknowledgment**

This research was supported by the National Natural Science Foundation of China (Nos. 61962029, 62362042, 62262033, and 62062045); the Jiangxi Provincial Natural Science Foundation of China (Nos. 20202BAB212006, 20202ACBL202005, 20224BAB202012, and 20232BAB202007); the Zhejiang Provincial Natural Science Foundation of China (No. LR23F020001); the Science and Technology Research Project of Jiangxi Education Department (Nos. GJJ201832, GJJ2201914, GJJ211815); the Hubei Natural Science Foundation Innovation and Development Joint Fund Project under Grant (No.2022CFD101), the Xiangyang High-tech Key Science and Technology Plan Project under Grant (No.2022ABH006848), the Hubei Superior and Distinctive Discipline Group of "New Energy Vehicle and Smart Transportation", and the Jiangxi Provincial Social Science Foundation of China (No. 23GL52D).

## 7 REFERENCES

[1] Zhang, S., Yang, L. T., Feng, J., Wei, W., Cui, Z., Xie, X., & Yan, P. (2021). A tensor-network-based big data fusion framework for Cyber-Physical-Social Systems (CPSS). *Information Fusion*, *76*, 337-354.

[2] You, L. & Wang, Z. (2022). A Cloud Based Network Intrusion Detection System. *Tehnicki vjesnik-Technical Gazette*, *29* (3), 987-992. https://doi.org/10.17559/TV-20211130024245

[3] Sari, T., Gules, H. K., & Yigitol, B. (2020). Awareness and readiness of Industry 4.0: The case of Turkish manufacturing industry. *Advances in Production Engineering & Management*, *15*(1), 57-68. https://doi.org/10.14743/apem2020.1.349

[4] Chen, Y., Yang, L. T., & Cui, Z. (2023). Tensor-Based Lyapunov Deep Neural Networks Offloading Control Strategy with Cloud-Fog-Edge Orchestration. *IEEE Transactions on Industrial Informatics*. https://doi.org/10.1109/tii.2023.3266401

[5] Laghari, A. A., He, H., Khan, A., Laghari, R. A., Yin, S., & Wang, J. (2022). Crowdsourcing platform for QoE evaluation for cloud multimedia services. *Computer Science and Information Systems*, 38-38. https://doi.org/10.2298/CSIS220322038L

[6] Muntean, M., Brândaş, C., Cristescu, M. P., & Matiu, D. (2021). Improving cloud integration using design science research. E*conomic computation & economic cybernetics studies & research*, *55*. https://doi.org/10.24818/18423264/55.1.21.13

[7] Song, Y. J. & Lee, J. K. (2020). A blockchain-based fog-enabled energy cloud in internet of things. *Journal of Logistics, Informatics and Service Science*, *7*, 45-64.

[8] Karvela, P., Kopanaki, E., & Georgopoulos, N. (2021). Challenges and Opportunities of Cloud Adoption in Supply Chain Management: A SWOT Analysis Model. *Journal of System and Management Sciences*, *11*(3), 215-234. https://doi.org/10.33168/JSMS.2021.0311

[9] Cui, Z., Lu, Z., Yang, L. T., Yu, J., Chi, L., Xiao, Y., & Zhang, S. (2023). Privacy and Accuracy for Cloud-Fog-Edge

Collaborative Driver-Vehicle-Road Relation Graphs. *IEEE Transactions on Intelligent Transportation Systems*. https://doi.org/10.1109/tits.2023.3254370

[10] Khoa, B. T. & Huynh, T. T. (2022). The Influence of Individuals' Concerns about Organization's Privacy Information Practices on Customers' Online Purchase Intentions: The Mediating Role of Online Trust. *Journal of Logistics, Informatics and Service Science*, *9*, 31-44.

[11] Wu, Z., Shen, S., Li, H., Zhou, H., & Lu, C. (2021). A basic framework for privacy protection in personalized information retrieval: An effective framework for user privacy protection. *Journal of Organizational and End User Computing (JOEUC)*, *33*, 1-26. https://doi.org/10.4018/joeuc.292526

[12] Wu, Z., Li, G., Shen, S., Lian, X., Chen, E., & Xu, G. (2021). Constructing dummy query sequences to protect location privacy and query privacy in location-based services. *World Wide Web*, *24*, 25-49. https://doi.org/10.1007/s11280-020-00830-x

[13] Boldyreva, A., Chenette, N., Lee, Y., & O'neill, A. (2009). Order-preserving symmetric encryption. *Advances in Cryptology-EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany*, *Proceedings 28*, 224-241. https://doi.org/10.1007/978-3-642-01001-9_13

[14] Boldyreva, A., Chenette, N., & O'Neill, A. (2011). Order-preserving encryption revisited: Improved security analysis and alternative solutions. *Advances in Cryptology-CRYPTO 2011: 31st Annual Cryptology Conference*, *Proceedings 31*, 578-595. https://doi.org/10.1007/978-3-642-22792-9_33

[15] Boneh, D. & Waters, B. (2007). Conjunctive, subset, and range queries on encrypted data. *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007*, *Proceedings 4*, 535-554. https://doi.org/10.1007/978-3-540-70936-7_29

[16] Hore, B., Mehrotra, S., & Tsudik, G. (2004). A privacy-preserving index for range queries. *Proceedings of the Thirtieth international conference on Very large data bases*, *30*, 720-731. https://doi.org/10.1016/b978-012088469-8.50064-4

[17] Hore, B., Mehrotra, S., Canim, M., & Kantarcioglu, M. (2021). Secure multidimensional range queries over outsourced data. *The VLDB Journal*, *21*, 333-358. https://doi.org/10.1007/s00778-011-0245-7

[18] Wang, P. & Ravishankar, C. V. (2013). Secure and efficient range queries on outsourced databases using Rp-trees. I*EEE 29th International Conference on Data Engineering (ICDE)*, 314-325. https://doi.org/10.1109/icde.2013.6544835

[19] Shi, E., Bethencourt, J., Chan, T. H. H., Song, D., & Perrig, A. (2007). Multi-dimensional range query over encrypted data. *IEEE Symposium on Security and Privacy (SP'07)*, 350-364. https://doi.org/10.1109/sp.2007.29

[20] Lu, Y. (2012). Privacy-preserving Logarithmic-time Search on Encrypted Data in Cloud, NDSS.

[21] Lee, Y. (2014). Secure ordered bucketization. *IEEE Transactions on Dependable and Secure Computing*, *11*, 292-303. https://doi.org/10.1109/tdsc.2014.2313863

[22] Wu, S., Li, Q., Li, G., Yuan, D., Yuan, X., & Wang, C. (2019). Serve DB: Secure, verifiable, and efficient range queries on outsourced database. *IEEE 35th International Conference on Data Engineering (ICDE)*, 626-637. https://doi.org/10.1109/icde.2019.00062

[23] Zheng, Y., Lu, R., Guan, Y., Shao, J., & Zhu, H. (2021). Towards practical and privacy-preserving multi-dimensional range query over cloud. *IEEE Transactions on Dependable and Secure Computing*, *19*, 3478-3493. https://doi.org/10.1109/tdsc.2021.3101120

[24] Zhan, Y., Shen, D., Duan, P., Zhang, B., Hong, Z., & Wang, B. (2022). MDOPE: Efficient multi-dimensional data order preserving encryption scheme. *Information Sciences*, vol. *595*, 334-343. https://doi.org/10.1016/j.ins.2022.03.001

[25] Kalloniatis, C., Lambrinoudakis, C., Musahl, M., Kanatas, A., & Gritzalis, S. (2021). Incorporating privacy by design in body sensor networks for medical applications: A privacy and data protection framework. *Computer Science and Information Systems*, *18*, 323-347. https://doi.org/10.2298/CSIS200922057K

[26] Ivanovic, M., Autexier, S., Kokkonidis, M., & Rust, J. (2023). Quality medical data management within an open AI architecture-cancer patients case. *Connection Science*, *35*. https://doi.org/10.1080/09540091.2023.2194581

[27] Paraschiv, D. M., Țițan, E., Manea, D. I., Bănescu, C. E. (2021). Quantifying the effects of working from home on privacy. An empirical analysis in the 2020 pandemic. *Economic Computation & Economic Cybernetics Studies & Research*, *55*. https://doi.org/10.24818/18423264/55.4.21.02

[28] Daemen, J. & Rijmen, V. (1999). AES proposal: Rijndael. https://doi.org/10.1007/springerreference_461

[29] Tian, P., Guo, C., Choo, K. K. R., & Liu, Y., Li, L., & Yao, L. (2021). EAFS: An efficient, accurate, and forward secure searchable encryptionscheme supporting range search. *IEEE Systems Journal*, *16*, 3450-3460. https://doi.org/10.1109/jsyst.2021.3071327

[30] Wong, W. K., Cheung, D. W., Kao, B., & Mamoulis, N. (2009). Secure kNN computation on encrypted databases. *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 139-152. https://doi.org/10.1145/1559845.1559862

[31] Li, R., Liu, A. X., Wang, A. L., & Bruhadeshwar, B. (2015). Fast and scalable range query processing with strong privacy protection for cloud computing. *IEEE/ACM Transactions on Networking*, *24*, 2305-2318. https://doi.org/10.1109/tnet.2015.2457493

[32] Liu, A. X. & Chen, F. (2008). Collaborative enforcement of firewall policies in virtual private networks. *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*. https://doi.org/10.1007/978-3-030-58896-0_6

[33] Shen, E., Shi, E., & Waters, B. (2009). Predicate privacy in encryption systems. *Theory of Cryptography: 6th Theory of Cryptography Conference, TCC 2009*, *Proceedings 6*, 457-473. https://doi.org/10.1007/978-3-642-00457-5_27

[34] Bethencourt, J., Sahai, A., & Waters, B. (2007). Ciphertext-policy attribute-based encryption. *2007 IEEE symposium on security and privacy (SP'07)*, 321-334. https://doi.org/10.1109/sp.2007.11

[35] Mei, Z., Zhu, H., Cui, Z., Wu, Z., Peng, G., Wu, B., & Zhang, C. (2018). Executing multi-dimensional range query efficiently and flexibly over outsourced ciphertexts in the cloud. *Information Sciences*, *432*, 79-96. https://doi.org/10.1016/j.ins.2017.11.065

[36] De Caro, A. & Iovino, V. (2011). jPBC: Java pairing based cryptography. *IEEE symposium on computers and communications (ISCC)*, 850-855. https://doi.org/10.1109/iscc.2011.5983948

[37] Wang, C., Cao, N., Li, J., Ren, K., & Lou, W. (2010). Secure ranked keyword search over encrypted cloud data. *IEEE 30th international conference on distributed computing systems*, 253-262. https://doi.org/10.1109/icdcs.2010.34

[38] Wang, B., Yu, S., Lou, W., & Hou, Y. T. (2014). Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. *IEEE INFOCOM 2014-IEEE conference on computer communications*, 2112-2120. https://doi.org/10.1109/infocom.2014.6848153

[39] Curtmola, R., Garay, J., Kamara, S., & Ostrovsky, R. (2006). Searchable symmetric encryption: improved definitions and efficient constructions. *Proceedings of the 13th ACM conference on Computer and communications security*, 79-88. https://doi.org/10.3233/jcs-2011-0426

[40] Kuzu, M., Islam, M. S., & Kantarcioglu, M. (2012). Efficient similarity search over encrypted data. *IEEE 28th*

*International Conference on Data Engineering*, 1156-1167.
https://doi.org/10.1109/icde.2012.23

**Contact information:**

**Zhuolin MEI**, Lecturer, PhD
School of Information Science and Technology,
Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: meizhuolin@126.com

**Jing YU**, Lecturer, PhD
School of Information Science and Technology,
Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: yujingellemma@gmail.com

**Jinzhou HUANG**, Associate Professor, PhD
(Corresponding author)
School of Computer Engineering,
Hubei University of Arts and Science,
No.296, Longzhong Road, Xiangyang, 441053, China
E-mail: huangjinzhou@hbuas.edu.cn

**Bin WU**, Lecturer, PhD
School of Information Science and Technology,
Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: wubincs@gmail.com

**Zhiqiang ZHAO**, Lecturer, PhD
School of Information Science and Technology,
Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: zqzhao2000@foxmail.com

**Caicai ZHANG**, Lecturer, PhD
School of Modern Information Technology,
Zhejiang Institute of Mechanical & Electrical Engineering,
No. 999, Qingnian Road, Hangzhou, Zhejiang 310000, China
E-mail: caicaizhng@gmail.com

**Zongda WU**, Professor, PhD
School of Mathematical Information,
Shaoxing Universitiy,
No. 508, Huancheng East Road, Shaoxing, Zhejiang 312000, China
E-mail: Wuzongda@126.com