IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 14, 2021, accepted December 3, 2021, date of publication December 13, 2021,
date of current version December 30, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3135050

An Efficient Ciphertext Retrieval Scheme Based on Homomorphic Encryption for Multiple Data Owners in Hybrid Cloud

HENG HE[1,2], RENJU CHEN[1,2], CHENGYU LIU[1,2], KE FENG[1,2], AND XIAOHU ZHOU[3]
[1]School of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430081, China
[2]Hubei Province Key Laboratory of Intelligent Information Processing and Real-Time Industrial System, Wuhan 430081, China
[3]School of Computing, Engineering and Built Environment, Birmingham City University, Birmingham B5 5JU, U.K.

Corresponding author: Heng He (heheng@wust.edu.cn)

This work was supported by the National Natural Science Foundation of China under Project 61602351 and Project 61802286.

ABSTRACT More and more individuals and enterprises outsource data and applications to cloud servers in recent years. Since the public cloud servers are not completely trusted, users usually encrypt important data before sending it to cloud servers. As a result, ciphertext retrieval technology has gradually become a research hotspot. In the existing related schemes, there are some defects such as not supporting ''multiple owners'' mode and multi-keyword retrieval, having low retrieval efficiency, accuracy and security, and difficult data updating. Hence, we propose an efficient Ciphertext Retrieval scheme based on Homomorphic encryption for Multiple data owners in hybrid cloud (CRHM), in which the public cloud server and the private cloud server cooperate to perform the ciphertext retrieval. In CRHM, an encrypted balanced binary index tree structure and a homomorphic encryption method based on large integer operations are designed to support ''multiple owners'' mode and multi-keyword ranked retrieval. The security analysis shows that CRHM can effectively guarantee the privacy and security of user file and retrieval, and the performance evaluation demonstrates that compared with the existing related schemes, CRHM has high efficiency in the index generation and retrieval processes, while keeps relatively high retrieval accuracy.

INDEX TERMS Hybrid cloud, ciphertext retrieval, homomorphic encryption, balanced binary index tree.

I. INTRODUCTION

With the rapid development of cloud computing [1], more and more individuals and enterprises outsource data and applications to cloud servers, so as to obtain many benefits brought by cloud services. In recent years the hybrid cloud has gradually become one of the main modes of cloud computing, and is also the further development direction of cloud computing [2], which integrates both the public cloud and the private cloud. Typically, public clouds are responsible for storing large-scale data, and private clouds are responsible for handling sensitive data. The main application scenarios of hybrid cloud technology are on-cloud disaster recovery, cross-cloud backup, and proprietary distributed storage services. However, the public cloud server providers are usually not completely trusted, they may snoop and analyze some important privacy information stored by users. The data needs to be encrypted before outsourcing to the public cloud server, but at the same time, it brings new challenges such as data access and operation difficulties.

The traditional retrieval scheme based on plaintext is no longer applicable, and the ciphertext retrieval technology has developed rapidly. There are already many ciphertext retrieval methods, such as the searchable symmetric encryption retrieval [3], [4], the searchable asymmetric encryption retrieval [5]–[7], the single keyword retrieval [8]–[10], and the multi-keyword retrieval [11]–[14], etc., which make the ciphertext retrieval technology gradually mature and more suitable for data retrieval in cloud environment.

Song *et al.* [4] first proposed a ciphertext retrieval scheme based on symmetric encryption, which is the beginning of the ciphertext retrieval. Boneh *et al.* [7] proposed an asymmetric retrievable encryption scheme for the first time, this scheme only supports the linear matching of single keyword, which will disclose the privacy information of users in the retrieval process. Zhou and Wornell [10] proposed a partial

The associate editor coordinating the review of this manuscript and approving it for publication was Aneel Rahim.

homomorphic encryption scheme over integer vectors, which improves the efficiency and security of retrieval, and supports the relevancy ranking of single keyword retrieval. Cao *et al.* [11] proposed a multi-keyword ciphertext retrieval scheme for the first time, to achieve the sorting of retrieval results, this scheme adopted the kNN algorithm, and calculated the inner product similarity between index vector and query vector. Li *et al.* [12] improved the secure kNN algorithm and built a B+ index tree to achieve efficient query. Fu *et al.* [13] proposed a personalized multi-keyword ciphertext retrieval scheme, which builds a user interest model for individual users by analyzing their search history with the help of semantic ontology WordNet to rank files.

The above schemes all follow the ''single owner'' mode, which means that users retrieve data in the outsourcing dataset belonging to only one certain data owner. In fact, different data owners have different fields of expertise, and there are large quality differences in the outsourcing data of multiple data owners, so users usually expect to retrieve high quality data from all the datasets of multiple data owners at the same time, which is ''multiple owners'' mode [15]. Obviously, the retrieval accuracy of ''multiple owners'' mode schemes is higher than that of ''single owner'' mode schemes. Subsequently, Peng *et al.* [16] proposed a multi-keyword ranked retrieval for multiple data owners over encrypted cloud data. The tree-based indexes are generated by different data owners and merged in the cloud server to improve retrieval efficiency. Yin *et al.* [17] proposed a secure joint multi-keyword ranked retrieval for multiple data owners over encrypted cloud data, where elliptic curve algorithm is used to calculate the similarity score of index vector and query vector to achieve top-k retrieval. In the above two schemes, each data owner needs to encrypt its own files and indexes, and the efficiency of creating indexes is relatively low. Guo *et al.* [18] proposed a secure multi-keyword ranked retrieval for multiple owners on cloud server, in which a trusted agent is introduced to manage key distribution, and a grouped balanced binary tree is built to achieve efficient query. However, although the asymmetric scalar-product-preserving encryption used in this scheme can easily implement searchable encryption, its security risk is relatively high. Moreover, it needs to regenerate all the security indexes every time the indexes need to be updated, which is inefficient. To sum up, at present there is no multi-keyword ciphertext retrieval scheme with high efficiency, high accuracy, and high security for ''multiple owners'' mode in the cloud environment. Thus, it is of great significance to research on designing such a scheme.

Based on the above research works, in this paper we propose an efficient Ciphertext Retrieval scheme based on Homomorphic encryption for Multiple data owners in hybrid cloud (CRHM). In CRHM, the public cloud server and the private cloud server cooperate to perform the ciphertext retrieval process. CRHM supports ''multiple owners'' mode and multi-keyword ranked retrieval. In addition, it does not disclose any privacy information related to data files, keywords and indexes to the public cloud server and users. Compared with the existing related schemes, CRHM improves the retrieval efficiency significantly while keeps relatively high retrieval accuracy. Our main contributions are as follows:

- We construct CRHM based on the characteristics of hybrid cloud, in which the public cloud server provides sufficient computing and storage resources to store ciphertext files and security indexes, and implement operations of ciphertext retrieval. The private cloud server performs the generation of security indexes and trapdoors for users to ensure the security and privacy of the retrieval, so that users can get rid of the heavy tasks.
- We collects the popularity information of documents belonging to multiple data owners to obtain the weighted indexes as scheme [17], so it can provide ideal ranking results that are not only relevant to the query data, but also of high-quality from multiple owners. The difference is that we partition and group the files, and we encrypt the weighted indexes and then construct an encrypted balanced binary index (EBBI) tree to generate and store security indexes for each partition of files. The EBBI tree structure can greatly improve the efficiency of index generation and retrieval. Furthermore, this structure makes the index update operation more convenient.
- We design a novel ciphertext retrieval method based on homomorphic encryption. The method realizes secure and efficient multi-keyword ciphertext ranked retrieval in hybrid cloud by using additive homomorphism of large integer modular operation.

II. PRELIMINARY KNOWLEDGE

A. DYNAMIC WEIGHT MODEL

The traditional TF-IDF model [19] is widely used in the ciphertext retrieval system of ''single owner'' mode. However, it is not useful in the ciphertext retrieval system of ''multiple owners'' mode. Guo *et al.* [18] designed a dynamic weight model to obtain the weighted indexes of files which belong to multiple data owners. In general, if the files of a data owner are more popular and professional in a subject, they should be retrieved preferentially when data users retrieve with keywords about this subject, so that the evaluation of files should not only consider the correlation with query keywords, but also consider the quality of files of multiple data owners.

In order to accurately calculate the weights of keywords, first the Jaccard similarity coefficient [20] is used to evaluate the degree of correlation between different keywords. The correlation coefficient Sx, y between each two keywords can be defined as follows:

$$S_{x,y} = \begin{cases} \beta . \dfrac{L(d_x \cap d_y)}{L(d_x \cup d_y)} & x \neq y \\ 1 & x = y \end{cases} \tag{1}$$

In formula (1), dx and dy represent the file set containing keyword x and file set containing keyword y respectively, L ($dx \cap dy$) represents the number of files containing both x

and y, L $(dx \cup dy)$ represents the number of files containing at least x or y, and β ($\beta \in [0,1)$) represents a variable parameter, which is used to adjust the correlation results of keywords. Obviously, the more files containing both x and y, the more relevant the two keywords are. Next, a correlation matrix S can be constructed which contains the correlation coefficients of each two keywords in the dictionary. Then, the popularity information of files is collected to obtain the popularity information of data owners for different keywords. The popularity information of files can be extracted from the information such as the number of files downloaded, browsed, and referenced. Finally, through a series of computing operations on the plaintext indexes of the files, the correlation matrix S and the popularity information of data owners, the weighted indexes of the files are generated.

B. HOMOMORPHIC ENCRYPTION

Homomorphic encryption is a kind of encryption method with special natural features, and this concept was first proposed by Rivest *et al.* in the 1970s [21]. Compared with the general encryption method, homomorphic encryption not only realizes the basic encryption operation, but also allows a variety of computing functions over ciphertexts, since by homomorphic encryption, first computing on the ciphertexts and then decrypting the result is equivalent to first decrypting the ciphertexts and then computing on the result. Homomorphic encryption technology can be used to handle ciphertext for users without key, and in the process of operation any information about plaintext is not disclosed [22]–[24]. This feature is of great significance to protect the security of information and improve the efficiency of information processing. More and more research efforts are put into the exploration of its theory and application. More specifically, if an encryption function f satisfies $f(a)+f(b)=f(a+b)$, we say that f has additive homomorphism, and if it satisfies $f(a)*f(b)=f(a*b)$, it has multiplicative homomorphism.

III. SCHEME DESIGN AND IMPLEMENTATION

A. SCHEME OVERVIEWE

There are four entities in CRHM:

1) Public Cloud Server (PuCS). PuCS is operated by large-scale enterprises, which can provide users with large storage space and data processing capacity.

2) Private Cloud Server (PrCS). PrCS is built by small and medium-sized enterprises, which is used to handle important security operations for users.

3) Data Owner (DO). DO refers the data provider who uploads files to PuCS, there are multiple DOs with expertise in various fields.

4) Data User (DU). DU refers the user who requests to retrieve files from PuCS.

Figure. 1 shows the system framework of CRHM, in which the system execution steps are as follows:

1) DO encrypts every one of his files using a symmetric encryption algorithm respectively, sends the ciphertexts of files to PuCS, and sends the plaintexts of files to PrCS.

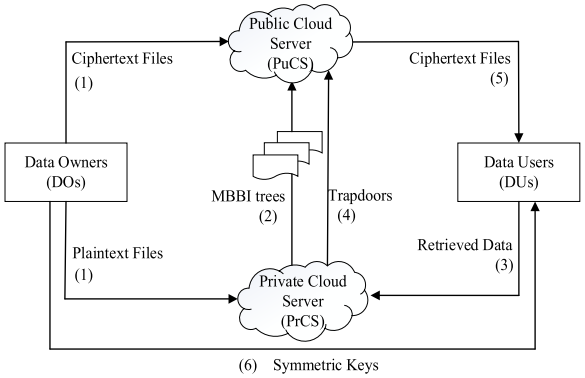


FIGURE 1. The system framework of CHRM.

2) PrCS randomly partitions all the files from all DOs. For each file partition, PrCS extracts keywords from the files in the partition, constructs a partition dictionary based on the keywords, and generates the plaintext indexes for all files in the partition. After that, PrCS uses the dynamic weight model to calculate the weighted indexes for all files in the partition and creates a balanced binary index tree to store the weighted indexes. Then, PrCS uses a homomorphic encryption method to encrypt all indexes in the nodes of the tree to get the security indexes, that is, an EBBI tree is constructed, which stores the security indexes. Finally, PrCS sends the EBBI tree of each partition to PuCS.

3) DU sends query data to PrCS.

4) PrCS extracts keywords from the query data, generates a trapdoor according to the partition dictionaries and sends it to PuCS.

5) PuCS searches in all the EBBI trees according to the trapdoor to find a certain number of top ranked ciphertext files and return them to DU.

6) DU contacts DO to get the symmetric key and decrypts the ciphertext files to obtain the plaintext files.

B. THREAT MODEL

In CRHM, we assume that PuCS is semi-trusted, that is, it executes programs honestly without tampering with the instruction sets from DO and DU, however, it also snoops data content contained in the user file and retrieval process as much as possible. We assume that PrCS is secure and reliable, which performs data operations with high security requirements for DO and DU. As described in literature [25], CRHM is also based on two threat models: 1) **Known Ciphertext Model**, in which PuCS can only obtain ciphertext files, security indexes, trapdoors and keyword searching results, and record the search results; 2) **Known Background Model**, that is based on known ciphertext model, and in which PuCS can further collect some additional background information such as the keyword frequency and distribution, which are used to analyze the correlation between the secure indexes and the received trapdoors to infer the keywords.

Function 1 Plaintext Index Generation

Input: The File set $F_i = \{f_{i,1}, f_{i,2}, \ldots, f_{i,r}\}$ of partition P_i
Output: The plaintext index set I_i of F_i
1. Generate a partition dictionary $Dp_i = \{D_{i,1} \cup D_{i,2} \cup \ldots \cup D_{i,r}\}$ for P_i, where $D_{i,j}$ ($j \in \{1, 2, \ldots, r\}$) is the keyword list of each file $f_{i,j}$ in F_i;
2. Generate a plaintext index set $I_i = (I_{i,1}, I_{i,2}, \ldots, I_{i,r})^T$ of F_i based on Dp_i, where $I_{i,j}$ ($j \in \{1, 2, \ldots, r\}$) is the plaintext index of each file $f_{i,j}$ in F_i; the length of $I_{i,j}$ is equal to d, which is the number of keywords in Dp_i, and each bit of $I_{i,j}$ depends on whether each keyword in Dp_i appears in $f_{i,j}$, if a keyword appears, the corresponding bit is 1, otherwise it is 0;
3. Return I_i.

C. ALGORITHM DESIGN

The algorithm of CRHM includes five functions. For a file partition P_i which contains a file set F_i, Function 1 generates a plaintext index set I_i of F_i.

Using the dynamic weight model, Function 2 generates a weighted index set $\tilde{I}_i$ of F_i according to I_i, the partition dictionary Dp_i, and the popularity information of files in F_i.

Function 3 generates a balanced binary index tree τ_i of F_i based on $\tilde{I}_i$.

Function 4 generates an EBBI tree τ_i^* of F_i based on τ_i.

In Function 4, we design a formula with additive homomorphism to get the security index, $C_t = B_t^* K_1 + K_1^* K_2^* q_t$. In Function 5, it is also used to calculate the trapdoor. Because of its additive homomorphism, Function 6 can rank the files by calculating the correlation scores. For one retrieval, Function 5 generates a trapdoor T_i for P_i according to the query data and Dp_i.

According to τ_i^* and T_i, Function 6 implements file retrieval in P_i, and the top k ciphertext files with the highest scores are selected.

D. EXECUTION STEPS

CRHM consists of five steps: data preprocessing, weighted index generation, security index generation, ciphertext retrieval and dynamic update, which are described as below.

1) Data preprocessing. Each DO sends the plaintext file set to PrCS, then uses a symmetric encryption algorithm (AES or DES) to encrypt each plaintext file respectively and sends the ciphertext file set to PuCS.

2) Weighted index generation. PrCS receives the file sets sent by DOs, and divides all files into m partitions randomly ($m = \log 2n$, n is the total number of files). The purpose of file partition is to improve the efficiency of index generation and retrieval, and provide convenience for data update. However, the larger the number of file partitions is, the shorter the length of partition dictionary is, which will lead to the decline of retrieval precision. Here files are divided into $\log 2n$ partitions to improve the retrieval efficiency significantly and ensure high retrieval precision simultaneously. Then, PrCS uses Function 1 to generate the partition dictionary and the plaintext index set of each partition.

PrCS collects the popularity information of files, and calls Function 2 to generate the weighted index set of each partition based on the plaintext index sets, partition dictionaries and popularity information. Then the weights of DO to keywords are calculated, based on which the weighted indexes of files of DO are generated. Thus, the weighted indexes can

Function 2 Weighted Index Generation

input: The plaintext index set $I_i = (I_{i,1}, I_{i,2}, \ldots, I_{i,r})^T$ of file set F_i, the partition dictionary Dp_i, the popularity information of files in F_i
Output: The weighted index set $\tilde{I}_i$ of F_i
1. Calculate a correlation coefficient between every two keywords in Dp_i using formula (1);
2. Generate a correlation matrix S_i which contains the correlation coefficients of each two keywords in Dp_i; S_i is a symmetric matrix with the same number of rows and columns, which is the number of keywords in Dp_i, and each element in S_i is the correlation coefficient between the corresponding keywords;
3. **For** each data owner O_l:
4. Generate a popularity vector $PV_l = (PV_{l,1}, PV_{l,2}, \ldots, PV_{l,t})$, in which t is the number of files of O_l in F_i, and each element is the popularity information such as download amount and click rate of the corresponding file of O_l;
5. Calculate an average popularity vector $AP_l = (PV_l \cdot I_l)\alpha_l$, in which $I_l = (I_{l,1}, I_{l,2}, \ldots, I_{l,t})^T$, where each element is the plaintext index vector of a file of O_l in F_i, $\alpha_l = (\alpha_{l,1}, \alpha_{l,2}, \ldots, \alpha_{l,d})$, and each element is the reciprocal of the number of files which contain the corresponding keywords in Dp_i, the operator "" represents the product of the corresponding elements of the two vectors;
6. Calculate an original weight vector $W_l^{raw} = AP_l \cdot S_i = (W_{l,1}^{raw}, W_{l,2}^{raw}, \ldots, W_{l,d}^{raw})$, in which each element represents the original weight of O_l to a keyword in Dp_i;
7. **End for**
8. Calculate a maximum weight vector $W_{max} = (W_{max}[1], W_{max}[2], \ldots, W_{max}[d])$, in which each element represents the maximum original weight to each keyword in Dp_i among different data owners;
9. Normalize $W_{max} = (1, 1, \ldots, 1)$;
10. **For** each data owner O_l:
11. Calculate a normalized weight vector $W_l = (W_{l,1}, W_{l,2}, \ldots, W_{l,d})$, in which $W_{l,t} = \frac{W_{l,t}^{raw}}{W_{max}[t]}, t \in \{1, 2, \ldots, d\}$;
12. For each file $f_{i,j}$ in F_i, if $f_{i,j}$ is owned by O_l, calculate a weighted index of $f_{i,j}$ as $\tilde{I}_{i,j} = W_l I_{i,j}$;
13. **End for**
14. Construct a weighted index set $\tilde{I}_i = (\tilde{I}_{i,1}, \tilde{I}_{i,2}, \ldots, \tilde{I}_{i,r})^T$ of F_i;
15. Return $\tilde{I}_i$.

Function 3 Balanced Binary Index Tree Generation

Input: The weighted index set $\tilde{I}_i$ of file set F_i
Output: The balanced binary index tree τ_i of F_i
1. **For** each weighted index $\tilde{I}_{i,j}$ in $\tilde{I}_i$:
2. Generate a leaf node u;
3. $u.id \leftarrow$ a unique identifier of u;
4. $u.fid \leftarrow$ the identifier of the corresponding file $f_{i,j}$;
5. $u.val \leftarrow$ the value of $\tilde{I}_{i,j}$;
6. Add u in a queue *que*;
7. **End for**
8. **For** each leaf node u in *que*:
9. Select a leaf node u' which satisfies the inner product of $u.val$ and $u'.val$ is the maximum among all other nodes in *que*;
10. Generate a parent node v for u and u';
11. $v.id \leftarrow$ a unique identifier of v;
12. Generate a new index $\tilde{I}_v$, each bit of which is the larger value of the corresponding elements of $u.val$ and $u'.val$;
13. $v.val \leftarrow$ the value of $\tilde{I}_v$;
14. Delete u and u' in *que*;
15. Add v in *que*;
16. **End for**
17. **For** each parent node v in *que*:
18. Select a parent node v' which satisfies the inner product of $v.val$ and $v'.val$ is the maximum among all other nodes in *que*;
19. Generate an upper parent node w for v and v';
20. $w.id \leftarrow$ a unique identifier of w;
21. Generate a new index $\tilde{I}_w$, each bit of which is the larger value of the corresponding elements of $v.val$ and $v'.val$;
22. $w.val \leftarrow$ the value of $\tilde{I}_w$;
23. Delete v and v' in *que*;
24. Add w in *que*;
25. **End for**
26. Continue to deal with all the upper parent nodes in *que* using the steps 17-25, until a root node is generated;
27. Return the generated balanced binary index tree τ_i.

Function 4 EBBI Tree Generation

Input: The balanced binary index tree τ_i of file set F_i
Output: The EBBI tree τ_i^* of F_i
1. Generate a security key $SK = (K_1, K_2)$, K_1 and K_2 are two random large integers and shared key respectively, $K_1 \ll K_2$;
2. **For** each index $\tilde{I}_u$ of nodes in τ_i:
3. **For** each bit b_t of $\tilde{I}_u$:
4. Encode b_t as $B_t = [1000 * b_t + A]$, in which A is a constant integer and [] is rounding operation;
5. Select a random large prime number q_t, $q_t \ll K_1$;
6. Encrypt b_t as $C_t = B_t{}^* K_1 + K_1{}^* K_2{}^* q_t$;
7. **End for**
8. Obtain the security index $\tilde{I}_u^* = (C_1, C_2, \ldots, C_d)$;
9. **End for**
10. **For** each node u in τ_i:
11. $u.val \leftarrow$ the value of $\tilde{I}_u^*$;
12. **End for**
13. Return the EBBI tree τ_i^*.

Function 5 Trapdoor Generation

Input: The query data, the partition dictionary Dp_i
Output: The trapdoor T_i for partition P_i
1. Generate a query vector Q_i based on the query data and Dp_i, the length of Q_i is equal to d, and each bit of Q_i depends on whether each keyword in Dp_i appears in the query data, if a keyword appears, the corresponding bit is 1, otherwise it is 0;
2. **For** each bit $b_t{}'$ of Q_i:
3. Encode $b_t{}'$ as $B_t{}' = [1000 * b_t{}' + A]$, in which [] is rounding operation;
4. Select a random large prime number q_t', $q_t{}' \ll K_1$;
5. Encrypt $b_t{}'$ as $C_t{}' = B_t{}' * K_1 + K_1{}^* K_2{}^* q_t{}'$, in which A, K_1, K_2 are the same as those in Function 4;
6. **End for**
7. Obtain the trapdoor $T_i = (C_1{}', C_2{}', \ldots, C_d{}')$;

represent the quality of files, and high-quality files will be obtained when retrieval is performed on them.

3) Security index generation. For improving the retrieval efficiency, PrCS calls Function 3 to generate a balanced binary index tree for each partition, whose leaf node stores the identifier and the weighted index of each file. An example of balanced binary index tree is shown in Figure. 2. Assume that there are 7 files in partition P_i which belong to 3 DOs, files $f_{i,1}, f_{i,2}, f_{i,3}$ belong to O_1, files $f_{i,4}, f_{i,5}$ belong to O_2 and files $f_{i,6}, f_{i,7}$ belong to O_3. Besides, we assume that the number of keywords in partition dictionary D_{pi} is 5. PrCS generates 7 leaf nodes for these files and constructs a balanced binary index tree τ_i according to Function 3. Then PrCS encrypts τ_i using Function 4. In Function 4, all the indexes in the nodes of τ_i are encrypted as the security indexes, and the structure of τ_i is kept unchanged, so that an EBBI tree τ_i^* is generated, in addition, τ_i and τ_i^* are isomorphic. In this way, PrCS generates m EBBI trees for m partitions, and sends them to PuCS for storage.

4) Ciphertext retrieval. When DU needs to retrieve files, he sends the query data to PrCS, PrCS calls Function 5 to generate a trapdoor T_i for partition P_i and total m trapdoors are combined to form a new integrated trapdoor T_q. PrCS sends T_q to PuCS, and PuCS performs retrieval in all m EBBI trees in parallel according to Function 6. When retrieval is finished, PuCS gets total m^*k scores, then selects top k scores from them, and returns the corresponding ciphertext files to DU. At last, DU contacts DO to get the symmetric encryption keys and decrypts the ciphertext files. In Function 6, PuCS calculates the correlation score Sc_u between T_i and the security index $\tilde{I}_u^*$ in τ_i^* by shared key K_2. Here Sc_u is a kind of similarity score and is proportional to the actual score $Sc_u{}'$. The proportion coefficient is the square of private key K_1.

Function 6 Retrieval

Input: The trapdoor T_i for partition P_i, the EBBI tree τ_i^* of file set F_i, the shared key K_2

Output: The top k ciphertext files in P_i with the highest scores

1. Traverse τ_i^* with the deep-first search strategy beginning from the root node of τ_i^*:
2. When traversing to a leaf node u:
3. Calculate the inner product R_u of T_i and the security index $\tilde{I}_u^*$ in u as:
$R_u = T_i \cdot (C_1, C_2, \ldots, C_d) = (C_1', C_2', \ldots, C_d') \cdot (C_1, C_2, \ldots, C_d) = C_1' \cdot C_1 + C_2' \cdot C_2 + \ldots + C_d' \cdot C_d$
4. Calculate the correlation score Sc_u as:
$Sc_u = R_u \bmod K_2 = K_1^2 * (B_1 * B_1' + B_2 * B_2' + \ldots + B_d * B_d') = K_1^2 * Sc_u'$, Sc_u' is the actual correlation score between Q_i and the index $\tilde{I}_u$;
5. Add Sc_u in a queue *que* with length k;
6. **If** there are already k scores in *que*, stop traversal and define Sc_m as the minimum score in *que*;
7. **Else** continue to traverse;
8. Continue to traverse τ_i^* with the deep-first search strategy beginning from the node next to the kth leaf node until traversal is complete:
9. When traversing to a non-leaf node u:
10. Calculate the correlation score Sc_u as the steps 3 and 4;
11. **If** $Sc_u <= Sc_m$, return to the previous node and continue to traverse;
12. **Else** continue to traverse the child nodes of u;
13. When traversing to a leaf node u:
14. Calculate the correlation score Sc_u as the steps 3 and 4;
15. **If** $Sc_u \leq Sc_m$, continue to traverse the next node;
16. **Else** replace the minimum score with Sc_u in *que*, redefine Sc_mas the minimum score in *que* and continue to traverse;
17. When traversal is complete, return k scores in *que* and the ciphertext files corresponding to the leaf nodes with these scores.

As a result, PuCS does not need to decrypt T_i, $\tilde{I}_u^*$ and then calculate Sc_u', it only needs to obtain Sc_u, sort the scores and finally return the ciphertext files corresponding to the leaf nodes with top k scores.

As the example shown in Figure. 2, we assume the query vector Q_i is $\{1, 0, 0, 0, 1\}$, the parameter A in Function 4 and Function 5 is 10, the parameter k in Function 6 is 3. The retrieval starts from the root node, and reaches the first leaf node u_1 through v_4 and v_1, the correlation score of $f_{i,1}$ is $460K_1^2$. Next the retrieval reaches leaf nodes u_2 and then u_3 through v_2, the correlation scores of $f_{i,2}$ and $f_{i,3}$ are $490K_1^2$ and $620K_1^2$ respectively. At this time, the score queue $que = \{460K_1^2, 490K_1^2, 620K_1^2\}$ and the length of que is limited to 3. After that, the nodes u_4, v_3, u_7 are reached in order and que is changed while reaching u_4 and u_7. Finally, $que = \{620K_1^2, 1010K_1^2, 1160\ K_1^2\}$, and the corresponding ciphertext files $f_{i,3}$, $f_{i,5}$ and $f_{i,7}$ are returned.

5) Dynamic update. When DO does not need to update files, the weights of the keywords in the partition dictionary still change dynamically with the popularity information of files on PuCS. Therefore, PrCS needs to update the average popularity information and normalized weights of all keywords in the dictionary of each partition at regular intervals. In this case, PrCS does not need to recalculate the correlation matrixes and just recalculates the weighted indexes of files of all DOs, then generates new EBBI trees and sends them to PuCS.

When DO needs to update files, the partition dictionary changes and the weights of keywords in the dictionary also change. In this case, PrCS regenerates the dictionary and plaintext indexes of the partition where the file is located, and sends the new ciphertext file to PuCS. After that, PrCS regenerates the correlation matrix and weighted indexes in this partition, then generates a new EBBI tree and sends it to PuCS. When DO needs to update files, the partition dictionary changes and the weights of keywords in the dictionary also change. In this case, PrCS regenerates the dictionary and plaintext indexes of the partition where the file is located, and sends the new ciphertext file to PuCS. After that, PrCS regenerates the correlation matrix and weighted indexes in this partition, then generates a new EBBI tree and sends it to PuCS.

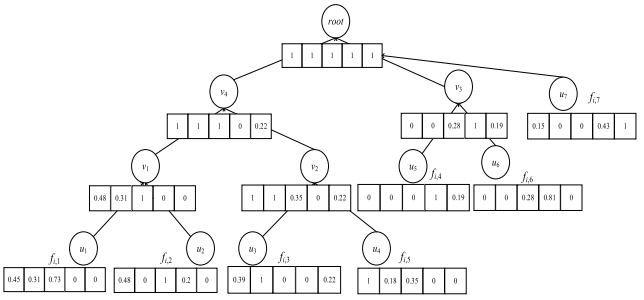


FIGURE 2. An example of balanced binary index tree T_i.

IV. SECURITY ANALYSIS AND PERFORMANCE EVALUATION

A. SECURITY ANALYSIS

We analyze and prove the security of CRHM under the known ciphertext model and the known background model respectively. And the notions used are listed as follows.

History: $\mathrm{H} = (\Delta_s, T_s, Q_k)$, in which Δ_s is a partition of the file set, T_s is an EBBI tree of Δ_s and $Q_k = \{q1, \ldots, qk\}$ is a series of queries from users.

View: $V(H) = (Enc_{SK}(\Delta_s), Enc_{SK}(T_s), \mathrm{Enc_{SK}}(Q_k))$, which is encrypted from H. The original history H is invisible for PuCS, while the view is visible.

Trace of a history: A trace of H is the set of the trace of queries $Tr(H) = \{Tr(q1), \ldots, Tr(q_k)\}$ which is get by PuCS from access pattern and retrieval results. And $Tr(q_i) = \{(\delta_j, \zeta_j)q_i \subset \delta_j, 1 \leq j \leq |\Delta s|\}$, where ζ_j is the similarity score between the query q_i and the file δ_j.

Theorem 1: CRHM is secure under the known ciphertext model.

Proof: If PuCS cannot distinguish two histories with the same trace generated by simulator, and then PuCS cannot explore more information about the index and the dataset except the access pattern and the retrieval results. Here, we introduce a simulator S to generate a V' that is distinguishable from PuCS's view $V(H)$. The generation process of a view V' as follows:

• S selects a random $\delta_i' \in (0, 1)\ |\delta_i|, \delta_i \in \Delta_s, 1 \le i \le |\Delta_s|$, then outputs $\Delta_s' = \{\delta_i', 1 \le i \le |\Delta_s'|\}$.

• S randomly generates a $SK' = (K_1', K_2')$, where K_1' is a 64-bit integer and K_2' is a 512-bit integer.

• S generates a query Q_k' to simulate Q_k and constructs trapdoor $Td'(Q_k') = Enc_{SK}'(Q_k')$, as follows: 1) for each $q_i \in Q_k, 1 \le i \le k$, is generated. Each position is a randomly selected as 1 or 0, but the number of 1s is the same as Q_k; 2) Encrypt Q_k' with SK': $Enc_{SK}'(Q_k') = \{Enc_{SK}'(q_1'), \ldots, Enc_{SK}'(q_k')\}$.

• S generates an index trees τ_i' for F_i to simulate τ_i and encrypt τ_i', as follows: 1) for each $\delta_i' \in \Delta_s', 1 \le i \le |\Delta_s'|$, a d-bit null vector $I\delta_i'$ is generated as the index; 2) for each $q_j \in Q_k$, if $q_j \in \delta_i, 1 \le j \le k$, then the d positions of $I\delta_i'$ are set as that of q_j'; 3) an index tree τ_i' is constructed based on these index vectors, τ_i' is encrypted with SK' as $Enc_{SK}'(\tau_i')$.

• S outputs the $V' = (\Delta_s', Enc_{SK}'(\tau_i'), Enc_{SK}'(Q_k'))$.

From the above process, according to the same trace as those of PuCS, the EBBI tree $Enc_{SK}'(\tau_i')$ and the trapdoor $Enc_{SK}'(Q_k')$ are generated. No PPT (probabilistic polynomial-time) adversary can distinguish view V' and V, or the encrypted file sets $Enc_{SK}(\Delta_s)$ and file sets Δ_s' with more than 1/2 probability since the semantic security of symmetric encryption. Some related schemes have demonstrated the indistinguishable of index tree and trapdoor which are generated by homomorphic encryption. So, theorem 1 has been proven.

Theorem 2: CRHM is secure under the known background model.

Proof: we still use a simulator S to generate a view V' that is indistinguishable from PuCS's view $V(H)$. The process of generating V' as follows:

• S selects a random $\delta_i' \in (0, 1)|\delta_i|, \delta_i \in \Delta_s, 1 \le i \le |\Delta_s|$, then outputs $\Delta_s' = \{\delta_i', 1 \le i \le |\Delta_s'|\}$.

• S randomly generates a $SK' = (K_1', K_2')$ as above, and a large prime number q_t is selected randomly, $q_t' \ll K_1$.

• S generates a query Q_k' to simulate Q_k and constructs trapdoor $Td'(Q_k') = Enc_{SK}'(Q_k')$, as follows: 1) for each $q_i \in Q_k, 1 \le i \le k$, is generated. Each position is a randomly selected as 1 or 0, but the number of 1s is the same as Q_k; 2) Encrypt Q_k' with SK': $Enc_{SK}'(Q_k') = \{Enc_{SK}'(q_1'), \ldots, Enc_{SK}'(q_k')\}$.

• S generates an index trees τ_i' for F_i to simulate τ_i and encrypt τ_i', as follows: 1) for each $\delta_i' \in \Delta_s', 1 \le i \le |\Delta_s'|$, a d-bit null vector $I\delta_i'$ is generated as the index; 2) for each $q_j \in Q_k$, if $q_j \in \delta_i, 1 \le j \le k$, then the d positions of $I\delta_i'$ are set as that of q_j'; 3) an index tree τ_i' is constructed based on these index vectors, τ_i' is encrypted with SK' as $Enc_{SK}'(\tau_i')$.

• S outputs the $V' = (\Delta_s', Enc_{SK}'(\tau_i'), Enc_{SK}'(Q_k'))$.

In this process, the conclusion proved in Theorem 1 applies equally to Theorem 2. Although PuSC has a series of keyword-trapdoor pairs, it cannot distinguish the output of the linear analysis form a random string because of the indistinguishability of the randomness of file partitioning and randomness of large prime number selection. So, theorem 2 has been proven.

B. PERFORMANCE EVALUATION

The function comparison among CRHM and related schemes in terms of "multiple owners" mode, trusted organization, tree-based index, simple update, and high-quality file retrieval is described in TABLE 1.

Then we evaluate the performance of CRHM from five aspects: weighted index generation, security index generation, trapdoor generation, retrieval efficiency and retrieval precision. CRHM is compared with EMRS [18], MRSE [11] and PRSE [13] in the above aspects. These schemes are implemented on Intel Core i5-3230M 2.60 GHz processor and Windows 10 operating system platform using Java language. The data set used in the experiment is crawled from Google Scholar, which contains 10000 papers related to different fields. And the average of each file is 2MB, and the number of keywords in each file is set to 20. So, the parameters $d = 100$, $A = 10$, $k = 3$, and K_1, K_2, q_t, q_t' are selected randomly, where K_1 is a 64-bit integer, K_2 is a 512-bit integer and q_t, q_t' are 8-bit prime numbers.

1) WEIGHTED INDEX GENERATION

We compare and analyze the efficiency of weighted index generation of these schemes under different file sets. Figure. 3 shows the variation of the generation time of weighted indexes with the total number of files. In CRHM, the whole process of weighted index generation mainly includes file partition, partition dictionaries generation, plaintext indexes generation and weighted indexes generation; In EMRS, there are no partitions, a public dictionary is prepared instead of partition dictionaries, and plaintext indexes and weighted indexes are generated according to the public dictionary. As for MRSE and PRSE, plaintext indexes are generated with a public dictionary like EMRS, but plaintext indexes do not need to be further processed. Thus, we treat the plaintext indexes here as weighted indexes. In CRHM, the sizes of keyword dictionaries are reduced by partitioning files, so that the time overheads are also decreased while generating plaintext indexes and weighted indexes. Therefore, for any number of files, the generation time of weighted indexes in CRHM is significantly less than that of EMRS, MRSE and PRSE.

2) SECURITY INDEX GENERATION

After the generation of weighted indexes, they need to be encrypted with specific methods to generate the security indexes. In CRHM, we divide files into multiple partitions,

TABLE 1. Function comparsion of related schemes.

Schemes	*MRSE* [11]	*MKQE* [12]	*PRSE* [13]	*TBMSM* [16]	*SSMDO* [17]	*EMRS* [18]	*CRHM*
"Multiple owners" mode	×	×	×	√	√	√	√
Trusted organization	×	×	×	×	×	√	√
Tree-based index	×	√	√	√	×	√	√
Simple update	√	×	×	×	√	×	√
High-quality file retrieval	×	×	×	×	×	√	√

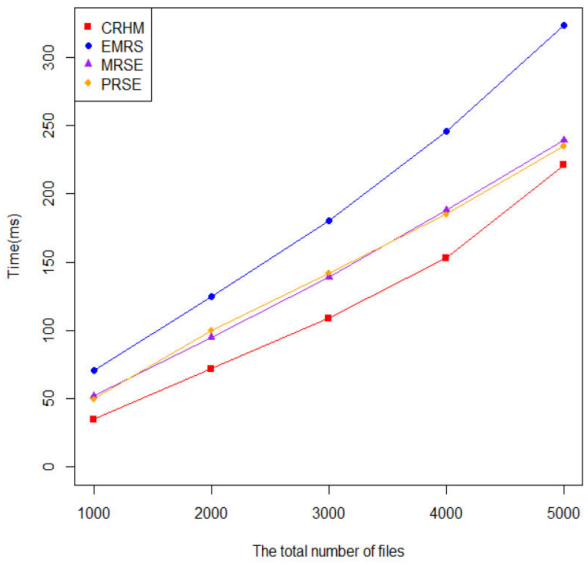


FIGURE 3. The variation of the generation time of weighted indexes with the total number of files.

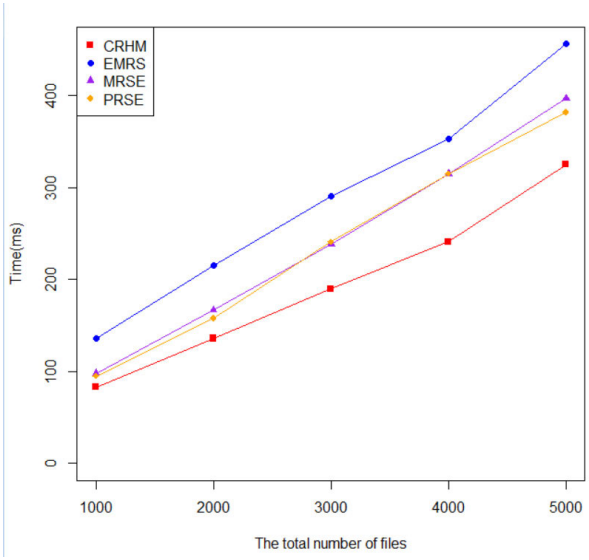


FIGURE 4. The variation of the generation time of security indexes with the total number of files.

and an EBBI tree is generated for each partition with the homomorphic encryption method. In EMRS, the secure kNN method is used to encrypt weighted indexes and generate

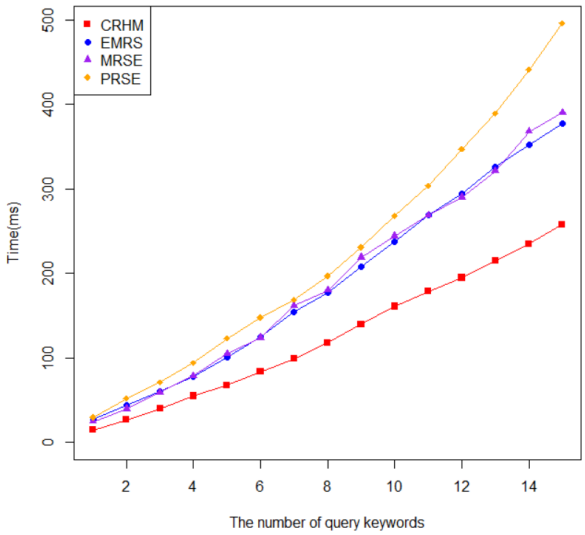


FIGURE 5. The variation of the generation time of trapdoors with the number of query keywords.

a large grouped balanced binary tree to store security indexes. In MRSE and PRSE, the plaintext indexes are also encrypted with the secure kNN method, but the security indexes are stored directly instead of using tree-based structure. The variation of the generation time of security indexes with the total number of files is as shown in Figure. 4. It can be seen that, MRSE and PRSE are more efficient than EMRS because of their simple processing of indexes. Moreover, because of the superiority of the homomorphic encryption and EBBI tree, the efficiency of security index generation in CRHM is the highest among all the schemes for any number of files.

3) TRAPDOOR GENERATION

According to query data, trapdoors are generated with the keyword dictionaries and encryption method. In EMRS and MRSE, query vectors are generated according to the query data and public dictionary, and trapdoors will be generated with the secure kNN method based on the query vectors. In PRSE, query vectors are generated as the same, and further processed as the weighted query vectors, based on which trapdoors are generated with the secure kNN method. As for CRHM, each trapdoor is generated by the partition dictionaries and homomorphic encryption method. Figure. 5 shows

the variation of the generation time of trapdoors with the number of query keywords. The capacity of the partition dictionaries in CRHM is much smaller than that of the public dictionary in EMRS, MRSE and PRSE, and the homomorphic encryption method in CRHM is very efficient, so that the efficiency of trapdoor generation in CRHM is higher than that in other three schemes for any number of query keywords.

4) RETRIEVAL EFFICIENCY

In order to illustrate the retrieval efficiency of these schemes, we use the same query data to retrieve files and then compare the retrieval time. In CRHM, EBBI trees are traversed in parallel and the files are sorted by the inner products of trapdoors and security indexes which are calculated with the feature of homomorphic encryption based on large integer operations. In EMRS, MRSE and PRSE, the inner product is calculated with the feature of complex matrix operations. In addition, In EMRS, the large grouped balanced binary tree is traversed while all indexes without any tree-based structure are linearly traversed in MRSE and PRSE. Figure. 6 shows the variation of retrieval time with the total number of files, where the number of files to be retrieved out is set to 10, and Figure. 7 shows the variation of retrieval time with the number of files to be retrieved out, where the total number of files is set to 1000. We can get similar results with other settings. It can be seen that CRHM has better retrieval efficiency compared with other three schemes in various situations.

5) RETRIEVAL PRECISION

We compare the retrieval precision of these schemes. The retrieval precision here refers to the ability of the scheme to distinguish the files with different qualities but involving similar topics. In MRSE, coordinate matching method is used to calculate the correlation score, in which all keywords in security index are regarded as equivalent. In PRSE, TF-IDF model is used and each keyword in security index is given weight that considers the importance of the keyword to files. In EMRS and CRHM, dynamic weight model is used and each keyword is given weight that considers both the importance of the keyword to files and the file popularity information of multiple DOs.

In the experiment, we choose two file sets, in which the corresponding files contain the same keywords, and the files in file set A are far more popular than the corresponding files in file set B. For these four schemes, we calculate the inner product results of the query vectors and weighted indexes of files in the two file sets respectively, to rank files for simplicity. The results are shown in Table 2. We number the files in the two file sets in order respectively, FID represents the file number, and the value in Table 2 (e.g., 10/10) contains two parts, in which the first part represents the inner product of the file in A, and the second part represents the inner product of the file in B. Since the qualities of files in A are higher than those in B, the larger the quotient value of the two parts is, the higher the retrieval precision is. As we can see, in MRSE and PRSE, the inner products of files in A are the same as those in B, so high quality files in A cannot be

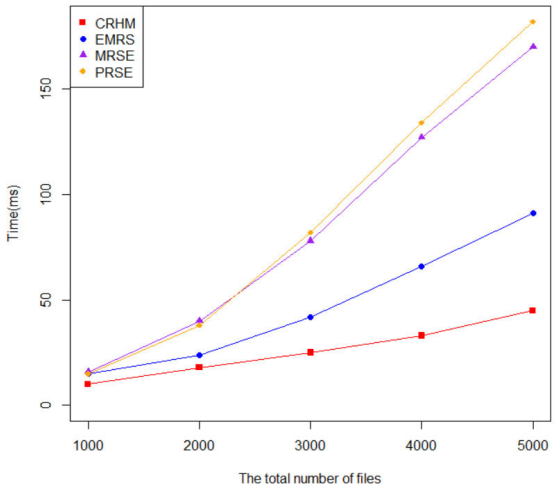


FIGURE 6. The variation of retrieval time with the total number of files.

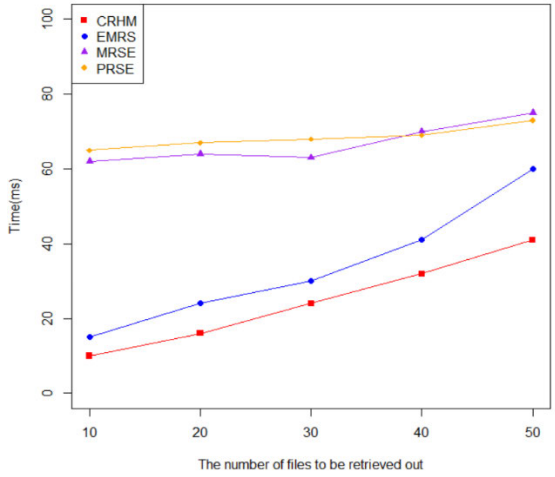


FIGURE 7. The variation of retrieval time with the number of files to be retrieved out.

TABLE 2. The inner product results of the query vectors and weighted indexes of files in two file sets.

FID	*MRSE*	*PRSE*	*EMRS*	*CRHM*
1	10/10	10/10	29/4	26/4
2	15/15	15/15	50/7	44/7
3	9/9	9/9	21/3	25/4
4	16/16	16/16	55/8	59/10
5	7/7	7/7	20/3	18/3
6	12/12	12/12	40/6	32/5
7	5/5	5/5	13/2	13/2
8	21/21	21/21	67/9	50/8
9	8/8	8/8	21/3	19/3
10	10/10	10/10	29/4	26/4

selected preferentially. When using dynamic weight model, the inner products of files in A are obviously larger than those in B, therefore CRHM and EMRS achieve more precise retrieval, which can find out higher quality files in A. In addition, the quotient value of the two parts in CRHM is a bit smaller than the corresponding value in EMRS, because affected by the file partition, the retrieval precision is slightly reduced in CRHM. In a word, CRHM can ensure relatively high retrieval precision and improve the retrieval efficiency significantly.

V. CONCLUSION

In this paper, we propose an efficient ciphertext retrieval scheme based on homomorphic encryption for multiple data owners in hybrid cloud, namely CRHM, where the public cloud server and the private cloud server cooperate to implement the ciphertext retrieval securely and efficiently. In CRHM, multiple EBBI trees are set up to generate and store security indexes, and a homomorphic encryption method based on large integer operations is designed to perform ciphertext retrieval process. CRHM can support multi-keyword ranked retrieval, and provide the results which are not only relevant to query data, but also of high quality from multiple owners. Compared with the existing related schemes, CRHM achieves both high retrieval efficiency and accuracy. In addition, it can effectively guarantee the privacy and security of user file and retrieval.

For future work, we will investigate on how to solve the validation problems of retrieval results to increase the reliability of system when the cloud server may not perform the operations correctly.

REFERENCES

[1] M. Ali, A. Abbas, M. U. S. Khan, and S. U. Khan, "SeSPHR: A methodology for secure sharing of personal health records in the cloud," *IEEE Trans. Cloud Comput.*, vol. 9, no. 1, pp. 347–359, Jan. 2021.

[2] K. Kritikos, P. Skrzypek, A. Moga, and O. Matei, "Towards the modelling of hybrid cloud applications," in *Proc. IEEE 12th Int. Conf. Cloud Comput. (CLOUD)*, San Diego, CA, USA, Jul. 2019, pp. 291–295.

[3] G. S. Poh, J.-J. Chin, W.-C. Yau, K.-K.-R. Choo, and M. S. Mohamad, "Searchable symmetric encryption: Designs and challenges," *ACM Comput. Surveys*, vol. 50, no. 3, pp. 1–37, Oct. 2017.

[4] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy. (SP)*, May 2000, pp. 44–55.

[5] D. Hofheinz and T. Jager, "Tightly secure signatures and public-key encryption," *Des., Codes Cryptogr.*, vol. 80, no. 1, pp. 29–61, Jul. 2016.

[6] J. H. Cheon and J. Kim, "A hybrid scheme of public-key encryption and somewhat homomorphic encryption," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 5, pp. 1052–1063, May 2015.

[7] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techniques* Berlin, Germany: Springer-Verlag, 2004, pp. 506–522.

[8] X. Song, C. Dong, D. Yuan, Q. Xu, and M. Zhao, "Forward private searchable symmetric encryption with optimized I/O efficiency," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 5, pp. 912–927, Sep. 2020.

[9] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Dallas, TX, USA, Oct. 2017, pp. 1465–1482.

[10] H. Zhou and G. Wornell, "Efficient homomorphic encryption on integer vectors and its applications," in *Proc. Inf. Theory Appl. Workshop (ITA)*, Washington, DC, USA, Feb. 2014, pp. 1–9.

[11] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2013.

[12] R. Li, Z. Xu, W. Kang, K. C. Yow, and C.-Z. Xu, "Efficient multi-keyword ranked query over encrypted data in cloud computing," *Future Gener. Comput. Syst.*, vol. 30, no. 1, pp. 179–190, Jan. 2014.

[13] Z. Fu, K. Ren, J. Shu, X. Sun, and F. Huang, "Enabling personalized search over encrypted outsourced data with efficiency improvement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2546–2559, Sep. 2015.

[14] P. Xu, S. Tang, P. Xu, Q. Wu, H. Hu, and W. Susilo, "Practical multi-keyword and Boolean search over encrypted E-mail in cloud server," *IEEE Trans. Services Comput.*, vol. 14, no. 6, pp. 1877–1889, Nov. 2021.

[15] Y. Miao, X. Liu, K.-K.-R. Choo, R. H. Deng, J. Li, H. Li, and J. Ma, "Privacy-preserving attribute-based keyword search in shared multi-owner setting," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 3, pp. 1080–1094, May 2021.

[16] T. Peng, Y. Lin, X. Yao, and W. Zhang, "An efficient ranked multi-keyword search for multiple data owners over encrypted cloud data," *IEEE Access*, vol. 6, pp. 21924–21933, 2018.

[17] H. Yin, Z. Qin, J. Zhang, L. Ou, F. Li, and K. Li, "Secure conjunctive multi-keyword ranked search over encrypted cloud data for multiple data owners," *Future Gener. Comput. Syst.*, vol. 100, pp. 689–700, Nov. 2019.

[18] Z. Guo, H. Zhang, C. Sun, Q. Wen, and W. Li, "Secure multi-keyword ranked search over encrypted cloud data for multiple data owners," *J. Syst. Softw.*, vol. 137, pp. 380–395, Mar. 2018.

[19] J. H. Paik, "A novel TF-IDF weighting scheme for effective ranking," in *Proc. 36th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, New York, NY, USA, Jul. 2013, pp. 343–352.

[20] S. Plansangket and J. Q. Gan, "A query suggestion method combining TF-IDF and Jaccard coefficient for interactive web search," *Artif. Intell. Res.*, vol. 4, no. 2, pp. 119–125, 2015.

[21] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Found. Secure Comput.*, vol. 4, no. 11, pp. 169–180, 1978.

[22] Z. Gong, Y. Xiao, Y. Long, and Y. Yang, "Research on database ciphertext retrieval based on homomorphic encryption," in *Proc. 7th IEEE Int. Conf. Electron. Inf. Emergency Commun. (ICEIEC)*, Jul. 2017, pp. 149–152.

[23] Y. Ding and X. Li, "Policy based on homomorphic encryption and retrieval scheme in cloud computing," in *Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSE), IEEE Int. Conf. Embedded Ubiquitous Comput. (EUC)*, Guangzhou, China, Jul. 2017, pp. 568–571.

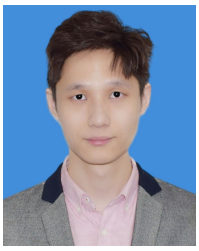
[24] S. Xiang and X. Luo, "Reversible data hiding in homomorphic encrypted domain by mirroring ciphertext group," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 11, pp. 3099–3110, Nov. 2017.

[25] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 340–352, Feb. 2016.

HENG HE received the B.S. degree in computer science and technology from the Hubei University of Technology, Wuhan, China, in 2004, and the M.S. and Ph.D. degrees in computer science and technology from the Huazhong University of Science and Technology, Wuhan, in 2007 and 2013, respectively. He is currently an Associate Professor with the Wuhan University of Science and Technology. His research interests include cloud computing, network security, information retrieval, and software defined networks.

RENJU CHEN received the B.S. degree in computer science and technology from the Wuhan University of Science and Technology, Wuhan, China, in 2020, where he is currently pursuing the M.S. degree in computer science and technology. His research interests include cloud computing, data encryption, and information retrieval.

CHENGYU LIU received the B.S. degree in computer science and technology from the Wuhan University of Science and Technology, Wuhan, China, in 2018, where he is currently pursuing the M.S. degree in computer science and technology. His research interests include cloud computing, blockchain, and information retrieval.



KE FENG received the B.S. degree in computer science and technology from the Wuhan University of Science and Technology, Wuhan, China, in 2018, where he is currently pursuing the M.S. degree in computer science and technology. His research interests include cloud computing, data encryption, and information retrieval.

XIAOHU ZHOU received the B.Sc. degree (Hons.) in business information technology from Birmingham City University, U.K., the Bachelor of Engineering degree in computer science and technology from the Guilin University of Technology, China, and the Master of Science degree in web technology from the University of Southampton, in 2013. She is currently pursuing the Ph.D. degree with Birmingham City University. She is working in secure health data sharing at Birmingham City University. She was a previously an IT Lecturer at Guilin Medical University, China, hosted, and joint several medical informatics projects. Her research interests include blockchains, healthcare information systems, and data sharing and security. Her research interests include security, trust, and data privacy, particularly in health applications.