




Searchable Encryption System for Big Data Storage

Yuxiang Chen^{1,2,3} , Yao Hao^{1,2}, Zhongqiang Yi^{1,2}, Kaijun Wu^{1,2}, Qi Zhao^{1,2},
and Xue Wang^{1,2}

¹ Science and Technology on Communication Security Laboratory, Chengdu 610041, China

² No. 30 Inst, China Electronics Technology Group Corporation, Chengdu 610041, China

³ School of Computer Science and Engineering, University of Electronic Science
and Technology of China, Chengdu 611731, China

Abstract. Big data cloud platforms provide users with on-demand configurable computing, storage resources to users, thus involving a large amount of user data. However, most of the data is processed and stored in plaintext, resulting in data leakage. At the same time, simple encrypted storage ensures the confidentiality of the cloud data, but has the following problems: if the encrypted data is downloaded to the client and then decrypted, the search efficiency will be low. If the encrypted data is decrypted and searched on the server side, the security will be reduced. Data availability is finally reduced, and indiscriminate protection measures make the risk of data leakage uncontrollable. To solve the problems, based on searchable encryption and key derivation, a cipher search system is designed in this paper considering both data security and availability, and the use of a search encryption algorithm that supports dynamic update is listed. Moreover, the system structure has the advantage of adapting different searchable encryption algorithm. In particular, a user-centered key derivation mechanism is designed to realize file-level fine-grained encryption. Finally, extensive experiment and analysis show that the scheme greatly improves the data security of big data platform.

Keywords: Big data platform · Searchable encryption · Fine-grained · Secure storage

1 Introduction

Currently, big data storage platforms store user's data in plain text, which brings the risk of data leakage, and it is difficult for users to protect their stored data from being stolen. At the same time encrypted data storage faces the following problems: if the encrypted data is downloaded to the user's terminal and then decrypted and searched, the search efficiency will be inefficient. Decrypting and searching the encrypted data on the server will reduce the security [1–4].

At the same time, when the distributed file system is applied to provide storage services for users, users usually upload and store plaintext data directly, and it is difficult for users to control whether their stored data is leaked or stolen. In addition, the file

storage service provider may monitor and analyze the user's file retrieval behavior, perform correlation analysis on the user's file content through the keywords retrieved by the user, and then focus on cracking and stealing user's data [5–9]. Besides, users store plaintext data in third-party organizations, losing the initiative to control data and failing to know whether their data has been stolen or leaked [10–13].

For example, in June 2017, the analysis report of the Shodan Internet device search engine showed that the Hadoop server was exposed due to insecure configuration and plaintext storage. It involves nearly 4500 servers using the Hadoop Distributed File System (HDFS), with a data volume of up to 5120 TB. The requirements corresponding to the above risks are:

1. File system data encryption protection.
2. The data of the file system can realize the search query of the ciphertext to obtain the corresponding file content.

A feasible solution is to use the ciphertext computing algorithm such as searchable encryption, homomorphic encryption, etc. They can efficiently perform retrieval operations in ciphertext and also have diverse scenarios, such as medical data, financial data and government data, etc. [13–16].

In response to the above risks and requirements, we developed our system based on searchable encryption algorithm and hierarchical key management to realize three major functions, including data encryption protection, encrypted search of encrypted data on the storages side, and data security sharing. By classifying users, distributing corresponding master keys, the keys of file contents are derived from users' master keys and file unique identifiers. Hierarchical and fine-grained control is realized through hierarchical keys, thus, users of corresponding levels can complete file encryption, upload and sharing only by holding encryption keys of corresponding levels, which is simple and convenient.

To sum up, our system can realize ciphertext storage, cipher retrieval and file access control based on user identity level and master key, effectively solve the content security problem of traditional distributed file system, and prevent file content leakage caused by malicious administrators and network attacks.

The rest of the paper is organized as follows: In Sect. 2, we first familiarize with the scheme's functional interaction model and system architecture. In Sect. 3, we list the usage of our algorithm and deployment of our system. Section 4 present the system analysis and feasibility. Section 5 concludes the paper.

2 Functional Interaction Model and System Architecture

2.1 Functional Interaction Model

Distributed file system supporting searchable encryption technology mainly provides users with file secure search and storage. Fig. 1 shows the functional model of distributed file system supporting searchable encryption technology proposed for this project. The core parts of functional interaction model mainly includes 3 parts: file data encryption protection, ciphertext search, download and decryption of the file data at the storage end and file secure sharing.

1. Data encryption protection: File data encryption protection means that users generate the encryption keys of files based on file meta data and their own master keys, then encrypt files, generate indexes and encrypt them at the same time, then upload ciphertext files and cipher text index to the cloud server.
2. Ciphertext search: Cipher search of encrypted file data at the storage end means that users generate ciphertext retrieval key value through data keywords and keys, then send the key value to the server, search the ciphertext index of the file through the retrieval algorithm to obtain the corresponding file, finally download and decrypt the retrieved file to obtain the plain text.
3. Data secure sharing: Security sharing of file data means that user A can directly share file ciphertext to user B by sharing the derived key and index, and user B updates its own ciphertext retrieval set, as long as user B's security level meets the security level of the file, the file can be obtained through ciphertext retrieval, download and decryption of file data.

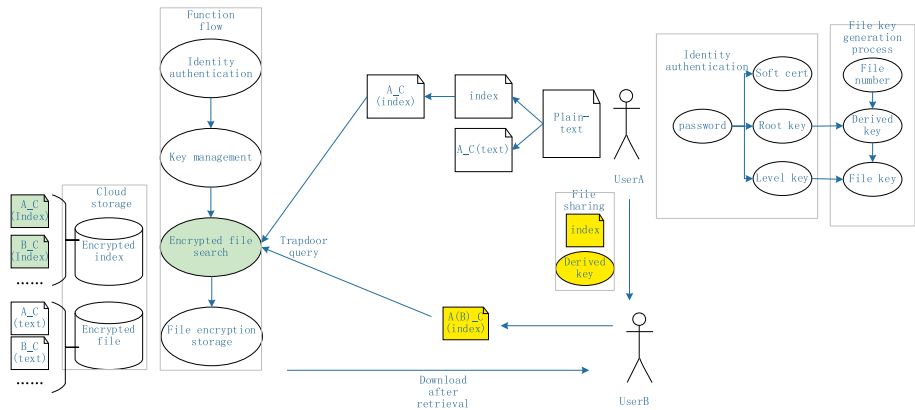


Fig. 1. Functional interaction model.

2.2 System Architecture

As is shown in Fig. 2, our system architecture mainly includes cipher search system client, cipher search service subsystem, key management subsystem and the existed big data platform it relies on (take HDFS as an example in the figure).

- The cipher search client uploads and downloads files to the HDFS storage system, performs encryption and decryption operations in the background of the client during uploading and downloading. When the client uploads the ciphertext file to the HDFS storage system, it also establishes a cipher keyword index list in the cipher search service subsystem for locating the ciphertext file. When the client initiates the sharing operation, it distributes the file key to the shared user's client through secure channel and notifies the shared user's client. After the

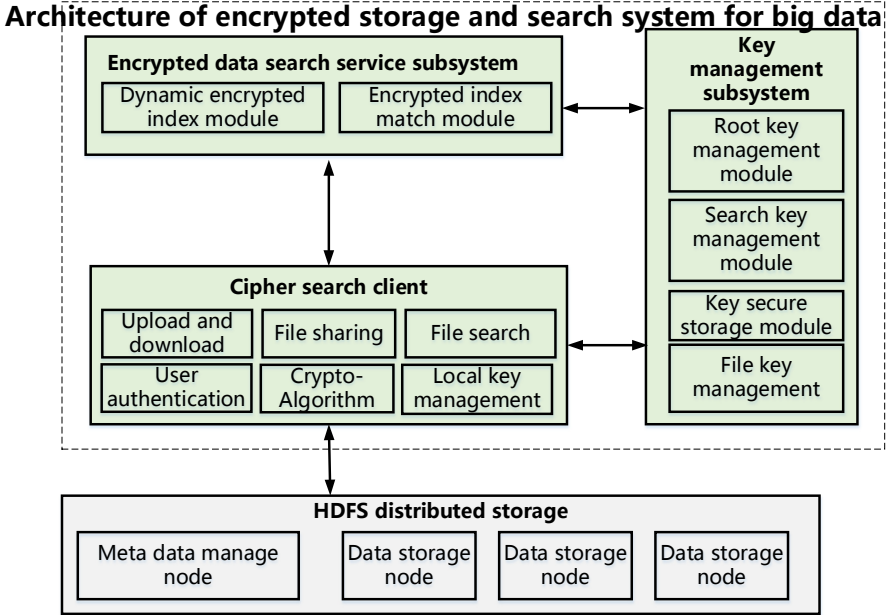


Fig. 2. System architecture.

shared user’s client receives the notification, its client background updates the shared ciphertext keyword index list to the search service subsystem to confirm sharing.

- The cipher search service subsystem obtain the search key from the key management subsystem, and establishes a ciphertext keyword index list for the user of the client when the client initiates uploading, receiving and sharing operations. It also provides storage, update and query of the index list.
- The key management subsystem provides file encryption keys for the clients when they perform upload and download operations, provides file encryption keys for shared user clients when sharing, provides search keys for clients when searching. At the same time, key management system provides search keys for search service system to establish, update and search the cipher keyword index.

3 Main Cryptographic Algorithm

3.1 Key Management Algorithm

Suppose $SKE = (Setup, Enc, Dec)$ is a secure traditional symmetric encryption algorithm, $Hash: \{0, 1\}^* \rightarrow \{0, 1\}^*$ is an anti-collision hash function, key management scheme includes following steps:

1. System boot: Key management center initiate, generate system public parameter Param and master key MK.

2. User register: User U_i use his identity and attributions to initiate registration, key management centre compute user's root key RK from its master key MK (using IBE scheme to generate secret key), then issue a certificate PK for user as the public key corresponding to RK. Meanwhile, issue hierarchical key LK_u to users according to their secure level, of which $LK_u \in \{LK_1, LK_2, LK_3\}$. Hierarchical keys are divided into several levels, users of same level have the same classification key, let 1st level key be LK_u , which is a pseudo-random number generated by using pseudo-random function Rand. The 2nd level key is $LK_2 = Hash(LK_1|2)$, 3rd level key is $LK_3 = Hash(LK_2|3)$, of which | means string concatenation.
3. Encryption key derivation, encryption and management algorithm: A file may have different security levels, in order to improve security levels, users have to encrypt different levels of files with different keys.

A user first generates a file key FK (where $FK = Hash(RK|Filename|LK_u)$) by using his root key RK, filename and file security level, of which LK_u corresponds to user's authorization file security level. Besides, the user randomly choose a string nonce to compute secret key SK for searchable encryption, of which $SK = Hash(RK|nonce)$. The user can use SK to compute cipher index of the file, which is used to identify the owner of the file and facilitate the query operation of other users.

Finally, the user set a password to encrypt the file key FK (that is, $C_K = Encrypt(FK, hash(password))$).

3.2 File Classification Encryption/Decryption Algorithm

For file encryption, we adopt the method of digital envelope, that is, use symmetric encryption algorithm to encrypt the file, use searchable encryption to generate the cipher index of the file, finally encrypt the file key. Further, we adopt national algorithm of China SM2 in the encryption/decryption procedure.

For example, when it comes to a file with security level A, the user first calculates its file key by using steps in key derivation, that is, $FK_{A_u} = Hash(RK|filename|LK_u)$. Then use file key to $FK = FK_{A_u}$ to encrypt file A to get its ciphertext C_F . ($C_F = Enc(FileA, FK_{A_u})$).

Finally, the user use his password key K to encrypt file key, get $C_K = Enc(FK_{A_u}, K)$, stores the final cipher result $C = (C_F, C_K)$.

When it comes to decryption, user first decrypt file key $FK = FK_{A_u} = Dec(C_K, K)$, of which $K = hash(password)$. Then restore his original file key FK_{A_u} according to his authorization. Finally, restore the file by computing $File = Dec(C_F, FK_{A_u})$.

3.3 Symmetric Searchable Encryption Algorithm

Suppose user has file collection $D = (d_1, d_2, \dots, d_n)$ corresponding unique identifiers $(\bar{d}_1, \dots, \bar{d}_n)$, all of them has keywords collection $W = (w_1, w_2, \dots, w_n)$, suppose SKE_1 and SKE_2 are two traditional secure symmetric encryption algorithms, $f : \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^{k+log_2s}$ is a pseudo-random function. $\pi : \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ and $\psi : \{0, 1\}^k \times \{0, 1\}^{log_2s} \rightarrow \{0, 1\}^{log_2s}$ are two pseudorandom permutation functions. The main steps listed as follows:

1. Key generation: generate random key K_1, K_2, K_3 , separately used for pseudorandom permutation function ψ , pseudorandom function f and pseudorandom permutation function π , generate key K_4 for SKE_2 .
2. Keyword matching: scan the file collection to get corresponding keywords collection $\delta(D)$, for each $w \in \delta(D)$, there exist a document subset $D(w)$ corresponding to this keyword, of which $D(w) \subseteq D$, we set a global counter $ctr = 1$ in the traversal.
3. Construction of file index array A: for $1 \leq i \leq |\delta(D)|$, construct linked list L_i corresponding to i -th keyword. Each L_i 's nodes (expressed as $N_{i,j}$) order is random and the length $|D(w)|$ may be different. Suppose $id(D_{i,j})$ is j -th unique identifier in $D(w)$, generate each node $N_{i,j} \leq id(D_{i,j}) || K_{i,j} || \psi_{K_1}(ctr + 1) >$, encrypt node $N_{i,j}$ with key and write the cipher in position $\psi_{K_1}(ctr)$, of which $K_{i,j}$ is used for encrypting next node $N_{i,j+1}$ while $\psi_{K_1}(ctr + 1)$ is the pointer of next node. Every time the encryption storage cycle is completed, the counter is incremented by 1 ($ctr + +$). In addition, the pointer and key of the last node in the linked list are empty, that is, $N_{i,|D(w_i)| \leq id(D_{i,|D(w_i)|}) || 0^k || NULL >$.
Let $s' = \sum_{w_i \in \delta(D)} |D(w_i)|$, it means adding up the number of documents to which each keyword belongs. if $s' < s$, fill the remaining positions with random strings, s is the upper limit of array A's element capacity.
4. Construction of head node index T: the size of T is $\{0, 1\}^l \times \{0, 1\}^{k+\log_{2^s} \times ||}$, $|\Delta|$ is the total number of dictionary set keywords. For all the $w_i \in \delta(D)$, set the index entry of the first node as $T[\pi_{K_3}(w_i)] = (addr_A(N_{i,1}) || K_{i,0}) \oplus f_{K_2}(w_i)$. Fill the remaining items $|\Delta| - |\delta(D)|$ with random numbers.
5. Encrypt original data: for each document $d \in D$, compute $c \leftarrow SKE_2.Enc(K_4, d)$, get the final result $I = (A, T), C = (c_1, \dots, c_n)$.
6. Generate query token: compute $t = (\pi_{K_3}(w), f_{K_2}(w))$ and send it to cloud server.
7. Cloud search: Parse the query token t as (γ, η) , where $\gamma = \pi_{K_3}(w_i), \eta = f_{K_2}(w_i)$, find out if there is a result θ at position γ in the head node index T (that is, $\theta \leftarrow T(\gamma)$). If θ exists, then compute:

$$\begin{aligned}
 & \theta + \eta \\
 & = (addr_A(N_{i,1}) || K_{i,0}) \oplus f_{K_2}(w_i) \oplus f_{K_2}(w_i) \\
 & = (addr_A(N_{i,1}) || K_{i,0})
 \end{aligned} \tag{1}$$

Use key K' to decrypt the node in position $addr_A(N_{i,1})$, finally output all the identifiers in list L_i one by one.

8. For each encrypted document, compute $d \leftarrow SKE_2.Dec(K_4, c)$, of which K_4 is derived as is shown in file classification encryption/decryption algorithm.

3.4 Dynamic Update of Cipher Index

Due to the cipher state exposed to cloud, dynamic update has become an extremely important issue, which is related to security.

When user uploads a new file d_k , which includes keyword w_n and w_m , of which w_n is a keyword that has been established in the cloud's ciphertext index, while w_m is a new keyword that hasn't been established.

- For w_n , user has already constructed head node in the cipher index server, that is, $T[\pi_{K_3}(w_n)] = (addr_A(N_{n,1}) || K_{n,0}) \oplus f_{K_2}(w_n)$, the file uploading process is equivalent to resubmitting the trapdoor of w_n , which means that the server need to parse out all the identifiers containing w_n .

The original linked list is:

$$L_n = N_{n,1} || \dots || N_{n,j-1} || N_{n,j} || N_{n,j+1} || \dots || N_{n,|D(w_n)|} \quad (2)$$

where $N_{n,j} \leq id(D_{n,j}) || K_{n,j} || \psi_{K_1}(ctr+1) >$, $N_{n,|D(w_n)|} \leq id(D_{n,|D(w_n)|}) || 0^k || NULL >$

The server will reconstruct the linked list L_n after parses out all the identifiers, that is reconstruct the tail node and attach a new node, which has the minimum computational overhead.

Generate random key $K'_{n,|D(w_n)|}$, then use it to encrypt the new tail node $N_{n,|D(w_n)|+1}$ through symmetric encryption algorithm SKE_1 . The tail node $N_{n,|D(w_n)|}$ of original linked list L_n is updated to

$$N'_{n,|D(w_n)|} \leq id(D_{n,|D(w_n)|}) || K'_{n,|D(w_n)|} || \psi_{K_1}(ctr_{|N_{ij}|} + 1) >$$

and then attach a new node $N_{n,|D(w_n)|+1} \leq id(D_{n,|D(w_n)|+1}) || 0^k || NULL >$, of which $id(D_{n,|D(w_n)|+1}) = id(d_k)$, at this time the new linked list is:

$$L'_n = N_{n,1} || \dots || N_{n,j-1} || N_{n,j} || N_{n,j+1} || \dots || N_{n,|D(w_n)|-1} || N'_{n,|D(w_n)|} || N_{n,|D(w_n)|+1} \quad (3)$$

So when the client updates the server array, it only needs to submit the server:

$$A[\psi_{K_1}(ctr_{|D(w_n)|})] \leftarrow SKE_1.Enc(K_{n,|D(w_n)|-1}, N'_{n,|D(w_n)|}) \quad (4)$$

$$A[\psi_{K_1}(ctr_{|N_{ij}|} + 1)] \leftarrow SKE_1.Enc(K'_{n,|D(w_n)|}, N_{n,|D(w_n)|+1}) \quad (5)$$

Equation (4) overwrite the encrypted node in the original position with the new node. Equation (5) overwrite the random string of position $A[\psi_{K_1}(ctr_{|D(w_n)|})]$ with the new tail node.

- For w_m , the user has not established the cipher index of w_m before, thus need to construct new linked list L_m , at this time, the linked list only needs to construct one node $N_{m,1} \leq id(D_{m,1}) || 0^k || NULL >$, where $id(D_{m,1}) = id(D_{n,|D(w_n)|+1}) = id(d_k)$, that is, the same identifier corresponds to different keywords.

Under this condition, the head node cipher index needs to be newly constructed: $T[\pi_{K_3}(w_m)] = (addr_A(N_{m,1}) || K_{m,0}) \oplus f_{K_2}(w_m)$, the node storage position is calculated by client: $addr_A(N_{m,1}) = \psi_{K_1}(ctr_{|N_{ij}|} + 1)$, of which $ctr_{|N_{ij}|}$ is the number of filled nodes in array A, besides, the remaining $s - s'$ positions still fill with random strings, $s' = ctr_{|N_{ij}|}$.

So when the client updates keyword w_m in the server array, it only needs to submit the server:

$$A[\psi_{K_1}(ctr_{|N_{ij}|} + 1)] \leftarrow SKE_1.Enc(K_{m,0}, N_{m,1}) \quad (6)$$

$$T[\pi_{K_3}(w_m)] \leftarrow (addr_A(N_{m,1}) || K_{m,0}) \oplus f_{K_2}(w_m) \quad (7)$$

File content encryption only needs to refer to step 5) in Sect. 3.

From the new uploaded items in index update Eq. (4), (5), (6), (7), we can see that every time a new document with k keywords is uploaded, whether it is an existing keyword or a new keyword, it is equivalent to the client re-executing the search process for each keyword, and updating the trapdoor of the keywords at $2k$ index positions in array A , without downloading the whole cipher index $I = (A, T)$ locally and updating it totally, thus reducing the computational overhead and bandwidth consumption of ciphertext update.

4 Deployment and System Test

4.1 Deployment

Figure 3 shows the application deployment and network connectivity of different modules of searchable encryption system. The client software is deployed on the user side, which provides encryption and decryption of user files, file indexes and search requests, sends encrypted ciphertext files to the distributed storage system, and sends encrypted search requests to the cipher search service subsystem.

Service side is constructed based on existed big data platform (take HDFS as an example), so users need to provide HDFS storage system and read-write interface for clients to call. Clients directly store ciphertext files in HDFS, and locate ciphertext files stored in HDFS based on ciphertext keyword index list stored in cipher search subsystem.

Cipher search service subsystem is deployed on a separate server, which is thought to be “honest but curious”. It establishes a ciphertext keyword index list for locating ciphertext files stored in HDFS. It responds to the client search request, searches the ciphertext file location and returns it to the client.

The key management subsystem provides search keys and file encryption keys for client and provides search keys for search service subsystem.

4.2 System Test

Based on the deployment, we can further construct the environment. We adopted Huawei’s big data platform FusionInsight HDFS system, which is mainly used to store encrypted files and cipher index uploaded by users. The performance indexes of the test equipment are shown in Table 1.

Test of Encryption

In terms of encryption efficiency, we perform 10 encryption tests on files of 100 Mb size, record the time consumption of each file encryption and calculate the average time consumption. As is Shown in Fig. 4. We can see that the encryption efficiency is close to 300 Mbps.

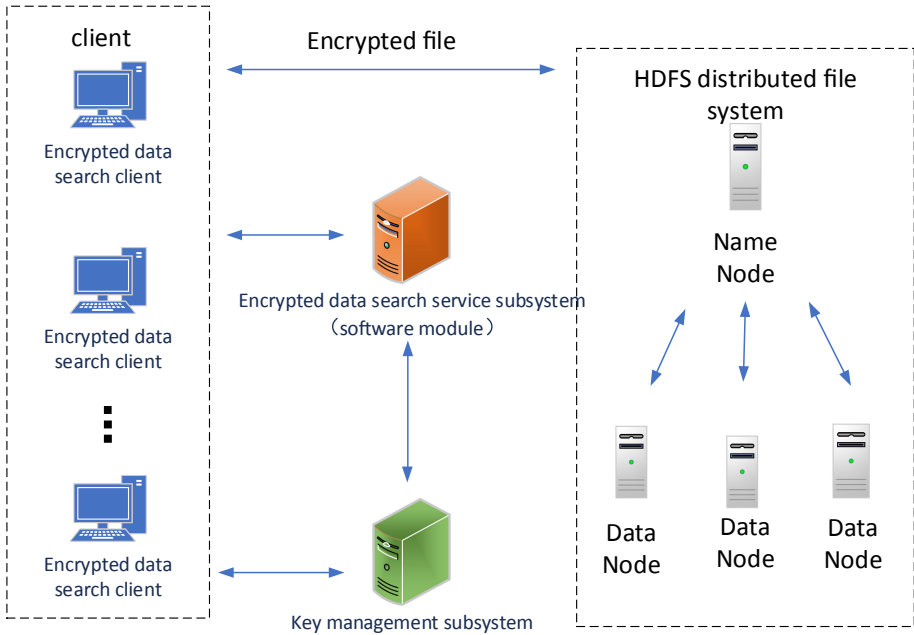


Fig. 3. Deployment of cipher search system.

Table 1. System configuration and experimental environment.

Hardware requirements	Operating system	Usage
2 servers, 8 core, Intel-i7, 32 GB memory	Ubuntu 16.04	Deployment of cipher search and key management software
Desktops ≥ 1 , Intel-i7, 16 GB memory	win7/win10	Deployment of client software
Servers ≥ 3 , Intel-i7, 8 core, 32 GB memory	Ubuntu 16.04	Deployment of HDFS

Test of Efficiency

When it comes to search efficiency, considering the index size of keywords is much smaller than the size of file. We constructed an index of 100 Mb size for retrieval tests, recorded time consumption of search efficiency, and calculated the average time consumption. As is shown in Fig. 5. We can see that the search efficiency is more than 4000 Mbps, All in all, whether it is encryption or search efficiency, even if the network transmission delay is included, the efficiency will be on the order of milliseconds, which will not affect user experience and gives consideration to security and availability.

```

Console  Tasks
<terminated> FileOperateAction [Java Application] C:\Program Files\Java\jre1.8.0_141\bin
The 0-th time:
encryption efficiency (200Mb)804ms
The 1-th time:
encryption efficiency (200Mb)675ms
The 2-th time:
encryption efficiency (200Mb)687ms
The 3-th time:
encryption efficiency (200Mb)658ms
The 4-th time:
encryption efficiency (200Mb)676ms
The 5-th time:
encryption efficiency (200Mb)707ms
The 6-th time:
encryption efficiency (200Mb)664ms
The 7-th time:
encryption efficiency (200Mb)685ms
The 8-th time:
encryption efficiency (200Mb)678ms
The 9-th time:
encryption efficiency (200Mb)645ms
average encryption efficiency(200Mb. excute 10 times)296.3820549927641Mbps

```

Fig. 4. Encryption efficiency of the system.

```

The 34-th time, search 100Mb keyword index file needs 6ms
The 35-th time, search 100Mb keyword index file needs 6ms
The 36-th time, search 100Mb keyword index file needs 6ms
The 37-th time, search 100Mb keyword index file needs 6ms
The 38-th time, search 100Mb keyword index file needs 7ms
The 39-th time, search 100Mb keyword index file needs 7ms
The 40-th time, search 100Mb keyword index file needs 7ms
The 41-th time, search 100Mb keyword index file needs 18ms
The 42-th time, search 100Mb keyword index file needs 7ms
The 43-th time, search 100Mb keyword index file needs 5ms
The 44-th time, search 100Mb keyword index file needs 6ms
The 45-th time, search 100Mb keyword index file needs 7ms
The 46-th time, search 100Mb keyword index file needs 7ms
The 47-th time, search 100Mb keyword index file needs 6ms
The 48-th time, search 100Mb keyword index file needs 6ms
The 49-th time, search 100Mb keyword index file needs 6ms
search keywords index file(100Mb). excute 50 times,search efficiency: 4347.83Mbps

```

Fig. 5. Search efficiency of the system.

4.3 Advantage Analysis

Our system supports the storage of ciphertext data and the direct query of the stored ciphertext data. In the query process, the storage end can not know what the user's query content is, nor can it know the user's file data.

The system supports the data owner to actively share the ciphertext data stored by himself with other users. In the process of sharing, the storage side cannot know the sharing behavior of users, nor can it know the plaintext of sharex file data. Meanwhile, encrypting the search request ensures that the server can't perform correlation analysis on the user's search behavior.

Users can completely control their own data through private keys and the storage end cannot know, steal or disclose users' plaintext data.

The cipher search system is loosely coupled with the big data platform, which means it can be deployed quickly only by providing the file data read-write interface of the big data platform, which is more practicability than other solutions.

5 Conclusion

In this paper, we have proposed a cipher search system for big data platform on the basis of searchable encryption algorithm. Especially, we have constructed a scheme.

that takes into account both the security and efficient use of the data. Meanwhile, we designed user-centric key management and file level fine-grained encryption and decryption, effectively preventing the risk of data leakage from getting out of control, which greatly improve the security of encrypted data storage and utilization. In the future work, we will further extend data protection to other ciphertext calculate algorithms. For instance, we will perform fully homomorphic encryption, order-preserving encryption and secure multi-Party computation in a big data fashion.

Acknowledgements. This work is supported by the Sichuan Science and Technology Program (2021JDR0077), the Sichuan Province's Key Research and Development Plan.

"Distributed Secure Storage Technology for Massive Sensitive Data" Project (2020YFG0298), and Applied Basic Research Project of Sichuan Province (No. 2018JY0370).

References

1. Li, H., Yang, Y., Dai, Y., Yu, S., Xiang, Y.: Achieving secure and efficient dynamic searchable symmetric encryption over medical cloud data. *IEEE Trans. Cloud Comput.* **8**(2), 484–494 (2020). <https://doi.org/10.1109/TCC.2017.2769645>
2. He, K., Chen, J., Zhou, Q., Du, R., Xiang, Y.: Secure dynamic searchable symmetric encryption with constant client storage cost. *IEEE Trans. Inf. Forensics Secur.* **16**, 1538–1549 (2021). <https://doi.org/10.1109/TIFS.2020.3033412>
3. Shen, J., Wang, C., Wang, A., Ji, S., Zhang, Y.: A searchable and verifiable data protection scheme for scholarly big data. *IEEE Trans. Emerg. Topics Comput.* **9**(1), 216–225 (2021). <https://doi.org/10.1109/TETC.2018.2830368>
4. Chen, G., et al.: Differentially private access patterns for searchable symmetric encryption. In: *IEEE Conference on Computer Communications*, Honolulu, USA, pp. 810–818 (2018)
5. Song, Q., et al.: SAP-SSE: protecting search patterns and access patterns in searchable symmetric encryption. *IEEE Trans. Inf. Forensics Secur.* **16**, 1795–1809 (2021). <https://doi.org/10.1109/TIFS.2020.3042058>
6. Mishra, P., et al.: Oblix: an efficient oblivious search index. In: *IEEE Symposium on Security and Privacy San Francisco, USA*, pp. 279–296 (2018)
7. Liu, X., Yang, G., Mu, Y., Deng, R.H.: Multi-user verifiable searchable symmetric encryption for cloud storage. *IEEE Trans. Dependable Secure Comput.* **17**(6), 1322–1332 (2020). <https://doi.org/10.1109/TDSC.2018.2876831>
8. Wang, Y., et al.: Towards multi-user searchable encryption supporting Boolean query and fast decryption. *J. Univ. Comput. Sci.* **25**(3), 222–244 (2019)

9. Pang, H., Zhang, J., Mouratidis, K.: Scalable verification for outsourced dynamic databases. *VLDB Endowment* **2**(1), 802–813 (2019)
10. Belguith, S., et al.: Phoabe: securely outsourcing multi-authority attribute based encryption with policy hidden for cloud assisted IOT. *Comput. Netw.* **133**, 141–156 (2018)
11. Liu, X., et al.: Privacy-preserving multi-keyword searchable encryption for distributed systems. *IEEE Trans. Parallel Distrib. Syst.* **32**(3), 561–574 (2021). <https://doi.org/10.1109/TPDS.2020.3027003>
12. Zhang, K., et al.: Lightweight searchable encryption protocol for industrial Internet of Things. *IEEE Trans. Industr. Inf.* **17**(6), 4248–4259 (2021). <https://doi.org/10.1109/TII.2020.3014168>
13. Ge, X., et al.: Towards achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification. *IEEE Trans. Dependable Secure Comput.* **18**(1), 490–504 (2021). <https://doi.org/10.1109/TDSC.2019.2896258>
14. Wang, H., et al.: Encrypted data retrieval and sharing scheme in space-air-ground integrated vehicular networks. *IEEE Internet of Things J.* <https://doi.org/10.1109/JIOT.2021.3062626>
15. Sultan, N.H., Laurent, M., Varadharajan, V.: Securing organization's data: a role-based authorized keyword search scheme with efficient decryption. *IEEE Trans. Cloud Comput.* <https://doi.org/10.1109/TCC.2021.3071304>
16. Mante, R.V., Bajad, N.R.: A study of searchable and auditable attribute based encryption in cloud. In: 2020 5th International Conference on Communication and Electronics Systems (ICCES), pp. 1411–1415 (2020). <https://doi.org/10.1109/ICCES48766.2020.9137860>