A DYNAMIC SYMMETRIC SEARCHABLE ENCRYPTION ALGORITHM WITH AN EXTENDED DUAL-INDEX STRUCTURE LEVERAGING IPFS

Yi CAI[1,*], Mingsheng FENG[2]

Traditional encryption methods typically encrypt the entire dataset and searching and querying can only be performed after decryption. Searchable Encryption enables searching, matching, and querying operations to be performed on encrypted data, providing search results without revealing plaintext information. Dynamic Symmetric Searchable Encryption allows dynamic insertion, updating, and deletion operations to be performed on encrypted data while maintaining the data's searchability and security. This paper improves the dynamic symmetric searchable encryption algorithm based on a dual-index structure. It extends the index structure and utilizes the IPFS self-verification mechanism to achieve rapid verification, thereby reducing the number of queries and time required. Additionally, it reduces the data storage costs on the blockchain. Finally, the proposed approach is tested on the Ethereum blockchain using the publicly available dataset Eron email, and the results effectively demonstrate the superiority of this solution in terms of time overhead.

Keywords: Blockchain, IPFS, Dynamic searchable encryption, Index

1. Introduction

Currently, many distributed storage and application systems have increasingly high-security requirements for data interaction. For example, smart grids [1] [2] have introduced a blockchain-based distributed architecture to protect the security of data interactions. During the process of data transactions in those Secure Data Exchange Systems (SDES) [3-5], caching mechanisms and cryptographic hash functions are often employed to improve data retrieval efficiency and reduce the number of data uploads to the system. The caching data is stored in the blockchain and it needs to keep privacy and security during the search process. So, the cache transaction data will be encrypted before upload. And searchable encryption techniques are used to achieve the search functionality while ensuring privacy and security.

[1] Prof., School of Computer Engineering, Guangzhou City University of Technology, Guangzhou, China, Corresponding author's e-mail: caiyi@gcu.edu.cn
[2] Eng., School of Software Engineering, South China University of Technology, Guangzhou, China, e-mail: fengmingsheng@bytedance.com

Most searchable encryption schemes [6][7] are designed for cloud storage, but there are few searchable encryption schemes based on blockchain [8][9]. In this paper, we first analyze other searchable encryption schemes based on blockchain. Then, addressing their limitations, we propose a dynamic searchable encryption scheme based on blockchain. We provide algorithm definitions and security proofs for the proposed scheme and finally validate the feasibility and advantages of the scheme through experimental analysis.

1.1 Related Work

Many scholars are dedicated to addressing various issues in data trading. Some researchers [10-12] are dedicated to addressing the contentiousness of data, including concerns such as data sources' accuracy, data integrity, and data authenticity. Another area of research [13] focuses on the security of the transaction process, ensuring that both parties cannot repudiate the transaction and that privacy is not compromised. Furthermore, there is research being conducted on secure and efficient data retrieval [6-9]. Lastly, researchers are exploring access control issues for data commodities, such as granting and revoking access control permissions [14-16] or using Ciphertext-Policy Attribute-Based Searchable Encryption [17-18] and also focus on search mode hidden [19-20], especially in the context of health data management.

Traditional applications of searchable encryption are mostly found in cloud storage, where encrypted data is stored along with the corresponding indices on cloud servers. Current searchable encryption schemes include public-key encryption search, symmetric encryption search, fuzzy search, Boolean search, and others. However, most of these schemes are based on a central server, and the correctness of the search results returned by an untrusted central server cannot be guaranteed.

With the development of blockchain technology, there have also been searchable encryption schemes related to blockchain. For example, in Scheme [8], a Bitcoin framework is used to achieve fast retrieval of data stored on the blockchain. However, this scheme does not consider dynamic updates. Scheme [6] combines dynamic searchable encryption with Ethereum, storing the index and encrypted data on the Ethereum blockchain. It designs a fair data retrieval scheme using smart contracts. However, storing both the index and encrypted data on the blockchain incurs significant storage costs.

Scheme [9] utilizes a dual-index structure to achieve fast update operations. This scheme also considers the verification of search result sets, with the verification data stored on the blockchain. However, the index data is stored in a one-to-one format using keyword files (w, f), and the number of index items increases with the growth of files. Similarly, the verification data increases linearly with the growth of the file set and the verification process adds to the query time.

1.2 Our Contribution

This paper proposes a blockchain-based dynamic searchable encryption scheme based on the foundation of Scheme [9] and integrating blockchain technology. Firstly, the scheme employs a dual-index structure to enable efficient dynamic updates. Secondly, by storing encrypted files on IPFS and referencing their IPFS hash values on the blockchain, the scheme ensures secure and efficient data access while reducing the storage load of the blockchain. Thirdly, it stores the document set associated with a keyword using IPFS, reducing the number of queries and query time. Finally, it provides a forward security guarantee by linking new index items to the head of the existing index chain. By incorporating these features, the proposed scheme aims to address the limitations of previous approaches and provide an advanced solution for dynamic searchable encryption based on blockchain.

2. Scheme Construction

The main components of this scheme include index construction, keyword search, file addition, and file deletion. The main steps are as follows:

2.1 Index construction algorithm

The index construction algorithm is executed by the data owner and generates two outputs: the index table *I* and the state table S. The index table *I* is stored on the blockchain network, while the state table S is stored locally on the user's device. The index table *I* contains the mappings between keywords and files, allowing for efficient retrieval of files based on the index. The state table S maintains various status information, such as the search frequency of keywords, file additions, and deletions.

The index in this scheme consists of a dual-index structure, which includes a keyword index and a file index. The keyword index enables fast file retrieval. The index is composed of the index table address entry $^{id}{}_{w}$, the keyword index entry I_w ,the encrypted data IPFS storage address ipfsadd, and the file deletion table B. During a query operation, the search trapdoor allows for quick location of the address in the index table, leading to the corresponding data retrieval. The file index contains the file id and the set of corresponding index table address entries $\left[id_w \cdots\right]$. When adding or deleting files, this index facilitates quick location of the index entries for updates. The index construction process is as follows:

1) For each keyword $w_i, i \in \{1, m\}$, update the status *a* ,*r*,*d* ,search count *s*, and file count *n*. *a* records the file addition operation corresponding to the keyword, *r* records the search operation corresponding to the keyword, and *d* records the file deletion operation corresponding to the keyword. *a, r, d* ensure forward security of

the search. *s* is initialized to 0, and n represents the number of files corresponding to the keyword.

2) For each keyword w_i , an index address entry $id^f_{w_i}$ is generated using the private key *K*, a pseudorandom function G_w , a hash function H_1 , and flags a and *r*. The encryption key k_{enc} is used for file encryption. The search trapdoor k_w is related to the private key *K* and the search state *r*, while the encryption key is only related to the search state *r*.

3) For each keyword, the keyword index I_w is generated using the hash function H_1 and the index address entry $id^f_{w_i}$. In the index construction phase, the I_w does not embed the next index address entry because there is only one address entry at this point. The entire file collection is encrypted using a pseudorandom function *F* and the encryption key k_{enc} , and then uploaded to IPFS to obtain the IPFS address, which is a hash of the data content and can be used for data verification. Finally, the encrypted index entry consists of [$id^f_{w_i} : I_{w_i}$,*ipfsadd, B, d*].

4) For each file *f* , its position in each keyword file collection is recorded and stored in the state table. The final state table *S* consists of [f_j : ($id^f_{w_i}$, j)...($id^f_{w_n}$, *m*)][w_i : ($id^f_{w_i}$, *j*), *a, r, n, s*].

Algorithm 1: Setup

Data:Given a private key *K*, the pseudorandom function {*F*, G_w}, the multiset hash function {H_1, *H*},keyword set {w_i }, i ∈ {1, m}, file set { f_j }, *j* ∈{*1, n}*

Result: The index table *I* and the state table *S*.

Initialize a empty hash table as a state table.
for w_i from w_1 to w_m do:
Update state *a,r,d,*set search count *s = 0*, and file count *n = 0*.

$id^f_{w_i}$ =H_1(w_i , H({f_j}))

k_w = G_w(K , w_i||r)
k_{enc} = G_f (K , w_i) //Key for encrypt files.
$f_{enc} = F\ (K_{enc}, \{fj\}), j \in \{1, n\}$
ipfsadd = F(k_{enc},ipfs.put(f_{enc}))
$I_{w_i} = H_1(k_w, id^f_{w_i}) \oplus \mathcal{L}$
Initializing file deletion table B.
for $f_j \in DB(w_i)$ do
if s[f_j] is not exist then
Add (f_j , ($id^f_{w_i}$,j)) to state table S;
else

S[fj].insert((f_j , ($id^f_{w_i}$,j)));

end

end

Add [$id^f_{w_i}$: I_{w_i} , ipfsadd, B, d] to index table *I*.

Add [w_i: $id^f_{w_i}$, a, r, n, s] to state table *S*.

i++

end

Uploading encrypted index table I to the blockchain network.

2.2 Query algorithm

The query process primarily involves searching keyword indexes with three main steps.

1) **Generating search trapdoor**. The user searches for the state table *S* locally based on keywords, obtaining the corresponding index item address and other state values. User use a pseudorandom function G_w , private key K,and the retrieved state *r* to generate the search tag k_w and encryption key k_{enc} . After each search, the search count *s* is updated. Then, the search trapdoor (k_w , k_{enc} , id_w) is sent to the blockchain network.

2) **Executing the query**. Once the blockchain network receives the query trapdoor, the querying process begins as follows: Firstly, the index table is queried based on the initial index item address id^f_w . It provides the corresponding encrypted data ipfsadd, deletion table *B*, and deletion status *d*. Next, by applying the hash function H_1 (k_w , id^f_w) and XOR operation with I_w , the next address item is obtained. This is because newly added data is temporarily stored in new index items and linked with the old index items. Once the entire search is completed, the result set containing ipfsadd is returned to the user.

3) D**ownloading the data and updating the index**. After decrypting the IPFS address using the decryption private key k_{enc}, the user downloads the file collection based on the IPFS address. For each file collection, the user checks the deletion flag *d* and examines if there are any files marked for deletion. If there are files to be deleted, the user updates each file collection based on the deletion table *B*. Once all the updates are completed, the user checks if new file collections were added before this query, indicated by the flag *a*. If new files were added user combines multiple file collections and reuploads them to IPFS to obtain a new address value. This is done to improve the efficiency of regular queries (i.e., without file updates). Then, the corresponding ipfsadd for the blockchain index table, keyword index item I_{w_i} , and local status table *S* are updated.

Algorithm 2: Search

Data:Given a private key K, the pseudorandom function $\{F, G_w\}$, the multiset hash function $\{H_1, H\}$,keyword w

Result: Matched result file collection.

//**Generating query token for the user.**

$id^f_{w_i}$, $a, r, n, s = S[w]$

$k_{enc}=G_w(K,w)$
$k_w=G_w(K,w||r)$
$s++$

Sending (k_w , $id^f_{w_i}$) to the blockchain network.

//**Executing the query on the blockchain.**

I_w , ipfsadd, B, d = I[$id^f_{w_i}$]

$I'_w = I_w$, $id^{f'}_w = id^f_w$

// Initializing the result set Res.
Res.insert(*ipfsadd, B, d*)

while H_1 (k_w , $id^{f'}_w$) $\oplus$ I'_w is not equal $\perp$ do

$id_w=H_1(k_w, id^{f'}_w \oplus I'_w$

$id^{f'}_w = id_w$

I'_w, newipfsadd, newB, newd=I[$id^{f'}_w$]

Res.insert((newipfsadd,newB,newd))

end

Return the result set Res.

//**The user downloads data and updates the index.**

//The user decrypts using the private key k_{enc} and obtains the plaintext IPFS address.

for $ipfs_i \in$ Res do

$data_i$=ipfs.get($ipfs_i$)

id'=H_1(w,H($data_i$))

for f $\in$ $data_i$ do

Update S[f], (id',n_f) $\rightarrow$ (id^f_w ,n_f)

if $d \neq 0$ then

Update $data_i$ with delete table B.

d=0

end

end

if $a \neq 0$ then

Integrate all file sets corresponding to w and upload the [$data_i$, ..., $data_j$] to IPFS and get the corresponding IPFS address value newipfsadd.

A=0

I_{w_i} = $H_1(k_w, id_w^f) \oplus \perp$

Send [$id_{w_i}^f$, I_{w_i} , newipfsadd, d] to blockchain to update the index table I.

Update state of S[w].

end

Decrypt $data_i$ using the private key k_{enc} to obtain the plaintext data $Data_i$.

end

2.3 File insert algorithm

The insertion algorithm focuses on the insertion of existing keywords because inserting new keywords follows the same index construction algorithm. The file insertion process is as follows:

1) Obtain the state values a, r, n, s, index address entries $id_{w_i}^f$ from the state table *S* based on the keyword w.

2) Update the insertion flag a and the file count n.

3) Generate a new index entry $id_{w_i}^{f'}$ based on the new file f and the hash function.

4) According to the search count s, update the search flag r. Then reset the value of s.

5) Generate an encryption key k_{enc} using a pseudorandom function and the private key K. Encrypt the file and upload it to IPFS to obtain the corresponding address value.

6) Using the hash function H_1 and the new index entry address $id_{w_i}^{f'}$, generate a keyword index I_{w_i} and embed the previous index address entry, linking the old data together. Then update the state table *S* and the index table *I*.

Algorithm 3: AddFile

Data:Given a private key *K*, the pseudorandom function {*F*, G_w}, the multiset hash function {H_1, *H*},new file f,state table S.

Result: The updated index table I_{upt} and the state table S_{upt}.

for $w_i \in$ DB(f) do

$id_{w_i}^f$, *a, r, n, s* =*S*[w_i]

a++

n++

$id_{w_i}^{f'}$ =H_1(w_i, H(f))

if $s \neq 0$ then
r++,s=0
end
k_{enc}=G_w(K,w_i) //Key for encrypt file.
f_{enc}= F (k_{enc},{f})
ipfsadd = F(k_{enc}, ipfs.put(f_{enc}))
k_w=G_w(K,w_i||r)
I_{w_i} = H_1(k_w, $id_{w_i}^{f'}$) $\oplus$ $id_{w_i}^{f}$
S[w_i] = ($id_{w_i}^{f'}$, a, r, n, s)
Add [$id_{w_i}^{f'}$: I_{w_i} , ipfsadd, B, d] to index table I′ .
Add [f, ($id_{w_i}^{f'}$, n)] to state table S.

End
Send new index I′ to blockchain to update.

2.4 File delete algorithm

The scheme adopts a dual indexing structure, where the index address entries corresponding to the files are recorded in the state table. When deleting a file, it is only necessary to locate the corresponding index address entries in the state table *S* based on the file f. These index address entries are then sent to the blockchain, where the blockchain updates the deletion flag and the pending deletion file table based on the index address entries. The actual file deletion is postponed until the next query, reducing the time consumption associated with file deletion.

Algorithm 4: DeleteFile

Data: Given the file f for delete, and the state table S.

Result: The updated index table I_{upt}.

//**User generate the trapdoor**.

[($id_{w_i}^{f}$, i), ..., ($id_{w_j}^{f}$, j)] = S[f]

Send [($id_{w_i}^{f}$, i), ..., ($id_{w_j}^{f}$, j)] to blockchain.

//**The blockchain network delete the file**.

for ($id_{w_i}^{f}$,i), i ∈ (i, j) do

B, d = I[$id_{w_i}^{f}$]

d + +

B.insert(i)

I[$id_{w_i}^{f}$] = B, d

end

3. Security Analysis

All data in this proposed solution is stored in encrypted form, adhering to the security definition of searchable symmetric encryption.Nodes on the blockchain only know the size of the index and cannot know any other potential information. By using IPFS to store data, IPFS addresses data using content hashes, and the IPFS address itself serves as data verification. Therefore, as long as the data can be retrieved through a search, it indicates that the data is correct and accurate. When dynamically updating and adding new files, new indexes are generated based on new private keys and new state values. This prevents old search traps from linking to new data, ensuring forward security. Additionally, newly inserted index entries will be linked to existing data, ensuring the integrity of the entire dataset. Therefore, the searchable encryption algorithm presented in this article is secure and reliable.

According to the security concepts of dynamic Searchable Symmetric Encryption (DSSE), this article provides a formal security definition and proves the forward security of the blockchain-based dynamic searchable encryption algorithm. From the perspective of an attacker, we define three leakage function as follows:

Leakage function L_{set} for index construction: This function describes the information that an attacker can obtain from the execution of the algorithm during the index construction process. It includes the extent of leakage of input data used to generate the index. L_{set}=(< $|I_w|$, |ipfsadd|, |B|, |d| >), where m identifies the count of entries in the index table, and < $|I_w|$, |ipfsadd| > refers to the size or length of a single encrypted index item.

Leakage function L_{search} for search: This function describes the information that an attacker can obtain from the execution of the algorithm during the search process. It includes the extent of leakage of search queries, search results, search patterns, etc. L_{search}=(K_w, < $|I_w|$, |ipfsadd|, |B|, |d| >), where K_w represents the query key corresponding to the keyword. During the querying process, it may lead to index updates. <$|I_w|$, |ipfsadd|, |B|, |d| > is the size of new index.

Leakage function L_{update} for update: This function describes the information that an attacker can obtain from the execution of the algorithm during the update process. It includes the extent of leakage of updated files, types of update operations, timestamps, etc. L_{update}=(op, < $|I_w|$, |ipfsadd|, |B|, |d| >$_u$), where op represents the two types of update operations, which are adding(add) or deleting (del) files. u represents the number of newly added or deleted indexes.

Definition 1. According to the definition of dynamic SSE algorithm, the encryption search algorithm designed in this paper can be represented as (KeyGen, Setup, Search, Update), where the leakage functions are represented as L_{set}, L_{search}, L_{update}. Additionally, two probabilistic models, $Real_A(k)$ and $Ideal_{A,S}(k)$, are defined to measure the security of the algorithm. These models involve a probabilistic polynomial time (PPT) attacker A and a simulator S.

$Real_A(k)$: First, the data owner generates a private key K using the KeyGen algorithm, which serves as the initial key parameter for subsequent algorithms. Attacker A provides a dataset DB(w). The data owner constructs the index using the index construction algorithm Setup or updates the encrypted index using the index update algorithm Update. Then, A adaptively constructs a series of probabilistic polynomial time(PPT) search queries by executing the search algorithm Search. Finally, A returns the observed results as output.

$Ideal_{A,S}(k)$: Attacker A is given a dataset DB(w), and simulator S utilizes the leakage functions L_{set} and L_{update} to simulate the encrypted index and the update process for attacker A. Then, A adaptively constructs a series of probabilistic polynomial-time (PPT) search queries. S utilizes the leakage function L_{search} obtained from each simulated query to simulate search tokens and encrypted data items. Finally, A returns the obtained results as output. Assuming that for all PPT attackers A with leakage functions L_{set}, L_{search}, L_{update}, the dynamic searchable encryption algorithm satisfies adaptive security, there exists a simulator S that satisfies the following condition: $Pr[Real_A(k) = 1] - Pr[Ideal_{A,S}(k) = 1] \leqslant negl(k)$, where the function negl(k) is negligibly small.

Theorem 1. If G and F are secure pseudorandom functions, then the algorithm with leakage functions (L_{set}, L_{search}, L_{update}) is adaptively secure in the random oracle model.

Proof: The simulator S can utilize the leakage functions L_{set} and simulate an indistinguishable encrypted index (I_w, ipfsadd, B, d) that contains m random items, where each index item is a random string of length $|I_w|+|ipfsadd|+|B|+|d|$. Through the leakage function L_{search}, S can simulate the first search trapdoor, and the simulated result is indistinguishable from the real result. For queries, S generates a random string representation of the query trapdoor K_w, performs a random search, and can simulate a random string ipfsadd representing a query after adding a file. The required ipfsadd to update can also be simulated by S, and the simulated results are indistinguishable from the real results. It is important to note that the leakage function L_{update} satisfies the definition of forward security. Based on L_{update}, S follows the simulation process of constructing the index to generate simulated additions. It is infeasible to determine the correlation between different search trapdoors by encoding new strings into newly added index items. Furthermore, the random oracle model exhibits pseudorandomness, and symmetric encryption is semantically secure. As a result, the attacker A cannot distinguish

between the outputs of the real probability model $\mathrm{Real}_A(k)$ and the ideal probability model $\mathrm{Ideal}_{As,Ss}(k)$. This proves the security of the designed encrypted search algorithm within the defined leakage functions.

4. Experimental Analysis

4.1 Experimental Setup

The experiment implemented the searchable encryption scheme using Python and Solidity. Python was used to implement various encryption methods, while Solidity was used to simulate the Ethereum blockchain using the Ganache Ethereum private chain tool. The encryption library in Python was used to implement symmetric encryption with AES-128 and pseudo-random functions with HMAC-256.The experiment utilized the publicly available dataset ”Eron Email” and preprocessed it into a (keyword, file set) format. After preprocessing, files of different sizes were associated with different keyword-file pairs. In this experiment, the comparative scheme used was the dual index scheme [9]. We reproduced the same scheme in the same experimental environment.

4.2 Experimental Metric

The main experimental metric is the runtime, which measures the time taken by various algorithms in the searchable encryption scheme. Specifically, we tested the time overhead of four algorithms: index construction, keyword searching, file addition, and file deletion.

4.3 Analysis of Experimental Results

The result of index construction is shown in Fig. 1. It can be observed that the proposed solution in this paper exhibits reasonable time overhead as the data volume of index items increases from 10K to 160K, with the time ranging from 1.7 seconds to 13.0 seconds. Similarly, the dual-index scheme [9] was reproduced in the experiment, and the time for index creation ranged from 2.4 seconds to 17.3 seconds. It can be observed that the proposed solution has a more favorable time overhead. The main time overhead of the proposed solution lies in storing the index table on the blockchain. Since each index entry is $(w : \mathrm{IPFSaddress}[f_1, f_2, ...f_n])$, that is a keyword corresponds to a document set and the document set is stored as an IPFS hash value, each index item has a fixed size of 256*2+ 8. The total number of stored indexes is equal to the number of keywords. Therefore, the time overhead for index construction mainly depends on the number of keywords. On the other hand, the dual-index scheme stores a verification table on the blockchain, which grows linearly with the number of files and has a size of 256*n where n is the number of files. Additionally, the dual-index scheme adopts an index format where each keyword corresponds to a file. The number of indexes and keywords is related

to the number of files. Therefore, the dual-index scheme incurs more time overhead in index construction and storage.

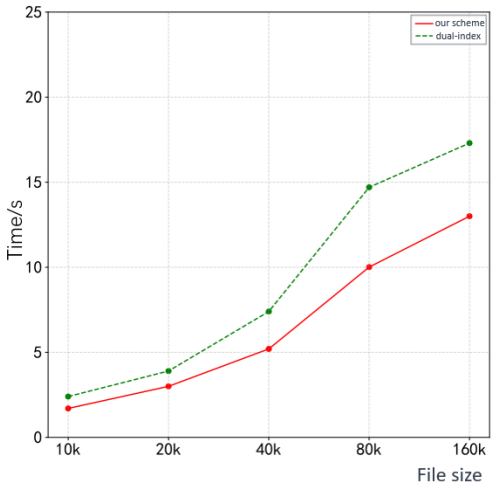

Fig. 1. Comparison Chart of Index Construction Time Overhead

Keyword queries were tested, as shown in Fig.2. The proposed approach was tested under two different conditions: one without adding any files, where a search keyword corresponds to only one index item, and another condition with added data. From Fig. 2, it can be observed that for queries without adding any files or updating the index, the time overhead remains consistent at around 0.07 seconds. This is because the indexing storage format in the proposed approach is (w : IPFSaddress[f1, f2, ...fn]), so querying the index table once provides all the results. However, when files are added, the query time increases as multiple index items need to be searched and the corresponding document set's IPFS hash value needs to be updated, merging multiple index items into one. Therefore, the query time increases based on the number and size of files added. The query data in the graph, ranging from 2k to 32k, corresponds to cases where files were added 1 to 5 times, and the query time increases from 0.12 seconds to 0.45 seconds.

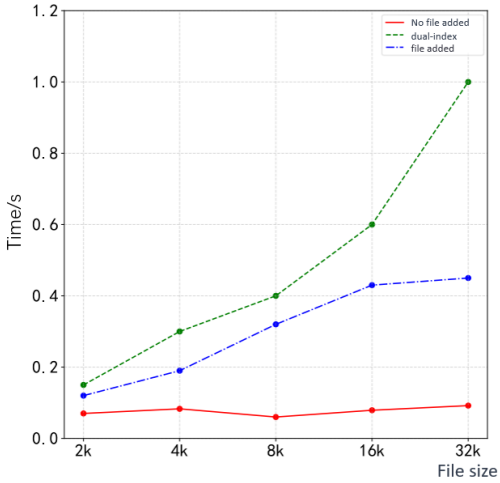

Fig. 2. Comparison Chart of Keyword Query Time Overhead

In contrast, the dual-index scheme requires multiple queries during the search since it stores data in the format (w : f), retrieving all files corresponding to the keyword. Thus, the query time increases with the size of the file set. Additionally, the dual-index scheme requires data verification, and the size of the verification table is also related to the file set size. In our proposed approach, verification is not needed as the IPFS hash value itself provides verification functionality.

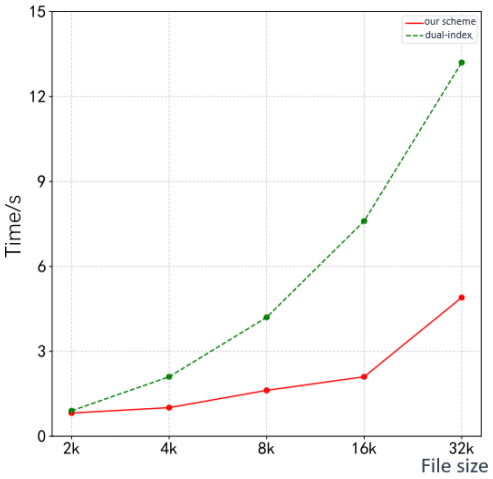

Fig. 3. Comparison Chart of Insert File Time Overhead

In the experiment, the dynamic update algorithm was tested for adding and deleting files. As shown in Fig. 3, our proposed solution has a time cost ranging from 0.82 seconds to 4.9 seconds when adding files from 2K to 32K. In comparison, the dual-index scheme has a time cost ranging from 0.9 seconds to 13.2 seconds. It can be observed that our solution outperforms the dual-index scheme in terms of

performance. This is because in our proposed solution, when adding new files, they are added in the form of (w : IPFSaddress[f1, f2, ...fn]), while the dual-index scheme adds files in the form of (w : f). Consequently, our solution has a larger number of additional index entries, resulting in relatively higher time costs. This difference in time costs becomes more apparent, especially when adding larger files.

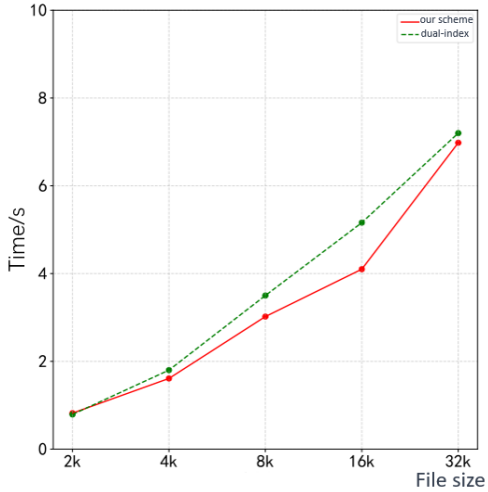

Fig. 4. Comparison Chart of File Remove Time Overhead

As shown in Fig. 4, the experiment also evaluated the performance of file deletion. The size of the deleted files ranged from 2K to 32K, and the time overhead increased from 0.82 seconds to 6.98 seconds. When deleting files, the main operation is to modify the index table on the blockchain. The deletion operation requires modifying each index entry associated with the file. Similarly, the dual-index scheme was tested, and its time overhead was similar to the proposed scheme, increasing from 0.79 seconds to 7.2 seconds. The dual-index scheme also modifies each associated index entry for the file and requires additional calculations when searching for the next index entry, resulting in slightly more time. Since both schemes only mark the files for deletion without actually deleting them, the overall time overhead is relatively small.

Through experimental comparative analysis, it can be seen that our proposed scheme has more advantages, especially in the query phase and file addition phase, where it demonstrates higher efficiency.

5. Conclusion

This paper analyzes blockchain-based searchable encryption schemes and addresses the issue of low efficiency in blockchain queries. To tackle this problem, we propose a dynamic searchable encryption scheme based on blockchain.

This scheme improves upon the existing algorithm based on a dual-index structure, improving the storage and retrieval efficiency of index key-value pairs. Leveraging the self-verification feature of IPFS in the algorithm enhances the verification efficiency of data stored on the blockchain. Improving the mapping relationship between keywords and files in index pairs enhances the multi-file hit rate in searches. Furthermore, by linking the initially added index items to the head of the index chain, forward security is ensured.

We provide a security analysis and proof for the proposed scheme, verifying its security guarantees. Finally, through a comparative experimental study, we compare our scheme with the dual-index scheme, validating the feasibility and advantages of our approach. Overall, this paper presents a viable and advantageous solution for blockchain-based searchable encryption. Our scheme enhances query efficiency and provides secure data storage and efficient secure queries. It demonstrates practical potential in protecting sensitive information on the blockchain.

Acknowledgment

This work is funded by the Science and Technology Program of Guangzhou, China (202102080644).

REFERENCES

[1] D. Sikeridis, A. Bidram, M. Devetsikiotis, and M. J. Reno, "A blockchain-based mechanism for secure data exchange in smart grid protection systems," in 2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC). IEEE, 2020, pp. 1–6.

[2] V. Dehalwar, M. L. Kolhe, S. Deoli, and M. K. Jhariya, "Blockchainbased trust management and authentication of devices in smart grid," Cleaner Engineering and Technology, vol. 8, p. 100481, 2022.

[3] A. Mosteiro-Sanchez, M. Barcelo, J. Astorga, and A. Urbieta, "End to end secure data exchange in value chains with dynamic policy updates," arXiv preprint arXiv:2201.06335, 2022.

[4] Z. A. Hussien, H. A. Abdulmalik, M. A. Hussain, V. O. Nyangaresi, J. Ma, Z. A. Abduljabbar, and I. Q. Abduljaleel, "Lightweight integrity preserving scheme for secure data exchange in cloud-based iot systems," Applied Sciences, vol. 13, no. 2, p. 691, 2023.

[5] Y. S. Kıyak, A. Poor, Işıl İrem Budakoğlu, Özlem Coşkun, "Holochain: A novel technology without scalability bottlenecks of blockchain for secure data exchange in health professions education," Discover Education, vol. 1, no. 1, p. 13, 2022.

[6] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 792–800.

[7] C. Ge, W. Susilo, J. Baek, Z. Liu, J. Xia, and L. Fang, "Revocable attribute-based encryption with data integrity in clouds," IEEE Transactions on Dependable and Secure Computing, vol. 19, no. 5, pp. 2864–2872, 2022.

[8] H. Li, F. Zhang, J. He, and H. Tian, "A searchable symmetric encryption scheme using blockchain," CoRR, 2017. [Online]. Available: http://arxiv.org/abs/1711.01030

[9] H. Li, H. Zhou, H. Huang, and X. Jia, "Verifiable encrypted search with forward secure updates for blockchain-based system," in Wireless Algorithms, Systems, and Applications, D. Yu, F. Dressler, and J. Yu, Eds. Cham: Springer International Publishing, 2020, pp. 206–217.
[10] Y. Zhao, Y. Yu, Y. Li, G. Han, and X. Du, "Machine learning based privacy-preserving fair data trading in big data market," Information Sciences, vol. 478, pp. 449–460, 2019.
[11] W. Q. Qiao R, Cao Y, "Traceability mechanism of dynamic data in internet of things based on consortium blockchain," Journal of Software, vol. 30, no. 6, p. 16141631, 2019.
[12] Z. Wang, Y. Tian, and J. Zhu, "Data sharing and tracing scheme based on blockchain," in 2018 8th international conference on logistics, Informatics and Service Sciences (LISS). IEEE, 2018, pp. 1–6.
[13] S. Delgado-Segura, C. Pérez-Solà, G. Navarro-Arribas, and J. Herrera-Joancomartí, "A fair protocol for data trading based on bitcoin transactions," Future Generation Computer Systems, vol. 107, pp. 832–840, 2020.
[14] Y. Chen, J. Guo, C. Li, and W. Ren, "Fade: a blockchain-based fair data exchange scheme for big data sharing," Future Internet, vol. 11, no. 11, p. 225, 2019.
[15] C.-H. Liao, X.-Q. Guan, J.-H. Cheng, and S.-M. Yuan, "Blockchainbased identity management and access control framework for open banking ecosystem," Future Generation Computer Systems, vol. 135, pp. 450–466, 2022.
[16] S. Alshehri, O. Bamasaq, D. Alghazzawi, and A. Jamjoom, "Dynamic secure access control and data sharing through trusted delegation and revocation in a blockchain-enabled cloud-iot environment," IEEE Internet of Things Journal, vol. 10, no. 5, pp. 4239–4256, 2022.
[17] H. Yin, J. Zhang, Y. Xiong, L. Ou, F. Li, S. Liao, and K. Li, "Cp-abse: A ciphertext-policy attribute-based searchable encryption scheme," IEEE Access, vol. 7, pp. 5682–5694, 2019.
[18] M. B. Hinojosa-Cabello, M. Morales-Sandoval, and H. M. Marin-Castro, "Novel constructions for ciphertext-policy attribute-based searchable encryption," in 2022 IEEE Mexican International Conference on Computer Science (ENC). IEEE, 2022, pp. 1–8.
[19] Y. Wang, S.-F. Sun, J. Wang, J. K. Liu, and X. Chen, "Achieving searchable encryption scheme with search pattern hidden," IEEE Transactions on Services Computing, vol. 15, no. 2, pp. 1012–1025, 2020.
[20] J. Su, L. Zhang, and Y. Mu, "Ba-rmkabse: blockchain-aided ranked multi-keyword attribute-based searchable encryption with hiding policy for smart health system," Future Generation Computer Systems, vol. 132, pp. 299–309, 2022.