

BISE: Enhance data sharing security through consortium blockchain and IPFS

Mingxuan Chen^a, Puhe Hao^a, Weizhi Meng^b, Yasen Aizezi^c and Guozi Sun^{a,*}

^aCollege of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, 210003, China

^bSchool of Computing and Communications, Lancaster University, UK

^cDepartment of Information Security Engineering, Xinjiang Police College, Urumqi, 830013, China

ARTICLE INFO

Keywords:

Blockchain

Searchable Encryption

IPFS

ABSTRACT

Data sharing is pivotal in sectors such as healthcare, finance, and social networking. Encrypting sensitive data, while essential for privacy protection, introduces complexity to data sharing and poses privacy risks when leveraging cloud servers. Blockchain-based searchable encryption offers a balance between privacy preservation and data availability; however, user anonymity remains a significant concern. Traditional storage systems, which rely on centralized servers, limit data stability and scalability. To address these challenges, we have introduced BISE, a solution that leverages the power of blockchain to achieve data integrity, using searchable encryption for secure searches and IPFS for decentralized storage. Constructed on Hyperledger Fabric and IPFS, our system demonstrates efficiency through simulations. This integrated approach ensures data privacy, integrity, and availability, with efficient updates and queries, making it a robust solution for sensitive data sharing in various domains.


1. Introduction

In the contemporary digital landscape, data sharing has emerged as a critical requirement across diverse domains, including healthcare, supply chain management, and the Internet of Things (IoT). For example, healthcare institutions frequently exchange patient medical records to deliver comprehensive care, while supply chain enterprises collaborate by sharing logistics and inventory data. Similarly, IoT devices continuously exchange data to enable seamless connectivity and functionality. These scenarios involve sensitive information such as personal health data, proprietary business policies, and user behavior patterns, which require strong privacy protection mechanisms. Existing methods mostly use traditional centralized storage systems with searchable encryption and cloud infrastructure.

Searchable Encryption (SE) technology allows for searching on encrypted data, protecting privacy while not sacrificing search efficiency. It prevents servers from stealing data and allows users to securely search for keywords in encrypted data. Although SE balances privacy and availability, it still faces challenges. In existing solutions, cloud servers are considered “honest but curious” (Guo et al., 2022). They will not tamper with data or attack uploaded data, but may attempt to pry into user data. This poses a risk for traditional searchable encryption schemes when facing malicious servers, as they may provide incomplete search results or users may raise objections to the correct results. Cloud computing allows more people to store data in the cloud, reducing local management pressure. But this way, users cannot directly control the data and may be illegally accessed. So, sensitive data such as financial and medical records are often encrypted. However, encrypted data is not easy to search, and users have to download all files and decrypt them before they can search. This is not practical in large amounts of data, as it consumes both resources and bandwidth.

Most solutions use blockchain to achieve reliable SE, utilizing its transparency and immutability to ensure open and trustworthy data sharing. The decentralized nature of blockchain solves the problems of traditional SE, supporting SE to execute in a decentralized, transparent, and tamper proof manner, and ensuring the correctness of retrieval results through the assistance of smart contracts. However, these solutions have limitations in terms of data sharing. SE in a single user environment (Chen et al., 2023) focuses on personal privacy and does not support multi-party data sharing. In multi-user schemes, precise search schemes (Gao et al., 2022) cannot tolerate spelling errors, and users often search for keywords that do not match perfectly. Due to storage structure limitations, the solution may not be able

*Corresponding author:

 sun@njupt.edu.cn (G. Sun)

ORCID(s):

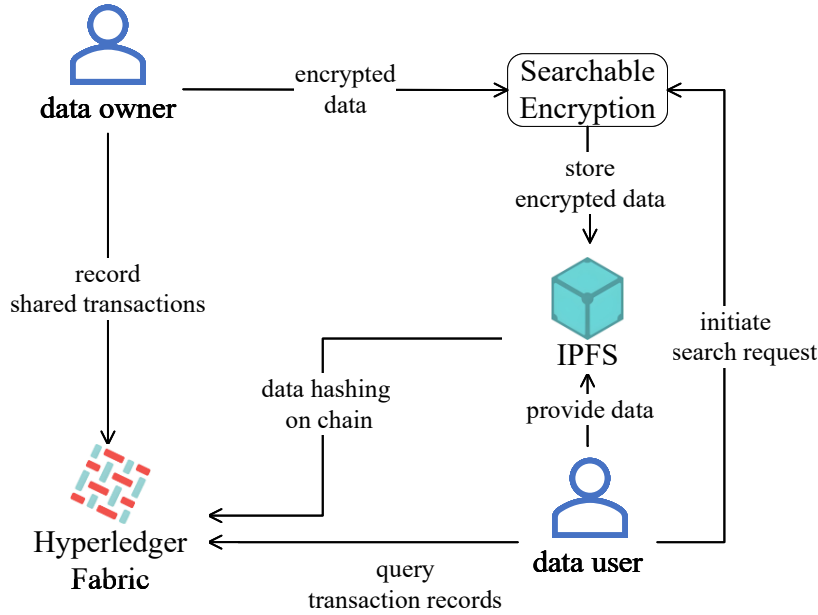


Figure 1: Proposed BISE system architecture

to dynamically update data (Chakraborty et al., 2022). Updating data requires retrieving it from the server, updating it locally, and then uploading it, which is inefficient. Most solutions store data locally or on cloud servers, and server failures may affect data stability.

To address these challenges, we propose BISE (Blockchain-based Indexed Searchable Encryption with IPFS), a novel framework that integrates blockchain technology, searchable encryption, and the InterPlanetary File System (IPFS), as shown in Figure 1 below. This integrated approach leverages the unique advantages of each technology to provide a comprehensive solution for secure and efficient data sharing. Blockchain technology, particularly through Hyperledger Fabric, offers a decentralized and transparent ledger for recording data-sharing transactions. Searchable encryption enables users to perform keyword searches on encrypted data without revealing the content or the nature of the queries. IPFS provides a decentralized storage solution that enhances data availability and resistance to censorship. Together, these technologies form a synergistic framework that ensures data privacy, integrity, and availability.

The key contributions of our proposed scheme are as follows:

- 1) To tackle cloud server trust issues in encrypted data sharing, we've proposed a scheme that combines blockchain with searchable encryption. This approach maintains data privacy and availability while ensuring the sharing process is transparent and trustworthy, thanks to blockchain's openness and immutability.
- 2) To fill the gap in blockchain-based searchable encryption for multi-user, multi-keyword fuzzy search, and dynamic updates, we've integrated blockchain with the trigram algorithm and quotient filter for index construction, allowing for these functionalities in a shared environment.
- 3) We've adopted IPFS over cloud servers for decentralized, tamper-proof encrypted data storage to enhance our data sharing scheme's efficiency and robustness. Utilizing Hyperledger Fabric and IPFS, along with searchable and ring signature technologies, we've developed a decentralized data sharing system. Functional analysis and testing confirm the system's rich features and strong performance.

The subsequent structure of this paper is as follows. Section 2 reviews the pertinent literature on blockchain, searchable encryption, and ring signatures. Section 3 outlines our proposed system model. Section 4 delves into the complexity of our system design, integrating searchable encryption with a data management framework based on Fabric and IPFS. Section 5 details the performance evaluation of BISE. Finally, Section 6 summarizes our work.

2. Background and Related Work

2.1. Blockchain

Blockchain is a trustworthy digital ledger that chains information blocks in time order using advanced cryptography, making past records impossible to change. With the widespread adoption of Bitcoin (Nakamoto, 2008; Andrychowicz et al., 2016), blockchain is hailed as a disruptive innovation in the computing paradigm (Yuan and Wang, 2018). Blockchain has become a new paradigm for secure and shareable computing that supports business innovation (Zhao et al., 2016). Blockchain has recently gained significant popularity, mainly due to its distributed nature. In order to expand the potential uses of blockchain technology, Buterin (Buterin, 2014) proposed the next generation of smart contracts and decentralized application platform Ethereum. Bonneau et al. (Bonneau et al., 2015) first systematically elucidated Bitcoin and many related cryptocurrencies. Andrychowicz et al. (Andrychowicz et al., 2016) proposed a secure multi-party lottery protocol based on the Bitcoin blockchain, in which the Bitcoin based timed commitment scheme is the main component. In order to achieve more universal computation, Andrychowicz et al. (Andrychowicz et al., 2014) proposed a two party computation protocol that is incompatible with the Bitcoin blockchain because it modifies the Bitcoin specification to resist scalability attacks. Bentov et al. independently proposed a similar idea (Bentov and Kumaresan, 2014).

2.2. Searchable encryption

In order to achieve secure keyword search on encrypted data, Song et al. (Song et al., 2000) first proposed the concept of searchable encryption and demonstrated a standard scenario for searchable encryption. Due to the use of symmetric encryption to construct indexes, their scheme is also known as Symmetric Searchable Encryption (SSE). Later, Goh et al. (Goh, 2003) and Chang et al. (Chang and Mitzenmacher, 2005) designed improved index structures to address the shortcomings of Song et al.'s scheme. Curtmola et al. (Curtmola et al., 2006) defined the concept of secure search for symmetric searchable encryption and proposed new structures for non adaptive and adaptive keyword selection attacks. Chase et al. (Chase and Kamara, 2010) extended SSE by introducing the concept of structured encryption and proposed an efficient scheme with adaptive security (i.e., sublinear time).

The searchable encryption scheme that supports update operations is called a dynamic searchable encryption scheme. Kamara et al. (Kamara et al., 2011) first proposed the concept of dynamic searchable encryption schemes. Their follow-up work (Kamara and Papamanthou, 2013) supports parallel search based on red black trees, while leaking less information. In order to improve I/O efficiency, Cash et al. (Cash et al., 2013) designed a dynamic scheme optimized for large datasets. However, this solution requires the server to maintain an additional undo list for deleted files, which makes search efficiency very low. For the DSSE scheme, attackers can use file injection attacks (Zhang et al., 2016) to inject some files into a dataset that has created keywords, and obtain the privacy of query keywords from the search results of keyword tokens.

In response to this attack, Stefanov et al. (Stefanov et al., 2013) first proposed the concept of forward security, which requires newly added files to not match previous query results. Bost et al. (Bost et al., 2016) formally defined this concept and used perforated encryption to efficiently achieve forward security. Demertzis et al. (Demertzis et al., 2019) proposed three new methods to alleviate client storage burden and achieve better performance in terms of forward and backward privacy. However, none of the above schemes have validated the accuracy of the search results. Bost et al. (Bost et al., 2016) proposed a scheme that can resist malicious servers and verify the integrity of the results. Later, Zhu et al. (Zhu et al., 2018) proposed the first verifiable SSE scheme, utilizing Merkle Precia trees and incremental hashing to construct proof indexes that support data updates. Miao et al. (Miao et al., 2020) designed a verifiable SE scheme that can resist Key Guessing Attack (KGA) while supporting verifiable searchability.

3. Proposed scheme

3.1. System architecture

A blockchain based searchable encrypted file sharing system called BISE has been proposed for secure data sharing. Files are stored in IPFS, and the system uses dynamic searchable encryption for personal spaces and multi-keyword fuzzy retrieval for sharing. The BSSShare (Blockchain-based Searchable Symmetric Encryption Share System) system integrates searchable encryption, IPFS, Fabric blockchain, and RBAC (Role-Based Access Control), as depicted in Figure 2.

In the system architecture, the Web Layer acts as the presentation layer. It uses Vue, Webpack, HTML, CSS, JQuery, and Bootstrap to form the Client Interface. Its main role is to display data to users. It also submits user -

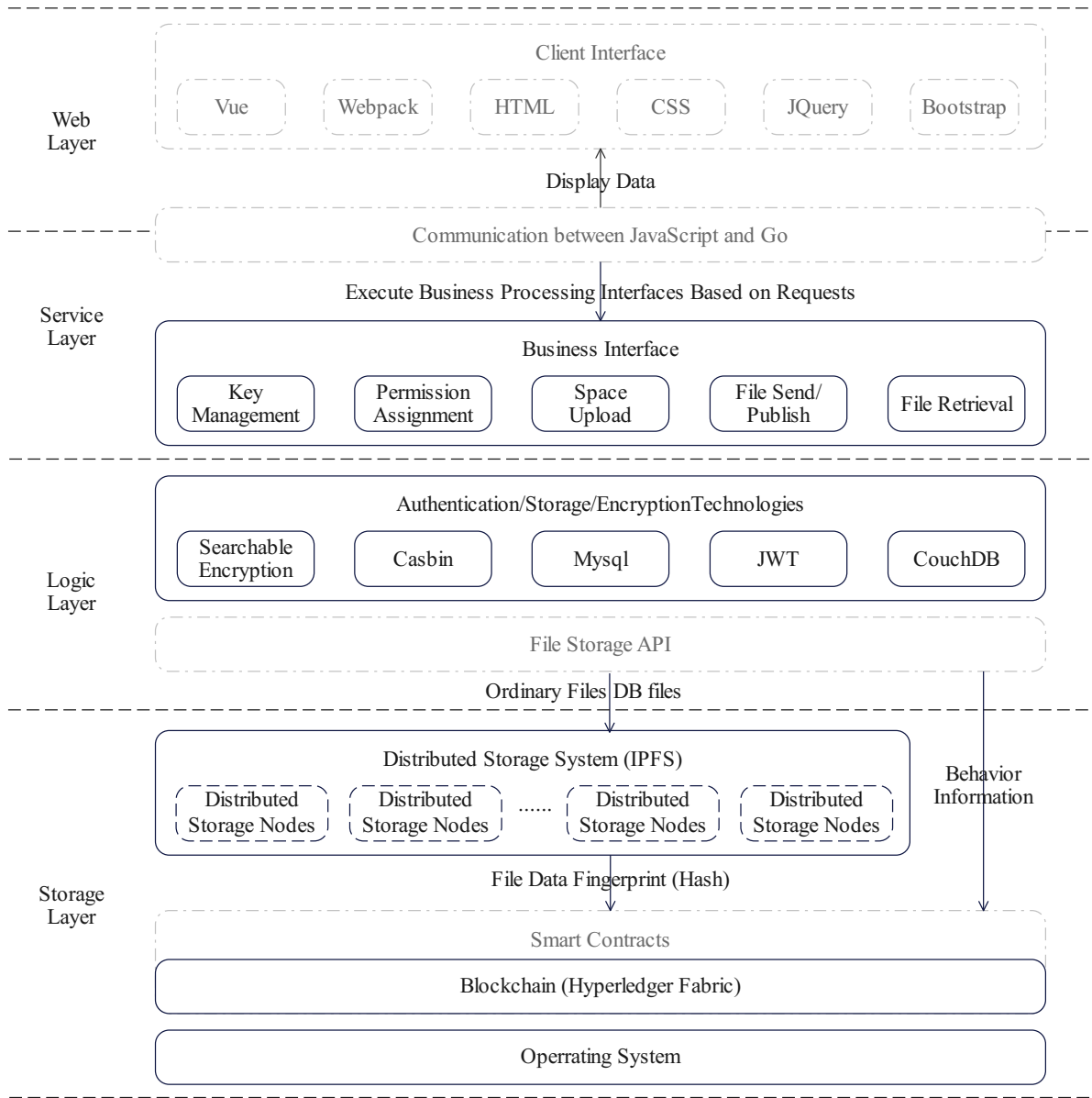


Figure 2: Proposed BISE system architecture

related operations to the Service Layer for processing via the communication between JavaScript and Go. The Service Layer is the business layer. After receiving requests from the Web Layer, it executes corresponding business processing interfaces based on the requests. These interfaces include Key Management, Permission Assignment, Space Upload, File Send/Publish, and File Retrieval. It interacts with the Web Layer during this process. The Logic Layer focuses on data operations and technical support. It covers Authentication/Storage/Encryption Technologies. Key technologies here are Searchable Encryption, Casbin, Mysql, JWT, and CouchDB. It provides a File Storage API to handle Ordinary Files and DB files. It interacts with the Storage Layer. The Storage Layer includes a Distributed Storage System (IPFS) and Blockchain (Hyperledger Fabric). IPFS, made up of multiple Distributed Storage Nodes, is used to store file data. Files are uniquely identified by their File Data Fingerprint (Hash). IPFS has strong fault tolerance, resistance to DoS attacks, and scalability. Hyperledger Fabric, with the support of Smart Contracts, stores Behavior Information and other relevant data. The entire Storage Layer runs on the Operating System.

3.2. Searchable encryption model

Using IPFS instead of cloud servers, the system stores encrypted data across a distributed network involving four main entities: Data Owner (DO), Blockchain System (BS), Data User (DU), and IPFS. Their roles are as follows:

The data owner is responsible for building a secure index. They hold datasets, extract keywords for each data and store them in indexes. Create encrypted indexes for each keyword in the data using Bloom filters. The data is encrypted and stored in IPFS, while the encrypted index is uploaded to the blockchain system to ensure the credibility of the search. At the same time, data owners and authorized data users share symmetric keys.

After being authenticated by the data owner, data users need to use shared keys and specific algorithms (triplet model+Bloom filter) to generate keyword trapdoors and send them to the blockchain system. After receiving search results containing data identifiers, users request files from IPFS through the data identifiers, and IPFS returns encrypted datasets. The blockchain system (consortium chain) stores encrypted indexes and matches search trapdoors with smart contracts. After successfully matching the node traversal index, consensus is reached and a list of data identifiers is sent back to the data user. IPFS stores encrypted data and big data is sharded for storage. Search for data based on the request identifier, retrieve and send encrypted data from each node to the user.

4. System design

4.1. System Requirements

The BISE framework's system requirements aim to ensure efficient, secure, and anonymous data sharing. On the hardware side, the server needs a high - performance CPU, such as a multi - core processor with a clock speed of at least 3.0 GHz, to handle complex cryptographic operations and concurrent data processing. It also requires sufficient RAM, with a minimum of 32 GB, to meet the large - scale data processing demands of blockchain and IPFS networks. For storage, high - capacity and high - speed SSDs with a capacity of at least 1 TB are necessary to store the blockchain ledger, encrypted indexes, and other related data. Moreover, the network environment should have high bandwidth and low latency to support efficient data transmission between nodes in the distributed network.

On the software side, the BISE framework is primarily built on Hyperledger Fabric and IPFS. Hyperledger Fabric provides a solid framework for constructing permissioned blockchain networks, while IPFS offers a decentralized storage solution to enhance data availability and censorship resistance. Go and Python are mainly used for developing smart contracts and application logic. For data encryption and decryption, the AES algorithm combined with salt values is adopted, and the SHA - 256 algorithm is used for data hashing. The Casbin framework is employed to implement fine - grained permission control based on the RBAC model. Other key software components include the Linux operating system, MySQL and CouchDB for data storage and management, and the Gin framework for building web service interfaces. These software requirements ensure the efficient and secure operation of the BISE framework. The following sections will detail the system design based on these requirements.

4.2. Multi person data sharing algorithm

This section details the core multi-person data sharing algorithm of the BISE framework, a critical component for enabling secure and efficient collaborative data access in decentralized environments. The algorithm integrates three key technical modules: symmetric encryption with salt values, trigram-based keyword processing, and quotient filter-based index construction, forming a closed-loop workflow for data encryption, index generation, and search trapdoor creation.

The algorithm first encrypts original files using the key K_s and salt to ensure encryption uniqueness and resist rainbow table attacks. It then decomposes file keywords into character triplets via the trigram algorithm, converts these triplets into compact numerical forms through ASCII encoding and bit-shift operations, and maps them to the quotient filter to construct encrypted indexes. Meanwhile, a trapdoor generation mechanism compatible with the index construction logic is designed to enable authorized data users to generate valid search trapdoors. The following will detail the pseudocodes of the index construction and trapdoor generation processes, which collectively ensure the security, efficiency, and fuzzy search capability of multi-person data sharing in the BISE system.

The pseudocode of the index construction process is shown in Algorithm 1, which encrypts each file f_i through AES using K_s and $salt$. Salt value refers to a randomly generated data fragment, typically used to enhance the security of cryptographic hashing. Using both keys and salt values to encrypt files simultaneously can improve the security and resistance of encryption algorithms compared to traditional methods that only use keys to encrypt files. It can also effectively prevent rainbow table attacks, increase resistance to dictionary attacks, improve entropy, and ensure that

the encryption result of each file is unique. By combining the use of keys and salt values, the security of files can be better protected. Afterwards, this article uses the trigram algorithm and quotient filter to generate indexes for files.

Algorithm 1 BuildIndex

Require: index I ; initial file collection F ; key K_s ; salt $salt$.

Ensure: encrypted index B .

```

1: for  $i = 1$  to  $n$  do
2:    $es_i \leftarrow \text{encrypt}(salt_i, K_{s_i})$ 
3:    $f_i \leftarrow f_i || es_i$ 
4:   for  $w \in W$  do
5:      $trigramIndex \leftarrow \text{strToTrigram}(w)$ 
6:     for  $t \in trigramIndex$  do
7:        $flag \leftarrow \text{hasElement}(t, b)$ 
8:       if  $flag == true$  then
9:         continue
10:      end if
11:       $hash\ k \leftarrow h_k(t)$ 
12:       $b[hash\ k] \leftarrow k$ 
13:    end for
14:  end for
15:   $B_i \leftarrow (f_i, b_i)$ 
16: end for
17: return  $B$ 

```

Trigrams is an algorithm based on statistical language models, using n-gram models where n is 3. In natural language processing, trigrams refer to a sequence of three consecutive words or characters. Trigrams processing a sentence means dividing it into adjacent combinations of three words or characters. The difference between the trigram algorithm used in this chapter and traditional trigram algorithms is that after simple disassembly, each part is converted into ASCII uint32 and stored through displacement. This is because if the disassembled characters are only saved as indexes, the subsequent retrieval process will be more time-consuming than the process string, and storing them as text will also occupy a larger storage space.

For a keyword “Query”, the algorithm first breaks it down into three parts: “Que”, “uer”, and “ery”, and then converts each part into ASCII uint32 and stores it through displacement. For example, Que is converted to (81117101) through the method of (uint32 (s[0]), uint32 (s[1]), uint32 (s[2])), and stored in decimal form through displacement, for example, $81 \ll 16 + 117 \ll 8 + 101 = 5338469$. When inserting a decimal key x , k is first hashed to obtain k , then $k \% \beta$ is used as the quotient (X_{kq}), $k \% \beta$ is used as the remainder X_{kr} , and finally X_{kq} is used as the stored index, X_{kr} is used as the corresponding slot to obtain the secure index QF_i of the corresponding file f_i , and finally the quotient filter QF_i is obtained, where the cloud server (corresponding to IPFS) cannot obtain K_s .

This Algorithm 2 takes a set of keywords W as input, and the data owner uses Algorithm to generate trapdoors for each keyword. The index generation algorithm first constructs a triplet for each keyword, and each item in the triplet is stored in decimal format. The corresponding hash value digest is calculated using K_p and the decimal number of the triplet. Unused space is filled with pseudo values, and the quotient filter combines index values and pseudo values to achieve randomness. The last cell is used to store the storage location of the last index of the hash value. Finally, the summary value is inserted into the trapdoor set, and the data user sends the trapdoor to IPFS through the blockchain.

Algorithm 2 TrapdoorGen**Require:** keyword collection W .**Ensure:** trapdoor T .

```

1:  $st \leftarrow \text{size}(T)$ 
2: for  $w \in W$  do
3:    $\text{trigramIndex} \leftarrow \text{strToTrigram}(w)$ 
4:   for  $t \in \text{trigramIndex}$  do
5:      $\text{flag} \leftarrow \text{hasElement}(t, b)$ 
6:     if  $\text{flag} == \text{true}$  then
7:       continue
8:     end if
9:      $\text{hash}_k \leftarrow h_k(t)$ 
10:     $\text{pos}[\text{hash}_k] \leftarrow k$ 
11:   end for
12: end for
13:  $\text{index} \leftarrow \text{getSetPosition}(\text{pos})$ 
14:  $T_{st} \leftarrow |\text{index}|$ 
15:  $T \leftarrow \text{randomize}(\text{index})$ 
16: return  $T$ 

```

The pseudocode of the keyword search process is shown in Algorithm 3. The smart contract will search the blockchain based on the uploaded matching items and provide a link to appropriately encrypted data. Users can decrypt the data using the corresponding key to obtain plaintext results that meet the search criteria. Once the blockchain system receives the trapdoor T , it triggers the search contract algorithm and compares T with QF . Since L defines the number of input errors or format inconsistencies allowed to be relaxed for searching for corresponding files containing keywords, we say that if the number of matches is greater than or equal to $3L$, the encrypted file identifier f_i will be stored in the result set R , otherwise no operation will be performed. Subsequently, the final result R will be transmitted to the data user.

Algorithm 3 Search Keyword**Require:** trapdoor T ; encrypted index B **Ensure:** results R

```

1: for  $i = 1$  to  $n$  do
2:    $f \leftarrow \text{compare}(B_i[b], T)$ 
3:   if  $f \neq \perp$  then
4:      $R \leftarrow R \parallel f$ 
5:   end if
6: end for
7:  $R \leftarrow R \parallel f_1 \parallel \dots \parallel f_q$ 
8:  $\text{salt} \leftarrow \text{decrypt}(\text{encryptedSalt}, K_s)$ 
9:  $\hat{R} \leftarrow \text{encrypt}(R, K_s) \oplus \text{salt}$ 
10:  $\text{found} \leftarrow \text{search}(\hat{R}, C)$ 
11: if  $\text{found} \neq \perp$  then
12:   return  $\hat{C} \leftarrow C_i$ 
13: else
14: end if
15: return  $\hat{C}$ 

```

After receiving the result set R from the blockchain, the data user forwards it to any node in the IPFS network. In order to decrypt data files, nodes first need to determine which nodes store the corresponding data blocks. Through IPFS's distributed hash table DHT, these nodes can be identified to contain the required data blocks. Afterwards, establishing connections with such storage nodes becomes crucial. In the IPFS environment, the BitSwap block

exchange protocol supports connections between nodes to request data from other nodes. This node can send the result R to these connected nodes and immediately obtain the required data blocks. Practice has proven that IPFS has high throughput, enabling data owners to efficiently obtain the required data files.

4.3. Fine grained permission control module

This article adopts the RBAC model of the Casbin framework to achieve fine-grained permission control. Casbin effectively manages user, role, and permission relationships by defining policies and models, enhancing the security of resource access. The permission control process after user operation request is shown in Figure 3. Using Casbin can conveniently manage access rules, enhance application security and data protection.

When users request operations, the system verifies their identity. For sensitive actions like login or deletion, incorrect verification attempts over three times lead to a lockout. Upon successful verification, Casbin checks user permissions across system initialization and request processing stages. If functional and data permissions are confirmed, the operation proceeds.

Initialization involves loading a user-defined model with access requests, policies, matchers, and effectors. Casbin's Adapter fetches policies from various data sources. Requests are processed and matched against policies, allowing for both exact and fuzzy matches. If a request aligns with multiple policies, the outcomes are combined to decide on permission.

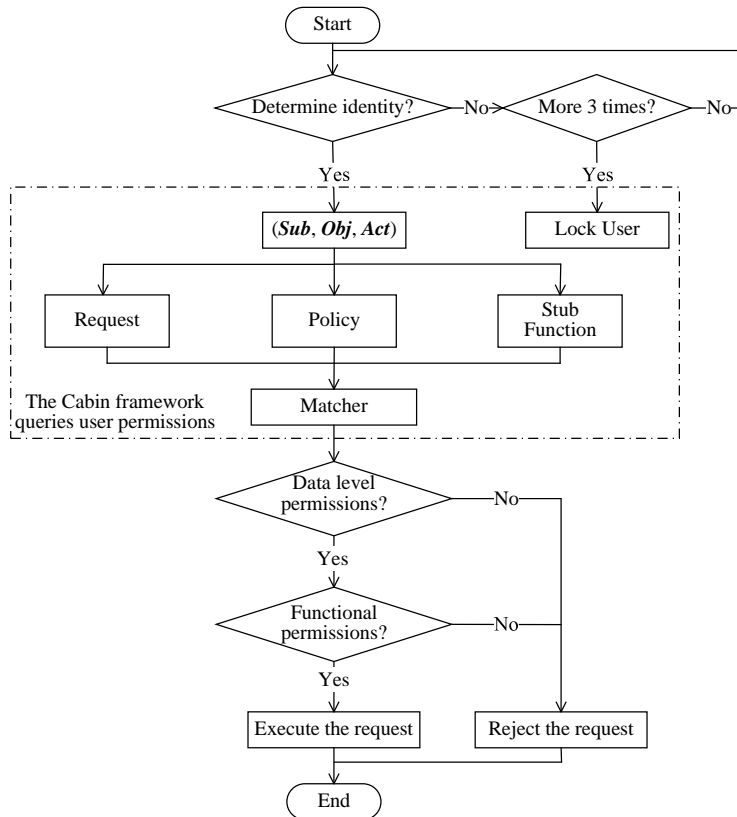


Figure 3: Role based permission management process

4.4. Data management process based on Fabric and IPFS

The system uses role-based management. New users get a regular role, and their SHA-256 encrypted passwords are on the blockchain. Logins are authenticated via hash values. It includes a data sharing module for encrypted content in IPFS and indexes on the blockchain. Data can be shared post-verification, and users can search and decrypt files using keywords.

4.4.1. User management module

Figure 4 illustrates the process of adding users in the system, using a role-based permission management model and the Casbin framework. When users register, they are assigned as regular user roles by default, and subsequent role assignments are controlled by administrators. After the user sets the password, the system encrypts it using the SHA-256 algorithm and stores the hash value on the blockchain. SHA-256 has irreversibility, collision resistance, and high security, protecting user password confidentiality, providing data integrity and security, and ensuring that user information is not easily stolen or tampered with.

Upon logging in, users create verification credentials with their password, then fetch their hash from CouchDB on the blockchain for validation. Data retrieval uses the Composer rest server from Hyperledger Composer, which swiftly generates REST APIs for blockchain interaction, allowing users to manage blockchain assets, participants, and transactions easily.

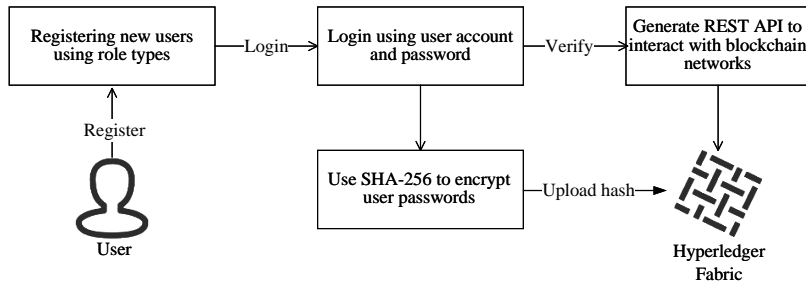


Figure 4: Process of user registration and login

4.4.2. Data sharing module

The system allows department admins to share data via IPFS and enable fuzzy keyword searches for users. The system leverages blockchain's features like identity verification, immutability, and traceability to support governance off the chain. The alliance chain data has the characteristics of identity recognition, tamper resistance and non repudiation, providing a data foundation and execution credentials for off chain governance. The data sharing process is shown in Figure 5, which includes administrator upload sharing (a) and regular user retrieval and download (b).

Department managers release data after role verification; non-managers need admin approval. The admin then selects sharees and grants keys. Data is encrypted and uploaded to IPFS, while indexes are made with trigrams and Bloom filters and stored on the blockchain. Users retrieve data through keywords, and the system generates trapdoors and sends them to the blockchain. Blockchain compares trapdoors with stored indexes, and if the matching degree exceeds 90%, data identifiers are added to the result set. After completion, it will be returned to the user, who will then download and decrypt the data from IPFS.

5. Theoretical analysis and simulation

The performance testing experiments in this section were conducted in an experimental environment consisting of six machines, with four machines serving as IPFS and Fabric servers, and the other two machines acting as file uploaders and file downloaders respectively. The server machine ran Ubuntu 20.04 system, while the client machine ran Windows 10 system. The server is equipped with Intel Xeon E-2225G processor, and the client machines are equipped with AMD Ryzen 7 5800H with Radeon Graphics processor. All machines are equipped with SSD solid-state drives. Specifically, the four servers are Lenovo models, each with 16GB memory and 1TB SSD; the two clients are Lenovo Xiaoxin models, each configured with 16GB memory and 512GB SSD. Meanwhile, the experiment relies on Golang 1.18.1, Fabric 2.0 and IPFS 0.4.23 to form a complete testing environment.

The performance testing part of the system is mainly divided into three main aspects: searchable encryption methods, blockchain fabric, and interstellar file system IPFS. Regarding the testing of searchable encryption methods, we mainly tested the performance of the designed scheme in three processes: index generation, trapdoor generation, and search keyword generation, and compared it with related schemes. The selection of the experimental dataset was evaluated using the email dataset (Klimt and Yang, 2004). This raw dataset covers 619446 emails from 158 users.

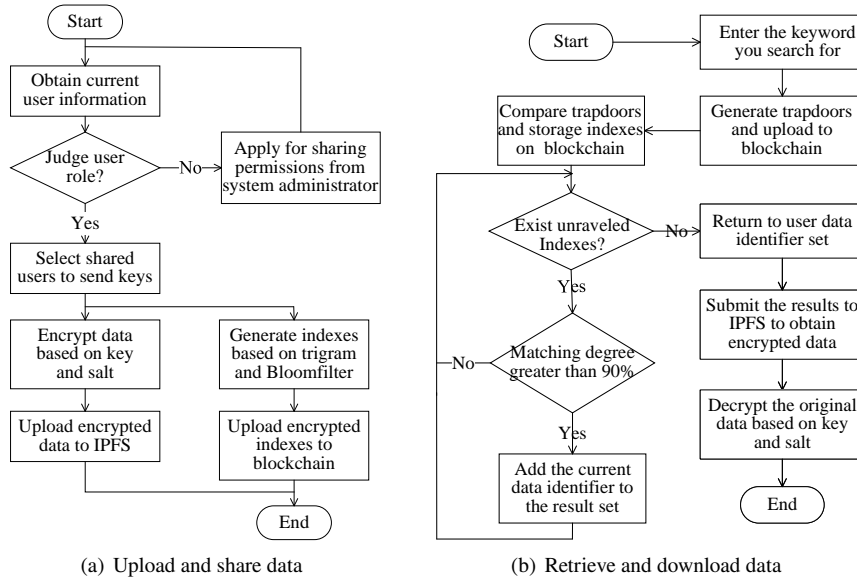


Figure 5: Data sharing process based on searchable encryption

Table 1
Scheme comparison

Scheme	Multi-user	Blockchain	Multiple keyword	Fuzzy retrieval	Dynamic update	IPFS storage	Identity hiding
BSMFS (Chakraborty et al., 2022)	✓	✓	✓	✓	×	×	×
BPMS (Bernabe et al., 2019)	✓	✓	✓	×	×	✓	×
SEMFS (Nayak and Tripathy, 2017)	✓	×	✓	✓	×	×	×
BVDSSE (Guo et al., 2022)	×	✓	×	×	✓	×	×
MKSSMDO (Yin et al., 2019)	✓	×	✓	×	×	×	×
CBDSS (Guan et al., 2020)	✓	✓	×	×	✓	×	✓
BC-RFMS (Zheng et al., 2024)	×	✓	✓	✓	×	✓	×
FMVDSSE (Xu et al., 2025)	✓	×	×	×	✓	×	×
BHSSE (Guan et al., 2025)	×	✓	×	×	✓	×	×
Our scheme	✓	✓	✓	✓	✓	✓	✓

1000 data files were randomly selected as the experimental dataset, and keywords were extracted using Scikit learn (Pedregosa et al., 2011) tool, which will be inserted into the dictionary D of each data file. To conduct performance analysis on the searchable encryption methods in this section, the number of keywords in the data file is set to 10 to 80 and 100 to 800, respectively. All experiments were conducted using Golang on computers running Ubuntu 20.04 desktop system.

To assess IPFS and Fabric performance in a cluster, this article sets up the system across four local network hosts, each running Fabric and IPFS. Fabric has 5 Order nodes (distributed 2-1-1-1 across hosts) and 4 peer nodes split into 2 orgs. IPFS has 4 nodes. MinIO (a high-performance distributed object storage system compatible with Amazon S3 API) is also set up for performance comparison with IPFS. The blockchain test uses Hyperledger Fabric v1.4.3 with Raft consensus, and Caliper measures blockchain performance.

5.1. Functional analysis

This section first compares the designed multi keyword fuzzy searchable scheme with other related schemes (Chakraborty et al., 2022; Bernabe et al., 2019; Nayak and Tripathy, 2017; Guo et al., 2022; Yin et al., 2019; Guan et al., 2020; Zheng et al., 2024; Xu et al., 2025; Guan et al., 2025), mainly comparing whether it supports the following content attributes: blockchain, multi-user environment, multi keyword retrieval, fuzzy retrieval, dynamic updates, IPFS storage, and identity hiding, as shown in Table 1:

In a rigorous comparative analysis of existing solutions, a clear observation emerged that the BVDSSE scheme has unique shortcomings in terms of multi-user search capabilities. This drawback is fundamentally attributed to its underlying symmetric searchable encryption. Although it ensures operational efficiency and forward privacy through

blockchain mechanisms, it hinders its applicability in environments that require concurrent user participation. In sharp contrast, most other solutions, except for the obvious omissions of SEMFS, MKSSMDO and FMVDSSE, use blockchain to enhance data integrity and security.

BSMFS and SEMFS are distinguished by their sophisticated search functionalities, particularly their support for multi-keyword and fuzzy search capabilities. These attributes are of paramount importance as they transcend the traditional boundaries of exact match queries, enabling a more exhaustive data retrieval process and enriching the user experience by accommodating semantic variations in search terms. The dynamic update feature, which facilitates the modification of encrypted data without the prerequisite of re-encryption, is a significant attribute supported by SEMFS, BVDSSSE, FMVDSSE and CBDSS. This feature is imperative for systems that require agility in data management, streamlining the data updating process and, consequently, optimizing the efficiency of storage and retrieval operations. Notably, BHSSE also partially supports dynamic updates—while it enables the secure addition of new encrypted data entries to the existing system by generating incremental indexes and synchronizing verification credentials, it lacks mechanisms for the deletion or modification of already stored encrypted data, thus realizing only a limited scope of dynamic operation capabilities. In the realm of storage efficiency, BPMS and BC-RFMS are distinguished by its adoption of IPFS, which is designed to construct a lightweight and efficient blockchain framework. This approach is particularly significant as it mitigates the scalability challenges commonly associated with conventional blockchain architectures, making it more suitable for applications that necessitate high throughput and low latency. Privacy preservation is a critical concern in the current data-centric paradigm, and both CBDSS and our proposed solution address this by incorporating identity hiding features. These schemes integrate consortium chains with searchable encryption, providing an additional layer of security that safeguards user identities and ensures data confidentiality, which is essential for maintaining user trust and adherence to privacy regulations.

In summation, our scheme excels in several critical domains, including search functionality, dynamic data management, storage efficiency, and privacy protection. By integrating these advanced features, our solution not only enhances the performance of data retrieval systems but also bolsters their resilience against potential security threats, positioning it as a robust solution for the complex data management challenges of the modern era.

5.2. Searchable encryption test

We compared the time cost of the searchable encryption part of the designed system BISE with BSMFS (Chakraborty et al., 2022) and MKSSMDO (Yin et al., 2019) in index construction algorithms through experiments. Figure 6 shows the time cost (in seconds) of generating indexes for three different schemes as the number of keywords in the data file changes. When the number of keywords in the data file varies between 10 and 80, the time cost of the designed scheme in the index construction process shows a linear increase trend. This is because each keyword is decomposed into multiple triplets and added to the index. Therefore, as the number of keywords increases, the number of generated triples also increases linearly, thereby increasing the complexity and size of the index. In this case, our solution requires much less time. This is because triplet indexing has certain advantages in efficiency when the number of keywords is low. However, as the number of keywords increases, building and maintaining triplet indexes typically requires more computing resources and storage space, which may lead to a decrease in efficiency for large-scale datasets. In contrast, the time cost of MKSSMDO does not vary with the number of keywords, as although more encryption operations are required, the operation of generating padding index elements in bilinear mapping pairs is reduced.

Next, we compared the time cost of the proposed scheme with BSMFS and MKSSMDO in the trapdoor generation algorithm. Figure 7 shows the time cost (in seconds) required to generate trapdoors for three different schemes as the number of keywords in the data file changes. All three schemes will show a linear growth trend as the number of search keywords (within the range of 10 to 80) increases. It is worth mentioning that the implementation of MKSSMDO only considers the time cost of exponentiation in bilinear mappings, while ignoring the time cost of hash operations for each keyword. Therefore, the actual time cost of this scheme may be higher. BSMFS is a scheme based on Bloom filters, and the efficiency of the trapdoor generation algorithm depends on the number of keywords and the size of the Bloom filter. Therefore, the symmetric searchable encryption scheme implemented with the help of commercial filters is slightly superior to the other two schemes.

Finally, we compared the time cost of our method with BSMFS and MKSSMDO in search algorithms. Figure 8 shows the time cost (in milliseconds) of searching keywords for three different schemes as the number of data files changes. As shown in the figure, the time cost of the three schemes is linearly related to the number of files. Due to the fact that the merchant filter only requires one hash function, the scheme is significantly better in search time cost than

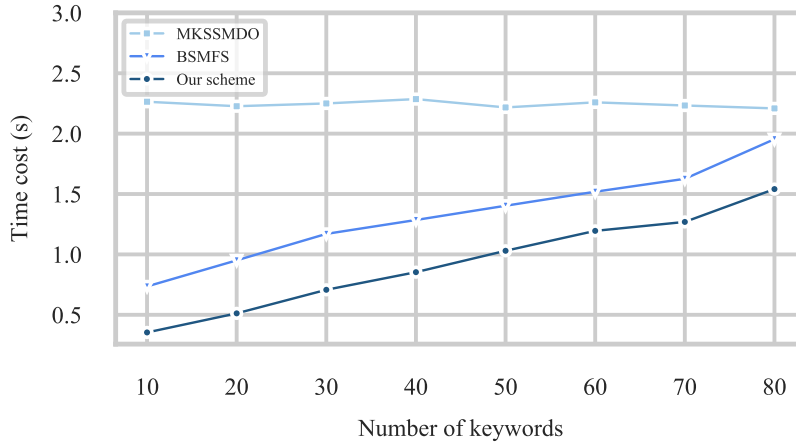


Figure 6: Comparison about index generation

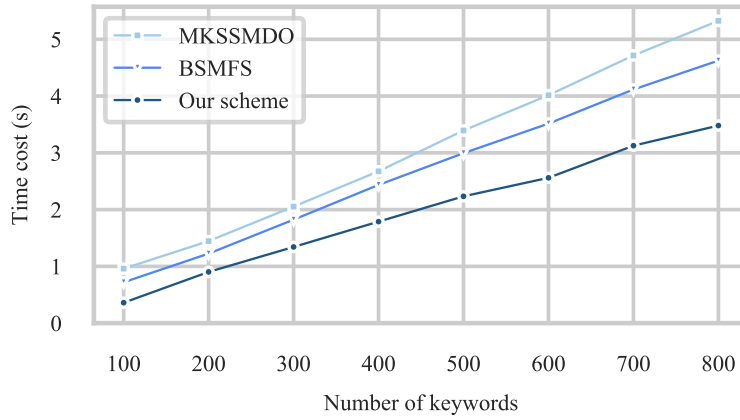


Figure 7: Comparison about trapdoor generation

BSMFS using Bloom filter. The MKSSMDO scheme is a public key searchable encryption scheme, and theoretically, the time cost should be significantly higher than our scheme. However, the actual results are not much different. It is possible that the scheme introduces blockchain, which requires sending search requests to the blockchain during the search process. Although this process increases additional time overhead, the scheme in this chapter can therefore support trusted search.

5.3. Blockchain test

In order to study the impact of different transaction rates on the performance of blockchain networks, this paper uses Hyperledger Caliper to conduct performance tests on blockchain, and then records and analyzes the final performance of the blockchain platform based on two performance indicators: throughput (defined as successful transactions per second, tps); Delay (defined as the response time for each transaction, in seconds). Figure 9 shows the average throughput (in tps) and average latency (in seconds) at different transaction rates, with transaction rates of 100, 200, 300, 400, and 500 tps respectively. The total number of transactions is fixed at 1000 tps. For storage enabled transactions (with each transaction performing one read and one write), the blockchain network can only handle a maximum of 300tps without any significant network latency. When the transaction speed increases to over 300tps, the load on the fabric reaches its limit, and transactions enter the queue waiting. The blockchain throughput decreases with the increase of transaction speed, and the delay increases significantly.

Moreover, as the transaction speed increases, the growth rate of average delay gradually increases. This means that this specific testing environment can handle up to 300tps of storage transaction type functionality without any

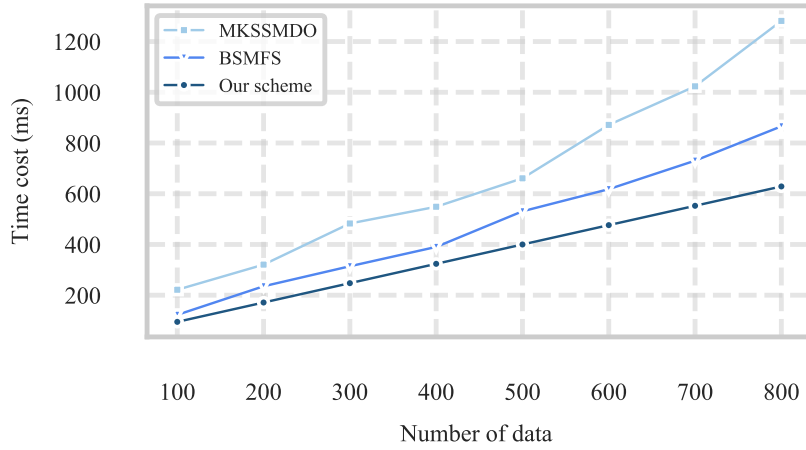


Figure 8: Comparison about search performance

significant network latency. For query type transactions (where only one read is performed in a transaction), blockchain networks can obviously handle 500tps with almost no latency, with an average latency of less than 0.1s. This means that the testing environment has not reached its maximum limit and can support higher transaction rates.

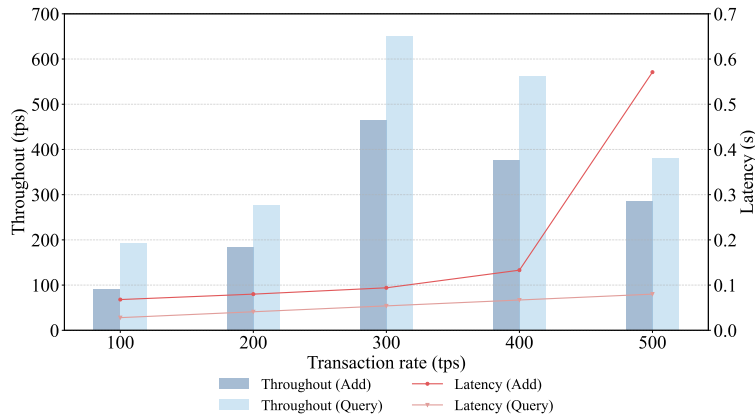


Figure 9: Comparison of average throughput and average latency

5.4. IPFS test

Figure 10 compares the upload and download time(in seconds) of IPFS and MinIO at different file sizes (in MB). Each experiment was tested 10 times and the average was taken as the result. It should be noted that due to IPFS's deduplication feature, files with the same content have the same hash value. This means that no matter how many times the same file is uploaded, only one copy will actually be stored. This deduplication feature will result in faster uploading of the same file in the future. Therefore, although the files used for testing in the experiment have the same size, their contents are as different as possible. As shown in the figure, when the file size is less than 100MB, IPFS outperforms MinIO in terms of upload and download performance. As the file size gradually increases, the performance advantage of MinIO gradually becomes apparent. Although IPFS takes a long time to process large files, its response time varies greatly between each operation, resulting in a high standard deviation in its average performance. In addition, when handling small file requests, the fluctuation range of IPFS is relatively small, but when executing large file requests, the fluctuation range of IPFS is significantly higher than MinIO. Overall, IPFS is more suitable for handling small file requests, while Minio is more suitable for handling large file requests. This may be because MinIO directly stores raw files on the hard drive, which is not user-friendly for storing large amounts of small files. In addition, the performance

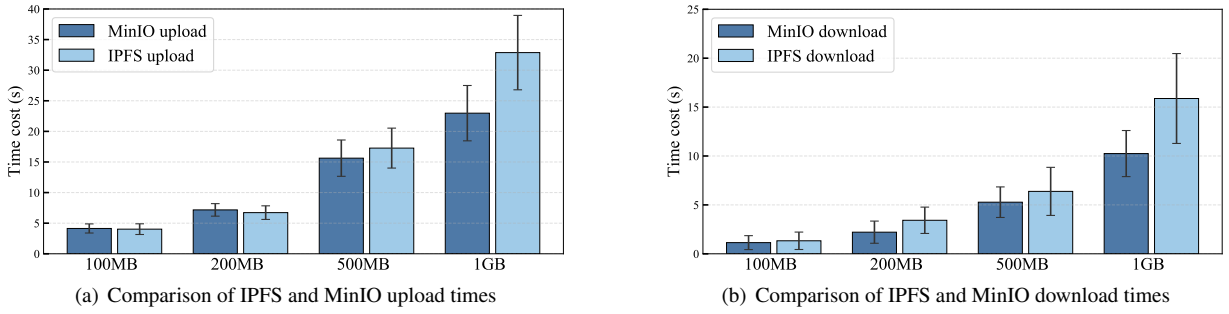


Figure 10: Comparison of IPFS and MinIO Performance

of IPFS depends on the number of available distributed nodes in the network. More nodes can improve the reliability and accessibility of files, while also helping to accelerate file transfer speed.

In addition to the file upload and download time (in seconds) of IPFS, the BSEShare system designed in this article also involves time-consuming processes such as encryption, decryption, and smart contract calling in actual file upload and download functions. Therefore, this section analyzed the upload and download performance of the system on files of different sizes (in MB). As shown in Figure 11, the time taken to write and read files of different sizes to the server was compared using the BSEShare client. These times cover the entire file operation process, including steps such as encrypting files using searchable encryption during file reading and writing, storing and downloading files, uploading and querying relevant information. Observing the entire process, it can be observed that the time required for file upload and download generally increases linearly with file size. Overall, the download time of files is significantly higher than the upload time, and the gap between the two becomes more significant as the file size increases. Part of the reason for this phenomenon may be that IPFS performs sharding when processing large files, and on the other hand, it is because the file upload process requires uploading index and other information to the blockchain, which increases the generation of new blocks and generates corresponding time consumption.

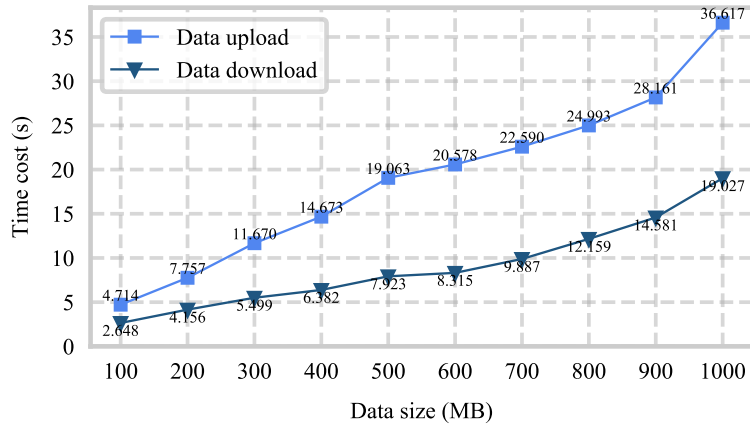


Figure 11: BSEShare file upload and download time

The file download time of IPFS is much shorter than the file upload time, because IPFS uses content addressing to search and access data through the hash value of files. When downloading a file, simply knowing the hash value of the file allows you to directly retrieve the file content from the nearest node, avoiding the time consumption of searching and addressing, and therefore the download speed may be faster.

6. Conclusion

In recent years, electronic data storage and circulation have been replacing traditional methods. Our decentralized BISE solution combines blockchain, searchable encryption, traceable ring signatures, and IPFS, balancing cost, efficiency, and privacy security. In this scheme, verification information is stored on the blockchain, data is shared in IPFS, and searchable encryption protects data privacy and security. The experiment shows that BISE has rich functions and good performance.

Based on blockchain, searchable encryption and IPFS, BISE tackles complex secure and efficient data sharing challenges, ensuring data privacy, integrity and availability. However, BISE has drawbacks. First, the advent of quantum computing poses a critical threat to conventional cryptographic algorithms employed in BISE for data encryption and privacy preservation. Future iterations must integrate quantum-resistant cryptographic algorithms to mitigate this vulnerability. Second, the scalability of the BISE model encounters significant challenges under high-load scenarios. As user access volumes and data processing demands escalate, the system experiences pronounced degradation in response latency and substantial increases in inter-node communication delays, leading to severe performance bottlenecks. Subsequent research should explore distributed load-balancing strategies and adaptive resource scheduling mechanisms to enhance the framework's processing capacity in concurrent environments.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 62472234, No. 62372245), the Natural Science Foundation of Xinjiang Uygur Autonomous Region (No. 2024D01A55).

Appendix : Key Symbols and Acronyms

Symbol	Description
BISE	Blockchain-based Indexed Searchable Encryption with IPFS
DO	Data Owner
DU	Data User
BS	Blockchain System
IPFS	InterPlanetary File System
K_s	Symmetric Key
$salt$	Random Salt Value
W	Keyword Set
B	Encrypted Index
QF_i	Quotient Filter for File f_i
T	Search Trapdoor
f_i	i -th Original File
f'_i	Encrypted i -th File
R	Result Set
$h_k(t)$	Keyword Hash Function
$strToTrigram(\omega)$	Trigram Conversion Function
tps	Transactions Per Second

References

- Andrychowicz, M., Dziembowski, S., Malinowski, D., et al., 2014. Fair two-party computations via bitcoin deposits, in: Financial Cryptography and Data Security: FC 2014 Workshops, BITCOIN and WAHC 2014, Springer Berlin Heidelberg. pp. 105–121.
- Andrychowicz, M., Dziembowski, S., Malinowski, D., et al., 2016. Secure multiparty computations on bitcoin. Communications of the ACM 59, 76–84.
- Bentov, I., Kumaresan, R., 2014. How to use bitcoin to design fair protocols, in: Annual Cryptology Conference, Springer Berlin Heidelberg. pp. 421–439.
- Bernabe, J.B., Canovas, J.L., Hernandez-Ramos, J.L., Moreno, R.T., Skarmeta, A., 2019. Privacy-Preserving Solutions for Blockchain: Review and Challenges. IEEE Access 7, 164908–164940.
- Bonneau, J., Miller, A., Clark, J., et al., 2015. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies, in: 2015 IEEE Symposium on Security and Privacy, IEEE. pp. 104–121.
- Bost, R., Fouque, P.A., Pointcheval, D., 2016. Verifiable dynamic symmetric searchable encryption: Optimality and forward security. Cryptology ePrint Archive .
- Buterin, V., 2014. A next-generation smart contract and decentralized application platform. White Paper 3, 2–1.
- Cash, D., Jarecki, S., Jutla, C., et al., 2013. Highly-scalable searchable symmetric encryption with support for boolean queries, in: Advances in Cryptology-CRYPTO 2013, Springer Berlin Heidelberg. pp. 353–373.
- Chakraborty, P.S., Chandrawanshi, M.S., Kumar, P., Tripathy, S., 2022. BSMFS: Blockchain assisted Secure Multi-keyword Fuzzy Search over Encrypted Data, in: 2022 IEEE International Conference on Blockchain (Blockchain), IEEE. pp. 216–221.

- Chang, Y.C., Mitzenmacher, M., 2005. Privacy preserving keyword searches on remote encrypted data, in: International Conference on Applied Cryptography and Network Security, Springer Berlin Heidelberg. pp. 442–455.
- Chase, M., Kamara, S., 2010. Structured encryption and controlled disclosure, in: Advances in Cryptology-ASIACRYPT 2010, Springer Berlin Heidelberg. pp. 577–594.
- Chen, B., Xiang, T., He, D., et al., 2023. BPVSE: Publicly verifiable searchable encryption for cloud-assisted electronic health records. *IEEE Transactions on Information Forensics and Security* 18, 3171–3184.
- Curtmola, R., Garay, J., Kamara, S., et al., 2006. Searchable symmetric encryption: improved definitions and efficient constructions, in: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 79–88.
- Demertzis, I., Chamani, J.G., Papadopoulos, D., 2019. Dynamic searchable encryption with small client storage. *Cryptology ePrint Archive*.
- Gao, S., Chen, Y., Zhu, J., et al., 2022. BPMS: Blockchain-based privacy-preserving multi-keyword search in multi-owner setting. *IEEE Transactions on Cloud Computing* 11, 2260–2272.
- Goh, E.J., 2003. Secure indexes. *Cryptology ePrint Archive*.
- Guan, S., Cao, Y., Zhang, Y., 2025. Blockchain-Enhanced Data Privacy Preservation and Secure Sharing Scheme for Healthcare IoT. *IEEE Internet of Things Journal* 12, 5600–5614.
- Guan, Z., Wang, N., Fan, X., Liu, X., Wu, L., Wan, S., 2020. Achieving Secure Search over Encrypted Data for e-Commerce: A Blockchain Approach. *ACM Transactions on Internet Technology* 21, 1–17.
- Guo, Y., Zhang, C., Wang, C., Jia, X., 2022. Towards public verifiable and forward-privacy encrypted search by using blockchain. *IEEE Transactions on Dependable and Secure Computing* 20, 2111–2126.
- Kamara, S., Papamanthou, C., 2013. Parallel and dynamic searchable symmetric encryption, in: Financial Cryptography and Data Security, Springer Berlin Heidelberg. pp. 258–274.
- Kamara, S., Papamanthou, C., Roeder, T., 2011. Cs2: A searchable cryptographic cloud storage system. Technical Report MSR-TR-2011-58. Microsoft Research.
- Klimt, B., Yang, Y., 2004. The enron corpus: A new dataset for email classification research, in: European Conference on Machine Learning, Springer Berlin Heidelberg. pp. 217–226.
- Miao, Y., Tong, Q., Deng, R.H., 2020. Verifiable searchable encryption framework against insider keyword-guessing attack in cloud storage. *IEEE Transactions on Cloud Computing* 10, 835–848.
- Nakamoto, S., 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*.
- Nayak, S.K., Tripathy, S., 2017. SEMFS: Secure and Efficient Multi-keyword Fuzzy Search for Cloud Storage, in: Shyamasundar, R., Singh, V., Vaidya, J. (Eds.), *Information Systems Security*, Springer.
- Pedregosa, F., Varoquaux, G., Gramfort, A., et al., 2011. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research* 12, 2825–2830.
- Song, D.X., Wagner, D., Perrig, A., 2000. Practical techniques for searches on encrypted data, in: Proceedings 2000 IEEE Symposium on Security and Privacy, S&P 2000, IEEE. pp. 44–55.
- Stefanov, E., Papamanthou, C., Shi, E., 2013. Practical dynamic searchable encryption with small leakage. *Cryptology ePrint Archive*.
- Xu, Z., Tian, C., Zhang, G., Tian, W., Han, L., 2025. Forward-Secure multi-user and verifiable dynamic searchable encryption scheme within a zero-trust environment. *Future Generation Computer Systems* 166, 107701.
- Yin, H., Qin, Z., Zhang, J., et al., 2019. Secure conjunctive multi-keyword ranked search over encrypted cloud data for multiple data owners. *Future Generation Computer Systems* 100, 689–700.
- Yuan, Y., Wang, F.Y., 2018. Blockchain and cryptocurrencies: Model, techniques, and applications. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 48, 1421–1428.
- Zhang, Y., Katz, J., Papamanthou, C., 2016. All your queries are belong to us: the power of file-injection attacks on searchable encryption, in: 25th USENIX Security Symposium (USENIX Security 16), pp. 707–720.
- Zhao, J.L., Fan, S., Yan, J., 2016. Overview of business innovations and research opportunities in blockchain and introduction to the special issue. *Financial Innovation* 2, 1–7.
- Zheng, L., Zhang, H., Ge, X., Lin, J., Kong, F., Yu, L., 2024. BC-RFMS: blockchain-based rankable fuzzy multi-keyword search scheme. *Blockchain* 2, 0004.
- Zhu, J., Li, Q., Wang, C., 2018. Enabling generic, verifiable, and secure data search in cloud services. *IEEE Transactions on Parallel and Distributed Systems* 29, 1721–1735.