Computers, Materials & Continua
DOI: 10.32604/cmc.2022.028284
Article

Tech Science Press

A Blockchain-Based Framework for Secure Storage and Sharing of Resumes

Huanrong Tang[1], Changlin Hu[1], Tianming Liu[2] and Jianquan Ouyang[1,*]

[1]Key Laboratory of Intelligent Computing and Information Processing, Ministry of Education, Computer Science College of Xiangtan University, Xiangtan, Hunan Province, China
[2]Department of Computer Science, The University of Georgia, Athens, Georgia, USA
*Corresponding Author: Jianquan Ouyang. Email: oyjq@xtu.edu.cn
Received: 06 February 2022; Accepted: 10 March 2022

Abstract: In response to problems in the centralized storage of personal resumes on third-party recruitment platforms, such as inadequate privacy protection, inability of individuals to accurately authorize downloads, and inability to determine who downloaded the resume and when, this study proposes a blockchain-based framework for secure storage and sharing of resumes. Users can employ an authorized access mechanism to protect their privacy rights. The proposed framework uses smart contracts, interplanetary file system, symmetric encryption, and digital signatures to protect, verify, and share resumes. Encryption keys are split and stored in multiple depositories through secret-sharing technology to improve the security of these keys. Corresponding key escrow incentives are implemented using smart contracts to automatically verify the correctness of keys and encourage the active participation of honest key escrow parties. This framework combines blockchain and searchable symmetric encryption technology to realize multi-keyword search using inverted indexing and Bloom filters and verify search results on the chain. Escrow search service fees are charged through contracts. Only after the search results are verified can the search service provider obtain the search fee, thus ensuring fair and efficient search for encrypted resumes. The framework is decentralized, secure, and tamper-evident, and achieves controlled sharing while protecting personal privacy and information security.

Keywords: Resume; blockchain; interplanetary file system; secret sharing; searchable encryption

1 Introduction

In the recruitment market, a personal resume is an essential document that highlights the qualifications of an individual for a job. The resume states the person's educational background and work experience, as well as contact information and even identification numbers.

Resumes are generally publicly stored on a third-party recruitment cloud server, exposing the owner's private information. Even if some third-party recruitment platforms also provide an encryption function, the keys for encrypting and decrypting resumes are stored centrally on a third-party

server. When a third party maliciously views or leaks keys, the encryption security is put at risk. Also, the resume owner on the existing recruitment platform cannot control the authorization or know when the document has been downloaded.

Many recruitment platforms have leaked user resumes. An example in the United States is Ladders, one of the most popular recruitment websites specializing in high-paying job recruitment. In 2019, this company mistakenly made its database public without a password such that anyone could access the data, resulting in the disclosure of user information belonging to more than 13.7 million individuals.

According to the published judgment files on China Judgments Online, from 2017 to 2020, a total of 48 cases involving resume trading were convicted of infringing on citizens' private information. These cases involved websites such as 58.com, Ganji.com, and Zhilian.com. Criminal cases for the sale of resumes on third-party recruitment platforms are rampant, and the security of personal privacy information contained in resumes cannot be guaranteed.

On Zhilian.com, as long as a corporate account pays for membership, it has unlimited ability to download resumes containing personal information. Similar problems occur on 51job.com and Liepin.com. Corporate accounts only need to pay a fee to download the complete resumes of job applicants. Thus, the privacy of applicants is compromised without their knowledge.

With the current centralized storage and management of resumes on third-party recruitment platforms, resumes' security, authenticity, and controllability are seriously challenged. For example, when a resume is shared, there is no guarantee that the owner has control over the resume, and there is no real way to know when the resume is shared with whom. This research proposes a blockchain-based framework for securely storing and sharing resumes to address these issues.

The main contributions of this study are the following:

1) A blockchain-based resume-sharing framework is proposed, which leverages interplanetary file system (IPFS), smart contracts, and digital signature technology to store and share resumes securely. Only authorized recruiters have access to plaintext resumes. Authorization and other access services are done by blockchain accounts to ensure identity privacy protection.
2) Secret sharing is used to implement secure key escrow and design incentives for it. A fair and verifiable multiple-keyword searchable encryption algorithm is proposed to achieve fair and efficient search of encrypted resumes.
3) We implemented our scheme and performed security and performance analysis. The time for index generation, trapdoor generation, search, and verification, as well as the economic consumption of the contract function, have been experimentally simulated. Results show that the proposed scheme is feasible.

The rest of this paper is organized as follows. Section 2 describes the research background and related work. Section 3 outlines the system model and definition of each stage. Section 4 summarizes the detailed design of the framework. Section 5 introduces the design of smart contracts. Section 6 discusses security and performance. Section 7 provides the research conclusions and future development directions.

2 Background and Related Work

2.1 Blockchain

Blockchain is a concept proposed by Satoshi Nakamoto in the "Bitcoin White Paper" [1]. Ethereum, a representative of Blockchain 2.0, provides smart contracts that can be automatically

executed on the blockchain and has the characteristics of strong tamper-proof modification and a high degree of decentralization [2,3].

2.2 IPFS

The IPFS is a distributed storage system that uses distributed hash tables to solve the problem of data transmission and positioning, and changes the single point of transmission through a peer-to-peer (P2P) network. The IPFS makes data storage safer and more efficient for a longer period compared with the traditional system [4].

2.3 Secret Sharing

Secret sharing is a technology for sharing secrets among a group of participants. It is mainly used to protect critical information and prevent information from loss, damage, or tampering. Shamir [5] and Blakley [6] first proposed it in 1979. Asmuth et al. [7] proposed a (t, n) threshold scheme based on the Chinese remainder theorem in 1983. Using secret sharing technology to keep the keys can prevent confidential information from risk exposure and being extremely concentrated in authoritative third parties, thereby improving the stability and robustness of the keys.

2.4 Symmetric Searchable Encryption

Searchable encryption technology can efficiently retrieve ciphertext data without leaking private information. As symmetric searchable encryption only requires one symmetric key, the scheme has a simple structure, the encryption and decryption processes are fast, and the method is suitable for data encryption and retrieval. Construction methods of symmetric searchable encryption include SWP scheme proposed by Xiaoding et al. [8] and SSE-1 scheme proposed by Curtmola et al. [9]. Cash et al. [10] proposed symmetric searchable encryption that can be extended to massive databases. Existing symmetric searchable encryption schemes are expanded from multiple perspectives, including security [11,12], dynamic [13,14], and expandability [15–18].

2.5 Bloom Filter

The Bloom filter [19] is a long binary vector and a series of random mapping functions that can quickly determine whether an element belongs to a certain set or not. Its advantage is that the space efficiency and query time are much better than the general algorithm. Its disadvantage is its misidentification rate and deletion difficulty.

2.6 Related Work

In recent years, many researchers have explored the application of blockchain technology in data sharing. For example, Fan et al. [20] proposed an information management system called MedBlock using blockchain, access control protocols, and symmetric cryptography. This system enables authorized users to share sensitive medical information.

However, a centralized database management system controls a large amount of data and involves many hidden dangers, such as denial of service attacks and single points of failure. Tan et al. [21] used the IPFS to store archive data and realized the management, verification, and sharing of such data on the chain through a combination of public and alliance chains. Nizamuddin et al. [22] used the smart contract of the Ethereum blockchain to manage the sales of e-books and realized the transaction sharing of e-books but not the access control of e-book owners and other users who obtained e-books. Alzubi et al. [23], based on AG codes over Hermitian curve, design a new crypto system

for IoT devices to protect data security, and in [24] propose an IoT network combining blockchain and artificial intelligence-driven secure medical data transfer (BAISMDT) for data transfer security and privacy in IoT networks. Uddin et al. [25] proposed an electronic health record (EHR) system incorporating the Hyperledger Fabric blockchain for storing, sharing, and exchanging EHRs in a P2P network of healthcare stakeholders. Alnssayan et al. [26] proposed a VacChain system in conjunction with blockchain to track children's vaccination records, creating a convenient information-sharing solution without loss of integrity and privacy, as well as alleviating the current problems of human error, intentional tampering, and untraceable vaccination records for vaccination certificates.

The aforementioned data-sharing schemes for file encryption can ensure data confidentiality, but they do not consider the safe custody of content-encryption keys. Naz et al. [27] used secret sharing to split and encrypt the hash address of the file stored in the IPFS and store it in other nodes. However, once the hash address of the file is leaked, anyone can directly access the plaintext data through the address. Sohrabi et al. [28] improved the security of encrypted file keys for cloud data by using the Shamir secret sharing algorithm, which enables the owner to split the decryption key into n parts distributed to miner nodes on the blockchain for storage through a secure channel. However, this algorithm does not verify the correctness of the miner nodes' keys, nor does it provide escrow rewards.

Searchable encryption can effectively retrieve ciphertext data, so some scholars have incorporated this technology into effective search when sharing encrypted data in their proposed schemes. Nevertheless, the traditional searchable encryption technology cannot guarantee the fairness of retrieval. Zhang et al. [29] used a method that involved submitting a deposit in the blockchain to achieve fair payment of the handling fee, ensuring that participants can obtain correct search results and service fees when they execute honestly. However, numerous signature verifications are required during the verification process, and the overhead is large. Li et al. [30] proposed a searchable symmetric encryption scheme using the Bitcoin blockchain system to solve the problem of data search and the unfairness of traditional symmetric searchable encryption schemes. However, the scheme requires six transactions to obtain search results. The correctness of the search results verified through the Bitcoin script makes the transaction cycle long and inefficient. Chen et al. [31] proposed a searchable encryption scheme for EHRs based on blockchain, supporting logical expression queries and fair payment. This scheme uses smart contracts to replace cloud servers. However, its ciphertext database and security index are stored in smart contracts, which requires a large amount of gas. Feng et al. [32] used attribute encryption and searchable encryption methods to control private data on the blockchain and fast ciphertext search, which solved the privacy leakage problem. However, this scheme cannot achieve efficient multi-keyword search or fair payment.

Blockchain-based technology research has made great progress in digital rights, finance, and electronic medical record sharing. However, relatively little research has been done related to resumes. The above studies still have the problems of inability to securely escrow keys and inefficiency of searchable encryption schemes. This paper uses secret sharing and blockchain technology to enhance the security of keys. A fair searchable encryption scheme supporting multi-keyword search is implemented using Bloom filters to improve the searching efficiency of encrypted resumes.

3 System Framework/Stage Definition

3.1 System Framework

In this section, the overall design of our framework is presented. Tab. 1 summarizes the symbols and corresponding descriptions used.

Table 1: Symbols and descriptions

Symbol	Description
enc	Use AES algorithm for encryption
dec	Use AES algorithm for decryption
inv	Solve the inverse element
PRF	Pseudo-random function
H	Hash function
||	Concatenate string
BF	Bloom Filter

The framework consists of the following four entities:

Applicant (Ap): **Ap** is the resume owner who has the absolute right to hold the document. **Ap** encrypts the resume and, at the same time, uploads the search index corresponding to the encrypted resume to the cloud server. No one can view **Ap**'s resume unless authorized by him/her.

Recruiter (Rt): Users request to download complete resume information. After the request is authorized, the full key is synthesized through a smart contract. After **Rt** enters the address of the encrypted resume on the client and calls the contract to receive the decryption key, the complete plaintext resume can be obtained. At the same time, **Rt** can use the contract to generate a search token to search for resumes containing keywords.

Key keeper (Kk): **Kk** receives **Ap**'s subkey through a secure channel and can receive the custodian fee from **Ap**. **Kk** can obtain a portion of the money from **Rt** for downloading the resume by providing the subkey correctly through a smart contract and assisting in recovering the symmetric key.

Third-party recruitment platform (TRP): **TRP** and **Kk** operate the IPFS. **TRP** operates the blockchain platform, keeping the encrypted index uploaded by Ap and providing the search function. If **TRP** returns correct search results, then **TRP** receives a service fee, but a penalty is applied if **TRP** returns incorrect results. Fig. 1 shows the framework diagram of the safe storage and sharing system.

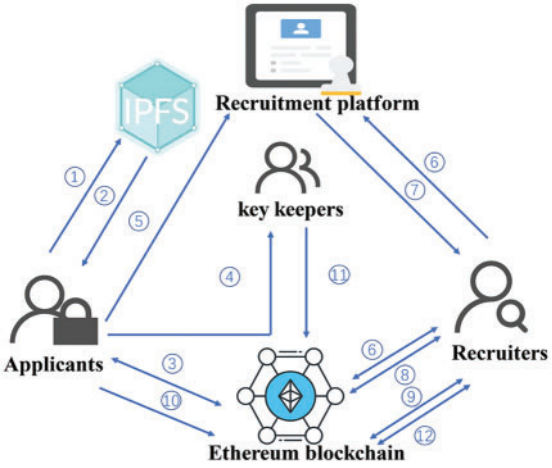

Figure 1: System framework

Each step in the figure is described as follows:

1) **Ap** enters his/her information, generates an encryption key in the local client, encrypts the complete resume containing personal private information, and then uploads the encrypted data to the IPFS.
2) The IPFS returns the hash address of the file.
3) **Ap** stores the hash address and the resume information on the blockchain.
4) **Ap** divides the encryption key into n subkeys via (t, n) secret sharing scheme and sends them to different **Kk** via secure channels.
5) **Ap** creates and uploads the search index related to the resume to **TRP**.
6) **Rt** selects the keywords, uploads to the blockchain to get the search token, and sends it to **TRP**.
7) **TRP** sends the search results to **Rt**.
8) **Rt** uploads the search results to the blockchain and obtains the verified search results.
9) **Rt** initiates the request and stores the request information on the blockchain.
10) **Ap** authorizes **Rt** after receiving the request and notifies **Kk**.
11) **Kk** enters the subkey into the key recovery function in the smart contract, and **Kk** is rewarded for correct subkey verification.
12) The contract recovers the original key and notifies **Rt** after receiving the correct subkey that meets the threshold. **Rt** empties the subkey stored on the contract after calling the contract method to obtain the original key.

Finally, **Rt** accesses the encrypted resume according to the hash address set of the searched resume and uses the recovered key for decryption to obtain a complete plaintext document.

3.2 Stage Definition

Based on the secret-sharing scheme by Asmuth [7] and the SSE scheme by Curtmola [9], a blockchain-based secure storage and sharing framework for personal resumes is constructed through seven stages, which can be defined as VSSE = (Gen, Enc, IndexGen, TokenGen, Search, Verify, Dec). Each stage is described in the following:

1) Key generation stage: $\mathrm{Gen}(1^{\lambda}) \rightarrow (\mathrm{K})$

Security parameter λ is entered and a set of keys $\mathrm{K} = (\mathrm{K1}, \mathrm{K2}, \mathrm{K3})$ is output. K1 is used as the key for encrypted files, K2 is used as the document verification set encryption key, and K3 is used as the key to generating the index.

2) Encryption stage: This stage is run by **Ap** and consists of the following two substages:

File encryption stage: $\mathrm{FileEncrpyt}(\mathrm{file}, \mathrm{K1}, \mathrm{K2}) \rightarrow (\mathrm{IpfsHash}, \mathrm{kw}, \mathrm{MAC_{kw}})$, where file and encryption key are input, and hash address set IpfsHash, keyword set kw, and verification set $\mathrm{MAC_{kw}}$ are output.

Key distribution stage: $\mathrm{Keydistrput}(\mathrm{k}) \rightarrow (\mathrm{k}_1, \mathrm{k}_2, \cdots, \mathrm{k}_n, \mathrm{MAC}_{ss}, \mathrm{r}, \mathrm{p})$, where the encryption key is input, and n subkeys, verification set MAC_{ss}, and random numbers r and p are output.

3) Index generation stage: $\mathrm{IndexGen}(\mathrm{kw}, \mathrm{K3}) \rightarrow \gamma_w, \tilde{\gamma}_w$, where the index generation stage takes a keyword set kw and encryption key as input. The encrypted keyword-file hash address as the key-value pair index table γ_w and the file hash address-Bloom filter as the key-value pair index table $\tilde{\gamma}_w$ are output.

4) Search token generation stage: $\mathrm{TokenGen}(\mathrm{K3}, \mathrm{w}) \rightarrow \tau_w$, where the search key and keywords are entered, and the token is output.

5) Search stage: Search $(\tau_w, \gamma_w, \tilde{\gamma}_w) \rightarrow I_w$. The search token τ_w and search index $\gamma_w, \tilde{\gamma}_w$ are input. A set of files containing keywords I_w is output.

6) Verification stage: Verify $(IpfsHash, K2, MAC_{kw}) \rightarrow IpfsHash_{true}/false$. The search result hash address set IpfsHash is returned by the server, and the key K2 and verification set MAC_{kw} are used as input. The correct output $ipfshash_{true}$ is verified; otherwise, the output is false.

7) Decryption stage: This stage consists of the following two sub-stages:

Key recovery stage: KeyRecover (addr, $u_1, u_2, \ldots, u_t$, MAC_{ss}, r, p) $\rightarrow$ k/*false*, where the account address on the chain, t subkeys, and subkey verification set cooperation MAC_{ss} are taken as input. The correct output file decryption key is verified; otherwise, the output is false.

Resume recovery stage: FileDecrypt (k, IpfsHash) $\rightarrow$ file, where the decryption key and hash address of the file stored on the IPFS are taken as input, and the output is a complete decrypted resume.

4 Specific Program

1) System initialization stage

Gen $(1^{\lambda}) \rightarrow$ (K): Random keys K1, K2, and K3 are generated using pseudo-random function (PRF), where K1 is used as the key for encrypting the file, K2 is used as the encryption key for the verification set, and K3 is used as the key for generating the index.

2) Encryption phase

Based on the assumption that **Ap** has several plaintext resume files $CV = \{CV_1, CV_2, \cdots, CV_n\}$, the following steps are performed for these files.

Step 1: File encryption, FileEncrpyt (file, K1, K2) $\rightarrow$ (IpfsHash, kw, MAC_{kw}). **Ap** encrypts $CV_i (i \in [1, n])$ and randomly generates the initial vector iv of corresponding bits. The encryption key is K1, and obtains $E_i = enc_{k1} (iv||CV_i) | i \in [1, n]$ and uploads E_i to IPFS, which returns the specific hash address of the file $IpfsHash_i = \{Upload(iv||E_i)|i \in [1, n]\}$.

Ap calls the Evidence() in the smart contract to record the hash address of the uploaded encrypted file and other related information. The verification key K2 is used to generate the verification set $MAC_{kwi} = \{PRF (K2, IpfsHash_i) | i \in [1, n]\}$. Then, the verification key is recorded along with a set of message verifications on the chain using addmac() in the verification contract.

Step 2: Key splitting, Keydistrput (k) $\rightarrow$ ($k_1, k_2, \cdots, k_n$, MAC_{ss}, r, p). **Ap** uses an Asmuth–Bloom secret-sharing scheme to split the file encryption key K1 into *n* copies and sends them to **Kk** through a secure channel. The key-splitting algorithm is shown in algorithm 1. Meanwhile, for each subkey, $MAC_{ssi} = \{H (i||x_i||m_i) | i \in [1, n]\}$ is computed, and **Ap** uploads the subkey verification set and r, p to the data-sharing contract storage.

Algorithm 1: Key Distribution

Input: K, t, n
Output: $k_1,\ k_2,\ \cdots,\ k_n,\ r,\ p$
01: $U \leftarrow []$
02: random choose prime number p, $p > K$
03: $m_1,\ m_2, \cdots,\ m_n \leftarrow$ getm()
04: $N \leftarrow m_1 m_2 \cdots m_t$
05: randomly choose r, $0 \le r \le N/p - 1$
06: $k' \leftarrow K + r \cdot p$
07: **for** $i \leftarrow 1\ \cdots\ n$ **do**
08: $\quad x_i \leftarrow k' \bmod m_i$
09: **end**
10: **for** $i \leftarrow 1\ \cdots\ n$ **do**
11: $\quad U[i] \leftarrow (i,\ x_i,\ m_i),\ k[i] \leftarrow U[i]$
12: **end**
13: **return** $k_1,\ k_2,\ \cdots,\ k_n,\ r,\ p$

3) Index generation stage

IndexGen (kw, K3) $\rightarrow \gamma_w,\ \tilde{\gamma}_w$, **Ap** extracts keywords from the CV set $CV = \{CV_1,\ CV_2, \cdots,\ CV_n\}$ to form a combination kw. For each keyword $w \in kw$, K3 is used to generate encrypted keywords $\tau_w = PRF(K3, 1||w)$, τ_w is used as key, file hash address is used as value to build index table γ_w, $\gamma_w[\tau_{wi}] = \{ipfshash_j,\ \cdots,\ ipfshash_k\}$, file hash address is used as key, the keyword $\{\tau_{wi},\ \cdots, \tau_{wj}\}$ contained in each file is added to the Bloom filter to obtain the binary vector $\{\cdots, 101, \cdots\}$, $\tilde{\gamma}_w[ipfshash_i] = BF(\{\tau_{wi},\ \cdots, \tau_{wj}\}) = \{\cdots, 101, \cdots\}$. Then, γ_w and $\tilde{\gamma}_w$ are sent to the server.

4) Search token generation stage

TokenGen (K3, w) $\rightarrow \tau_w$, where the search token is generated in the smart contract by TokenGen(). In **TRP**, qualified recruiters have the right to store the relevant service fees through the TokenGen() function after inputting keywords to calculate the search token $\tau_w = PRF(K3, 1||w)$.

5) Search stage

Search $(\tau_w,\ \gamma_w,\ \tilde{\gamma}_w) \rightarrow I_w$, where the server uses the search token provided by **Rt** to search its saved index table γ_w and $\tilde{\gamma}_w$. Only γ_w is searched when searching with a single keyword. When searching with multiple keywords $\{\tau_{w1},\ \cdots, \tau_{wk}\}$, for the first keyword τ_{w1}, γ_w is searched to confirm exactly which files contain the keyword. Then, for all files that contain the first keyword, $\tilde{\gamma}_w$ is searched to confirm whether $\{\tau_{w2},\ \cdots, \tau_{wk}\}$ are in the file. The set of hash addresses matching the keywords $I_w = \{ipfshash_j,\ \cdots,\ ipfshash_k\}$ are output, and I_w is sent to **Rt**.

6) Verification stage

Verify (IpfsHash, K2, MAC_{kw}) $\rightarrow$ $IpfsHash_{true/false}$. After **Rt** receives the IpfsHash set from the server in the client backend, IpfsHash is sent to the verification contract. The verify() function is called in the verification contract and incoming the parameter MAC_{kwi} and K2. MAC_{kwi} and K2 are stored as private variables on the verification contract, which can only be used within a contract and not viewed from an external call. If there is $MAC_{kwi} = PRF(K2,\ IpfsHash_i)$ for all search results of $IpfsHash_i$, then the search results returned by the server are correct, and the smart contract returns the search results to the user and transfers the service fee temporarily stored in the search token generation

stage to the account of **TRP**. If it is not equal, i.e., then the search result returned by **TRP** is wrong. The smart contract will deduct a small portion of the deposit of **TRP** and return it to **Rt**'s account together with the service fee.

Algorithm 2: Key Recover

Input: addr, u_1, u_2, $\cdots$, u_v, MAC_{ss}, r, p
Output: K
Require: addr Authorized
Contract storage subkey execution:
01: TemoMap ← []
02: count ← 0
03: **for** i ← 1 $\cdots$ n **do**
04: i, x_i, m_i ← parse (u_i)
05: **if** H (i ||x_i|| m_i) == MAC_{ssi}
06: reward()
07: TempMap.add (x_i, m_i)
08: count ← count + 1
09: **end**
10: **else**
11: punishment()
12: **end**
13: **end**
When equal count t, stop receiving subkeys and call the correct subkey to recover:
14: (x_1, m_1), (x_2, m_2), $\cdots$, (x_t, m_t) ← get x_i, m_i from TempMap
15: M ← $m_1 m_2 \cdots m_t$
16: **for** i ← 1 $\cdots$ t **do**
17: y_i ← inv $\left(\frac{M}{m_i}, m_i\right)$
18: **end**
19: k1 ← 0
20: **for** i ←1 $\cdots$ t **do**
21: k1 ← k1 + $\left(\frac{M}{m_i}\right) x_i y_i$
22: **end**
23: k1← k1 mod M
24: Empty the TemoMap
25: **return** k1-rp

7) Decryption phase

Key recovery phase: KeyRecover (addr, u_1, u_2, …, u_t, MAC_{ss}, r, p) → k/*false*. After confirming that **Rt** is successfully authorized, **Kk** enters subkeys into the key recovery contract, notifies **Rt** when there are enough correct subkeys, and recovers the key. The recovery algorithm is in algorithm 2. The input addr is **Rt**'s account to confirm that it is authorized, and MAC_{ss} is used to verify that the subkeys provided by **Kk** are correct. If the validation is correct, then the reward() function is called to reward. If the validation is incorrect, then the punishment() function penalizes **Kk** by deducting a portion of its deposit in the data-sharing contract.

Decryption stage: FileDecrypt (k, IpfsHash) → file, where Rt uses the symmetric key returned from the key recovery smart contract and the hash address of the resume stored on IPFS as the input. The encryption key is K1, by accessing the IPFS address of the resume to obtain $E_i = enc_{k1}(iv||CV_i)|i \in [1, n]$. The iv is parsed out, and $D_i = \{Dec_{k1}(CV_i)|i \in [1, n]\}$ can be obtained. which automatically converts into a complete decrypted resume.

5 Smart Contract Design

This section describes the interface and algorithm logic of the smart contract used in this study. The smart contract is programmed using Solidity and mainly includes a data-sharing contract and verification contract.

1) Data-sharing contract. The main functions used in the data-sharing contract and resume-sharing interaction are shown in Fig. 2. The implementation of the AddShareAccess() function is mainly used for authorization.

2) Verified contract. The main verification operation is implemented in the contract, as shown in algorithm 3, the add() function is used to add the processed search results, and loop is used to compare whether elements in R are equal to those in Mac_{kw}. If the elements are equal, then 1 is added to the counter, and if the counter is equal to the size of R, then the results are verified correctly, the $reward()_{TRP}$ function is called to transfer the service fee temporarily stored by **Rt** to the server account, and the correct result is returned. Otherwise, $punishment()_{TRP}$ is called to deduct part of the guaranteed amount stored by the server in the contract to **Rt** to return *false*. (The function complete description is shown in Tab. 2).

Algorithm 3: Verify

Input: $return_{Ipfshash}$, Mac_{kw}, K
Output: $return_{Ipfshash}$/*false*

```
count ← 0
R ← []
R.add(PRF(K, return_Ipfshash[i])
for i ← 0 ··· length of return_ipfshash do
    for j ← 0 ··· length of Mac_kw do
        if R[i] == Mac_kw[j]
            count ← count + 1
            break
        end
    end
end
if count == length of R
    reward()_TRP
    return return_ipfshash
end
else
    punishment()_TRP
    return false
end
```

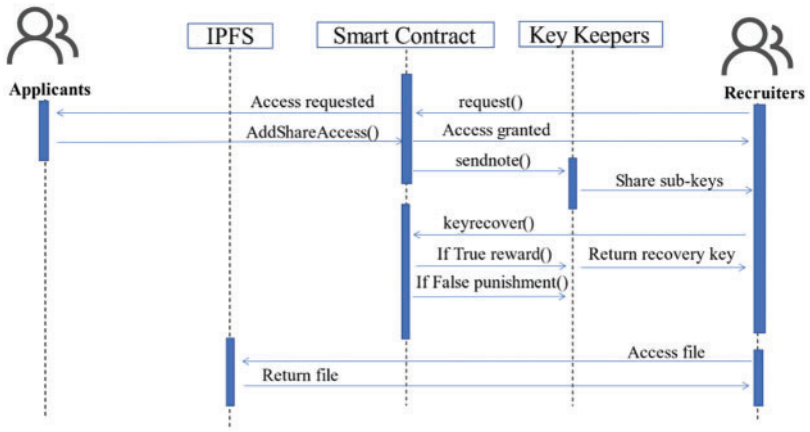

Figure 2: Complete resume sharing interaction

Table 2: Contract functions and descriptions

Function	Description
Evidence()	Store the resume hash and other related information, so that **Ap** can confirm the right.
addkeeper()	Adding a key keeper account.
deletekeeper()	Deleting a key keeper account.
AddShareAccess()	Record shared information, authorize access.
Isauthorized()	Verify that the account is authorized.
request()	**Rt** initiates the request information and records it on the chain.
keyrecover()	Call this function to restore the symmetric key.
addmac()	Add information about the subkey verification set.
reward() reward()$_{TRP}$	reward(): **Kk** provides the correct subkey to participate in key recovery will receive a portion of the amount paid by **Rt** for downloading the resume. reward()$_{TRP;}$ Transfers the search fee to the **TRP** address that provided the correct search result.
punishment() punishment()$_{TRP}$	punishment(): Deduct part of the deposit of the Kk that provided the wrong subkey to **Ap** and **Rt**. The node that applied to be a **Kk** (**TRP**) will first have to store the deposit in the contract. punishment()$_{TRP}$: Deduct part of the deposit to **Rt** for the **TRP** that provided the wrong search result.

6 Analysis

This section evaluates the proposed framework from three aspects: security, function, and performance.

6.1 Security Analysis

The proposed framework combines Ethereum blockchain, IPFS, secret-sharing technology, and searchable encryption technology, which have more advantages than the traditional **TRP** for storing and sharing resume files. This section discusses the benefits of the framework in the following four aspects:

1) Controlled sharing and recording

Only authorized recruiters can recover the key for decrypting resumes in the proposed framework. **Rt** is entitled to the key only when the candidate adds **Rt** to the authorized list of the data-sharing contract. Meanwhile, the resume information, **Rt**'s request, and other operation records related to the resume are stored in the blockchain and cannot be tampered with, making it easy to track and confirm the right resume with high traceability.

2) Resume security and privacy

Instead of traditional cloud servers, the IPFS is used to store files. Even if a node is breached, the stored data are fragments instead of complete files, and outsiders can only view messy code. Thus, the proposed framework guarantees secure storage and privacy protection of resumes.

3) Key management

The file encryption key is divided into n parts by a secret-sharing scheme, and they are assigned to different custodians. Therefore, at least t trustees need to collude maliciously to recover the key and decrypt the resume. Smart contracts provide automatic reward and penalty mechanisms for **Kk**. Those who send correct subkeys honestly can obtain certain rewards. By contrast, after sending wrong subkeys, malicious **Kk** incurs a deduction on deposit to achieve incentives and fairness.

4) Retrievability and retrieval fairness

The traditional scheme does not support the retrieval of encrypted files, which causes great inconvenience. This condition requires the use of searchable encryption technology to ensure the efficient retrieval of encrypted files. Still, most traditional searchable encryption solutions only support single keyword searches and require semi-honest cloud servers. The cloud server may maliciously return the wrong results while obtaining service fees. This framework constructs two index tables for multi-keyword search through inverted indexes and Bloom filters. It generates a search token using a smart contract by temporarily storing the service fee at the contract address. After performing the retrieval task, the server returns the result to the validating smart contract, and the validating contract verifies whether the result is correct. If the result is correct, then the corresponding service fee is transferred to the server account; if it is incorrect, then the server loses the service fee and receives a penalty. Therefore, to obtain the service fee and avoid penalties, the server returns the correct search results to ensure a fair search.

6.2 Performance Analysis

1) Functional comparison

On the one hand, the functional features of the proposed framework are compared with the schemes presented by [25–32], as shown in Tab. 3. Y indicates that this feature is implemented, and N indicates that it is not.

Table 3: Function comparison

Paper	IPFS	Access control	Key security	Searchable	Fair payment	Multiple Keywords
Paper [25]	Y	Y	N	N	N	N
Paper [26]	N	Y	N	N	N	N
Paper [27]	Y	Y	Y	N	N	N
Paper [28]	N	Y	Y	Y	Y	N
Paper [29]	N	N	N	Y	Y	N
Paper [30]	N	N	N	Y	Y	N
Paper [31]	N	Y	N	Y	Y	Y
Paper [32]	N	Y	N	Y	N	N
Our	Y	Y	Y	Y	Y	Y

As can be seen from Tab. 3, in terms of data sharing, the advantages of the scheme in this paper are the design of a secret sharing scheme with an incentive mechanism to improve the security of the key. And the support of multiple keywords for encrypted data retrieval. In terms of resume application scenarios, this paper uses IPFS and blockchain to replace the centralized storage and management of resumes in current recruitment platforms to avoid the problems of resume forgery, loss, and leakage. Further, the existing recruitment platform only has public and confidential options. Once a resume is confidential, it cannot be searched. Applicants are faced with the dilemma of protecting their privacy and not being retrieved by recruiters. This paper combines searchable encryption to ensure that the recruiter can search the resume by keywords. The disadvantage of the solution is that the construction and maintenance of IPFS and the Ethereum platform will increase the cost.

2) Performance comparison

The computational cost of the blockchain-based framework for secure storage and sharing resumes (BBFSSR) in the present study is compared with the schemes TKSE [29] in index generation, token generation, search phase, and verification phase, as shown in Tab. 4. H denotes hashing operation, PRF denotes pseudo-random function, CP denotes string comparison operation, |N| denotes the number of keywords extracted by **Ap** in the index generation phase, |F| denotes the number of files containing keyword w, |R| denotes the number of returned ciphertext files, |C| denotes the number of files, |W| denotes the number of keywords in a multi-keyword query, and |BF| denotes the time required to determine whether a keyword is in Bloom's filter. Furthermore, S denotes the signature algorithm, which contains two processes: signature and verification. SE denotes the symmetric encryption algorithm, which contains two processes: encryption and decryption.

Table 4: Performance comparison

Paper	Index generation	Token generation	Search	Validation
TKSE	|N||F|(PRF+H+ S)	|C|PRF	|C|(H+S)	|C|(H+S)
BBFSSR	|N|PRF+|C||BF|	PRF	1/|W||C||BF|	|R|(PRF+|C|CP)

In the index generation phase, TKSE needs to execute 1 S, 1 PRF, and 1 H for each keyword. BBFSSR needs to perform 1 PRF to build the first inverted index γ_w and then build a Bloom filter on |C| files to build the second index $\tilde{\gamma}_w$. In the token generation phase, TKSE and BBFSSR both use PRF to generate tokens, TKSE as the number of files increases, the number of PRF performed increases, and BBFSSR only performs one PRF. In the search phase, when searching with a single keyword, BBFSSR uses the keyword-file set approach to build the index, so enter the search token will get the results immediately, and the search efficiency is O(1). When searching with multiple keywords, the user has to determine for each file whether |W| keywords are in corresponding Bloom filters, and the search efficiency is |W||C||BF|. In the validation phase, TKSE requires the user to perform |C| times H and S locally, and BBFSSR performs |R| times H and |R||C| times CP in the smart contract.

3) Experimental analysis

To more accurately evaluate the actual performance of the framework, this paper conducts simulation tests in terms of index generation time, search token generation time, search time, and verification time. The experiments use English word dataset, and hashing algorithm H is SHA-256, pseudo-random function PRF is HMAC-SHA256, symmetric encryption algorithm SE is AES-256 CBC mode, signature algorithm is ECDSA, the smart contract is solidity language, Ethereum blockchain is built locally, embark framework is used to compile, deploy and run. The hardware environment for this experiment is Intel Core™ i5-10400 CPU (2.9 GHz) with 32 GB RAM. The experimental results are shown below.

As shown in Fig. 3, suppose each file contains 200 keywords in the index generation phase. TKSE and BBFSSR increase index generation time as the files increase and the corresponding keywords increase; TKSE involves the signature operation; the index takes a long time to generate, 10,000 keywords to build the index, TKSE takes 4889.95 ms, BBFSSR takes only 324.26 ms. As shown in Fig. 4, in the token generation phase, BBFSSR executes 1 PRF algorithm. TKSE runs more PRF depending on the number of files. In Fig. 4, assuming that the total number of files |C| is 5, TKSE performs 5 PRFs.

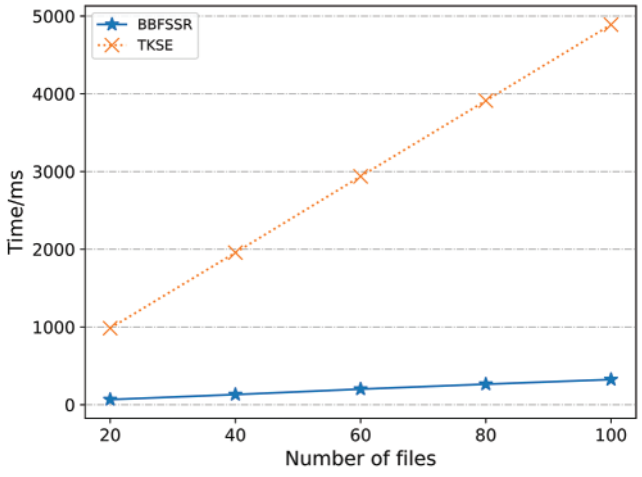

Figure 3: Index generation time

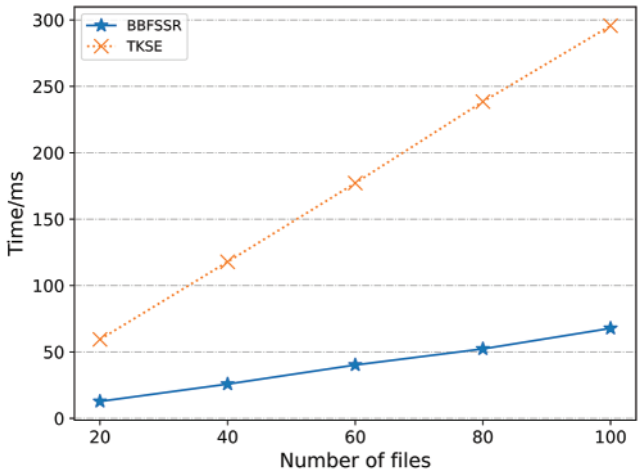

Figure 4: Token generation time

As shown in Fig. 5, a single keyword search in BBFSSR is O(1) operation, and the search time in the index table is established by 2000 to 10000 keywords is maintained at 0.1 ms. TKSE increases the search time due to increasing the number of files, and when the number of files is 100, the search time is 47.65 ms. TKSE does not support multi-keyword search; to search for multiple keywords, not only to conduct multiple single-keyword searches also need to constantly filter out the files containing all the keywords, very time-consuming, as shown in the figure, the search in 100 files to meet the files containing 10 keywords took 950.12 ms. In contrast, BBFSSR supports multi-keyword search only takes 85.72 ms. In the validation phase, the number of files returned by the search |R| is the same as the total number of files |C|, and the validation times of the two schemes are shown in Fig. 6.

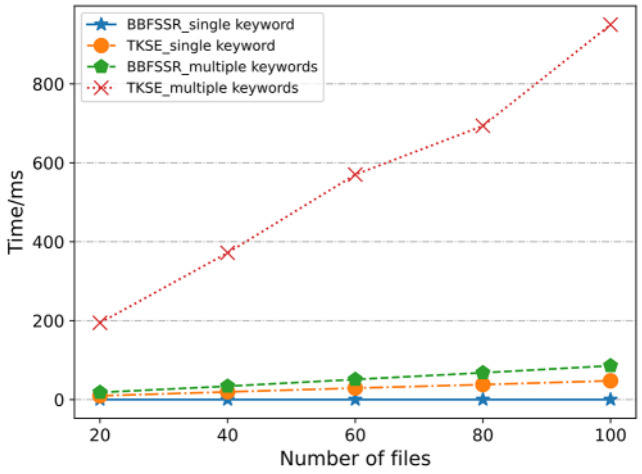

Figure 5: Search time

We simulated the cost of the smart contract, where the price of ETH is 1 ether = 1,906 USD, the price of gas is 1 Gwei (1 Gwei = 10^{-9} and Wei = 10^{-9} ether). Figs. 7 and 8 show the cost of contract deployment and several functions used. The contract deployment consumes a large amount of gas, but the deployment of the functions in the contract consumes a small amount of gas, which is within the acceptable range.

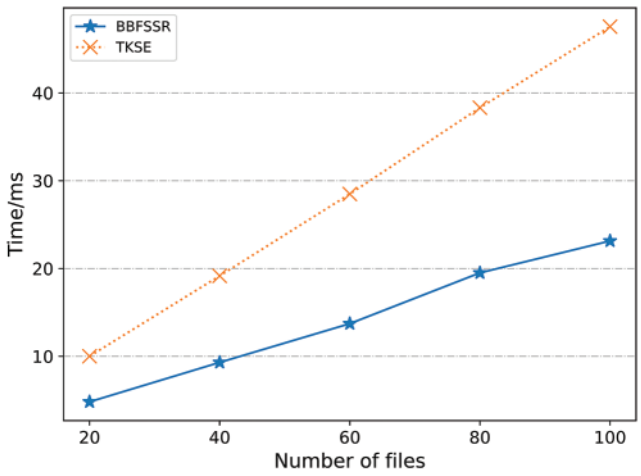

Figure 6: Verification time

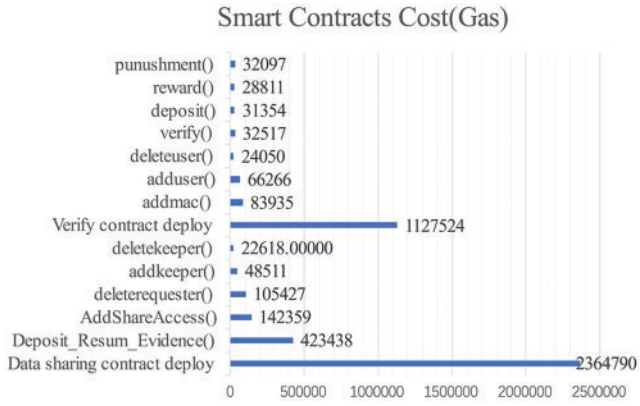

Figure 7: Smart Contracts Cost (Gas)

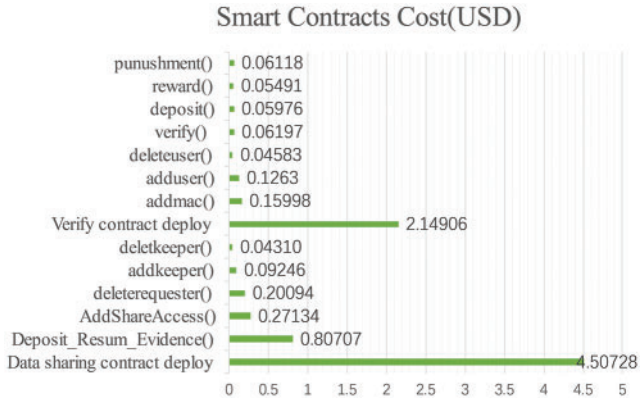

Figure 8: Smart Contracts Cost (USD)

This paper tested the time to perform 1000 secret distributions and reconstructions for 256-bit integers using the Asmuth-Bloom algorithm with different thresholds. as shown in Appendix A, in Fig. 9 1000 times (10, 11) threshold secret distribution takes 51.34 s, and secret reconstruction takes only 0.62 s. Therefore, the contract can reconstruct the key quickly when it receives enough subkeys.

7 Conclusion

This paper has presented a new solution based on blockchain, secret sharing, and searchable encryption technology to address the problems of centralized data management, resume leakage, falsification, and the inability to pursue responsibilities in traditional recruitment platforms accurately. The solution uses blockchain to replace traditional third-party agencies. It combines on-chain and off-chain distributed storage mechanisms and secret sharing technology to ensure the security and authenticity of resumes. It also realizes fair and efficient resume search based on searchable encryption technology. The analysis and experiments on the security and performance of the scheme show that the scheme can support resume sharing and meet the requirements of privacy protection and sufficient system performance. However, further refinement is needed in consensus mechanism selection and framework deployment cost optimization.

Funding Statement: The authors gratefully acknowledge the financial supports by Key Projects of the Ministry of Science and Technology of the People's Republic of China (2018AAA0102301).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, pp. 21260, 2008.
[2] K. Christidis and M. De vetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
[3] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, pp. 1–21, 1997.
[4] J. Benet, "IPFS - content addressed, versioned, P2P file system," *arXiv*, arXiv:1407.3561, pp. 1–11, 2014.
[5] A. Shamir, "How to share a secret," *Communications of the ACM*, 1979, vol. 22, no. 11, pp. 612–613, 1979.
[6] G. R. Blakley, "Safeguarding cryptographic keys," in *Proc. MARK*, New York, NY, USA, pp. 313–318, 1979.
[7] C. Asmuth and J. Bloom, "A modular approach to key safeguarding," *IEEE Transactions on Information Theory*, vol. 29, pp. 208–210, 1983.
[8] S. D. Xiaoding, D. Wagner and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. S&P 2000*, Berkeley, CA, USA, pp. 44–55, 2000.
[9] R. Curtmola, J. Garay, S. Kamara and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. CCS '06*, New York, NY, USA, pp. 79–88, 2006.
[10] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk *et al.,* "Dynamic searchable encryption in very-large databases: Data structures and implementation," *IACR Cryptol. ePrint Arch*, vol. 2014, pp. 853, 2014.
[11] J. Li, Y. Huang, Y. Wei, S. Lv, Z. Liu *et al.,* "Searchable symmetric encryption with forward search privacy," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 1, pp. 460–474, 2021.
[12] Z. Li, J. Ma, Y. Miao and X. Liu, "Forward and backward secure keyword search with flexible keyword shielding," *Information Sciences*, vol. 576, pp. 507–521, 2021.

[13] C. Xu, L. Yu and L. Zhu, "A Blockchain-based dynamic searchable symmetric encryption scheme under multiple clouds," *Peer-to-Peer Netw*, vol. 14, pp. 3647–3659, 2021.
[14] S. Cui, X. Song and M. R. Asghar, "Privacy-preserving dynamic symmetric searchable encryption with controllable leakage," *ACM Transactions on Privacy and Security*, vol. 24, no. 18, pp. 1–35, 2021.
[15] J. Chen, K. He, L. Deng, Q. Yuan, R. Du *et al.,* "EliMFS: Achieving efficient, leakage-resilient, and multi-keyword fuzzy search on encrypted cloud data," *IEEE Transactions on Services Computing*, vol. 13, no. 6, pp. 1072–1085, 2020.
[16] L. Li, C. Xu, X. Yu, B. Dou and C. Zuo, "Searchable encryption with access control on keywords in multi-user setting," *Journal of Cyber Security*, vol. 2, no. 1, pp. 9–23, 2020.
[17] J. Zhang, G. Xu, X. Chen, H. Ahmad, X. Liu *et al.,* "Towards privacy-preserving cloud storage: A blockchain approach," *Computers, Materials & Continua*, vol. 69, no. 3, pp. 2903–2916, 2021.
[18] C. Xu, L. Mei, J. Cheng, Y. Zhao and C. Zuo, "IoT services: Realizing private real-time detection via authenticated conjunctive searchable encryption," *Journal of Cyber Security*, vol. 3, no. 1, pp. 55–67, 2021.
[19] P. Almeida, C. Baquero, N. Preguiça and D. Hutchison, "Scalable bloom filters," *Information Processing Letters*, vol. 101, pp. 255–261, 2007.
[20] K. Fan, S. Wang, Y. Ren, H. Li and Y. Yang, "Medblock: Efficient and secure medical data sharing via blockchain," *Journal of Medical Systems*, vol. 42, pp. 1–11, 2018.
[21] H. Tan, T. Zhou, H. Zhao, Z. Zhao, W. Wang *et al.,* "Archival data protection and sharing method based on blockchain," *Journal of Software*, vol. 30, pp. 2620–2635, 2019.
[22] N. Nizamuddin, H. Hasan, K. Salah and R. Iqbal, "Blockchain-based framework for protecting author royalty of digital assets," *Arabian Journal for Science and Engineering*, vol. 44, pp. 3849–3866, 2019.
[23] O. A. Alzubi, J. A. Alzubi, O. Dorgham and M. Alsayyed, "Cryptosystem design based on hermitian curves for IoT security," *the Journal of Supercomputing*, vol. 76, no. 11, pp. 8566–8589, 2020.
[24] O. A. Alzubi, J. A. Alzubi, K. Shankar and D. Gupta, "Blockchain and artificial intelligence enabled privacy-preserving medical data transmission in internet of things," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 12, pp. e4360, 2021.
[25] M. Uddin, M. S. Memon, I. Memon, I. Ali, J. Memon *et al.,* "Hyperledger fabric blockchain: Secure and efficient solution for electronic health records," *Computers, Materials & Continua*, vol. 68, no. 2, pp. 2377–2397, 2021.
[26] A. A. Alnssayan, M. M. Hassan and S. A. Alsuhibany, "Vacchain: A blockchain-based emr system to manage child vaccination records," *Computer Systems Science and Engineering*, vol. 40, no. 3, pp. 927–945, 2022.
[27] M. Naz, F. A. Al-zahrani, R. Khalid, N. Javaid, A. M. Qamar *et al.,* "A secure data sharing platform using blockchain and interplanetary file system," *Sustainability*, vol. 11, no. 24, pp. 7054, 2019.
[28] N. Sohrabi, X. Yi, Z. Tari and I. Khalil, "BACC: Blockchain-based access control for cloud data," in *Proc. (ACSW '20)*, New York, NY, USA, pp. 1–10, 2020.
[29] Y. Zhang, R. H. Deng, J. Shu, K. Yang and D. Zheng, "TKSE: Trustworthy keyword search over encrypted data with two-side verifiability via blockchain," *IEEE Access*, vol. 6, pp. 31077–31087, 2018.
[30] H. Li, H. Tian, F. Zhang and J. He, "Blockchain-based searchable symmetric encryption scheme," *Computers & Electrical Engineering*, vol. 73, pp. 32–45, 2019.
[31] L. Chen, W. K. Lee, C. C. Chang, K. K. R. Choo and N. Zhang, "Blockchain based searchable encryption for electronic health record sharing," *Future Generation Computer Systems*, vol. 95, pp. 420–429, 2019.
[32] T. Feng, H. Pei, R. Ma, Y. Tian and X. Feng, "Blockchain data privacy accesscontrol based on searchable attribute encryption," *Computers, Materials & Continua*, vol. 66, pp. 871–890, 2021.

A Appendix Secret sharing time consumption

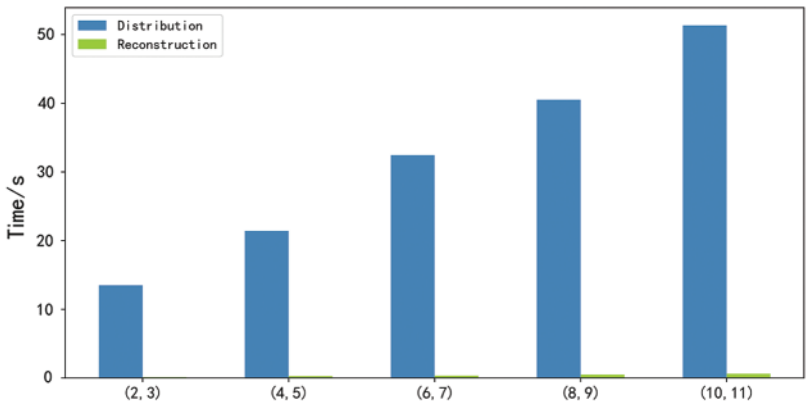

Figure 9: Secret sharing time consumption