Computers and Electrical Engineering 73 (2019) 32–45

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

# Blockchain-based searchable symmetric encryption scheme☆

Huige Li[a,b], Haibo Tian[a,b], Fangguo Zhang[a,b,*], Jiejie He[a,b]

[a] School of Data and Computer Science, Sun Yat-Sen University, Guangzhou 510006, China
[b] Guangdong Key Laboratory of Information Security, Guangzhou 510006, China

ARTICLE INFO

*Article history:*
Received 20 April 2017
Revised 24 October 2018
Accepted 24 October 2018

*Keywords:*
Searchable symmetric encryption
Bitcoin
Blockchain
Fairness
Cloud-computing

ABSTRACT

The mechanism for traditional searchable symmetric encryption (SSE) is pay-then-use. This situation is not fair to user because the cloud server may return wrong results. Besides, the user needs to verify these results locally. In order to ensure fairness and reduce user's calculations, we combined blockchain with SSE, and proposed a fair SSE scheme based on blockchain. Our scheme can guarantee fairness for both parties. That is, if the user is not honest, he cannot get right results from the server, and at the same time the server cannot get any information related to the plaintexts during this search process. If the server is not honest, except for the service charge, it will be punished automatically. Moreover, the user in our scheme does not need to verify the results locally. The security and performance analyses showed our scheme was semantic secure and feasible.



## 1. Introduction

With the rapid development of cloud computing, more and more service providers have issued a series of cloud products, such as Amazon Web Services, Google Cloud, etc. These cloud storage systems have changed the original storage way of data. Users can store their data on the cloud directly, and then get access to them on any device conveniently.

The data usually include some sensitive information, such as name, telephone number and so on. The cryptography plays an important role in protecting data privacy. Namely, to protect the privacy of data, users can encrypt them before uploading. However, which encryption algorithm is chosen will affect the search efficiency.

Searchable symmetric encryption (SSE) was firstly proposed by Song et al. [1]. It allows a user to outsource his data to the cloud in a private manner, while maintains the ability of selectively search segments of the data. It involves three participants in SSE: Data owner, cloud server and user. The data owner encrypts documents $D_1, D_2, \ldots, D_n$ and stores them on the cloud. The cloud server manages these ciphertexts and provides search service to users. If a user gets the permission from the data owner, he can obtain search results with the help of the cloud server by using an encrypted keyword. Finally, the user decrypts them locally.

There are two ways to construct SSE. The first is to use an index, and the second is not to use an index. Because the former can improve search efficiency, the subsequent SSE works mostly accept this approach. A SSE scheme is secure if anyone except authorised person cannot learn any information about the plaintexts when he/she only gets ciphertexts.

☆ This paper is for CAEE special section SI-bciot.Reviews processed and recommended for publication to the Editor-in-Chief by Area Editor Dr. G. Martinez Perez.

* Corresponding author at: School of Data and Computer Science, Sun Yat-Sen University, Guangzhou 510006, China.
*E-mail address:* isszhfg@mail.sysu.edu.cn (F. Zhang).

https://doi.org/10.1016/j.compeleceng.2018.10.015
0045-7906/

Meanwhile, when search, except the search results, he/she also cannot learn anything about the plaintexts and the keywords that queried.

For existing SSE schemes, users need to pay before enjoying search service the cloud server provided. However, this manner is very unfair to users. Once the search results are incorrect, the user needs to ask a third party to redeem the service fee they paid, which may take a long time. If the third party colludes with the cloud server, the user will never be able to defend their rights. This situation usually happens in medical systems, insurance company, etc. In order to prevent such things from happening, it is necessary to solve the problem of fairness in SSE.

### *1.1. Related work*

The search efficiency of scheme [1] is $O(n)$, where $n$ denotes the length of document. In order to improve it, Goh et al. introduced an index[2] in the construction of SSE. However, this scheme may return wrong results, therefore, Curtmola et al. [3] used the data structure to create a new index for all documents, whose search complexity was $O(|D(w)|)$, where $|D(w)|$ denotes the number of documents that contain keyword $w$.

In order to enrich the function of SSE, the subsequent researchers proposed parallel search [4], Boolean query search [5–7], fuzzy search [8–10], dynamic updates [11–13], search with multi-level access policy [14].

Most SSE schemes can resist honest but curious adversary, such as [1,3,7]. While, Kurosawa et al. firstly gave a protocol to resist malicious adversary [15], which uses the message authentication code technology in the construction of index. Cheng et al. designed an algorithm that also can resist such adversary by using the indistinguishability obfuscation [16]. Other works that resist malicious adversary are [11–13].

Undoubtedly, a stronger privacy guarantee for SSE scheme can be achieved when using oblivious Random Access Machines [17]. However, it needs multiple interactions between server and user. Therefore, the search efficiency is low, as a result, it is not practical. Song et al.[1] pointed that to improve the search efficiency, it is feasible to weaken the privacy security level appropriately.

**Bitcoin** is an emerging virtual digital currency which happened in peer-to-peer (P2P) network. It was firstly proposed by Nakamoto [18] in 2008, who generated the first bucket of bitcoin in 2009. The issuance of bitcoin does not depend on a trusted entity. Anybody may issue a certain amount of bitcoin as long as he mines a right nonce. It is a purely decentralized system. It uses proof-of work (POW) to confirm a transaction. In this system, it requires that the majority of nodes in P2P network are honest.

Because the bitcoin system is decentralized and irreversible, there has a boom of research of bitcoin mechanism [19–21]. Andrychowicz et al. and Bentov et al. introduced bitcoin into multiparty computations [22–24] to resolve the fairness problem respectively.

Blockchain is the core tool in bitcoin, which can be used to issue other forms of cryptography currency, such as Ethereum [25]. In the Ethereum system, everyone can build some smart contracts, which can be invoked by anyone. In fact, the protocol [23] can be seen as a smart contract, because it introduces a commitment algorithm $h(x)$ in the out-script of transaction.

**Our contributions:** In this paper, we constructed a fair SSE scheme by using blockchain. The documents are encrypted and stored on the cloud server. When searching, the user and the server need to build a series of transactions to get the final search results. Our contributions are listed as follows:

- We introduced blockchain into SSE, and design a fair SSE scheme. Our blockchain-based SSE scheme can maintain fairness for both parties automatically. As long as one party does not perform the protocol honestly, it will lose its own deposit. Moreover, our blockchain-based SSE scheme also can resist malicious server.
- In our scheme, it needs 6 transactions to obtain the search results. Nonetheless, the search efficiency is still linear with the number documents that contain the querying keyword.
- Our scheme can verify the results automatically. Compared with the existing SSE schemes, our scheme can reduce the calculation of users.
- We implemented our scheme in Java and Go language, and the experimental results showed that our scheme was feasible.

**Organization.** The remainder of this paper is organized as follows. In Section 2, we review some preliminaries that will be used in our construction. Then we propose our model of blockchain-based SSE and its security definition. In Section 4 we present our concrete construction. Next we give the analyses of performance and security for our scheme. The last section is our conclusion.

## 2. Preliminaries

In this section, we review some cryptographic tools and terms that will be used in our construction. It mainly includes searchable symmetric encryption, bitcoin currency system and negligible function.

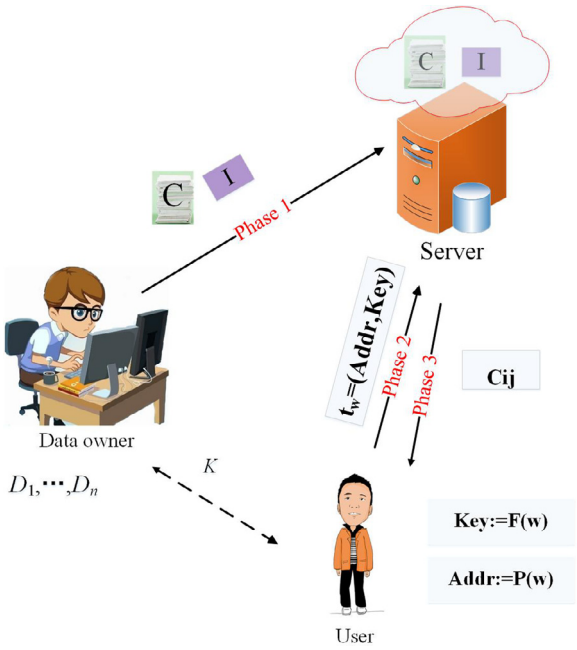


**Fig. 1.** Traditional SSE model.

### 2.1. Searchable symmetric encryption

As shown in Fig. 1, there are three participants: Data owner, server and user. The data owner has $n$ documents $D_1, D_2, \ldots, D_n$. Before uploading, he encrypts them into ciphertexts $C = (C_1, C_2, \ldots, C_n)$ by using his secret key $K$. Besides, he generates an encrypted index $I$. Then, he sends $C$, $I$ to the server. Suppose that the data owner and the user share the private key $K$. Now, the user wants to search documents that contain keyword $w$. Therefore, he computes a search token $t_w$ for keyword $w$ by using key $K$, and sends it to the server. The server then computes the results $C_{ij}$ by combining $t_w$, $C$, and $I$. Finally, the user decrypts $C_{ij}$ locally.

A searchable symmetric encryption is secure if the following properties hold:

- The server cannot learn anything about the plaintexts when it only gets ciphertexts.
- When the server executes search algorithm, it also can not learn anything about the plaintexts and the keywords queried except the search results.

### 2.2. Bitcoin currency system

The blockchain is a core technology in bitcoin system. There does not have a standard definition for blockchain. In order to understand it clearly, we will review the bitcoin.

A bitcoin system consists of addresses and transactions between them. The address usually is a hash value of the user's public key. Each user can have a pair of keys when he wants to build a transaction, i.e., a private key and a public key [23]. The public key is used to verify whether the signature of a transaction is valid or not, while the private key is used to sign this transaction. For brevity, we will use the capital letter (e.g. $A$) to denote this pair key ($A.pk$, $A.sk$). Let $\sigma = sig_A(T)$ denote the signature of transaction $T$ with respect to the private key $A.sk$ of $A$, and $ver_A(T, \sigma)$ be the verification result by using the public key $A.pk$ of $A$.

In the bitcoin system, each transaction can have multiple inputs and two outputs at most. The transaction describes the circulation of digital cryptocurrency, namely, the money is transferred from an address of user $A$ to an address of user $B$. A transaction can be denoted as $T_x = ((y_1, a_1, \sigma_1), \ldots, (y_l, a_l, \sigma_l), (v_1, \pi_1), \ldots, (v_l, \pi_l), t)$, where $y_i$ is a hash value of previous transaction $T_{y_i}$, $a_i$ is an index of the output of $T_{y_i}$ and $\sigma_i$ is the input-script. The $(v_1, \pi_1), \ldots (v_l, \pi_l)$ can be seen as the outputs of $T_x$, where the $\pi_i$ is output-script. The input-script and output-script are written in bitcoin scripting language, it is a stack-based language [22]. The input-script of a transaction is associated with the output-script of previous transaction. Each transaction can have a time lock $t$, which means it is valid only after $t$ time. The $((y_1, a_1), \ldots, (y_l, a_l), (v_1, \pi_1), \ldots (v_l, \pi_l), t)$ usually is called as the body of $T_x$, which we denote by $[T_x]$.

A transaction is valid if it satisfies that:

- The defined time $t$ is reached.

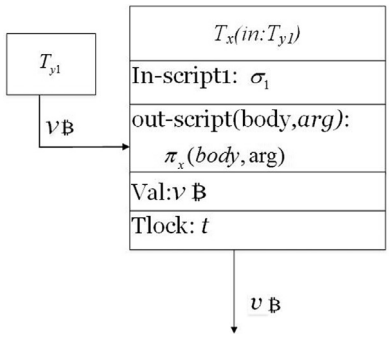


**Fig. 2.** A transaction for a situation when a user locks $v$Ƀ. $y_1$ and $x$ denote the pairs of keys hold respectively by the sender and the receiver. $t$ is a moment of time, when the receiver can take his deposit back. $T_{y_1}$ is an unredeemed transaction with value $v$Ƀ, which can be redeemed with key $y_1$. $\sigma$ denotes a signature of transaction $T_x$ created by $x$.

**Table 1**
Symbols/Functions list that we will use in our blockchain-based SSE scheme.

| Symbol/Functions | Meaning |
|---|---|
| $D$ | document |
| $C$ | ciphertext of D |
| $w$ | keyword |
| k | system parameters |
| Ƀ | the form of currency in the blockchain |
| $F_1$, $F_2$ | pseudorandom functions, where $F_i$: $\{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^k$. |
| $H$ | a keyed hash function, such as $SHA-256$. |
| $\varepsilon = (\varepsilon.Enc, \varepsilon.Dec)$ | indistinguishability against chosen-plaintext attacks ($IND-CPA$) secure symmetric encryption (SE) scheme, where $\varepsilon.Enc$ denotes the encryption process and $\varepsilon.Dec$ denotes the decryption process. |
| $\delta = (\delta.Enc, \delta.Dec)$ | a determinate SE scheme, where $\delta.Enc$ denotes the encryption process and $\delta.Dec$ denotes the decryption process. |

- The verification of $\pi_i([T_x], \sigma_i)(1 \le i \le l)$ holds.
- The involved transactions are not redeemed.
- It was accepted by at least 6 nodes.

When the transaction is finished, it will be collected by a block. As shown in Fig. 2, it is a simple transaction.

**Definition 1.** A function $f$ is negligible if for every polynomial $p(\cdot)$ there exists an Integer $N$ such that for all integer $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.

Here, we also list some functions and symbols that we will use in our construction, which are shown in Table 1.

## 3. Our system model

In fact, SSE technology can be used in many scenarios. For example, in a financial sector, employees can store their encrypted data on the cloud, which not only protects the interests of company, but also can provide convenient search services for employees. Since the cloud is not trustworthy, when employee searches, it may return wrong results. If employees do not check them locally, it may lead to them making a poor decision. Moreover, if the size of returned results is large, they will take more time to finish this verification. Therefore, it is necessary to design a SSE scheme which can automatically check the search results and guarantee the fairness for both parties.

In this section, we will give our model of blockchain-based SSE and its security definition.

### 3.1. Definition for blockchain-based SSE

As shown in Fig. 3, in the blockchain-based SSE system, there mainly have three participants: Data owner, server and user. The data owner has $n$ documents $\mathbf{D} = (D_1, D_2, \ldots, D_n)$. Suppose the data owner and the user share the private key $k_1$.

In the first stage, the data owner uploads encrypted documents $\mathbf{C} = (C_1, \ldots, C_n)$ and index $I$ on the cloud server. In the second stage, if the user wants to search documents that contain keyword $w$, he generates an encryption value $T_w$ of keyword $w$, and sends it to the server to invoke it to participate in the following stages. Besides, the user builds a transaction *Appoint* whose receiver either is himself or the server. In the third stage, the server creates a transaction *ask* to get the decryption key of $T_w$. If the user provides it correctly, he can uses the transaction *pay* to redeem the transaction

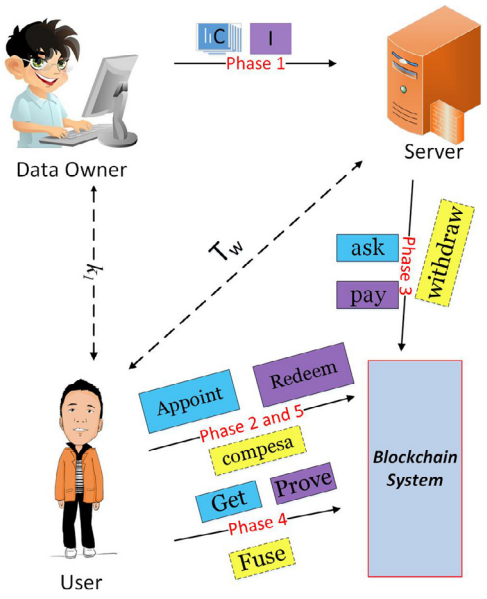


**Fig. 3.** Blockchain-based SSE model.

*ask*, and at the same time the server needs to compute search results that contain keyword $w$ locally; otherwise, the server uses the transaction *withdraw* to get the money in the transaction *ask* back. In the fourth stage, to get the search result, the user creates a transaction *Get*. As long as the server provides the right result, it can use the transaction *Prove* to redeem the transaction *Get*, otherwise the user uses the transaction *Fuse* to get his money in the transaction *Get* back. Finally, the user uses the search results to build a transaction *Redeem* to get his money in the transaction *Appoint* back, otherwise the server will redeem it by using the transaction *compesa*.

**Definition 2** (Blockchain-based SSE)**.** A blockchain-based SSE scheme is a tuple composed of seven polynomial time algorithms $\pi = (Gen, Enc, Srchtoken, Search, Verify, Dec, Rede)$ shown as follows:

- $\boldsymbol{K} \leftarrow Gen(1^k)$: It is a probabilistic algorithm that takes a security parameter $k$ as input, and outputs a key array $\boldsymbol{K}$.
- $(\boldsymbol{I}, \boldsymbol{C}) \leftarrow Enc(\boldsymbol{K}, \boldsymbol{D})$: It takes the key array $\boldsymbol{K}$, data file collection $\boldsymbol{D}$ as input, and outputs ciphertexts $\boldsymbol{C}$ and an invertible Index $\boldsymbol{I}$.
- $(Appoint, T_w) \leftarrow$ Srchtoken$(w, \boldsymbol{K}, T_u)$: It is a deterministic algorithm. The user takes the keyword $w$, the private key array $\boldsymbol{K}$ and an unredeemed transaction $T_u$ as input, and outputs a search token $T_w$ and a transaction *Appoint*.
- $(ask, Pay/Withdraw) \leftarrow Search(T_w, T_{s1}, \boldsymbol{K}, w)$: It takes the search token $T_w$, the key array $\boldsymbol{K}$, the keyword $w$ and an unredeemed transaction $T_{s1}$ as input, and outputs transactions *ask* and *Pay* (or *Withdraw*).
- $(Get, Prove/Fuse) \leftarrow Verify(ask, Pay, \boldsymbol{C}, \boldsymbol{I}, \boldsymbol{K}, T_{u1}, T_{u2})$: It takes the transactions *ask* and *Pay*, the ciphertexts $\boldsymbol{C}$, the index $\boldsymbol{I}$, the key array $\boldsymbol{K}$ and unredeemed transactions $T_{u1}$, $T_{u2}$ as input, and outputs transactions *Get* and *Prove*(or *Fuse*).
- $Redeem/compesa \leftarrow Rede(Appoint, Get, Prove)$: It takes transactions *Appoint, Get* and *Prove* as input, and outputs transaction *Redeem* (or transaction *compesa*).
- $D_j \leftarrow Dec(Prove, \boldsymbol{K})$: The users takes the secret key array $\boldsymbol{K}$, and transaction *Prove* as input, and outputs the plaintext $D_j$.

### 3.2. Security definition

Our security definition mainly adopts the real/ideal simulation paradigm stated in [3].

**Definition 3.** Let $\pi = (Gen, Enc, Srchtoken, Search, Verify, Dec, Rede)$ be a block chain-based SSE scheme, $k \in \mathbb{N}$ be the security parameter, $\mathcal{A} = (\mathcal{A}_0, \ldots, \mathcal{A}_q)$ be an adversary such that $q \in \mathbb{N}$, and $\mathcal{S} = (\mathcal{S}_0, \ldots, \mathcal{S}_q)$ be a simulator, then consider the probabilistic experiments $Real_{\mathcal{A}}^{\pi}(k)$ and $Ideal_{\mathcal{A},\mathcal{S}}^{\pi}(k)$ shown in Figs. 4 and 5:

We say $\pi$ is semantically secure if for all the probabilistic polynomial time (PPT) $\mathcal{A}$, there exists a PPT $\mathcal{S}$ such that for all polynomial size distinguisher $\mathcal{D}$,

$$|Pr[\mathcal{D}(v, st_{\mathcal{A}}) = 1 : (v, st_{\mathcal{A}}) \leftarrow Real_{\mathcal{A}}^{\pi}(k)] - Pr[\mathcal{D}(v, st_{\mathcal{A}}) = 1 : (v, st_{\mathcal{A}}) \leftarrow Ideal_{\mathcal{A},\mathcal{S}}^{\pi}(k)]| \leq negl(k), \quad (1)$$

where $negl(k)$ is a negligible function in $k$.

In the experiment $Ideal_{\mathcal{A},\mathcal{S}}^{\pi}(k)$, $\tau(D)$ is a trace related to documents $D$, which is composed of search pattern, access pattern, the size of ciphertexts and the transactions that the user and the server built. The readers can refer to [3] to get more detailed definitions about them.

$\mathbf{Real}^{\Pi}_{\mathbb{A}}(k)$

$K \leftarrow \mathbf{KeyGen}(1^k)$
$(\mathcal{D}, st_{\mathcal{A}}, UTXO) \leftarrow \mathcal{A}_0(1^k)$
$(I, C) \leftarrow Enc_K(D)$
$(w_1, st_{\mathcal{A}}, T_{u1}) \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, I, C, UTXO)$
$(T_{w_1}, Appoint_1) \leftarrow Srchtoken(K, w_1, T_{u1})$
$T_{s1} \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, UTXO)$
$(ask_1, withdraw_1/pay_1) \leftarrow Search(T_{s1}, T_{w1}, K)$
$T_{x1}, T_{y1} \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, UTXO)$
$(Get_1, Prove_1/Fuse_1) \leftarrow Verify(K, T_{x1}, T_{y1}, I, C)$
$Redeem_1/compesa_1 \leftarrow Rede(Appoint_1, Get_1, Prove_1)$
for $2 \leq i \leq q$
$(w_i, st_{\mathcal{A}}, T_{ui}) \leftarrow \mathcal{A}_i(st_{\mathcal{A}}, I, C, \alpha_1, \cdots, \alpha_{i-1}, UTXO)$
$(T_{w_i}, Appoint_i) \leftarrow Srchtoken(K, w_i, T_{ui})$
$T_{si} \leftarrow \mathcal{A}_i(st_{\mathcal{A}}, UTXO)$
$(ask_i, withdraw_i/pay_i) \leftarrow Search(T_{si}, T_{wi}, K)$
$T_{xi}, T_{yi} \leftarrow \mathcal{A}_i(st_{\mathcal{A}}, UTXO)$
$(Get_i, Prove_i/Fuse_i) \leftarrow Verify(K, T_{xi}, T_{yi}, I, C)$
$Redeem_i/compesa_i \leftarrow Rede(Appoint_i, Get_i, Prove_i)$
Let $\alpha_i = (ask_i, withdraw_i/pay_i)(1 \leq i \leq q)$
Let $\alpha = (\alpha_1, \cdots, \alpha_q)$
Let $\beta_i = (Get_i, Prove_i/Fuse_i, Redeem_i/compesa_i)(1 \leq i \leq q)$
Let $\beta = (\beta_1, \cdots, \beta_q)$
Output $V = (I, C, \alpha)$ and $\beta$ and $st_{\mathcal{A}}$

**Fig. 4.** Experiment that adversary $\mathcal{A}$ plays in.

$\mathbf{Ideal}^{\Pi}_{A,S}(k)$

$(\mathcal{D}, st_{\mathcal{A}}, UTXO) \leftarrow \mathcal{A}_0(1^k)$
$(I, C, st_{\mathcal{S}}) \leftarrow \mathcal{S}_0(\tau(D))$
$(w_1, st_{\mathcal{A}}, T_{u1}) \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, I, C, UTXO)$
$(T_{w_1}, Appoint_1) \leftarrow \mathcal{S}_1(st_{\mathcal{S}}, \tau(D, w_1), T_{u1})$
$T_{s1} \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, UTXO)$
$(ask_1, withdraw_1/pay_1) \leftarrow S_1(st_{\mathcal{S}}, \tau(D, w_1), T_{w1}, T_{s1})$
$T_{x1}, T_{y1} \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, UTXO)$
$(Get_1, Prove_1/Fuse_1) \leftarrow S_1(st_{\mathcal{S}}, \tau(D, w_1), T_{x1}, T_{y1}, I, C)$
$Redeem_1/compesa_1 \leftarrow S_1(st_{\mathcal{S}}, \tau(D, w_1), Get_1, Prove_1)$
for $2 \leq i \leq q$
let $\alpha_i = (ask_i, withdraw_i/pay_i)$
$(w_i, st_{\mathcal{A}}, T_{ui}) \leftarrow \mathcal{A}_i(st_{\mathcal{A}}, I, C, \alpha_1, \cdots, \alpha_{i-1}, UTXO)$
$(T_{w_i}, Appoint_i) \leftarrow \mathcal{S}_1(st_{\mathcal{S}}, \tau(D, w_1, \cdots, w_i), T_{ui})$
$T_{si} \leftarrow \mathcal{A}_i(st_{\mathcal{A}}, UTXO)$
$(ask_i, withdraw_i/pay_i) \leftarrow Search(T_{si}, T_{wi}, K)$
$T_{xi}, T_{yi} \leftarrow \mathcal{A}_i(st_{\mathcal{A}}, UTXO)$
$(Get_i, Prove_i/Fuse_i) \leftarrow S_i(st_{\mathcal{S}}, \tau(D, w_1, \cdots, w_i), T_{xi}, T_{yi}, I, C)$
$Redeem_i/compesa_i \leftarrow S_i(st_{\mathcal{S}}, \tau(D, w_i), Get_i, Prove_i)$
Let $\alpha_i = (ask_i, withdraw_i/pay_i)(1 \leq i \leq q)$
Let $\alpha = (\alpha_1, \cdots, \alpha_q)$
Let $\beta_i = (Get_i, Prove_i/Fuse_i, Redeem_i/compesa_i)(1 \leq i \leq q)$
Let $\beta = (\beta_1, \cdots, \beta_q)$
Output $V = (I, C, \alpha)$ and $\beta$ and $st_{\mathcal{A}}$

**Fig. 5.** Experiment that adversary $\mathcal{A}$ and simulator $\mathcal{S}$ played in.

**Definition 4.** The blockchain-based SSE scheme satisfies fairness if the following properties hold:

- If both parties execute the protocol honestly, the user can get the right result and the cloud server also can get the service fee;
- If the user firstly terminates the search protocol or he is malicious, except losing the deposit, he also cannot get the right result. Moreover, the server also cannot get any information about plaintext.
- If the server is malicious, in addition to not receiving the service fee, it will be punished.

## 4. The detailed scheme

Suppose a data owner has $n$ documents $D_1, D_2, \ldots, D_n$ need to upload onto the server. In this section, we still use capital letter (e.g. $A$) to denote the pair of keys ($A.pk$, $A.sk$), and use $\lambda$ to denote the output length of $\delta.Enc$.

Now the proposed blockchain-based SSE scheme is as follows:

- ***Gen***: The data owner takes the security parameter $k$ as input, and outputs a secret key array $\boldsymbol{K} = (K_1, K_2, K_3, K_4, K_5)$, where $K_i \leftarrow \{0,1\}^k (i = 1, \ldots, 5)$.
- ***Enc***: At this stage, it will output ciphertexts and an invertible index.
  - Firstly, the data owner uses the private key $K_1$ to encrypt documents $D_i (1 \le i \le n)$:

$$C_i = \varepsilon.Enc_{K_1}(D_i)(1 \le i \le n) \tag{2}$$

$$MAC(C_i) = H(K_5, C_i) \tag{3}$$

  He then sets $\boldsymbol{C} \leftarrow ((C_1, MAC(C_1)), \ldots, (C_n, MAC(C_n)))$.
  - In order to generate an index, the data owner firstly extracts keywords from document collection $\boldsymbol{D}$. Suppose the keyword collection is $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$. For each keyword $w_i \in \mathcal{W}$, he chooses an empty array $DB(w_i)$ of size $n$. He then assigns $DB(\cdot)$ in this way: If $j$th document contains keyword $w_i$, then $DB(w_i)[j] = 1$, otherwise, $DB(w_i)[j] = 0$. Next, he computes:

$$t_{w_i} = F_1(K_2, w_i \parallel 0) \tag{4}$$

$$k_{w_i} = F_1(K_2, w_i \parallel 1) \tag{5}$$

$$e_{w_i} = \delta.Enc(k_{w_i}, DB(w_i)) \tag{6}$$

$$K_{w_i} = F_1(K_3, w_i) \tag{7}$$

$$\mathcal{M}ac_{w_i} = H(K_{w_i} \parallel DB(w_i)) \tag{8}$$

  The data owner puts $(t_{w_i}, e_{w_i}, \mathcal{M}ac_{w_i})$ into $\boldsymbol{I}$ of length $m \cdot (2k + \lambda)$ with lexicographically order. At last, he uploads ***(C, I)*** onto the cloud.
- ***Srchtoken***: Suppose the data owner shares key array ($K_1$, $K_2$, $K_3$, $K_4$, $K_5$) with a user. In order to search documents that contain keyword $w$, the user computes:

$$t_w = F_1(K_2, w \parallel 0), \tag{9}$$

$$k_w = F_1(K_2, w \parallel 1), \tag{10}$$

$$k_{31} = F_1(K_4, w), \tag{11}$$

$$T_w = \delta.Enck(k_{31}, t_w \parallel k_w \parallel H(k_{31})), \tag{12}$$

  Then, he sends $T_w$ to the server, and waits for reply.

  Let $D(\cdot)$ be a contract built by the server, which can be invoked by anyone. In this contract it mainly performs hash operations, that is to say, when inputting $(x, y)$, this contract can verify $h(x) \stackrel{?}{=} y$.

  Meanwhile, the user builds a transaction *Appoint* shown in Fig. 6 in the following way:

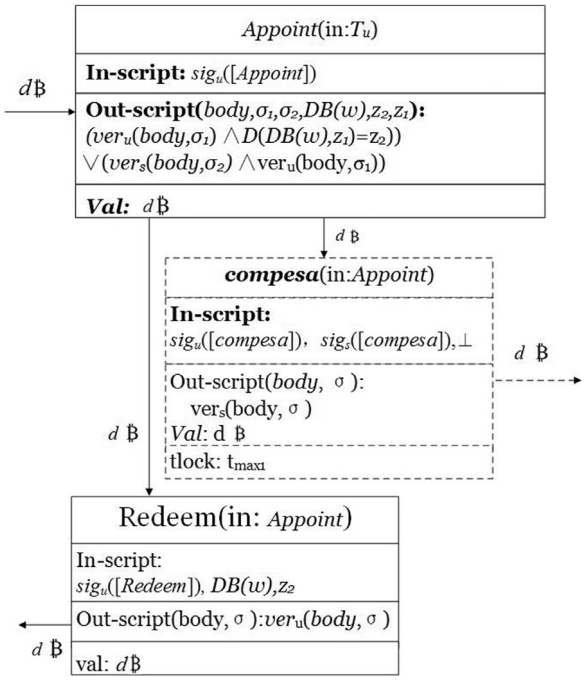


**Fig. 6.** The process of the user deposits.

- Finds an unredeemed transaction $T_u$ of value $d\overset{\cdot\cdot}{\mathbb{B}}$, whose receiver is himself.
- Embeds $D(\cdot)$ into the out-script of transaction *Appoint*.
- Computes the body of transaction *Appoint*.
- Both the user and the server compute the body of the transaction *compesa* by taking transaction *Appoint* as input. Then the user sends his signature of transaction *compesa* to the server, who will add signature in it. Here the transaction *compesa* has a time-lock $t_{max1}$, it means that after time $t_{max1}$, the server can broadcast transaction *compesa*.
- The user signs transaction *Appoint*, and broadcasts it on the blockchain.
- If transaction *Appoint* does not appear on the blockchain until $t_{max1} - max_0$, the user can immediately redeem transaction $T_u$ by using his private key and quits the protocol, where the $max_0$ means the maximal possible delay of including *Appoint* into the blockchain.

Let $\mathcal{V}(x, y)$ denote another contract created by the user. It will firstly perform a decryption algorithm $\delta$, then executes a hash verification.

- ***Search***: After receiving $T_w$, the server establishes a transaction *ask* shown in Fig. 7 to get the decryption key $k_{31}$ of $T_w$ in the following way:
  - Find an unredeemed transaction $T_{s1}$ of value $d\overset{\cdot\cdot}{\mathbb{B}}$, whose receiver is the server.
  - Embeds $\mathcal{V}(k_{31}, T_w)$ into the out-script of transaction *ask*.
  - Computes the body of transaction *ask*.
  - Both the server and the user compute the body of the transaction *withdraw* by taking transaction *ask* as input. The user sends his signature of transaction *withdraw* to the server to add its signature in it. The transaction *withdraw* has a time-lock *t*, it means that after time *t*, the server can broadcast transaction *withdraw*.
  - The server signs transaction *ask* and broadcasts it on the blockchain.
  - If transaction *ask* does not appear on the blockchain until $t - max_1$, where the $max_1$ means the maximal possible delay of including *ask* into the blockchain, the server can immediately redeem transaction $T_{s1}$ by using its private key and quits the protocol.

The user computes the body of transaction *Pay* by using transaction *ask*, and embeds $k_{31}$ into the in-script of transaction *Pay*. After signing it, he broadcasts transaction *Pay*.

The nodes collect transaction *Pay* in the P2P network, and verify it in this way:

- Decrypts $T_w$ by using key $k_{31}$: $t_w \parallel k_w \parallel H(k_{31}) = \delta.Dec(k_{31}, T_w)$;
- Verifies whether $H(\tilde{k}_{31}) \stackrel{?}{=} H(k_{31})$ holds or not. If it holds, the transaction *Pay* will be accepted, otherwise it will be rejected.

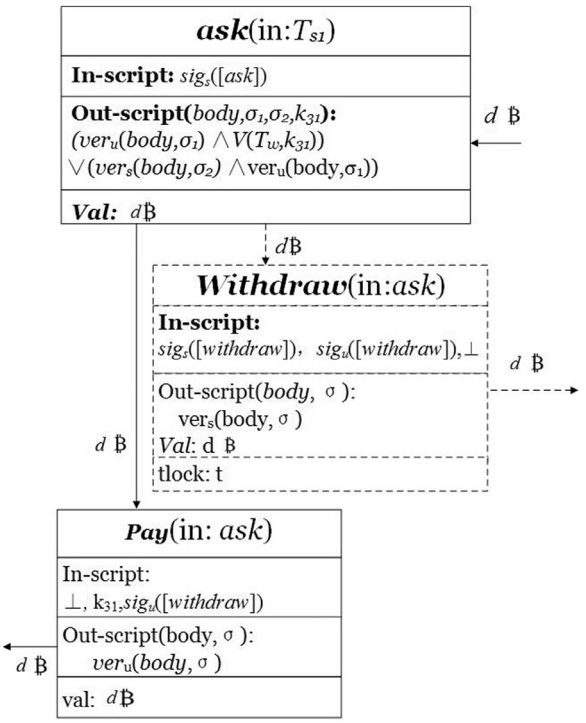


**Fig. 7.** The process of getting the decryption key of search token $T_w$.

If the transaction *Pay* does not appear on the blockchain until time $t$, the server will broadcast the transaction *withdraw* and gets his money back.

- ***Verify***: In order to redeem $d$฿ and charge $d_1$฿ service fee from the user, the server will do:
  - Gets $t_w \| k_w$ from transaction *Pay*, and finds $(t_w, e_w, \mathcal{M}ac_w)$ from the index $I$ by using $t_w$.
  - Decrypts $e_w$ by using $k_w$: $DB(w) = \delta.Dec(k_w, e_w)$.
  - Chooses an empty set $C_w$, and sets it in this way: If $DB(w)[j] = 1$, it puts document $(C_j, MAC(C_j)$ into $C_w$.

Let $V_1(x_1, x_2, x_3, x_4, \{y_i, mac_{y_i}\}, z_1, z_2)$ be a hash verification algorithm, which also can be seen as a smart contract. It takes $x_1, x_2, x_3, x_4, \{y_i, mac_{y_i}\}$ as input, and verifies whether $x_2 \stackrel{?}{=} H(x_3 \| x_1)$, $mac_{y_i} \stackrel{?}{=} H(x_4 \| y_i)$ and $z_2 \stackrel{?}{=} H(x_1 \| z_1)$ hold or not. If they hold, it outputs 1, otherwise it is 0.

The user constructs a transaction *Get* to get the search results shown in Fig. 8 as follows:

- Finds two unredeemed transactions $T_{u1}$ of value $d$**B**and $T_{u2}$ of value $d_1$**B**, whose receiver is the user.
- Embeds $V_1(DB(w), Mac_w, K_w, K_5, C_w, z_1, z_2)$ in the out-script of transaction *Get*, where $z_1$ is a session key generated by server, and $z_2 = H(DB(w) \| z_1)$.
- Takes $T_{u1}$ and $T_{u2}$ as input, and computes the body of transaction *Get*.
- Both the user and the server compute the body of transaction *Fuse* by taking transaction *Get* as input. Then, the server sends the signature of transaction *Fuse* to the user, who will add his signature into it. The transaction has a time-lock $t_1$, it means that the user can broadcast transaction *Fuse* after time $t_1$.
- The user signs transaction *Get* and broadcasts it on the blockchain.
- If transaction *Get* is not appeared in the blockchain within time $t_1 - max_3$, where the $max_3$ means the maximal possible delay of including *Get* into blockchain, the user can immediately redeem $T_{u1}$ and $T_{u2}$ and quits the protocol.

The user computes the body of transaction *Prove* by using transaction *Get*, and puts $K_5$, $K_w$ into the in-script of transaction *Prove*. Then, he sends his signature of transaction *Prove* to the server.

The server takes the transaction *Get* as input, and computes the body of transaction *Prove*. It then adds $DB(w)$, $Mac_w$, $C_w$, $z_1$, $z_2$ into the in-script of transaction *Prove*. After signing it, the server broadcasts the transaction *Prove*.

The nodes collect the transaction *Prove* in the P2P network, and do:

- verify whether $\mathcal{M}ac_w \stackrel{?}{=} H(K_w \| DB(w))$, $\mathcal{MAC}_{C_i} \stackrel{?}{=} H(K_5 \| C_i)$ and $z_2 \stackrel{?}{=} H(DB(w) \| z_1)$ hold or not. If they hold, the transaction *Prove* will be accepted.

If the transaction *Prove* does not appear on the blockchain until time $t_1$, the user can broadcast transaction *Fuse* to redeem transaction *Get*.

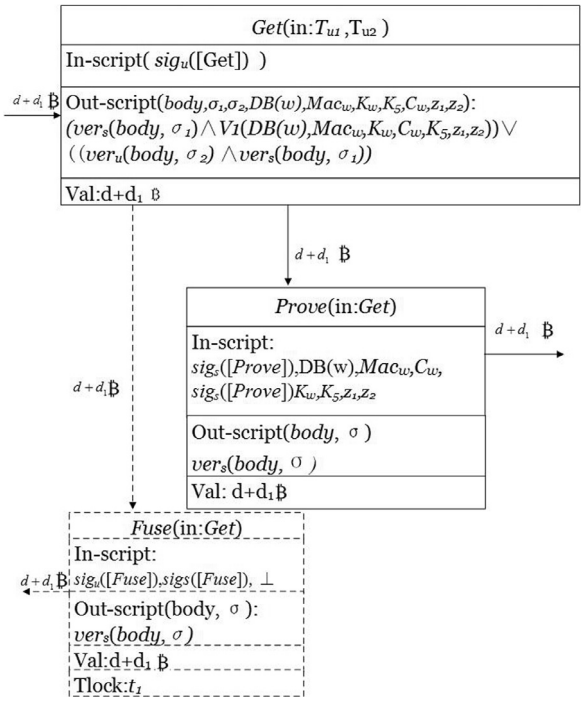


**Fig. 8.** The process of returning and verifying the search results.

- *Rede*: The user takes transaction *Appoint* as input, and puts $z_1$, $z_2$, $DB(w)$ into the in-script of transaction *Redeem* shown in Fig. 6. After signing it, he broadcasts it onto the blockchain. The nodes on the P2P network do:
  - Verify whether $z_2 \stackrel{?}{=} H(DB(w)||z_1)$ holds or not.
  - If it does, it outputs 1, which means the transaction *Redeem* will be accepted.

  If the transaction *Redeem* does not appear on the blockchain until $t_{max}$, the server broadcasts transaction *compesa* shown in Fig. 6 to redeem transaction *Appoint*.
- *Dec*: If the transaction *Prove* appears on the blockchain, the user can reads $\{C_j\}$ from it. Then, he uses the secret key $K_1$ to decrypt $C_j$: $D_j = \varepsilon.Dec(K_1, C_j)$.

Lastly, the user updates the $MAC(C_i)$ for each ciphertext $C_i$ by using a new key $K_5$, and deletes the old $MAC(C_i)$ that stored on the cloud.

## 5. Security and performance analysis

### 5.1. Security analysis

In this section,we will give two theorems to prove that our scheme is secure and fair.

**Theorem 1.** *If $F_1$, $F_2$ are pseudorandom functions, H is a collision resistant hash function, and $\varepsilon = (\varepsilon.Enc, \varepsilon.Dec)$ is IND-CPA-secure SE scheme, then the scheme we present above is adaptively secure.*

**Proof 1.** We need to construct a *PPT* simulator $\mathcal{S} = \{\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_q\}$ for the adversary such that $\mathcal{A} = \{\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_q\}$, the output of $Real_A^{\pi}(k)$ and $Ideal_{A,S}^{\pi}(k)$ are computationally indistinguishable.

Suppose that the simulator $\mathcal{S}$ is given the trace $\tau$ of documents $D$, then it can generate ($\mathbf{I}^*, \mathbf{C}^*$, *Appoint**, *Redeem**/*compesa**, *ask**, *Withdraw**/*Pay**, *Get**, *Prove**/ *Fuse**) as follows:

- If $q = 0$, the simulator $\mathcal{S}_0$ can set $I^*$ be a random strings whose size is $|\mathcal{I}|$. $\mathcal{S}_0$ puts the $I^*$ in the state $st_S$, and set $C_i^* \leftarrow \{0,1\}^{|D_i|}$. Besides it sets $MAC^*(C_i) \leftarrow \{0,1\}^k$ at random. Since the state $st_{\mathcal{A}_0}$ does not have the key $K_2$, $K_3$, $K_4$, the simulator will uniformly choose $t_w^* \leftarrow \{0,1\}^k$, $k_w^* \leftarrow \{0,1\}^k$, and $\mathcal{M}ac_w{}^* \leftarrow \{0,1\}^k$ at random for each keyword $w$. It also chooses $k_{31}^* \leftarrow \{0,1\}^k$, and sets $T_w^* = \delta.Enc(k_{31}^* \parallel t_w^* \parallel k_w^* \parallel H(k_{31}^*))$. The adversary $\mathcal{A}_0$ chooses some unredeemed transactions $T_u^*, T_{s1}^*, T_{u1}^*, T_{u2}^*$. Because the state $st_{\mathcal{A}_0}$ does not have the private key used to generate transactions, so the simulator chooses key pairs ($u^*.sk$, $u^*.pk$) and ($s^*.sk$, $s^*.pk$) randomly. Then, it builds transaction *Appoint** by using the private key $u^*.sk$, transaction *ask** by using the private key $s^*.sk$ and $T_w^*$, transaction *Pay** by using the private key $u^*.sk$ and $k_{31}^*$ (or transaction *withdraw* by using private keys $u^*.sk$ and $s^*.sk$), transaction *Get** by using the private key $u^*.sk$

and $V_1(DB(w)^*, \mathcal{M}ac_w^*, K_w^*, K_5^*, C_w^*, z_1^*, z_2^*)$, where $DB(w)^* \leftarrow \{0,1\}^n$, $K_w^* \leftarrow \{0,1\}^k$, $\mathcal{M}ac_w^* = H(K_w^*||DB(w)^*)$, $K_5^* \leftarrow \{0,1\}^k$, $C_j^* \leftarrow \{0,1\}^n$, $MAC(C_j^*) = H(K_5^*, C_j^*)$, $z_1^* \leftarrow \{0,1\}^k$ and $z_2 = H(DB(w)^*||z_1)$.

Since the state $st_{\mathcal{A}_0}$ does not have the keys $K_3$, $K_5$, therefore, the adversary $\mathcal{A}_0$ cannot build a valid transaction *Prove*$^*$ to redeem transaction *Get*$^*$. It also cannot build a valid transaction *Fuse*$^*$ to redeem transaction *Get*$^*$.

Because $\varepsilon$ is IND-CPA secure, the adversary cannot distinguish $C_i^*$ from $C_i$ that generated in the real experiment. Because $F_1$ is a pseudorandom function and $H$ is collision-resistant, the adversary cannot distinguish $(t_w^*, e_w^*, \mathcal{M}ac_w^*)$ from true $(t_w, e_w, \mathcal{M}ac_w)$ generated in *Enc* step. Therefore, the index $I^*$ is undistinguishable from true index $I$ generated in *Enc* step.

- When $q = 1$, the $\mathcal{S}_1$ can read true $DB(w_1), \mathcal{M}ac_{w_1}, C_{w_1} = \{C_j, MAC(C_j)\}, z_2$ from trace $\tau$ of documents $D$, then it can set $e_{w_1}^* = \delta.Enc(k_{w_1}^*, DB(w_1))$, where $k_{w_1}^* \leftarrow \{0,1\}^k$. It puts $(t_{w_1}^*, e_{w_1}^*, \mathcal{M}ac_{w_1})$ into $I^*$, where $t_{w_1}^* \leftarrow \{0,1\}^k$. The $\mathcal{S}_1$ sets $T_{w_1}^* = \delta.Enc(k_{31}^* \parallel t_{w_1}^* \parallel k_{w_1}^* \parallel H(k_{31}^*))$, where $t_{w_1}^* \leftarrow \{0,1\}^k, k_{w_1}^* \leftarrow \{0,1\}^k$ and $k_{31}^* \leftarrow \{0,1\}^k$. Then, it builds a transaction *ask*$^*$ that embedded function $V(T_{w_1}^*, k_{31}^*)$.

  Because the adversary $\mathcal{A}_1$ does not have the private key, it cannot give a valid signature for transaction *Pay*$^*$, as well as for transaction *withdraw*$^*$. Therefore, it cannot use transaction *Pay*$^*$ or *withdraw*$^*$ to redeem transaction *ask*$^*$.

  The simulator $\mathcal{S}_1$ embeds $V_1(DB(w_1), \mathcal{M}ac_{w_1}, K_{w_1}^*, K_5^*, C_{w_1}, z_1^*, z_2)$ into transaction *Get*$^*$ and broadcasts it to the blockchain, where $K_5^* \leftarrow \{0,1\}^k$ and $z_1^* \leftarrow \{0,1\}^k$.
  Because the adversary $\mathcal{A}_1$ does not have the key array $\mathbf{K}$, it cannot build a valid signature for transaction *Prove*$^*$, as well as *Fuse*$^*$. Therefore, it cannot redeem the transaction *Get*$^*$ normally.

  Because the $F_1$ is undistinguishable from function $f$: $\{0,1\}^k \rightarrow \{0,1\}^k$, and $H$ is collision resistant, we can get $I^*$ is undistinguishable from $I$ generated in *Enc* step.
- For $2 \le i \le q$: The simulator $\mathcal{S}_i$ checks whether $w_i$ was queried or not. If $w_i = w_j (1 \le j \le i-1)$, it sets $\sigma(i,j) = 1$ in $\tau$. If it does not be queried, it generates transactions *ask*$^*$ and *Get*$^*$ in the same way that $\mathcal{S}_1$ does. If $w_i$ did previously appear, the $\mathcal{S}_i$ returns the corresponding transactions *ask*$^*$, *Get*$^*$ previously used for $w_i$. The adversary $\mathcal{A}_i$ builds the appropriate transactions *Pay*$^*$/*withdraw*$^*$ and *Prove*$^*$/*Fuse*$^*$.
  Because the adversary does not have the key array $\mathbf{K}$, it cannot build a valid signature for transactions *Pay*$^*$ and *Prove*$^*$, as well as *withdraw*$^*$ and *Fuse*$^*$. Therefore, it cannot redeem the transactions *ask*$^*$ and *Fuse*$^*$ normally. □

**Theorem 2.** *If the blockchain is irreversible, our scheme can satisfy fairness.*

- If the user is not honest, it means that the transaction *Pay* cannot be accepted by the blockchain. As a result, the server cannot get $t_w$, $k_w$ which were used to find the search results. During this process, the user will be penalized, while the server cannot get any information about plaintexts.
- It the server is not honest, it either means the value $T_w$ embedded in transaction *ask* is wrong, or it does not provide right results. For the former, the server cannot get any information about plaintext; for the latter, it cannot get the service fee.
- If both of them honestly execute the protocol, the user can get right results from transaction *Prove*, and the server can get service fee as well.

### 5.2. Performance analysis

In this section, we will illustrate the practicality of our construction through experiments. These experiments are implemented in Java and Go and consist of 751 lines of codes which is not optimized. The reason that we use different platforms is that the constructions of current transaction and smart contract do not support Java language flexibly.

Our system configuration is Intel(R) Core (TM) $I7-8700k@3.70$ GHz, 4GB RAM. We instantiate pseudorandom functions $F_1$, $F_2$ with $HMAC-SHA256$, the keyed hash function $H$ with $HMAC-SHA256$, and the SE scheme $\varepsilon$, $\delta$ with *AES* in the CTR mode with a 256 bit key. The type of the test data we chose was (*w, ind*), where *w* denoted a keyword and *ind* represented a file identifier. The number of (*w, ind*) pairs ranged from 5000 to $2 \times 10^4$.

We will use the following characteristics to show feasibility of our scheme. The first is setup time, that is, how long it takes to produce an invertible index. The second is the time used to generate a search token for a single keyword *w*. The third is the search time needed to finish a search task.

**Setup time**. Since the construction of encrypted Index table does not depend on the blockchain, we implement it in Java. Firstly, we extracted the keyword set from the test data with different size, and classified the file identifiers based on these keywords. To generate the index, we invoked *AES* two times and *SHA*256 four times. From Fig. 9, we can see that the setup time increased linearly in the number of (*w, ind*) pairs.

**Search token generation time**. Here we only focus on single keyword search. To generate a search token, the user needs to use pseudorandom functions $F_1$ three times, hash function $H$ once, and *SE* once. Therefore, we invoked *SHA*256 four times, and *AES* once. As shown in Fig. 10, we can get that the generation time of a search token was not depended on the number of (*w, ind*) pairs. It took about 1 ms to generate a search token for a keyword with size 1 kB.

**Search time**. When search, the user needs to interact with the server on the blockchain and build transactions six times. Therefore, we put the search process on the Fabric 1.0 ran on a VMware Workstation on the ubuntu 16.04*LTS* system. We

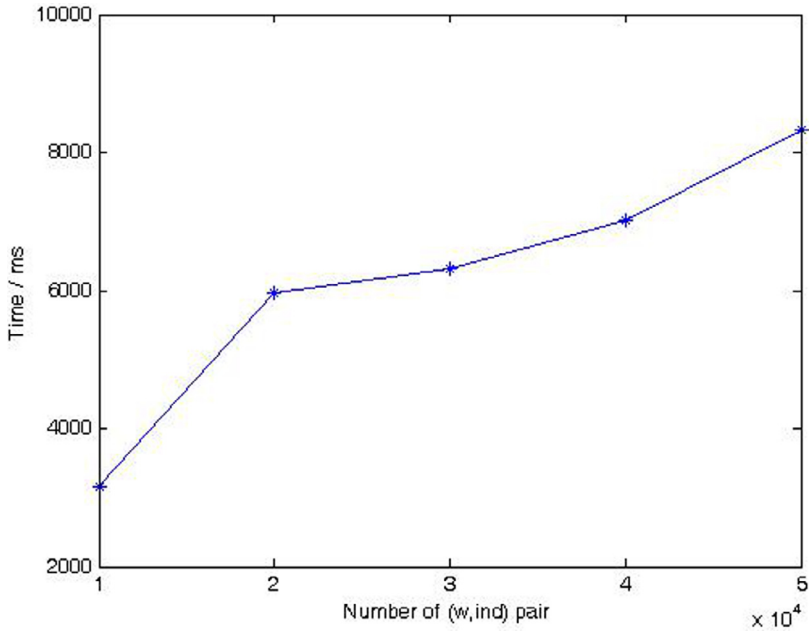


**Fig. 9.** Time of building an invertible index for keyword/identifier pairs with different size.

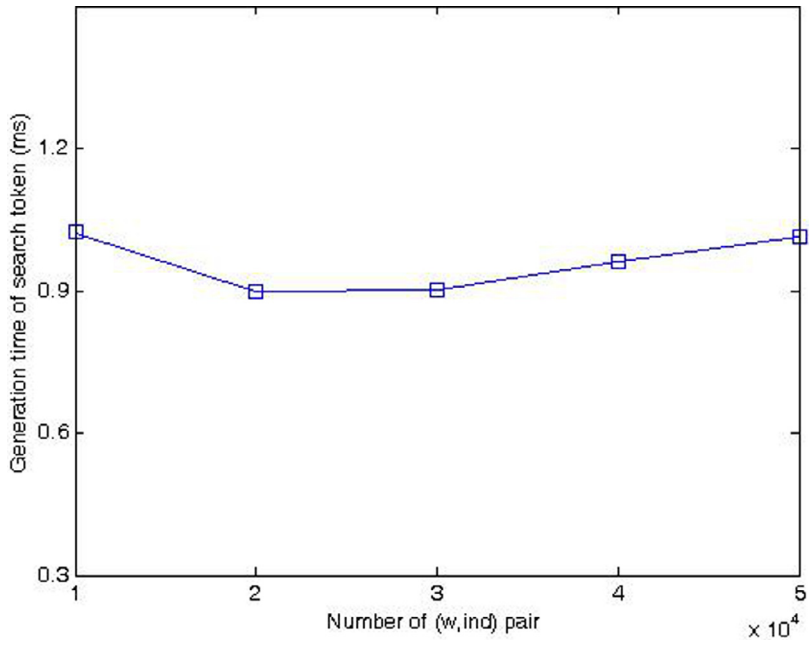


**Fig. 10.** Time of building a search token for a single keyword.

instantiated the smart contract $D(\cdot)$ with *SHA*256, $\mathcal{V}(x, y)$ with a combination of *AES* and *SHA*256, and $V_1(\ldots)$ with a function invoked *SHA*256 three times on the Fabric. The language we used is Go. In this process, the server also needs to decrypt $T_w$ into $t_w$, $k_w$, and uses $t_w$, $k_w$ to find the search results from index $I$ locally. Therefore, the search time is equal to the time it took to find the search results locally plus the time spent by 6 transactions from creation to finish. Fig. 11 shows the search time was linear with the number of $(w, ind)$ pairs. Since these contracts can verify the result automatically, the users can reduce their calculations locally. Moreover, in this process, only the user provided correct results in transactions *Redeem*, he could redeem his deposit. Similarly, only the server provided right search results, it could charge the service fee.

Once the transactions we built invokes the smart contracts, it can automatically judge whether the results embedded in them are valid. If they are not accepted by the blockchain, the user cannot get the search results or will be punished,

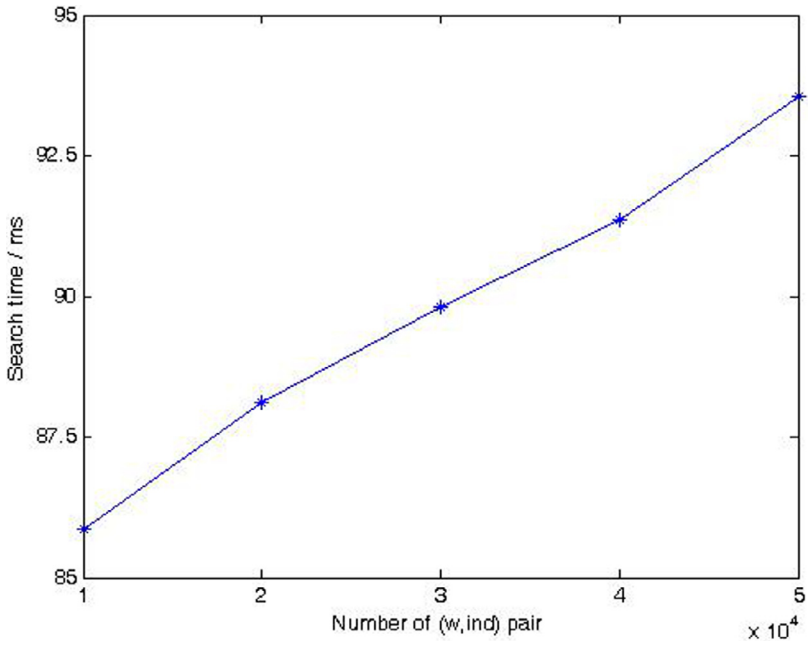


**Fig. 11.** Time of returning the search results to user.

while the server cannot obtain any information about plaintexts. Therefore, compared with [4,15], our scheme can guarantee fairness and also can resist the malicious user.

## 6. Conclusion

Although the blockchain is decentralized, it can automatically protect the rights of users. Therefore, based on this technology, we give a feasible solution to solve the fairness problem happened in the searchable symmetric encryption. From our scheme, the user can obtain the search results automatically without verification. If the server is malicious, except losing the deposit, it will not be able to obtain the service fee.

Because our scheme can safeguard the interests of users, it has many practical scenarios, such as in medical system, securities company, and so on. In addition, our scheme can not only support single keyword search, but also can be extended to multiple keywords search. However, the transaction time on the blockchain is long, it makes the search time in our experiment test phase low. Therefore, our next work is to design a new cryptography currency system, which can deal with our scheme cheaply and quickly.

## Acknowledgments

This work is supported by the National Key R&D Program of China (2017YFB0802503), the National Natural Science Foundation of China (no. 61672550), the Fundamental Research Funds for the Central Universities (no. 17lgjc45), and Guangxi Key Laboratory of Cryptography and Information Security (no. GCIS201711).

## References

[1] Song DX, Wagner D, Perrig A. Practical techniques for searches on encrypted data. In: 2000 IEEE symposium on security and privacy, Berkeley, California, USA; 2000. p. 14–17. May
[2] Goh E. J.. Secure indexes, IACR cryptology eprint archive 2003. 2003. https://gnunet.org/sites/default/files/secureindex.pdf. 216.
[3] Curtmola R, Garay J, Kamara S, Ostrovsky R. Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM conference on computer and communications security. ACM; 2006. p. 79–88.
[4] Kamara S, Papamanthou C. Parallel and dynamic searchable symmetric encryption. In: International conference on financial cryptography and data security. Springer; 2013. p. 258–74.
[5] Cash D, Jaeger J, Jarecki S, Jutla CS, Krawczyk H, Rosu MC, Steiner M. Dynamic searchable encryption in very-large databases: Data structures and implementation. In: 21st annual network and distributed system security symposium, NDSS 2014, San Diego, California, USA; 2014. p. 23–6. February
[6] Moataz T, Shikfa A. Boolean symmetric searchable encryption. In: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. ACM; 2013. p. 265–76.
[7] Cash D, Jarecki S, Jutla C, Krawczyk H, Roşu MC, Steiner M. Highly-scalable searchable symmetric encryption with support for boolean queries. In: Advances in cryptology–CRYPTO 2013. Springer; 2013. p. 353–73.
[8] Boldyreva A, Chenette N. Efficient fuzzy search on encrypted data. In: Fast software encryption. Springer; 2014. p. 613–33.
[9] Li J, Wang Q, Wang C, Cao N, Ren K, Lou W. Fuzzy keyword search over encrypted data in cloud computing. In: INFOCOM, 2010 proceedings IEEE. IEEE; 2010. p. 1–5.

[10] Wang B, Yu S, Lou W, Hou YT. Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In: INFOCOM, 2014 proceedings IEEE. IEEE; 2014. p. 2112–20.
[11] Stefanov E, Papamanthou C, Shi E. Practical dynamic searchable encryption with small leakage. NDSS 2014;14:23–6.
[12] Bost R., Fouque P.A., Pointcheval D.. Verifiable dynamic symmetric searchable encryption: optimality and forward security. In: Cryptology ePrint archive: report 2016/062 (2016). https://eprint.iacr.org/2016/062.
[13] Bost R. $o\varphi o\varsigma$: forward secure searchable encryption. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. ACM; 2016. p. 1143–54.
[14] Alderman J, Martin KM, Renwick SL. Multi-level access in searchable symmetric encryption. In: International conference on financial cryptography and data security; 2017. p. 35–52.
[15] Kurosawa K, Ohtaki Y. Uc-secure searchable symmetric encryption. In: Financial cryptography and data security. Springer; 2012. p. 285–98.
[16] Cheng R, Yan J, Guan C, Zhang F, Ren K. Verifiable searchable symmetric encryption from indistinguishability obfuscation. In: Proceedings of the 10th ACM symposium on information, computer and communications security, ASIA CCS '15, Singapore; 2015. p. 14–17. April
[17] Goldreich O, Ostrovsky R. Software protection and simulation on oblivious rams. J ACM 1996;43(3):431–73.
[18] Nakamoto S.. Bitcoin: a peer-to-peer electronic cash system. http://www.cryptovest.co.uk/resources/Bitcoin%20paper%20Original.pdf.
[19] Reid F, Harrigan M. An analysis of anonymity in the bitcoin system. In: Security and privacy in social networks. Springer; 2013. p. 197–223.
[20] Eyal I, Gencer AE, Sirer EG, Renesse RV. Bitcoin-ng: a scalable blockchain protocol. In: 13th USENIX symposium on networked systems design and implementation (NSDI 16); 2016. p. 45–59.
[21] Garay J, Kiayias A, Leonardos N. The bitcoin backbone protocol: analysis and applications. In: Annual international conference on the theory and applications of cryptographic techniques. Springer; 2015. p. 281–310.
[22] Andrychowicz M, Dziembowski S, Malinowski D, Mazurek L. Secure multiparty computations on bitcoin. In: 2014 IEEE symposium on security and privacy. IEEE; 2014. p. 443–58.
[23] Andrychowicz M, Dziembowski S, Malinowski D, Mazurek L. Fair two-party computations via bitcoin deposits. In: International conference on financial cryptography and data security. Springer; 2014. p. 105–21.
[24] Bentov I, Kumaresan R. How to use bitcoin to design fair protocols. In: International cryptology conference. Springer; 2014. p. 421–39.
[25] Buterin V.. A next-generation smart contract and decentralized application platform. https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf.

**Huige Li** received her M.S degree from the School of Mathematics and Information Science, Shaanxi Normal University in 2013. She is currently reading for her Ph.D. at the school of Electronics and Information Technology of Sun Yat-sen University, China. Her research focuses on Searchable Encryption.

**Haibo Tian** received his Ph.D. from the School of Communication Engineering, Xidian University in 2006. He is currently an associate Professor at the School of Data and Computer Science of Sun Yat-sen University, China. His research mainly focuses on security protocol analysis and its design.

**Fangguo Zhang** received his Ph.D. from the School of Communication Engineering, Xidian University in 2001. He is currently a Professor at the School of Data and Computer Science of Sun Yat-sen University, China. He is the co-director of Guangdong Key Laboratory of Information Security Technology. His research mainly focuses on cryptography and its applications.

**Jiejie He** received his B.E. degree from the School of Information Technology, Minzu University of China in 2015. At present, he is reading for his M.E. at the school of Data and Computer Science of Sun Yat-sen University, China. His research focuses on Blockchain and its applications.