Blockchain Based Multi-keyword Similarity Search Scheme over Encrypted Data

Mingyue Li[1,2], Chunfu Jia[1,2(✉)], and Wei Shao[1,2]

[1] College of Cyber Science, Nankai University, Tianjin, China
{limingyue,wei.shao}@mail.nankai.edu.cn, cfjia@nankai.edu.cn
[2] Tianjin Key Laboratory of Network and Data Security Technology, Nankai University, Tianjin, China

Abstract. Traditional searchable encryption schemes focus on preventing an honest-but-curious server. In practice, cloud servers may delete user data, perform partial queries and even falsify search results to save computing and storage resources. Although there is some previous work to verify the correctness of search results, these verification mechanisms are highly dependent on the specially appointed index structures.

In this paper, we propose a blockchain based multi-keyword similarity search scheme over encrypted data (BMSSED), which is a general scheme that keeps users from worrying about potential misbehaviors of a malicious server. To solve the problem that the size of transactions is limited, we use an index partition method to divide the traditional binary tree index into a plurality of sub-indexes. The new structure of sub-indexes not only circumvents the *gasLimit* problem, but also reduces the dimension of file vectors and improves the search efficiency using smart contracts. In addition, we propose an access control mechanism for transaction data, which is implemented by a new smart contract. It can reduce the computation burden of data owners and prevent the leakage of confidential information. We then define the security model and conduct repeated experiments on real data sets to test the efficiency. Experimental results and theoretical analysis show the practicability and security of our scheme.

Keywords: Searchable encryption · Blockchain · Ethereum · Smart contract · Access control.

1 Introduction

With the rapid development of computer technology and Internet applications, the demand for data access and information storage is increasing [1]. Although cloud computing enables users to enjoy high-quality services and ubiquitous network access, outsourcing data to cloud servers makes data owners lose the control over their sensitive data, resulting in data privacy disclosure [2,3]. To

N. Park et al. (Eds.): SecureComm 2020, LNICST 336, pp. 350–371, 2020.
https://doi.org/10.1007/978-3-030-63095-9_23

protect data privacy, the data owners usually opt to encrypt their data before outsourcing to clouds. This in turn limits the availability of encrypted data, e.g., the widely used keyword retrieval technology of plaintext information cannot be directly applied to encrypted data. To address this problem, Song et al. [4] firstly proposed a searchable encryption scheme based on ciphertext scanning that enables users to store encrypted data in clouds and perform keyword searches on the ciphertext domain. Their scheme is simple to implement and has virtually no additional storage overhead. Subsequently, much effort has been devoted to designing effective mechanisms to enable search over encrypted data.

Traditional searchable encryption (SE) schemes focus on problems brought by an honest-but-curious server. More severely, any insider attacker may use discovered vulnerabilities and unlawfully alter the computations performed over the outsourced data. Recently, there is some work proposing verifiable designs that enable the data owner to verify the correctness and integrity of search results and outsourced data (e.g., using MHST [5] or verifiable matrix [6]). Unfortunately, these verification mechanisms are highly dependent on specially appointed index structures and do not support expressive queries or complex data structures (e.g., fuzzy search [7], similarity search [8,10] or multimedia data [9]).

With the emergence of blockchain, symmetric searchable encryption (SSE) scheme based on Bitcoin was proposed to guarantee the privacy and confidentiality of data. However, the transaction cycle of Bitcoin system is long. Besides, its script language is not Turing complete and cannot be applied to more scenarios. Therefore, Hu et al.[11] proposed a symmetric searchable encryption scheme based on Ethereum blockchain and smart contract. This scheme not only ensures the privacy of data, but also solves the problem of fairness of retrieval. Subsequently, there is some work based on Ethereum [12]. Nevertheless, they focus on single-keyword queries and cannot meet actual demands of users. More importantly, all of the above schemes control who has access to the data through data owners. That is to say, if Bob wants to access transactions of Alice, he has to send the query keywords to Alice. Alice asserts Bob have permissions according to the access control list stored locally and generates trapdoor for executing query (or Alice sends the trapdoor to Bob). Meanwhile, Alice has to pay for $Search(\cdot)$ function in advance. There is no doubt that Alice has a huge computational burden, which violates the original intention of outsourcing.

1.1 Our Contributions

To address the above concerns, we propose a blockchain based multi-keyword similarity search scheme over encrypted data (BMSSED), utilizing smart contracts in Ethereum [11] to allow users of different roles to request access permission and interact with confidential documents. To give an exemplary instantiation, we build BMSSED on the classic binary tree index [10] and design the corresponding smart contract to circumvent various barriers in Ethereum. For example, the size of the binary tree index increases as the amount of data increases. If the size of an index is larger than the maximum of $gasLimit$, then blockchain network will reject the transaction. Therefore, we divide the binary

tree in [10] to break the limit (see discussion in Subsect. 3.1 for details). In addition, to reduce the computation burden of client and simplify the query process, we novelly use smart contract to create an access control mechanism, where data owners write access control policy, roles, permissions and other information of data users to a smart contract and install it on each node. Before executing the $Search(\cdot)$ function, the access control policy is executed in advance. Note that our framework is a general one, which can be better applied to many practical scenarios (e.g., management of personal electronic medical profiles). Generally, the main contributions are as follows.

- We design a new index partition method to avoid exceeding $gasLimit$ and improve the query efficiency. If the size of an index tree is greater than the maximum value of $gasLimit$, we divide the traditional binary tree index into multiple sub-indexes and embed each sub-index into transactions. After that the transactions are sorted in ascending order of their keys, added to blocks in turn and written to disk in block order.
- Considering the complicated query process and the huge computational burden of data owners, we novelly use smart contracts to create an access control mechanism. Meanwhile, to simplify the access control mechanism and avoid the complexity of individual authorization policies, we use an effective role-based access control (RBAC) strategy to check whether users have access to transaction data.
- We accomplish a prototype of our scheme and also deploy it to a locally simulated network and an Ethereum test network as [11]. We conduct repeated experiments on real data sets. Experimental results and theoretical analysis show the practicability and security of our scheme over encrypted data.

1.2 Related Work

To fulfill the retrieval of encrypted data, Song et al. [4] firstly proposed a searchable encryption scheme based on ciphertext scanning, where each plaintext is divided into equal length keywords. This scheme uses the double-layered XOR encryption to encrypt each keyword in plaintext by the stream cipher. Authorized users only provide the encrypted keywords to the server. The server makes a linear query for the encrypted files and returns the search results. The cryptographic model of the scheme is simple. There is no additional storage space overhead. However, the statistical distribution of plaintext is vulnerable to attack. In addition, retrieval performance is limited by the size of files. Considering the problems of retrieval efficiency and security, much effort has been devoted to enabling search over encrypted data [13–16].

However, the above schemes are based on the ideal assumption, that is, cloud servers are honest-but-curious. In practice, cloud servers may delete user data and perform partial queries or falsify search results to save computing and storage resources. To verify the correctness and integrity of the search results, Chen and Zhu [5] designed the structure of minimum hash subtree (MHST). The cloud server returns the MHST and the root signature to the data owner (or data user). The client uses the MHST to recalculate the hash value of the node for verification. To resist the malicious behavior of cloud servers, Wan and Deng [8] designed a trusted privacy protection keyword search scheme (VPSearch). The VPSearch scheme generates a binary vector for each keyword and uses homomorphic MAC (Message Authentication Code) technology to check the authenticity of the returned ciphertext. Liu et al. [6] proposed a dynamically verifiable SSE scheme that allows a user to perform a top-k search on a set of dynamic files while effectively verifying the correctness of search results. In this scheme, file nodes are ordered according to their ranks for such a keyword. The information about a node's prior/following neighbor is encoded with the RSA accumulator. Zhang et al. [17] proposed a verifiable keyword ranking retrieval scheme based on deterrence. Throughout the verification process, the cloud server cannot know which data owners, or how many data owners exchange anchor data that will be used for verifying the misbehavior of a cloud server. To recapitulate, these authentication mechanisms are highly dependent on the encrypted query index structure. It lacks a verification mechanism suitable for all search schemes.

The decentralized and tamper-proof characteristics of blockchain can prevent user data from malicious tampering and ensure the correctness of search results. There are many works about blockchain recently. Ron and Shamir [18] made a quantitative analysis of the full Bitcoin transaction graph. Vitalik et al. [19] firstly introduced smart contract to Bitcoin systems and proposed Ethereum. Andrychowicz et al. [20] and Bentov and Kumaresan [21] introduced Bitcoin to multi-party computing to solve the fairness problem. Swan put forward several blockchain schemes that can be applied to [22], one of which is Blockchain health. It provides a framework for storing health medical data on the blockchain. Patients who place their own electronic medical records (EMR) on the blockchain can obtain a certain amount of healthy coin.

Unfortunately, the above schemes fail to give an effective search method. Therefore, Li et al. [23] proposed a searchable symmetric encryption scheme based on blockchain. The scheme stores encrypted data and indexes on the blockchain and realizes the effective query of encrypted data in blocks. However, the script language of Bitcoin is not Turing complete and cannot be applied to more scenarios. Therefore, Tahir and Rajarajan [24] proposed a new privacy protection framework to fulfill keyword search for ciphertext stored on blockchain networks. The framework firstly implements a searchable encryption scheme based on probabilistic trapdoor on Hyperledger Fabric. The probabilistic trapdoor can resist distinguishable attacks and ensure a higher level of security and

privacy for the scheme. Hu et al. [11] built a decentralized, reliable and fair search scheme by replacing cloud servers with smart contract to ensure that data owners can obtain correct search results without worrying about malicious servers. Shortly afterwards, Chen [25] proposed blockchain based searchable encryption for electronic health records (EHR) sharing on the basis of Hu et al.'s scheme. The index of EHRs is constructed by complex logical expressions and is stored in smart contract, so that data users can use expressions to search for indexes.

2 Preparatory Knowledge

2.1 Ethereum

Ethereum is a new and open source blockchain platform where users can write code that controls digital assets runs exactly as programmed, and is accessible anywhere in the world. The top layer of its architecture is decentralized applications (DApp) that exchanges through the web3.js with the smart contract layer. All smart contracts run on the EVM and call the Remote Procedure Call (RPC) protocol. The EVM and RPC support four core elements of the Ethereum, including the Blockchain, the Consensus algorithm, the Miner and the Network layer. As a whole, Ethereum provides us with two appealing properties:

- Ethereum does not give users a set of preset operations (such as Bitcoin). It allows users to create complex operations as they wish.
- The design of the Ethereum is very flexible and adaptable. It is easy to create new applications on the Ethereum.

Smart contracts in Ethereum are applications with a state stored in the blockchain that can run automatically on each decentralized network node [26]. Similar to the way of the software library works, developers can create smart contracts to provide functionality for other smart contracts. Alternatively, smart contracts can be simply used as an application to store information on the Ethereum.

Gas system is designed to mitigate Denial-of-Service attack on the Ethereum network. When Alice sends a token, executes a contract or transfers etheric money, each opcode will cost a certain pre-defined amount of gas, so that Alice has to pay $gasprice{*}gasUsed$ for the gas used. In addition, to avoid unpredictable fuel consumption caused by errors in contracts, the user sets a maximum allowable gas consumption when sending the transactions, that is, $gasLimit$.

2.2 Notations

Following are the notations used in our manuscript (Table 1).

Table 1. Notations and descriptions

F:	The plaintext collection of m documents $F = \{f_1, ..., f_m\}$. Each file f_i in the collectioncan be considered as a sequence of keywords
m:	The number of files in F.
W:	The dictionary, namely, the set of keywords, denoted as $W = \{w_1, ..., w_n\}$.
n:	The size of W.
C:	The encrypted files collection stored in the blockchain, denoted as $C = \{C_1, ..., C_m\}$.
I:	The index tree encompassing multiple sub-indexes for the whole file collection F, denoted as $I = \{I_1, I_2, \ldots, I_c\}$
c:	The number of keyword groups.
$\widehat{I}$:	The encrypted form of index tree I.
T_q:	The set of query vector.
$\hat{T_q}$:	The encrypted form of T_q, named as trapdoor for the search request.
$TXID$:	The identifier of a transaction.
TX_{inx}:	The transaction embedded in index.
TX_{trap}:	The transaction embedded in trapdoor.
K:	The number of files returned.
h:	The number of roles in the access control list.
u:	The dimensions added for the security of the files.
l:	The upper bound of file size

2.3 System Overview

In Fig. 1, we outline the architecture of our system. There are three entities in the system model: data owner (Uo), data user (Us) and Ethereum. The Uo has m files $f_1, ..., f_m$. To protect the privacy of confidential documents, the Uo uses the symmetric encryption algorithm (e.g., AES) to transform them into ciphertext $C_1, ..., C_m$, which will be uploaded to the Ethereum blockchain in the form of transaction $TX_i (i = 1, ..., m)$. After the transactions are successful, each of them will have a corresponding transaction identifier $TXID$. The Uo uses these $TXIDs$ to generate an index I and encrypts the index to $\widehat{I}$. Then, the Uo uploads $\widehat{I}$ to the Ethereum in the form of transaction TX_{ind}. The Us is a data user authorized by the Uo. When the Us queries the encrypted data stored in Ethereum, he/she sends the transaction TX_{trap} embedded in the trapdoor to the Ethereum. After receiving legitimate query requests from Us, the smart contract executes search algorithms with a search token $\hat{T_q}$ and the previously stored index $\widehat{I}$ and saves the search results (i.e., file identifiers) to its state, which is publicly known including the Uo. After receiving the search results, the Us uses the secret keys to decrypt the documents.

2.4 Threat Models

Depending on what information an adversary (e.g., a malicious server node) knows, we adopt the following two threat models.

Known Ciphertext Model. In this model, the adversary only knows the encrypted file collection C, the encrypted index tree $\widehat{I}$, and the search trapdoor $\hat{T_q}$ submitted by the Us. It means that the adversary can conduct ciphertext-only attack (COA) [10].

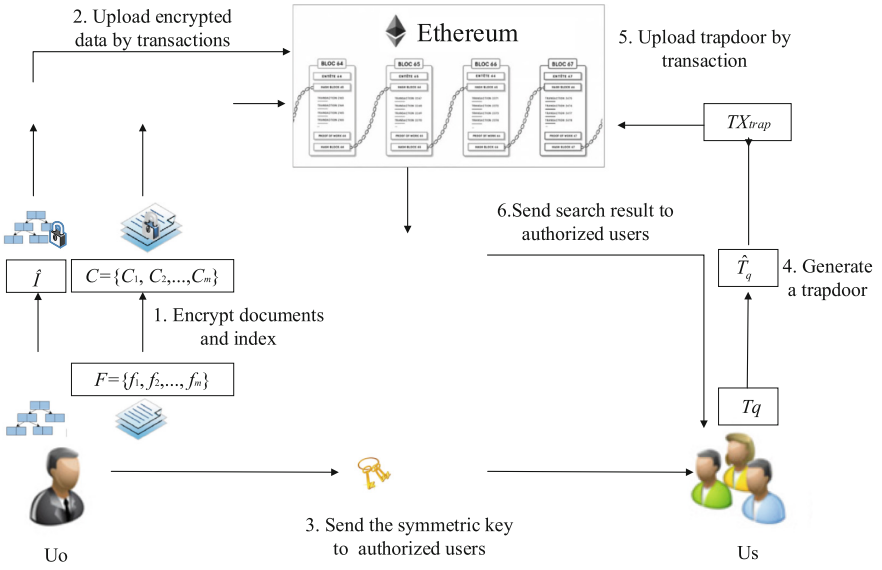

Fig. 1. System overview

Known Background Model. In this stronger model, the adversary not only equips with the knowledge of known ciphertext model, but also some other knowledge, such as the term frequency statistics of the document collection. That is to say, the adversary equipped with such statistical information can record how many documents there are for each term frequency of a specific keyword in F. Specifically, the adversary can conduct term frequency (TF) statistical attack to infer or even identify certain keywords through analyzing histogram and value range of the corresponding frequency distributions [27].

2.5 Design Goals

Soundness. It indicates that if there is a malicious server that does not correctly perform a protocol, it will get caught and obtain nothing. Generally, the previous works achieve this goal through a range of verifications of search results. In this paper, our scheme can obtain correct query results without verification operations and effectively prevent malicious nodes from unauthorized access.

Privacy-Preserving. In our scheme, we aim to protect the index and query confidentiality, query unlinkablitity as well as keywords privacy from adversaries.

- Index and query confidentiality. The adversaries cannot detect or infer the plaintext information about the content of trapdoor and index.
- Query unlinkablitity. The adversaries cannot identify whether or not two trapdoors are from the same search request.
- Keywords privacy. The adversaries cannot deduce whether the specific keyword is contained in a trapdoor through analyzing the TF distribution histogram.

2.6 Vector Space Mode

Vector space model and the $TF \times IDF$ rule are widely used in information retrieval [10]. In vector space model, for each document f_i, the Uo generates an n-dimensional document vector $D_i (1 \leq i \leq m)$. If the keyword w_j is not included in f_i, $D_i[j] = 0$; otherwise $D_i[j] = p_j == \frac{TF(w_j) \times IDF(w_j)}{\sqrt{\sum_{n=1}^{N} TF(w_i) \times IDF(w_i)}}$, where p_j is the weight of the keyword w_j. The $TF(w_j)$ is the frequency of the characteristic item, that is, the times that the keyword w_j appears in the document. The $IDF(w_j)$ is the inverse document frequency, inversely proportional to the number of documents in which the keyword w_j appears, $IDF = lg\frac{N}{idf}$. N denotes the number of words in the dictionary, and idf denotes the number of documents containing w_j.

3 Design Challenges and Countermeasures

3.1 Binary Indexed Tree

In Ethereum, each operation, including sending/storing data and executing computations, has an upper bound of consumed gas called $gasLimit$ as described in Subsect. 2.1. This restricts the size of transactions and the complexity of designed functions. Therefore, we divide the encrypted data to make privacy-preserving search over a large database feasible. Specifically, the main challenge is to divide the traditional binary index tree into multiple sub-indexes. The index partition method is described in detail as follows.

- The Uo produces an inverted index $INV = (inv(w_1), inv(w_2), ..., inv(w_n))$ for dictionary W, where $inv(w_i)$ is an inverted list of w_i. Each inverted list only stores top-P documents corresponding to keyword wi, where P is a positive integer, $P > K$.
- The Uo divides the set of keywords W into c groups $Wg = \{Wg_1, Wg_2, ..., Wg_c\}$, where each group Wg_i in Wg contains d keywords. Then, according to the inverted index INV, we generate inverted index set $Vg = \{Vg_1, Vg_2, ..., Vg_c\}$ for each group Wg_i.
- For each keyword group Wg_i, the Uo constructs the keyword balance binary (KBB) tree I_i as the sub-index by the top-P documents and obtains the index tree $I = \{I_1, I_2, ..., I_c\}$ for the whole file collection, where $1 \leq c$. The number of sub-indexes is determined by the value of d. Suppose N_i represents a node of index I_i and it is in the form of $< TXID, l_N, r_N, value >$, where l_N and r_N are the left and right nodes of N_i, and $value$ is a d-dimensional file vector. If N_i is a leaf node, then $TXID$ is transaction identifier, and $value$ stores the document vector D_i of keyword group Wg_i; else, the N_i is an intermediate node, then the $TXID$ is empty and the $value$ is calculated as follows:

$$value[j] = max\{l_N.value[j], r_N.value[j]\} + |rand()|\%max\{l_N.value[j], r_N.value[j]\}$$

As shown in Fig. 2, we give an example to illustrate the detailed process of partition. In summary, the structure of sub-indexes not only circumvents the

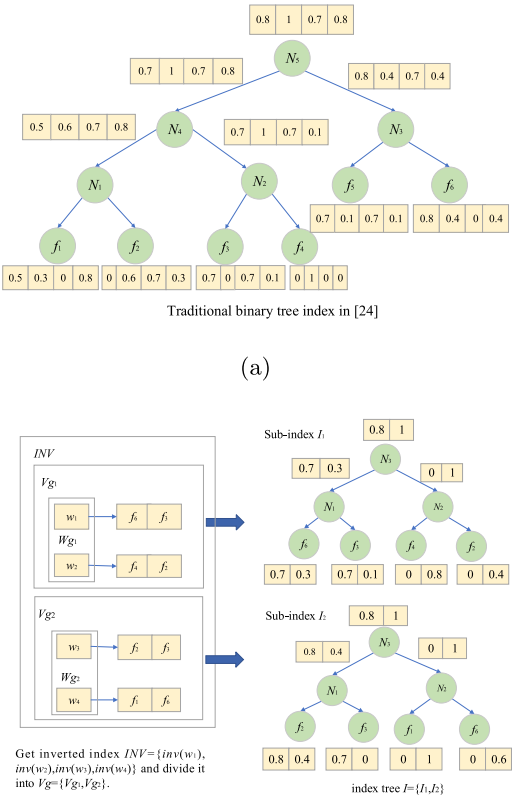

Fig. 2. An example of dividing the traditional binary tree index in [10] with the file set $F = f_1, ..., f_6$ and the dictionary $W = \{w_1, ..., w_4\}$. In the process of constructing the index, we divide (a) the traditional binary tree index into (b) two sub-indexes I_1 and I_2 by dividing the set of keywords W into two groups Wg_1 and Wg_2 (where $P = 2$ and $d = 2$). The sub-index of each group Wgi is generated according to the traditional index generation algorithm in [10]. Note that the file vectors in the figure is unencrypted to simply describe the partition process.

above *gasLimit*, but also reduces the n-dimensional file vector to d-dimensional and lowers the height of index tree to improve the retrieval efficiency in the smart contract.

3.2 Access Control

Due to the restriction of *gasLimit* in Ethereum, the operation of access control may become significantly expensive when there are abundant and complex expressions in a smart contract. Therefore, to simplify the whole access control mechanism and avoid the complexity of individual authorization, we introduce an effective role-

based access control (RBAC) strategy for transaction data. It can assert who have permission to access the requested transactions by checking whether the user roles are the root of the target polynomial. The detailed process is as follows.

Suppose there are h roles in the role set $\widehat{h}$. A data user whose role is in η_i will have the right to query transaction TX_i, where $\eta_i \subseteq \widehat{h}$. Let SC be a smart contract running on the Ethereum network, Uo be the dispatcher of SC and Us be a user authorized to invoke SC. The access control process is as follows.

(1) The Uo writes the access control policies, roles information and permissions of the users to SC and install it on each node.
(2) The Us searches the encrypted data by submitting the transaction TX_{trap} that contains the trapdoor $\hat{T}_q$, the transaction identifier TX_{ind_i} of the index and its role information z_1.
(3) After receiving the search request of the Us, the SC executes the access control strategy in advance.
   – Construct a variable polynomial of degree z for the transaction TX_{ind_i}, denoted as $y_i(z) = \prod_{r_j \in \eta_i} (z - r_j) = \sum_{j=0}^{j=h} \widehat{r_j} z^j$ (when $j > |\eta_i|$, $\widehat{r_j} = 0$).
   – Substitute the role z_1 of Us into the polynomial $y_i(z)$. If the role z_1 is the root of $y_i(z)$, the Us can access the transaction TX_{ind_i}. Otherwise, the Us is an unauthorized user and does not have permission to access the transaction data.

Therefore, the problem of whether the Us has access permission for TX_{ind_i} can be reduced to the problem to check whether its role is the root of $y_i(z)$.

Correctness. If Us is assigned to a role $z_1 \in \widehat{h}$ and transaction possesses subset η_i, it holds that

$$y_i(z) = \begin{cases} 0 & z_1 \in \eta_i \\ y_i(z_1) \neq 0 & other \end{cases} \tag{1}$$

4 The Detailed Scheme

In this section, we construct a decentralized privacy-preserving search scheme BMSSED that consists of six polynomial-time algorithm = (Setup, EncFiles, EcryptIndex, GenTrapdoor, Query, Dec).

(1) $Setup(1^\lambda)$:
   – The Uo enters the security parameter λ to generate the key $\Pi = \{sk_1, sk_2\}$. Let $\varepsilon = (\varepsilon.Enc, \varepsilon.Dec)$ be a secure symmetric encryption scheme, and sk_1 be the symmetric key of ε for encrypting files. Then, the Uo shares the key to the authorized users. The key $sk_2 = \{S, M_1, M_2\}$ is used to encrypt the index, where S contains the c groups $(d + u + 1)$ dimension vector, denoted $S = \{S_1, S_2, ..., S_c\}$. M_1 and M_2 are two groups of matrices, both of which contain c invertible matrices that are $(d + u + 1) \times (d + u + 1)$. Besides, the Uo sets a price of $\$offer$ for each search.

– The Us makes a deposit $\$deposit$ from $\$Buser$, where $\$Buser$ is the balance of the Us and $\$deposit$ is the deposit currency by the Us.

(2) $EncFiles(sk_1, F) \rightarrow C$: The Uo uses sk_1 to encrypt the document set $F = \{f_1, f_2, ..., f_m\}$ and obtains the ciphertext collection $C = \{C_1, C_2, ..., C_m\}$ as follow. Immediately, the Uo embeds the encrypted documents in C into transactions.

$$C_i = \varepsilon.Enc(sk_1, f_i)(1 \leq i \leq m) \tag{2}$$

– If $C_i > l$, the Uo divides C_i into s blocks $C_{i1}, C_{i2}, ..., C_{is}$, $|C_{ij}| + p \leq l$, $\forall j \in \{1, ..., s\}$, $s = \lfloor |C_i|/p \rfloor$. To store $C_i, C_{i1}, C_{i2}, ..., C_{is}$ are respectively embedded in the transaction $TX_{f_{i,k}}(k = 1, ..., s)$, the process is as follows:
  - When $k = 1$:
    * Embed $C_{i1}||0^p$ into the transaction $TX_{f_{i,1}}$.
    * Upload it to the blockchain and record its transaction identifier $TXID_{f_{i,1}}$.
    * For $2 \leq k \leq s$:
    * Embed $C_{ik}||TXID_{f_{i,k-1}}$ into the transaction $TX_{f_{i,k}}$.
    * Upload it to the blockchain and record its identifier $TXID_{f_{i,k}}$.
– If $|C_i| \leq l(1 \leq i \leq n)$, it is embedded directly into the transaction TX_{f_i} after signing it, upload it to the blockchain and record its corresponding transaction identifier $TXID_i$.

(3) $EncryptIndex(sk_2, S, F) \rightarrow \widehat{I}$:
– The Uo uses the $TXIDs$ of encrypted data to generate index $I = \{I_1, I_2, \ldots, I_c\}$. The generation process is described in Subsect. 3.1.
– The dimension of each data vector in the sub-index I_i is extended from d to $d + u + 1$. The value of the corresponding positions are randomly set to 0 or 1. The value of the $(d + u + 1)$-th dimension is set to 1.
– We use the secure kNN algorithm in [10] to encrypt the index. Suppose N_i denotes a node in index I_i, ND_i represents the stored data vector and S_i is the i-th vector in S.
  - Split ND_i into two random vectors $ND_i^{'}$ and $ND_i^{''}$ with the splitting indicator S_i as follow. For $1 < j < d + u + 1$,

$$\begin{cases} ND_i^{'}[j] = ND_i^{''}[j] = ND_i[j], & S_i[j] = 0 \\ ND_i^{'}[j] + ND_i^{''}[j] = ND_i[j], & S_i[j] = 1 \end{cases}$$

  - Encrypt these two vectors as $\{M_{1,i}^T ND_i^{'}, M_{2,i}^T ND_i{''}\}$ and store them on node N_i, where $M_{1,i}^T$ and $M_{2,i}^T$ represent the i-th matrices in the matrix groups M_1 and M_2, respectively. The encryption form of index I is denoted as

$$\begin{aligned} \widehat{I} &= \{\{M_{1,1}^T ND_i^{'}, M_{2,1}^T ND_i^{''}\}, ..., \{M_{1,c}^T ND_i^{'}, M_{2,c}^T ND_i^{''}\}\} \\ &= \{\widehat{I}_1, ..., \widehat{I}_c\}, \end{aligned} \tag{3}$$

  where $\widehat{I}_i \leq l$ and $1 \leq i \leq c$

- For each sub-index $\widehat{I}_i$ in $\widehat{I}$, the Uo embeds it into the transaction TX_{ind_i} and submits to the blockchain.
  - Embed $\widehat{I}_i||0^p$ into the transaction TX_{ind_1} , upload it to the blockchain and record its transaction identifier $TXID_{ind_1}$ that can be seen as a pointer to the TX_{ind_1} .
  - For each $\widehat{I}_i(2 \leq i \leq c)$, the Uo inserts $\widehat{I}_i||TXID_{ind_{i-1}}$ into transaction TX_{ind_i}, uploads it to blockchain and records its transaction identifier $TXID_{ind_i}$.

(4) $GenTrapdoor(w_Q, sk_1) \rightarrow TX_{trap}$
- When the Us wants to search with keyword set w_Q, he/she generates the trapdoor $T_q = \{T_{q1}, T_{q2}, ..., T_{qc}\}$, where T_{qi} is a query vector of length d. If $Wg_{i,j}$ exists in the keyword set w_Q, the value of $T_{qi}[j]$ is 1; else, it is 0. We emphasize that when all the dimensions of the query T_{qi} are 0, remove T_{qi} from T_q (see an example in Fig. 3).
- The dimension of each query vector T_{qi} in T_q is extended from d to $d+u+1$. The values of the corresponding positions are randomly set to $\gamma_{i,j}$, where $i \in \{1, ..., c\}$ and $j \in \{1, ..., u\}$. The $(d+u+1)$-th dimension of query T_{qi} is set to another random number ψ_i. Besides, the first $d+u$ dimensions of each vector are multiplied by a random positive number r. The process that the Us encrypts T_{qi} with sk_1 is as follows.
  - Split T_{qi} into two random vectors T'_{qi} and T''_{qi} with the splitting indicator S_i as follow. For $1 < j < d+u+1$,

$$\begin{cases} T'_{qi}[j] + T''_{qi}[j] = T_{qi}[j], & S_i[j] = 0 \\ T'_{qi}[j] = T''_{qi}[j] = T_{qi}[j], & S_i[j] = 1 \end{cases}$$

  - The encryption form of trapdoor T_q is denoted as

$$\begin{aligned} \hat{T_q} &= \{\{M_{1,1}^{-1}T'_{q1}, M_{2,1}^{-1}T''_{q1}\}, ..., \{M_{1,c}^{-1}T'_{qc}, M_{2,c}^{-1}T''_{qc}\}\} \\ &= \{\hat{T_{q1}}, ..., \hat{T_{q1}}\}, \end{aligned} \tag{4}$$

- For each $\hat{T_{qi}}$ in $\hat{T_q}$, the Us embeds it into the transaction TX_{trap} and submits it to the smart contract.
  - Specify a time limitation T_1.
  - Embed $(\hat{T_q}, TXID_{ind}, T_1, z_1)$ into the transaction TX_{trap} and upload it to the smart contract.

(5) $Query(TX_{trap}, TX_{ind}) \rightarrow C_l$:
- Assert current time $T < T_1$.
- Assert that the Us is the authorized user according to the access control policy described in Subsect. 3.2.
- Assert $\$deposit > GL_{srch} \times \$gasPrice + \$offer$, where GL_{srch} is the $gasLimit$ for calling $Search(\cdot)$ function.

– Execute algorithm Search $(\hat{T}_q, \hat{I}, K)$, get search result $Rlist = \{TXID_{\rho 1}, ..., TXID_{\rho K}\}$ and obtain corresponding $C_l = \{C_{l1}, ..., C_{lK}\}$ according to $Rlist$ (see an example in Fig. 3). The relevance score is

$$\begin{aligned} &Score(\hat{T_{qi}}, \hat{ND_i}) \\ &= (M_{1,i}^{-1} T_{qi}^{'}, M_{2,i}^{-1} T_{qi}^{''}) \cdot (M_{1,i}^{T} ND_i^{'}, M_{2,i}^{T} ND_i^{''}) \\ &= r(Score(T_{qi}, ND_i) + \sum_{j=1}^{u} \gamma_{i,j}) + \psi_i, \end{aligned} \tag{5}$$

where $Score(T_{qi}, ND_i)$ is real score.

– Let $\$cost \leftarrow \$offer + G_{srch} \times \$gasPrice$ and send $\$cost$ to $\$B_{owner}$. G_{srch} is the $gascost$ for calling $Search(\cdot)$ function.

– Let $\$deposit \leftarrow \$deposit \times \$cost$ and send $\$deposit$ to $\$Buser$. .

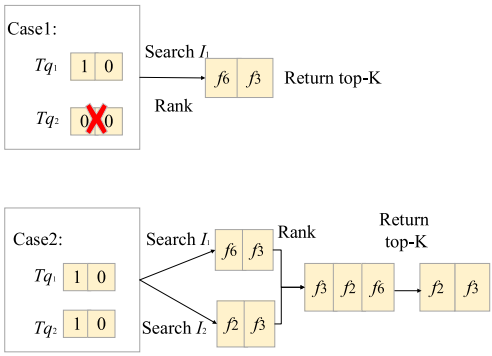

Fig. 3. An example of generating the trapdoor T_q for Case1: $w_Q = \{w_1, w_2\}$ or Case 2: $w_Q = \{w_1, w_3\}$ and returning the search results $(K = 2)$ by querying the sub-indexes in Fig. 2.

(6) $Dec(C_l, sk_1) \rightarrow D$:
After receiving the ciphertext set C_l, the Us uses the screct key to calculate $D_{\rho i} = \varepsilon.Dec(sk_1, C_{\rho i})(1 \leq i \leq r)$.

Algorithm 1. Search $(\hat{T_q}, \widehat{I}, K) \rightarrow Rlist$

Require: The query $\hat{T_q}$, the searchable index $\widehat{I}$;
Ensure: K transaction identifiers with highest scores.
for query $\hat{T_{qi}}$ in query group $\hat{T_q}$ **do**
  TOPK($\hat{T_{qi}}$, root of $\widehat{I_i}$, 0,K)
  Merge top-K transaction identifiers $list_i$ of $\hat{T_q}$ into $RList$
end for
return top-K transaction identifiers of $RList$
TOPK($\hat{T_{qi}}, node, sco, K$)
if $sco <$ K-th score in $list_i$ **then**
  return $null$
end if
if $node$ is leaf node **then**
  Insert the $TXID$ of $node$ into $list_i$
else
  $leftScore = Score(\hat{T_{qi}}, node.l_N)$
  $rightScore = Score(\hat{T_{qi}}, node.r_N)$
  if $leftScore > rightScore$ **then**
    TOPK($\hat{T_{qi}}, node.l_N, leftScore, K$)
    TOPK($\hat{T_{qi}}, node.r_N, rightScore, K$)
  else
    TOPK($\hat{T_{qi}}, node.r_N, rightScore, K$)
    TOPK($\hat{T_{qi}}, node.l_N, leftScore, K$)
  end if
end if

5 Security Proof

In this section, we analyze the security of the BMSSED protocol.

5.1 Soundness

It is obvious that the security of our scheme depends on Ethereum. Therefore, BMSSED can achieve soundness so long as the security of Ethereum is ensured. When the smart contracts are correctly executed on Ethereum, the decision and search results will be stored as contract states permanently and publicly. Moreover, miners in Ethereum network can verify the data. The consensus property of Ethereum can guarantee that access control policies and search operations are executed correctly.

5.2 Privacy-Preserving

Index Confidentiality and Query Confidentiality. Adversaries can calculate the real values of indexes and queries by establishing liner equations from

the exposed ciphertext [10]. In our scheme, we use the $(d+u+1)$-dimensional segmentation indicator vector S_i and the $(d+u+1)\times(d+u+1)$ reversible matrices $(M_{1,i}, M_{2,i})$ to encrypt each subindex I_i in the index I, where each data vector is randomly split into two different $(d+u+1)$-dimensional vectors, $ND_i^{'}$ and $ND_i^{''}$. That is to say, adversaries can establish $2\varpi(d+u+1)$ equations from the ciphertext of this index, where ϖ represents the number of nodes that sub-index $\widehat{I}_i$ contains. However, subindex $\widehat{I}_i$ contains $2(d+u+1)^2$ unknown numbers in each node. Besides, matrices $M_{1,i}, M_{2,i}$ also have $2(d+u+1)^2$ unknown numbers. It is obvious that the size of unknown numbers is more than the known equations. Similarly, the query vector $T_{qi}^{'}$ and $T_{qi}^{''}$ can be regarded as two $(d+u+1)$-dimensional random vectors. There are $2(d+u+1)$ unknown numbers in two query vectors and $2(d+u+1)(d+u+1)$ unknowns in matrices $M_{1,i}, M_{2,i}$. However, adversaries have only $2(d+u+1)$ equations, which are less than the unknown numbers. Therefore, there is not enough information to calculate the query vector or matrices $M_{1,i}, M_{2,i}$. Moreover, adversaries always use the known plaintext-ciphertext pair of queries to construct linear equations to calculate the value of index [28]. In our scheme, the relevance scores learned by adversaries are shown in formula (5), where $Score(T_{qi}, ND_i)$ is disturbed by the random value $r, \gamma_{i,j}$ and ψ_i. It means that the corresponding values of $Score(\hat{T_{qi}}, \hat{N_i})$ are different for the same queries. Furthermore, each linear equation may introduce u unknowns $\gamma_{i,j}$, two unknowns r and $\gamma_{i,j}$, that is, the unknowns in equations are always more than the number of linear equations. Therefore, adversaries cannot calculate the real value of index and secret key.

In summary, the BMSSED is strong enough to protect the security of index and query in known ciphertext model.

Query Unlinkability. By introducing the random value $r, \gamma_{i,j}$ and ψ_i, the same search requests will generate different query vectors and receive different relevance score distributions. Thus, adversaries are unable to establish the correspondence between query vectors and documents. However, adversaries can analyze the similarity of search results to judge whether the retrieved results come from the same requests. In the proposed BMSSED scheme, the data user can control the level of unlinkability by adjusting the value of $\sum_{j=1}^{u} \gamma_{i,j}$. This is a trade-off between accuracy and privacy, which is determined by the user.

Keyword Privacy. In the known background model, the adversary is equipped with the TF statistics of the document collection. The proposed BMSSED scheme is designed to obscure the TF distributions of keywords with the randomness of $\sum_{j=1}^{u} \gamma_{i,j}$. In order to maximize the randomness of relevance score distributions, we need to get as many different $\sum_{j=1}^{u} \gamma_{i,j}$ as possible. However, the query accuracy will be reduced. Therefore, data users can balance the trade off between query accuracy and keyword privacy by adjusting the value of u.

6 Performance Evaluation

6.1 Experimental Environment

Implementation. Experiments are performed on a computing system with an Intel(R)Core(TM) i5-6500 CPU(3.2 GHz) processor, and the Windows 10 (64 bit) operation system. We deploy the smart contract to the locally simulated network TestRPC and the official Ethereum test network Rinkeby, where TestRPC is initiated with the default configuration, similar to the configuration of the real Ethereum environment. Its mining block time is set to 0 so that we can focus on the performance of the search part of the smart contract. The smart contract is written in Python and combines with Solidity and Javascript as an intermediate interaction language.

Data. We make an assessment for BMSSED on four different, synthetically generated test data sets provided by the Natural language processing group, international database center, Department of computer information and technology [1]. These data sets have also been used in prior work, such as [8]. A quick summary of the statistics of the data sets, the size of the resulting encrypted databases, and the size of keyword set W are shown in Table 2.

Table 2. Database evaluation

DB name	#DB	EDB	Distinct keywords
DB1	348	5.62 MB	709
DB2	742	11.8 MB	1244
DB3	1184	14.8 MB	2176
DB4	1472	23.0 MB	1802

Table 3. Key differences between our scheme and other blockchain based SE schemes

Scheme	Query support	Application context	Access control
Tahir et al. [24]	Single keyword	General	By data owner
Hu et al. [11]	Single keyword	General	By data owner
Chen et al. [25]	Boolean, range	EHRs	By data owner
Ours	Similarity search	General	By smart contract

6.2 Performance in Ethereum

In this section, we design a number of experiments to test the performance of the proposed BMSSED scheme. It is mainly composed of the overhead on

[1] https://download.csdn.net/download/zgj_gutou/12009292.

TestRPC and Rinkeby. The indicators in the experiment are the running time of the algorithm, where we take the average value of 30 independent runs of the algorithm as the running time of the algorithm. Prior to presenting our evaluation, we first provide a comparative summary of our proposed scheme and other blockchain based SE schemes. As shown in Table 3, our scheme supports similarity search for the general data and novelly uses smart contracts to create an access control mechanism, having a certain advantage in terms of function.

On the Locally Simulated Ethereum Network TestRPC. To avoid exceeding *gasLimit*, we experimentally set the limited size of a transaction as l =16 kB and the number of keywords in each sub-index as $d = 34$. Table 4 presents the time overhead for building an index, uploading encrypted data, and generating a trapdoor on different data sets. We can observe from the experimental results that the overhead of uploading the encryption data to the Ethereum blockchain is much higher than other overheads. This is because data owners store their confidential data in blockchain by embedding it into thousands of transactions (e.g., storing DB4 requires 1501 transactions), and each transaction costs about 4 s on average. Besides, we compare the query time overhead corresponding to the change of matching document numbers on different data sets. As shown in Fig. 4, the size of DB4 is the largest, with the longest time of searching. Accordingly, the query time overhead on the DB1 is the least. We explain that by a larger number of mined blocks leads to a longer time for loading.

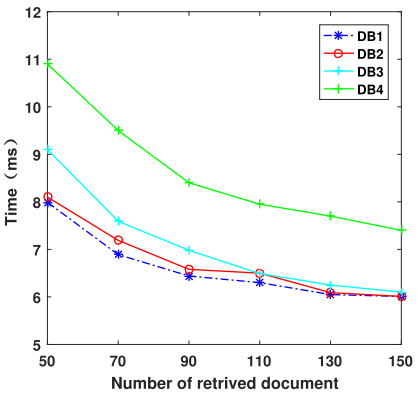

Fig. 4. Search time per matching document in TestRPC

We also observe that the running time of the query algorithm is lower as the size of search result become larger. This is caused by the constant cost of loading past mined blocks from disk into memory before each search runs. After the TestRPC startes, it creates 10 accounts by default and the following *Available Accounts* is the account list. Therefore, we use these default accounts to test the efficiency of access control policies based on smart contracts. As shown in

Table 4. Evaluations in TestRPC

DB name	#Tx	Construct index	#Tx	Upload encrypted files and index	Generate trapdoor	#Tx	Search Time
DB1	359	0.0045 s	10	24 min	≈ 0.001 s	1	7.23 s
DB2	944	0.0078 s	37	65 min	≈ 0.001 s	1	8.13 s
DB3	1258	0.0150 s	64	88 min	≈ 0.001 s	1	8.97 s
DB4	1501	0.0201 s	53	99 min	≈ 0.001 s	1	10.21 s

Fig. 5, in the case where the number of accounts is determined, the access control decision time is evenly distributed around 0.4 s and does not change with the size of the data sets. The exception point in the figure is mainly due to the instability of the network when the contract is running.

Available Accounts

==========

(0) 0x74650142c29e358b8f94a8c5d43345649009a4cd

(1) 0x450f2896c47c6e8763b6d389c40166584d0ced40

.....

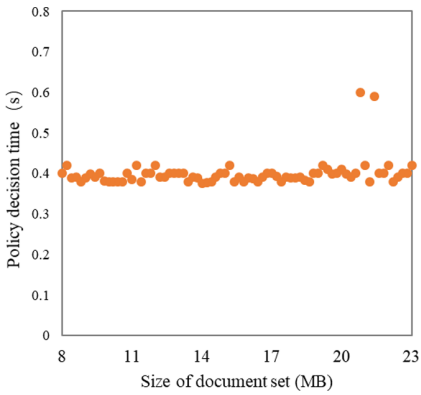

Fig. 5. The efficiency of access control policies based on smart contracts

On the Official Ethereum Test Network Rinkeby. Due to the limited balance, we only use the smallest database DB1 to perform experiments. There are 359 transactions to store database DB1. The average block time for mining is 15s and it costs 89 min to upload the whole encrypted data set. Besides, each command executed in a smart contract has a specific consumption, counted in units of gas. The average gas usage for a transaction is 4,201,232. However, when the SSE smart contract is deployed to the Ethereum blockchain for the first time, the cost is high. After completing the deployment of the smart contract, the consumption cost that the users invoke the function interface provided by

the smart contract is obviously reduced. For instance, the search time is about 10 s, 11 s, and 12.5 s for 50, 110, and 150 matched documents respectively in one transaction. The average gas usage for searching is 1,676,958.

Comparison of Query Efficiency with EDMRS in [10]. As shown in Fig. 6, we compare the search efficiency of our scheme with EDMRS in [10] under different numbers of retrieved documents to demonstrate the validity of our partition algorithm. The experimental results reveal that the sub-index structure is more efficient than the binary tree index of EDMRS. This is because our method divides the query T_q into c queries $\{T_{q1}, ..., T_{qc}\}$ and only sends those that are not empty to the cloud server. Therefore, the server does not have to search all sub-indexes.

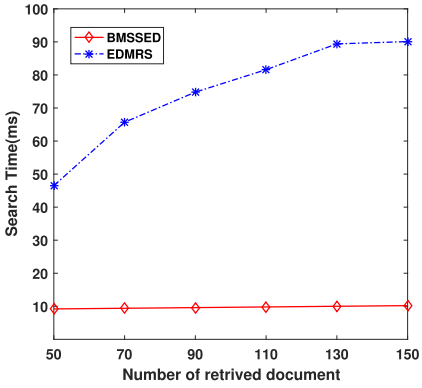

Fig. 6. Time cost of search with different numbers of documents that the Us wants to retrieve

6.3 Search Precision

Data users not only pay attention to search efficiency, but also care about search precision. The higher the similarity among the documents in search result, the higher the search precision. In our scheme, without loss of generality, the search precision SP_k is defined as

$$SP_k = \sum_{i=1}^{K} Score(\hat{T_{qi}}, \hat{ND_i}) / \sum_{i=1}^{K'} Score(T_{qi}, N_i) \tag{6}$$

where K is the top-K file returned by the ciphertext retrieval, and $K^{'}$ is the top-K file returned in the plaintext query. $Score(\hat{T_{qi}}, \hat{ND_i})$ is the similarity between the encrypted query vectors and the returned documents in result set and $Score(T_{qi}, ND_i)$ is real score.

As described in Subsect. 3.1, to avoid exceeding *gasLimit* and improving the query efficiency, each sub-index only stores the top-P documents. Obviously, this method may lead to lower accuracy of queries whereas the Us in BMSSED can control the accuracy by adjusting the value of P. Figure 7(a) indicates that the Us wants to get a higher retrieval accuracy, he/she just sets a larger value of P. Besides, when the value of P is beyond a certain point (such as, P =90 in Fig. 7(a)), it has little effects on search results. We explain that by the union of search results for one single keyword query embrace most of the top-K documents in a multi-keyword query. Note that the search precision of scheme is also affected by the size of the data set with the same P. The results are shown in Fig. 7(b). The smaller data set has a higher search precision.

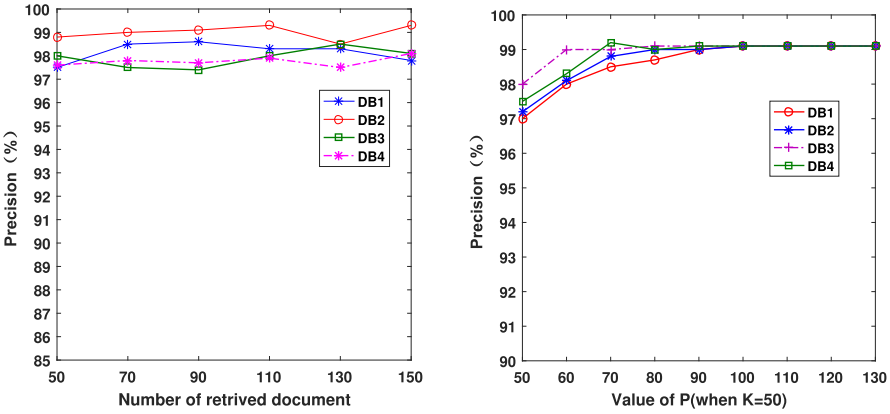

Fig. 7. The impact of the different values P (a) and K(b) on precision

7 Conclusion

In this work, we propose a blockchain based multi-keyword similarity search scheme over encrypted data (BMSSED). Different from the existing SSE scheme based on blockchain, our novelty is to use smart contracts to create an access control mechanism and fulfill multi-keyword similarity query. To avoid exceeding *gasLimit*, we divide the traditional binary tree index into a plurality of sub-indexes. The structure of sub-index not only helps us circumvent the *gasLimit*, but also reduces the dimensional of file vector and lower the height of index tree to improve the retrieval efficiency in the smart contracts. In addition, to simplify the whole authorization mechanism, we use the polynomial-based RBAC strategy to assert who have the permission to access the transactions. We conduct repeated experiments on real data sets. Experimental results and theoretical analysis show the practicability and security of our scheme over encrypted data.

The possible further research direction is to establish a protocol that enables access control crossing the domain on the basis of the present work.

Acknowledgments. We would like to thank anonymous reviewers for their helpful comments. This work is supported by National Key R&D Program of China(2018YFA0704703); National Natural Science Foundation of China(61972215, 61702399, 61972073); Natural Science Foundation of TianJin(17JCZDJC30500)

References

1. Shangqi Lai, Sikhar Patranabis, Amin Sakzad. Result pattern hiding searchable encryption for conjunctive queries, in: the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15–19, 2018, pp. 745–762
2. Kermanshahi, S.K., Liu, J.K., Steinfeld, R.: Generic multi keyword ranked search on encrypted cloud data, In: Proceedings of ESORICS 2019–24th European Symposium on Research in Computer Security, Luxembourg, 23–27 September 2019, Part II, pp. 322–343 (2019)
3. Xu, L., Yuan, X., Wang, C.: Hardening database padding for searchable encryption. In: 2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, pp. 2503–2511 (2019)
4. Song, D.X., Wagner, D.A., Perrig, A.: Practical techniques for searches on encrypted data. In: 2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, pp. 44–55. 14–17 May 2000
5. Chen, C., Zhu, X., Shen, P.: An efficient privacy-preserving ranked keyword search method. IEEE Trans. Parallel Distrib. Syst. **27**, 951–963 (2015)
6. Liu, Q., Nie, X., Liu, X.: Verifiable ranked search over dynamic encrypted data in cloud computing. In: 25th IEEE/ACM International Symposium on Quality of Service, IWQoS 2017, Vilanovaila Geltrú, Spain, pp. 1–6. 14–16 June 2017
7. Homann, D., Wiese, L.: Inference attacks on fuzzy searchable encryption schemes. Trans. Data Priv. **12**(2), 91–115 (2019)
8. Wan, Z., Deng, R.H.: Vpsearch: achieving verifiability for privacy-preserving multi-keyword search over encrypted cloud data. IEEE Trans. Dependable Secure Comput. **15**, 1083–1095 (2016)
9. Wang, Q., He, M., Minxin, D.: Searchable encryption over feature rich data. IEEE Trans. Dependable Sec. Comput. **15**, 496–510 (2018)
10. Xia, Z., Wang, X., Sun, X.: A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. IEEE Trans. Parallel Distrib. Syst. **27**(2), 340–352 (2016)
11. Hu, S., Cai, C., Wang, Q.: Searching an encrypted cloud meets blockchain: a decentralized, reliable and fair realization. In: 2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, pp. 792–800 (2018)
12. Jiang, S., Liu, J., Wang, L.: Verifiable search meets blockchain: a privacy-preserving framework for outsourced encrypted data. In: 2019 IEEE International Conference on Communications, ICC 2019, Shanghai, China, pp. 1–6. 20–24 May 2019
13. Lai, S., Patranabis, S., Sakzad, A.: Result pattern hiding searchable encryption for conjunctive queries. In: Proceedings of the ACM Conference on Computer and Communications Security, pp. 745–762 (2018)

14. Kerschbaum, F., Tueno, A.: An efficiently searchable encrypted data structure for range queries. In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) ESORICS 2019. LNCS, vol. 11736, pp. 344–364. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29962-0_17
15. Golle, P., Staddon, J., Waters, B.: Secure conjunctive keyword search over encrypted data. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 31–45. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24852-1_3
16. Asharov, G., Segev, G., Shahaf, I.: Tight tradeoffs in searchable symmetric encryption. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 407–436. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_14
17. Zhang, W., Lin, Y., Qi, G.: Catch you if you misbehave: Ranked keyword search results verification in cloud computing. IEEE Trans. Cloud Comput. **6**, 74–86 (2018)
18. Ron, D., Shamir, A.: Quantitative analysis of the full bitcoin transaction graph. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 6–24. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_2
19. Lavery, K.: Smart Contracting for Local Government Services: Processes and Experience. Praeger Publishers Inc, Westport (1999)
20. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on Bitcoin. Commun. ACM **59**(4), 76–84 (2016)
21. Bentov, I., Kumaresan, R.: How to use Bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_24
22. Portmann, E.: Rezension blockchain: blueprint for a new economy. HMD Prax. der Wirtschaftsinformatik **55**(6), 1362–1364 (2018). https://doi.org/10.1365/s40702-018-00468-4
23. Li, H., Zhang, F., He, J.: A searchable symmetric encryption scheme using blockchain. CoRR abs/1711.01030 (2017)
24. Tahir, S., Rajarajan, M.: Privacy-preserving searchable encryption framework for permissioned blockchain networks. In: IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), iThings/GreenCom/CPSCom/SmartData, Halifax, NS, Canada, pp. 1628–1633 (2018)
25. Chen, L., Lee, W.-K., Chang, C.-C.: Blockchain based searchable encryption for electronic health record sharing. Future Gener. Comput. Syst. **95**, 420–429 (2019)
26. Torres, C.F.: Mathis steichen, radu state: the art of the scam: demystifying honeypots in ethereum smart contracts. In: USENIX Security Symposium, pp. 1591–1607 (2019)
27. Zhang, Q., Fu, S., Jia, N., Xu, M.: A verifiable and dynamic multi-keyword ranked search scheme over encrypted cloud data with accuracy improvement. In: Beyah, R., Chang, B., Li, Y., Zhu, S. (eds.) SecureComm 2018. LNICST, vol. 254, pp. 588–604. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01701-9_32
28. Yao, B., Li, F., Xiao, X.: Secure nearest neighbor revisited. In: 29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, pp. 733–744 (2013)