IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 3, 2019, accepted September 14, 2019, date of publication September 24, 2019,
date of current version October 9, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2943356

Blockchain-Based Verifiable Multi-Keyword Ranked Search on Encrypted Cloud With Fair Payment

YANG YANG[1,2,3,4,5], HONGRUI LIN[1,5], XIMENG LIU[1,5], (Member, IEEE), WENZHONG GUO[1], XIANGHAN ZHENG[1], AND ZHIQUAN LIU[3]

[1]College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China
[2]Guangxi Key Laboratory of Cryptography and Information Security, Guilin 541004, China
[3]Guangdong Provincial Key Laboratory of Data Security and Privacy Protection, Guangzhou, Guilin 510632, China
[4]Fujian Provincial Key Laboratory of Information Processing and Intelligent Control, Minjiang University, Fuzhou 350108, China
[5]University Key Laboratory of Information Security of Network Systems, Fuzhou University, Fuzhou 350108, China

Corresponding authors: Yang Yang (yang.yang.research@gmail.com) and Wenzhong Guo (guowenzhong@fzu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61872091, in part by the Guangxi Key Laboratory of Cryptography and Information Security under Grant GCIS201721, in part by the Opening Project of Guangdong Provincial Key Laboratory of Data Security and Privacy Protection under Grant 2019B030301004-13, and in part by the Open Fund Project of Fujian Provincial Key Laboratory of Information Processing and Intelligent Control, Minjiang University, under Grant MJUKF-IPIC201906.

ABSTRACT In traditional cloud computing system, searchable encryption is deemed as a core technology to realize data confidentiality protection and information retrieval functions. However, the online payment problem and mutual distrust between cloud platforms and users may hinder the wide adoption of cloud service. In this paper, we construct a **b**lockchain based **m**ulti-keyword ranked search with **f**air **p**ayment (BMFP) system, which leverages smart contracts to verify the correctness and completeness of the search result, and automatically execute the fair payment operations. The system realizes public verifiability on a multi-keyword ranked search result. The data owner manages the search authority, and a concrete fair payment smart contract is designed. The BMFP is compatible with Ethereum, and the verification algorithm executed by the smart contract is cost-efficient.

INDEX TERMS Blockchain, cloud computing, fair payment, verifiable searchable encryption, top-*k* ranked search, multiple keywords.

I. INTRODUCTION

With the advent of cloud computing platform [1], an increasing amount of enterprises and individuals have the intention to take advantage of this emerging technology and migrate the large volume of data to a cloud platform to save the local storage cost. The cloud platform offers immediate services for remote storage and computation, which facilitates whenever and wherever data access and usage. Meanwhile, personal data privacy protection is enforced by the GDPR (General Data Protection Regulation)/HIPAA (Health Insurance Portability and Accountability Act) in Europe/USA. To guarantee security and data usability, searchable encryption technology [2], [3] simultaneously realizes data confidentiality and search over encrypted data, which becomes a hot research topic for cloud computing. Majority work only supports single keyword search [4], [5] such that a large number of files are sent back to the user without recommended order. A practical scheme should realize multiple keywords search with a ranked result and high efficiency, which allows a user to search multi-keywords in each query and the most relevant encrypted files containing these keywords are returned to save the bandwidth.

Searchable encryption system also faces a new attack paradigm, where the cloud server is not honest to execute the search operation (to save computation resources) and sends incorrect or incomplete search result to the users [6]. In the pay-for-use business model, the user is forced to pay the service fee to the cloud platform even though the above scenario occurs. If the payment model is changed to pay-after-use, the dishonest or malicious data user may slander the cloud platform and refuse to pay the service fee even though

The associate editor coordinating the review of this manuscript and approving it for publication was Petros Nicopolitidis.

he receives the correct result. Also, the value of data needs be considered such that the data owner should be paid message fee for providing digital resources. Therefore, the searchable encryption system should ensure fair payment among data owner, data user and cloud platform. The traditional payment schemes have several limitations: it requires a fully trusted party (such as a bank) to deal with the payment with fairness; the trustworthy party maybe not have the ability to verify the search results or other outsourced computation operations; the privacy of data owner or data users maybe leaked.

Blockchain technology [7] brings a new decentralized payment paradigm to deal with these problems, which is not controlled by any centralized authority. Smart contract in the blockchain is a self-executing contract with the terms and clauses (between buyer and seller) being directly written into lines of codes. Smart contracts [8] permit trusted transactions and agreements to be carried out among anonymous parties without the participation of a central authority, legal system, or external enforcement mechanisms. Thus, blockchain and smart contract are suitable to execute the verification operation and realize fair payment [9] in a searchable encryption system.

A. OUR CONTRIBUTION

To tackle with the above challenges, we propose a **b**lockchain based **m**ulti-keyword ranked search with **f**air **p**ayment system (BMFP), and the contributions are summarized as below.

We design a verifiable multi-keyword search system to realize (weighted zone score based) top-k ranked search, where only the most relevant encrypted files are returned to data users. A novel multi-keyword ranked inverted index data structure and an efficient look-up table are designed. Thus, the search efficiency increases with the keyword number rather than the number of total documents. The verification algorithm in BMFP is more efficient than the verification tree-based searchable encryption schemes.

We propose a framework that combines Ethereum blockchain and smart contract to realize fair payment in a searchable encryption system. The data owner has full authority to manage the data and search privileges, where trusted public key generation center is not necessary. The data user needs to pay message fee to data owner for using the encrypted data, and pay the service fee to the cloud platform for using the search service. The fair payment smart contract of BMFP guarantees: if the correct and complete search result is returned to data user, the message and service fees are transferred to data owner and cloud, respectively; otherwise, the data user's fees are returned to his account. Also, the search result can be publicly verified by all the nodes in the blockchain.

B. PAPER ORGANIZATION

The rest of the paper is organized as follows. Section II overviews the related works of blockchain, fair payment and searchable encryption. Section III introduces the background knowledge of weighted zone score, inverted index and Ethereum. Section IV describes the system model and workflow. Section V and Section VI show the concrete construction and smart contracts of the BMFP system. The performance and security analysis are discussed in Section VII. The conclusion is shown in Section IX.

II. RELATED WORK

A. SEARCHABLE ENCRYPTION

The concept of searchable encryption was put forward by Song *et al.* [10] to enable single keyword search over encrypted data through building the encrypted searchable index. Multiple keyword searchable encryption schemes are proposed to enable the conjunctive keyword search. Cao *et al.* [11] used inner product similarity to quantitatively evaluate the similarity between query and encrypted index, and proposed a multi-keyword ranked search system over encrypted data. Yang *et al.* [13] put forward a multi-user multi-keyword ranked search scheme to support arbitrary language query, which is constructed based on Paillier homomorphic encryption algorithm. He *et al.* [14] suggested an attribute-based hybrid Boolean keyword search scheme based on prime-order bilinear groups. A lightweight traceable system supporting keyword search is proposed in [15], which offloads most of the heavy cryptographic computations to the cloud and realizes traitor tracing function.

To verify the correctness of the search results, Wang *et al.* [12] proposed a verifiable data retrieval algorithm based on Bloom filter and Merkel hash tree. Liu *et al.* [16] utilized RSA accumulator to support two conjunctive keyword search, which makes use of broadcast encryption technology to support multiple user application. Zhu *et al.* [17] leveraged Merkel Patricia tree and incremental Hash to build a proof index with a dynamic data update. Ge *et al.* [5] constructed a searchable encryption scheme with symmetric-key based verification scheme utilizing an accumulative authentication tag. Yang *et al.* [19] proposed a novel Chinese multi-keyword fuzzy rank searchable encryption scheme, which achieves efficient fuzzy keyword search without constructing a large fuzzy set.

B. BLOCKCHAIN

Recently, designing verifiable privacy-preserving search schemes over encrypted data has received considerable research interest, where the correctness and integrity of search results can be verified. Although many verification techniques (e.g., Homomorphic MAC [6] or RSA accumulator [16]) can detect a dishonest behavior of cloud service provider who returns incorrect search result, it cannot properly work without a trusted third party. To address this problem, Hu *et al.* [20] proposed a blockchain-based searchable encryption: the search index is stored in the smart contract, and the search algorithm is executed by smart contract rather than the cloud service provider. A similar method in [20] is also adopted by Chen *et al.* [21], Wang *et al.* [22] and Wu *et al.* [23]: the search operation of the smart contract

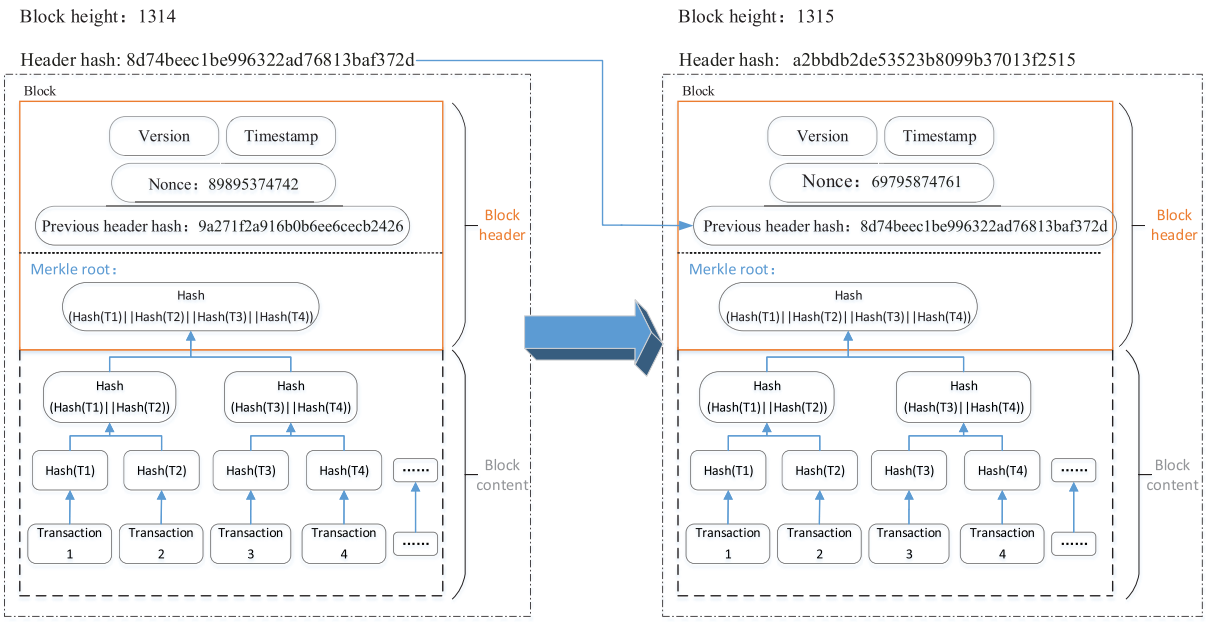


FIGURE 1. Structure of blockchain.

is always trustworthy to return correct result such that the result verification is not necessary. However, these scheme [20]–[23] suffer from low scalability and high cost since the complex search operation executed by smart contract costs a lot of ETH/ETC. Zhang *et al.* [24] leverages Bitcoin-based timed commitment [25] protocol to design a fair payment system. However, the smart contract of Bitcoin is not Turing-complete and the function is too limited. Similar ideal to [24], Cai *et al.* [9] used Ethereum to design a t-time-locked payment protocol (rather than Bitcoin) to realize pay-after-use with fairness in searchable encryption.

III. PRELIMINARY

A. WEIGHTED ZONE SCORE

Term frequency is always used to evaluate the importance of a keyword in a document. However, a document has different zones (such as zones of title, abstract and main body), and the keywords appear in different zones have different importance. For example, the keywords in the title are more important than that in abstract, and the keywords in the main body have the least importance compared with the other zones. We adopt the weighted zone scoring method [18] to calculate the relevance score. Consider a set of documents and each of them has t zones, which are assigned with weights $g_1, \ldots, g_t \in [0, 1]$ such that $\sum_{i=1}^{t} g_i = 1$. For $1 \leq i \leq t$, let s_i be the Boolean score denoting a match (or absence) between a keyword w and the i-th zone of a file F. Then, the weighted zone score is defined as $S_{w,F} = \sum_{i=1}^{t} g_i s_i$. For keyword set $W = (w_1, \ldots, w_m)$, the weighted zone score is denoted as $S_{W,F} = \sum_{j=1}^{m} S_{w_j,F}$.

TABLE 1. An example of inverted index.

w_1	$F_1, F_2, F_3, \cdots$
w_2	$F_2, F_3, F_7, F_9, \cdots$
$\cdots$	$\cdots$
w_m	$F_1, F_5, F_{10}, \cdots$

B. INVERTED INDEX

The inverted index is an efficient information retrieval data structure for accelerating the search process, which stores a mapping from a keyword to a set of documents (containing the keyword). An example of inverted index is shown in Table. 1, where the first line indicates that the identifiers of the files containing keyword w_1 are F_1, F_2, F_3, and so on.

C. BLOCKCHAIN TECHNOLOGY AND ETHEREUM

1) BLOCKCHAIN TECHNOLOGY

Essentially, blockchain is a distributed and shared public ledger storing the transactions permanently in an ordered way, and each block is identified with a unique ID (created by a hash algorithm). The structure of blockchain is shown in Fig. 1. The header of blockchain contains the following core fields.

- Previous header hash: stores the hash value of the previous block, which is a cryptographic link that connects the blocks together and guarantees the tamper-proof of the blockchain.
- Nonce: is a field whose value is adjusted by miners so that the hash of the block will be less than or equal to the current target of the network.

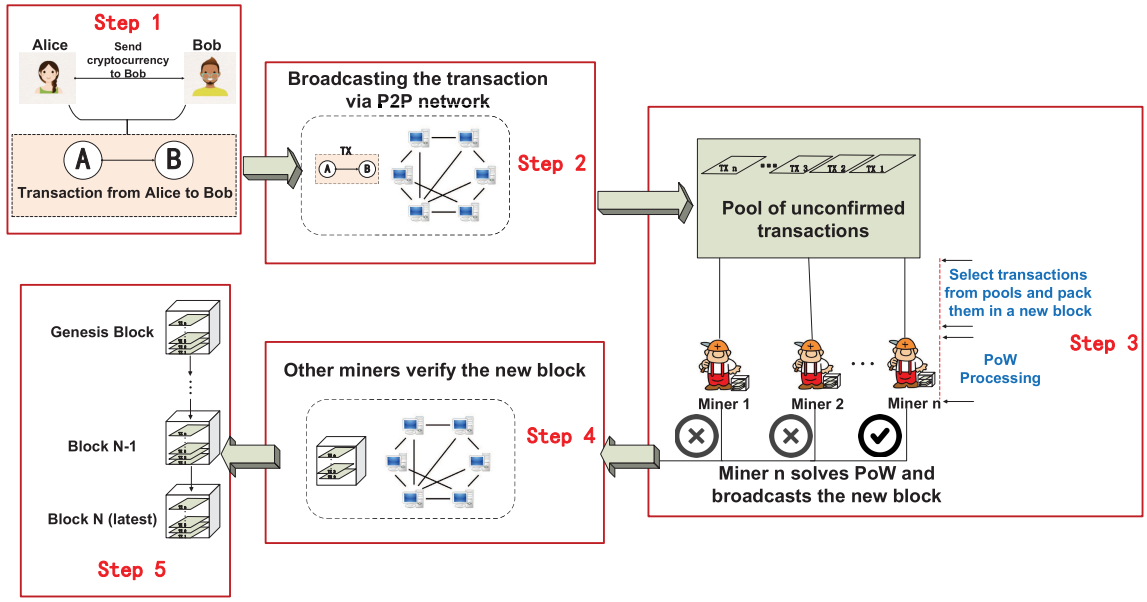


FIGURE 2. Blockchain transaction and POW consensus.

- Merkle root: is the root of a Merkle tree (or say a hash tree) that encodes the block transactions in an efficient and secure manner, which enables the quick verification of block transactions. Merkle root can be deemed as the hash of all the hashes of all the transactions in the block.

The content of blockchain contains a set of transactions (with inputs and outputs), where the inputs include the unspent transaction output (UTXO) and a signature (signed by the UTXO owner), and the outputs include the address (public key) of the recipient and the value to be transferred.

2) PROOF OF WORK AND TRANSACTION

Proof of work (PoW) is a consensus protocol used widely by many cryptocurrencies (such as Bitcoin and Ethereum), and the process is known as mining and the nodes participate in the mining process are called miners. In PoW, the miners struggle to solve a mathematical problem, which requires considerable work to complete it, but the result can be easily verified. The miner who is the first one to solve the problem has the right to mine the next block, and is rewarded a certain mount of cryptocurrency associated with the block. The average time to mine a block is around 10 minutes for Bitcoin network, and that is 20 seconds for the Ethereum network.

The workflow of blockchain transaction is shown in Fig. 2 and described below.

1) A user attempts to send a certain amount of cryptocurrency to someone else. A transfer transaction with a signature should be generated by the user.
2) The user broadcasts the transaction in P2P network, which is included in a pool of unconfirmed transactions and wait to be processed by the miners.
3) The miners select a collection of unconfirmed transactions from the pool and pack them in a new block. Then, these miners competes to solve a hard cryptography problem by finding a nonce (random number).
4) The miner who is the first one to get the satisfactory nonce broadcasts the new block in the network. The other miners verify the legitimacy of the new block. If it is valid, the other miners confirm its validity to reach a consensus.
5) Then, the new block is added to the blockchain.

PoW is a computationally intensive mathematical problem since the multiple miners working towards a common objective lead to tremendous wastage of computing power and electricity.

3) SMART CONTRACT AND ETHEREUM

The conception of smart contract was first proposed by Nick Szabo in 1995 [26]. A smart contract, in fact, is a digitized form of legal contract, which is represented by a program executed by a computer. Smart contract is utilized to build a trust relationship between the participates, which does not require a trusted third part (TTP). The implementation of smart contracts had not been realized for years for the sake of lacking a programmable digital system until the emergence of Bitcoin and Ethereum. The scripting language of Bitcoin is a weak version of smart contract since it has several limitations: lack of Turing-completeness and low scalability. Compared with Bitcoin, Ethereum is called a programmable blockchain. Instead of pre-defining a set of operations (as in Bitcoin), Ethereum allows the users to execute any complex operations according to demand. In this way, Ethereum serves as a platform for various types of decentralized blockchain applications, including but not limited to cryptocurrencies.

Ethereum platform allows a user (with a external owned account (EOA)) to call a smart contract (with a contract account) to implement specific function. Both EOA and contract account are featured with a 20-bytes hexadecimal string e.g., 0xca35b7d915458ef540ade6068dfe2f44e8fa733c. An Ethereum smart contract is deployed on Ethereum blockchain (in bytecode format) and executed in Ethereum Virtual Machine (EVM). A smart contract may include multiple functions. Therefore, an application binary interface (ABI) is necessary to specify which function in the contract is to be invoked and what is the format of the output. The contract account stores the code of smart contract, which can be triggered by EOA. In Ethereum, the user leverages private key to control his EOA, such as transferring *ether* to another address or triggering the execution of a smart contract. In this paper, we use smart contracts as a fair arbiter to verify the completeness and correctness of the search results provided by CP and guarantee the fair payment among data users, data owners and service providers.

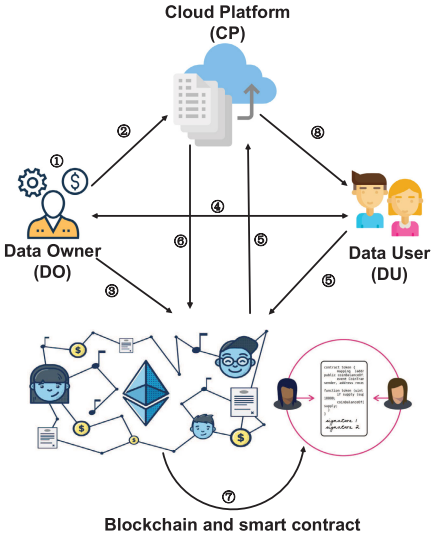


FIGURE 3. System workflow.

IV. SYSTEM MODEL

A. SYSTEM ARCHITECTURE

The blockchain based multi-keyword ranked search with fair payment (BMFP) system consists of the following entities.

- ***Data owner (DO)***: owns a set of documents to be outsourced to the cloud platform (CP). DO extracts multiple keywords from the plaintext file and constructs secure index and file ciphertext, which are sent to CP for remote storage. DO is responsible to authorize search privilege to the data users (DU) and earns message fee when DU searches on DO's data.
- ***Data user (DU)***: is authorized by DO who grants the search authority to him. To issue a search query, DU generates a search token and submits the search request to CP using smart contract. If the search result (returned by CP) is verified to be correct and complete (by smart contract), DU pays service fee to CP and message fee to DO; otherwise, no fee is payed.
- ***Cloud platform (CP)***: stores the encrypted index and encrypted files for the DOs and provides online search service to the DUs. CP is responsible to execute the search operation on encrypted index and returns the correct results to DUs in order to earn the service fee.
- ***Blockchain and smart contract***: Blockchain utilizes smart contract to record the verification data of the stored encrypted files for DO, such that smart contract could verify the correctness and completeness of the search results. Diverse smart contracts are deployed by DO or DU to execute the user management, fair payment and search related functions.

B. SYSTEM WORKFLOW

The workflow of BMFP is shown in Fig. 3, which includes the following steps.

1) DO generates the system parameter and secret keys.
2) DO extracts a keyword set from the files, and generates encrypted keyword index. DO also encrypts the files with symmetric encryption algorithm, and then outsources encrypted index and ciphertext to CP.
3) DO deploys smart contracts for user management and fair payment, and records the verification data in the smart contract for fair payment.
4) DU requests for search authorization, and DO grants the search privilege to DU utilizing the user management smart contract. After that, DO grants the authorized search key to DU.
5) DU deploys a smart contract for search related functions, generates a multi-keyword search token (using the authorized search key), and sends it to blockchain to trigger the fair payment smart contract. Before the request, DU is required to deposit enough search fee (including message fee and service fee) in the smart contract. If DU is verified to be an authorized user and enough search fee is deposited, the smart contract will automatically broadcast the search token in the blockchain and CP will receive it.
6) After executing the search operation according to the search token, CP returns the top-k ranked document identifiers to smart contract for verification.
7) According to DO's verification data, the fair payment contract verifies the correctness and completeness of the returned search result. If it is verified valid, the contract will automatically use DU's deposited search fee to pay message fee to DO and service fee to CP (according to the predefined allocation proportion); otherwise, DU's search fee is returned to his own account.
8) The ciphertext documents is sent to DU when the verification is passed. After receiving the ciphertext documents from CP, DU decrypts the encrypted files.

TABLE 2. Notations.

$\mathcal{D}$	plaintext documents collection $\mathcal{D} = \{D_1, D_2, ..., D_n\}$
$\mathcal{C}$	ciphertext documents collection $\mathcal{C} = \{C_1, C_2, ..., C_n\}$
$\mathcal{F}$	document identifiers collection $\mathcal{F} = (F_1, \cdots, F_n)$
$\mathcal{W}$	the keyword dictionary of $\mathcal{D}$
W	a keyword set
$token$	search token for keyword set W
$\mathcal{D}(W)$	plaintext document collection containing keyword set W
$\mathcal{C}(W)$	ciphertext document collection containing keyword set W
$\mathcal{F}(W)$	document identifiers collection containing keyword set W
$\mathcal{EF}(W)$	encrypted document identifiers collection containing keyword set W
$\|\mathcal{F}(W)\|$	the number of document identifiers containing keyword set W
$\mathcal{D}_k(W)$	top-k plaintext documents (containing W) with the highest weighted zone score
$\mathcal{C}_k(W)$	top-k ciphertext documents (containing W) with the highest weighted zone score
$\mathcal{F}_k(W)$	top-k document identifiers (containing W) with the highest weighted zone score
$F_j(W)$	the j-th document identifier in $\mathcal{F}(W)$
$EF_j(W)$	the j-th encrypted document identifier in $\mathcal{EF}(W)$
$\mathcal{EI}/proof$	encrypted index/verification data for encrypted index
$\mathcal{T}$	look-up table
$SEnc/SDec$	symmetric encryption/decryption algorithm
ek/sk	symmetric encryption key/symmetric search key
γ_{κ_1}	pseudo random function (PRF) with key κ_1
μ_{κ_2}	message authentication code (MAC) with key κ_2
$a \parallel b$	concatenation of strings a and b
FPC	fair payment contract
UMC	user management contract
UIC	user interface contract

C. FORMAL DEFINITION

In this work, the BMFP system contains seven algorithms. The main notations of this paper is shown in Table. 2.

- $Setup(1^\lambda) \rightarrow PP$: This algorithm is executed by DO. Taken as input a security parameter λ, it outputs the public parameter PP. (See step 1 of Fig. 3.)
- $KeyGen(1^\lambda) \rightarrow (ek, sk, vk)$: Taken as input a security parameter λ, DO generates the symmetric encryption key ek, search key sk and verification key vk. (See step 1 of Fig. 3.)
- $Enc(\mathcal{D}, \mathcal{W}, ek, sk, vk) \rightarrow (\mathcal{EI}, \mathcal{C}, proof)$: Taken as input the document collection $\mathcal{D}$, DO extracts the keyword dictionary $\mathcal{W}$ from $\mathcal{D}$ to generate encrypted index $\mathcal{EI}$ utilizing sk. Plaintext document collection $\mathcal{D}$ is encrypted to ciphertext collection $\mathcal{C}$ using symmetric encryption algorithm $SEnc$ and encryption key ek. DO utilizes vk to generate verification data $proof$ for $\mathcal{EI}$. In addition, DO deploys user management and fair payment smart contracts on blockchain. (See step 2-3 of Fig. 3.)
- $Trapdoor(W, sk) \rightarrow token$: DU deploys search related smart contract on blockchain and requests search privilege from DO. If it is permitted, DO grants search key sk to DU. Taken as input a keyword set W and search key sk, DU generates the multi-keyword search token $token$. (See step 4-5 of Fig. 3.)
- $Search(\mathcal{EI}, token) \rightarrow (\mathcal{C}_k(W), \mathcal{F}_k(W))$: Taken as input encrypted index $\mathcal{EI}$ and search token $token$, CP outputs the top-k most relevant results $(\mathcal{C}_k(W), \mathcal{F}_k(W))$ and $proof$ to smart contract for verification. (See step 6 of Fig. 3.)
- $Verify(vk, proof, token, \mathcal{F}_k(W)) \rightarrow 1/0$: Taken as input verification key vk, verification data $proof$, search token $token$, search result $\mathcal{F}_k(W)$, smart contract verifies the correctness and completeness of the result. If it is valid, the fair payment contract outputs 1 and transfers the message/service fee to DO/CP. Otherwise, the contract outputs 0 and returns the search fee to DU. (See step 7 of Fig. 3.)
- $Dec(\mathcal{C}_k(W), ek) \rightarrow \mathcal{D}_k(W)$: Taken as input ciphertext collection $\mathcal{C}_k(W)$ and symmetric encryption key ek, DU recovers the plaintext collection $\mathcal{D}_k(W)$. (See step 8 of Fig. 3.)

V. CONCRETE CONSTRUCTION

In this section, the concrete construction of BMFP system is introduced according to the seven algorithms shown in Section IV. Here, we use Ethereum as the blockchain platform to design the concrete scheme.

• $Setup(1^\lambda) \rightarrow PP$: Given security parameter λ, DO selects a pseudo random function (PRF) $\gamma_{\kappa_1} : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^d$ and a MAC function $\mu_{\kappa_2} : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^l$, where d is the length of document identifier and l is the output length of a standard MAC function (such as SHA256 based HMAC). DO chooses symmetric encryption/decryption algorithm pair $SEnc/SDec$ with symmetric key space $\mathcal{EK}$. Set public parameter as $PP = (\gamma_{\kappa_1}, \mu_{\kappa_2}, SEnc/SDec)$.

• $KeyGen(1^\lambda) \rightarrow (ek, sk, vk)$: Given security parameter λ, DO randomly selects secret keys $\kappa_1, \kappa_2 \in_R \{0,1\}^\lambda$ and a symmetric encryption key $ek \in_R \mathcal{EK}$. Set the search key as $sk = \kappa_1$ and the verification key as $vk = \kappa_2$.

• $Enc(\mathcal{D}, \mathcal{W}, ek, sk, vk) \rightarrow (\mathcal{EI}, \mathcal{C}, proof)$: Given a plaintext document collection $\mathcal{D}$, DO extracts several keywords from each document to form a keyword dictionary $\mathcal{W} = (w_1, \ldots, w_\tau)$.

TABLE 3. Inverted index structure of BMFP.

W	$\mathcal{F}(W)$	W	$\mathcal{F}(W)$
(w_1, w_1, w_1)	$(F_1, F_2, F_3, \cdots)$	(w_1, w_2, w_3)	$(F_2, F_6, F_7, \cdots)$
...	...	...	...
(w_τ, w_τ, w_τ)	$(F_1, F_5, F_{10}, \cdots)$	(w_1, w_2, w_τ)	$(F_{11}, F_{14}, F_{15}, \cdots)$
(w_1, w_2, w_2)	$(F_2, F_3, F_8, \cdots)$	(w_1, w_3, w_4)	$(F_6, F_{10}, F_{11}, \cdots)$
...	...	...	...
(w_1, w_τ, w_τ)	$(F_1, F_5, F_8, \cdots)$	(w_1, w_3, w_τ)	$(F_1, F_2, F_4, \cdots)$
(w_2, w_3, w_3)	$(F_9, F_{12}, F_{15}, \cdots)$	(w_2, w_3, w_4)	$(F_4, F_{22}, F_{31}, \cdots)$
...	...	...	...
(w_2, w_τ, w_τ)	$(F_4, F_{16}, F_{18}, \cdots)$	(w_2, w_3, w_τ)	$(F_7, F_{11}, F_{14}, \cdots)$
...	...	...	...
$(w_{\tau-1}, w_\tau, w_\tau)$	$(F_2, F_4, F_{11}, \cdots)$	$(w_{\tau-2}, w_{\tau-1}, w_\tau)$	$(F_5, F_6, F_8, \cdots)$

Firstly, BMFP adopts the inverted index data structure to realize multi-keyword ranked search. To simplify the notation, Table. 3 shows an example of the inverted index supporting three conjunctive keyword search. Assume the three keyword set is denoted as $W = (w_i, w_j, w_k)$, and the keywords are arranged in lexicographical order. If DU wants to query less than three keywords, the query should be extended: a single keyword query for w_i is extended to (w_i, w_i, w_i); a two keyword query for (w_i, w_j) is extended to (w_i, w_j, w_j). The identifiers of files containing the keyword set W is denoted as $\mathcal{F}(W) = (F_{\theta_1}, F_{\theta_2}, \ldots)$, and the files in $\mathcal{F}(W)$ are sorted

according to the weighted zone score $S_{W,F} = \sum_{\alpha=1}^{3} S_{w_\alpha,F}$ introduced in Subsection III-A.

Based on the above inverted index, DO utilizes search key $sk = \kappa_1$ and verification key $vk = \kappa_2$ to build the encrypted index containing verification data. The encrypted index is composed of a look-up table $\mathcal{T}$ and the associated encrypted file identifiers. The look-up table is a $< key, value >$ structure where the *key* field contains the output of a pseudo random function γ_κ, and the *value* field contains a tuple $< value, proof >$. The *value* field stores the encrypted address of the file identifier array; and the *proof* field stores the proof data for the multi-keyword ranked search result.

Concretely, DO calculates $\gamma_{\kappa_1}(W)$ for each keyword set W in the inverted index, sets $\mathcal{T}[\gamma_{\kappa_1}(W)].value = address(\mathcal{F}(W)) \oplus \gamma_{\kappa_1}(W \parallel 0)$ and $\mathcal{T}[\gamma_{\kappa_1}(W)].proof = \mu_{\kappa_2}(\gamma_{\kappa_1}(W) \parallel F_1(W) \parallel \cdots \parallel F_k(W))$, where $\mathcal{F}_k(W) = (F_1(W), \ldots, F_k(W))$ are the top-k identifiers of the documents that have the highest weighted zone scores. The notation $address(\mathcal{F}(W))$ denotes the address pointed to the file identifier collection $\mathcal{F}(W)$. If the number of files containing W is β and $\beta < k$, we construct $\mathcal{T}[\gamma_{\kappa_1}(W)].proof = \mu_{\kappa_2}(\gamma_{\kappa_1}(W) \parallel F_1(W) \parallel \cdots \parallel F_\beta(W))$ and $\mathcal{F}_k(W) = (F_1(W), \ldots, F_\beta(W))$. The file identifiers in $\mathcal{F}_k(W)$ are encrypted to $\mathcal{EF}(W) = (EF_1(W), \ldots, EF_k(W)) = (F_1(W) \oplus \gamma_{\kappa_1}(W \parallel 1), \ldots, F_k(W) \oplus \gamma_{\kappa_1}(W \parallel 1))$. The plaintext document collection $\mathcal{D}$ is encrypted to ciphertext document collection $\mathcal{C}$ using symmetric encryption algorithm *SEnc* and symmetric key ek. DO sets the encrypted index as $\mathcal{EI} = (\mathcal{T}, \{\mathcal{EF}(W)\}_{W \subseteq \mathcal{W}})$, and outsources $(\mathcal{EI}, \mathcal{C})$ to cloud platform.

After that, DO deploys a fair payment contract (FPC) in Ethereum and the verification key $vk = \kappa_2$ is embedded in FPC as a *private argument* (coded in Solidity), where κ_2 is only known to DO. FPC is the core component in BMFP, which is responsible for checking the validity of each DU requesting for a search service, recording and broadcasting the search token, verifying the search results from CP and eventually achieving fair payments. After FPC is deployed, DO builds user management contract (UMC) to enroll the authorized users. The detail of FPC and UMC are given in subsection VI-D and VI-C.

- $Trapdoor(W, sk) \rightarrow token$: The trapdoor generation algorithm is run by DU. When DU requests a search service from CP for the first time, he should firstly request search privilege from DO. If the request is permitted, DO grants search key sk to DU. Taken as input a query keyword set W and a search key sk, DU generates the multi-keyword search token *token*. Then, DU deploys a user interface contract (UIC) to deposit enough search fee in the *deposit pool* of FPC (associated with his own account). Specifically, DU generates the search trapdoor as $token = (\gamma_{\kappa_1}(W), \gamma_{\kappa_1}(W \parallel 0), \gamma_{\kappa_1}(W \parallel 1))$, and DU invokes the *initRequest*() function of FPC to upload the token to FPC. UIC is also used to receive the verified results from FPC. The details of UIC is shown in subsection VI-E.
- $Search(\mathcal{EI}, token) \rightarrow (\mathcal{C}_k(W), \mathcal{F}_k(W))$: Receiving the search token *token*, FPC invokes UMC to check whether DU is an authorized user. If DU is authorized, FPC emits an Ethereum *event* for informing CP to execute search operation. Capturing the *event* emitted by FPC, CP parses the *event* into a tuple $(\gamma_{\kappa_1}(W), \gamma_{\kappa_1}(W \parallel 0), \gamma_{\kappa_1}(W \parallel 1))$ to execute the search operation. In the look-up table $\mathcal{T}$, CP uses $\gamma_{\kappa_1}(W)$ to search for $\mathcal{T}[\gamma_{\kappa_1}(W)].value$ and $\mathcal{T}[\gamma_{\kappa_1}(W)].proof$. CP recovers an address $address(\mathcal{F}(W))$ pointed to $\mathcal{F}(W)$ by computing $\mathcal{T}[\gamma_{\kappa_1}(W)].value \oplus \gamma_{\kappa_1}(W \parallel 0)$. CP recovers the file identifier $F_j(W)$ by computing $EF_j(W) \oplus \gamma_{\kappa_1}(W \parallel 1)$ for each $F_j(W) \in \mathcal{F}_k(W)$. Then, CP sends $\mathcal{F}_k(W)$ and $\mathcal{T}[\gamma_{\kappa_1}(W)].proof$ to FPC for verification.
- $Verify(vk, proof, token, \mathcal{F}_k(W)) \rightarrow 1/0$: This process is run independently by the fair payment contract (FPC). Receiving an identifier collection $\mathcal{F}_k(W)$ ranked by weighted zone score, FPC verifies the correctness and completeness of the identifier collection $\mathcal{F}_k(W)$. Suppose the received proof data from CP is *Proof* and the search token from DU is $token = (\gamma_{\kappa_1}(W), \gamma_{\kappa_1}(W \parallel 0), \gamma_{\kappa_1}(W \parallel 1))$, FPC re-computes $proof' = \mu_{\kappa_2}(\gamma_{\kappa_1}(W) \parallel F_1(W) \parallel F_2(W), \ldots, F_k(W))$ and verifies whether the equation $proof = proof'$ holds. If it is true, FPC transfers DU's search fee to DO and CP (as message fee and service fee, respectively) from *deposit pool* according to the predefined allocation proportion, and sends the search result to UIC smart contract. Otherwise, FPC transfers DU's deposited search fee back to his own account.
- $Dec(\mathcal{C}_k(W), ek) \rightarrow \mathcal{D}_k(W)$: DU gets the search result $\mathcal{C}_k(W)$, and decrypts the ciphertexts using symmetric key ek to obtain the plaintext document collection, i.e., $\mathcal{D}_k(W) \leftarrow SDec_{ek}(\mathcal{C}_k(W))$.

VI. SMART CONTRACT DESIGN

In this section, we introduce the basic variables and functions in Solidity [32], and illustrate the interactions among the smart contracts. Then, we construct the concrete smart contracts (programmed in Solidity) for Ethereum.

A. BASIC VARIABLES AND FUNCTIONS

We introduce some variables in the global namespace to provide information about the blockchain, and some functions of general utility in Solidity. The following variables are used in this work.

- *msg.value* indicates number of *wei* associated with the transaction. We use \$fee to represent the number of *wei* that user should pay for a searching operation. Note that 1 ETH = 10^{18} *wei*.
- *tx.origin* corresponds to an initial address when a full call chain is created. Suppose an external owned account (EOA) with address A invokes a smart contract with address B, and another contract with address C is invoked in contract B, in that way a full call chain A → B → C is formed and the origin address A is *tx.origin*.
- *msg.sender* refers to an account address that directly invokes a smart contract, which can be an EOA or

a contract account. When a smart contract is deployed, the *msg.sender* is the address of the smart contract creator; when a function is called, the *msg.sender* is associated with the address of the smart contract caller.

The following functions are used in this paper.

- *Function assert (bool prerequisite)*: The bool value of prerequisite means that whether the prerequisite for running the following statement is satisfied or not. If and only if the prerequisite is met (i.e., the bool value of the prerequisite is true), the following statements of program will be executed. Otherwise, the exception will be thrown and the program execution will be suspended.
- *Function address.transfer (uint amount)*: Send a given amount of *wei* to the *address*, which can be an external owned account or a contract account.

B. INTERACTIONS AMONG SMART CONTRACTS

Since both of EOA and contract account accept ETH/ETC, the EOA of DUs can establish a transaction to transfer *ether* to the contract account of FPC. Denote *deposit pool* as the amount of *ether* that FPC holds. Recall that we leverage smart contract to verify the search results from CP, and both the completeness and correctness of search results are guaranteed using smart contract. The contract interaction in BMFP is shown in Fig. 4, which includes the following steps.

1) DO negotiates with CP on the allocation proportion and search fee. Then, DO deploys FPC and UMC, and DU deploys UIC on Ethereum.
2) DU deposits $amount *wei* into the *deposit pool* of FPC.
3) DU issues a search token to FPC attached with the address of his own UIC.
4) FPC checks whether the DU has enough *wei* in *deposit pool* and whether DU is an authorized user by invoking UMC.
5) If the conditions in 4) are satisfied, FPC broadcasts the token, and then CP receives it and returns search results after executing search operation.
6) FPC verifies the results from CP using verifying key stored in FPC.
7) If the verification function in FPC outputs true, it transfers the message fee and service fee to DO and CP from *deposit pool*, respectively, and invokes the UIC to save the identifiers collection. (DU's search fee is divided into service fee and message fee according to the predefined allocation proportion);
8) Otherwise, the search fee in the *deposit pool* is sent back to DU.

In BMFP, DU gets the correct results if the search fee is transferred to CP and DO; CP is rewarded the service fee as long as it faithfully executes the search algorithm; DO is rewarded the message fee when a search process is successfully operated.

C. USER MANAGEMENT CONTRACT

DO deploys user management contract (UMC) to manage an authorization user list *userList*, which maps a user address to a Boolean value ("1" represents an authorized user address, "0" represents a revoked user address). DO is able to add/delete user by invoking *addUser*/*removeUser* functions in UMC, which can only be executed by DO. The function *verifyUser* is called by FPC for user authentication utilizing the *userList*. The sketch of UMC is shown in Listing. 1.

```
contract userManagement{
  address DO;
  address FPC;
  mapping (address=>bool) private userList;

  function addUser(address userAddr){
    if(msg.sender == DO){
      add the userAddr into the userList.
    }
  }
  function removeUser(address userAddr){
    if(msg.sender == DO){
      remove the userAddr from userList.
    }
  }
  function verifyUser(address userAddr) public
      view returns(bool){
    if(msg.sender == FPC){
      verify the validity of tx.origin
    }
  }
}
```

Listing 1. Sketch of user management contract.

D. FAIR PAYMENT CONTRACT

FPC is deployed by DO after DO and CP has completed the negotiation about the allocation proportion of search fee. FPC verifies the search result of the submitted query *token*: once CP offers wrong results or does not offer all of the results, the search results will be rejected by FPC and CP will receive no payment. Oppositely, once FPC gets the correct results associated with a specific token, the search results will be authenticated and FPC will transfer the search fee from the *deposit pool* to CP and DO (according to the allocation proportion), which solves the problem that CP deliberately returns partial or wrong results to save computing resources. On the other hand, automatic payment from the *deposit pool* will be triggered if a correct search result is provided. Thus, DU cannot interrupts the process of payment since the money will be automatically deducted from his account in *deposit pool* of FPC. DO should record the application binary interface (*ABI*) and the account address *addr* of FPC, and make them public in the network. All the parties involved in this system can validate that the FPC deployed on the *addr* is exactly identical with source code via *Etherscan* [33]. The code sketch of FPC is shown in Listing. 2. The following three function interfaces are offered by FPC:

- *deposit()* $\rightarrow$ *balance value*: DU invokes this function to deduct money from his EOA account and deposits it into the *deposit pool* of FPC. When FPC receives a deposit from a DU, it updates the balance value of this DU by adding the deposit value into this users' account.
- *initRequest*(*token,address*) $\rightarrow$ *Ethereum event*: This function is used to be called by DUs to request a search request. The *initRequest* function will check the validity

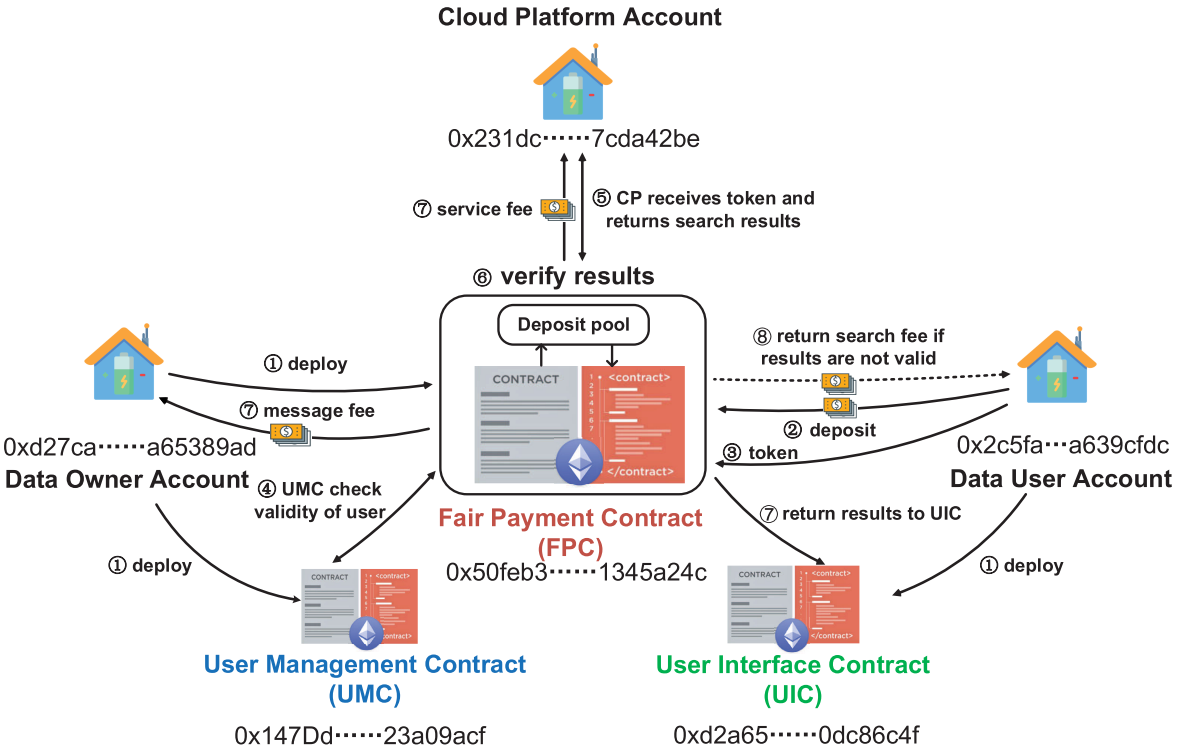


FIGURE 4. Interactions among smart contracts.

of caller by invoking the *verifyUser* function in UMC. Once the address of DU is an element in *userList*, that is to say, DO is willing to share his documents with this DU, and DU has enough amount of *wei* in *deposit pool*, the last line of *initRequest* function emits an *Ethereum event* related to this token. CP listens to those events emitted by FPC. CP receives and parses the *event* into tuple (*userAddr, token*), which is used as the input of the search function. Once the search operation is completed, CP calls the *verifyResultFromCP* function in FPC to authenticate the results and get the service fee.

- *verifyResultFromCP*(*userAddr, identifiers, proof*) → *Boolean*: The function is invoked by CP. As long as the search results from CP is verified to be complete and correct (by *verify* algorithm in sectionV), FPC transfers $\$fee \times (1 - proportion)$ amount *wei* to CP and $\$fee \times proportion$ amount *wei* to DO. Otherwise, the search fee is returned to DU. Eventually, the function invokes *receiveResults* function of UIC associated with specific *userAddr* to save the results. In listing 2, we assume that CP and DO divide the search fee equally, i.e., the allocation proportion of search fee is 1:1.

E. USER INTERFACE CONTRACT

In the peer-to-peer network of Ethereum, user's contract interfaces (as well as server's interfaces) can listen for events being emitted on the blockchain without much cost by running web3.js libraries of JavaScript, which makes it easy to track transactions. However, if CP returns the search results by emitting events, it may has security risks. Everyone monitoring the blockchain will get some results without any authentication mechanism. In order to solve this problem, we introduce the user interface contract (UIC) deployed by DU. Once the search results are authenticated with completeness and correctness, FPC invokes the UIC to record the search results. Only the creator of UIC (i.e., the data user) has the right to check the search results: *receiveResults* function receives valid results sent by FPC; *getSearchResults* function returns search results after assuring that the caller of this function is the contract creator. The code sketch of UIC is described in Listing 3.

VII. PERFORMANCE ANALYSIS

In this section, we compare the functionalities of BMFP with the other schemes in [9], [20]–[24]. Then, BMFP is implemented in Ethereum testing network to evaluate its performance.

A. FUNCTION COMPARISON

As shown in Table. 4, we compare the functions of BMFP with the available searchable encryption schemes: Hu's scheme [20], Chen's scheme [21], Wang's scheme [22], Wu's scheme [23], Zhang's scheme [24], and Cai's scheme [9].

Consider the following properties in a searchable encryption scheme: privacy-preserving multi-keyword search, search results ranking and verification of search results.

TABLE 4. Comparison of various schemes.

Properties	[20]	[21]	[22]	[23]	[24]	[25]	BMFP
Multi-keyword	×	×	×	×	×	×	√
Result ranking	×	×	×	×	×	×	√
Result Verification	√	√	√	√	√	√	√
Decentralization	√	√	√	×	√	√	√
Fair Payment	√	√	√	×	√	√	√
Access control	×	×	√	√	×	×	√
Blockchain	Ethereum	Ethereum	Ethereum	Ethereum	Bitcoin	Ethereum	Ethereum
Executor of search algorithm	blockchain	blockchain	blockchain	blockchain	cloud	cloud	cloud
Expenditure in blockchain	high	high	high	high	high	low	low

```
contract fairPayment{
  struct dataUser{
    bytes32 token;
    uint256 balance;
    address UIC;
  }
  address payable DO;
  address payable CP;
  mapping(address=>dataUser) private addrToDU;
  bytes32 private Key;
  uint256 constant fee;
  event getToken(bytes32 token, address userAddr);

  constructor(bytes32 verifykey){
      key = verifykey;
  }
  function deposit() public returns(uint256){
    if(msg.value != 0){
      addrToDU[msg.sender].balance += msg.value;
    }
    return addrToDU[msg.sender].balance;
  }
  function initRequest(bytes32 token,address UIC){
    invoke verifyUser function in UMC
    assert(addrToDU[msg.sender].balance >= fee);
    addrToDU[msg.sender].token = token;
    addrToDU[msg.sender].UIC = UIC;
    emit getToken(token,msg.sender);
  }
  function verifyResultFromCP(userAddr,ids,proof){
    Tag=Hmac(Key,addrToDU[userAddr].token,ids);
    if(proof == Tag){
      DO.transfer(fee/2);
      CP.transfer(fee/2);
      addrToDU[userAddr].balance -= fee;
    }else{
      userAddr.transfer(fee);
      addrToDU[userAddr].balance -= fee;
    }
    invoke receiveResults function from
    addrToDU[userAddr].UIC
  }
}
```

Listing 2. Sketch of fair payment contract.

```
contract userInterface{
  address DU;
  address FPC;
  mapping(bytes32=>string[]) resultHistory;
  function receiveResults(bytes32 token, ids){
    if(msg.sender == FPC){
      save the ids into resultHistory;
    }
  }
  function getSearchResults(bytes32 token) public
      view returns(string[] results){
    if(msg.sender == DU){
      return results from resultHistory.
    }
  }
}
```

Listing 3. Sketch of user interface contract .

The schemes in [9], [20]–[24] only support single-keyword search over encrypted inverted index. The schemes in [20]–[23] use smart contracts to store encrypted index and execute the single-keyword search algorithm. In order to store the large volume index in blockchain, these schemes [20]–[23] have to partition the complex searchable index into thousands of pieces, and store them in thousands of blockchain transactions (due to the low storage capacity of each transaction). These transactions have to be dealt with (i.e., upload to blockchain) one-by-one (rather than in a concurrent way), which takes a tremendous amount of time. The schemes in [9], [24] and BMFP scheme, the cloud platform (rather than smart contract) is tasked to store the encrypted index and implements the keyword search operation, which is much more efficient than [20]–[23]. However, [9], [24] only support single keyword search, while BMFP scheme is more flexible to realize multi-keyword search.

The schemes [9], [20]–[24] do not rank the retrieval result, which is not convenient for usage. Compared to these schemes, our proposed BMFP system returns ranked documents in a specific order based on the weighted zone score. The schemes in [20]–[23] execute the search operation by Ethereum smart contract without the need of verifying the result, while Zhang's scheme [24] exploits pre-defined input-script and output-script of Bitcoin to verify the search result. Therefore, these schemes [20]–[24] achieve result verification based on the trustworthy of smart contract execution. Cai's scheme [9] does not execute the verification algorithm unless the data user applies for an arbitration request. Receiving the request, each arbiter node independently performs the judgment process, who re-implements the keyword search algorithm to verify whether the search result is correct. Then, these individual arbitration results converge to an arbitration smart contract to make a decision. It can be seen that the scheme in [9] wastes a lot of computation resources in the arbitration process.

Wu's scheme [23] is not a fully decentralized scheme since it involves a key manager, while the other schemes do not rely on any trusted entity. Exploiting the built-in payment mechanism of Bitcoin and Ethereum, all these schemes [9], [20]–[24], BMFP include a fair payment protocol except Wu's scheme [23]. The schemes in [22], [23] and BMFP support access control, where the data owner specifies the authorized user set in smart contract. The expenditures in blockchain are quite high in [20]–[24] due to the following reasons. The schemes in [20]–[23] leverage Ethereum smart contracts to execute the whole search algorithm, which results in a mass of expenditure due to the costly smart contract executing. And the Bitcoin based time commitment scheme in [24] also consumes a respectable amount of Bitcoin and the price of Bitcoin is too high. Cai's scheme [9] and our BMFP simply use smart contract to implement the verification process rather than the costly search algorithm, which greatly reduces the expenditure in blockchain. Compared with the above schemes, our proposed BMFP scheme is more flexible in multi-keyword ranking search and cost-efficient in search result verification.

B. PERFORMANCE TEST

In this section, we implement the proposed BMFP scheme to analyze its feasibility and performance. We implement the searchable encryption algorithms in BMFP using Java and construct the Ethereum smart contracts using Solidity. The pseudo random function γ_{κ_1} is instantiated by HMAC-MD5 and message authentication code function μ_{κ_2} by HMAC-SHA256. Since BMFP involves blockchain and searchable encryption, the specific configuration of experimental platform for blockchain and searchable encryption are listed as follows:

- Searchable encryption test environment: The searchable encryption algorithms are implemented by Java language and run on a laptop with a Intel core i5-7300HQ@2.50GHz processor, 8GB RAM, a 512 GB SSD and the system operation is ubuntu 18.04LTS.
- Ethereum platform: Solidity language is used to program smart contract, and the simulation is conducted on Ethereum Virtual Machine (EVM). We implement the smart contracts with Ethereum TestRPC, which is a fast and customizable blockchain emulator and allows making calls to the blockchain without the overheads of running an actual Ethereum node. At the time of this writing, the exchange rate of *ether* and the USD is 1 *ether* = 236 USD.

1) PERFORMANCE OF BUILDING ENCRYPTED INDEX AND SEARCH

The database of the experiment is 5000 English academic papers, which is used to test the encrypted index building algorithm and search algorithm. For each paper, a weighted zone score is assigned to the document, which is calculated as below. Each paper contains 3 zones: title, abstract, and body, where the weights of "title/abstract/body" zones are 0.6/0.3/0.1, respectively. The extracted keywords are assigned with different weights according to the specific zone. The keyword index dataset (built from 5000 papers) contains 5,018,562 tuples of (*kws*, *ids*) pairs, where *kws* represents the three keywords (arranged in lexicographical order) and *ids* represents the set of file identifiers (sorted with the weighted zone score in descending order). We test the performance of searchable encryption algorithm in BMFP with different numbers of entries (*kws*, *ids*), which ranges from 295,240 to 5,018,562 shown in Fig. 5. It can be seen that it costs only 2.089 seconds to build the encrypted index for 295,240 keyword set entries; and the building time for 5,018,562 entries is less than 40 seconds.

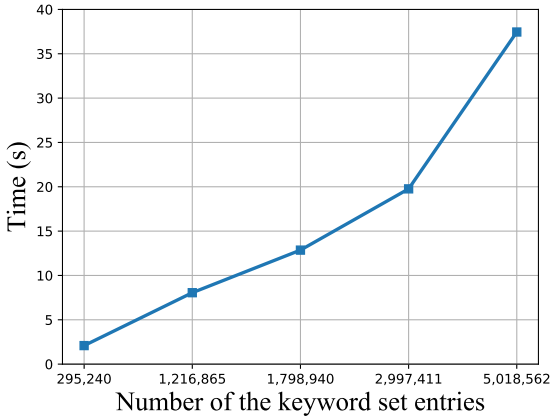


FIGURE 5. Time for encrypted index building.

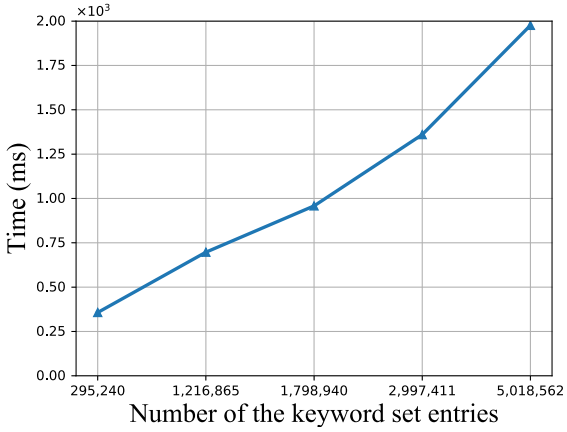


FIGURE 6. Time for search.

Fig. 6 shows the search time that varies with the keyword set entries, where the search time is the average time calculated from 100 independent queries. It can be observed that the search time grows with the number of (*kws*, *ids*) entries. The average search time for a query is less than 1 second when the entries is less than 1,798,940; and the search time is 1.976 second for 5,018,562 entries. The experimental result shows that BMFP is efficient for practical usage.

2) PERFORMANCE OF SMART CONTRACTS

The deployment and invoking of smart contract will produce cryptocurrency cost. The Ethereum gas consumed in

TABLE 5. Cost of smart contracts (1 Gas = 2×10^{-9} ETH, 1 ETH = 236 USD).

Operation / Cost	Gas ($\times 10^5$)	ETH ($\times 10^{-3}$)	USD
deployment of user management contract (UMC)	3.55669	0.711338	0.1678
deployment of fair payment contract (FPC)	12.40226	2.480452	0.5853
deployment of user interface contract (UIC)	6.61865	1.323730	0.3124
execution of *addUser* function	0.23798	0.047596	0.0112
execution of *removeUser* function	0.14321	0.028642	0.0067
execution of *verifyUser* function	0.23264	0.046528	0.0109
execution of *getSearchResults* function	0	0	0

the experiment of the smart contracts is shown in Table. 5, where Ethereum gas is an unit that measures the amount of computational effort that it will take to execute certain operations. At the time of paper writing, the Ethereum transaction price is 1 ETH $=$ 236 USD. Suppose the gas price is 1 $gas = 2 \times 10^9$ wei. Since 1 $wei = 10^{-18}$ ETH, it can be deduced that 1 $gas = 2 \times 10^{-9}$ ETH $= 4.72 \times 10^{-7}$ USD.

As shown in Table. 5, the deployment cost of deployment of user management contract (UMC), fair payment contract (FPC), user interface contract (UIC) are 0.1678 USD, 0.5853 USD and 0.3124 USD, respectively. These expenditures are payed only once during the smart contract deployment phase.

The *addUser* interface is executed by DO to register a new DU and grants the data access authority. The *removeUser* interface is invoked by DO to revoke the search and access authority of a DU. The *verifyUser* interface is used by FPC to verify the validity of DU who has submitted a search token. The execution costs of these three functions are 0.0112 USD, 0.0067 USD and 0.0109 USD, respectively. Since the *getSearchResults* interface only does the read operation to get *resultHistory*, which does not modify the state of blockchain and thus has no cost.

Since the data retrieval is frequently operated in the system, it is necessary to provide an efficient and inexpensive verification mechanism. The Solidity function library contains SHA256 algorithm, but HMAC function is not provided. Following the standard method of HMAC [27], we construct HMAC-SHA256 algorithm (based on SHA256) in Solidity language, which is invoked by the *verifyResultFromCP* function. The efficiency and gas consumption of *verifyResultFromCP* are measured with different numbers N of document identifiers, which ranges from 10 to 50. As shown in Fig. 7, the gas cost of result verification slightly increases with N, which is 1.54234×10^5 $gas = 0.0727$ USD for $N = 10$ and 1.80632×10^5 $gas = 0.0852$ USD for $N = 50$. The execution time of *verifyResultFromCP* interface is shown in Fig. 8, which is 2.928 seconds for $N = 10$, and 3.778 seconds for $N = 50$. The simulation results indicate the efficiency and low cost of our result verification algorithm.

In Fig. 9, we evaluate the gas consumption of *receiveResults* interface with different numbers of file identifiers

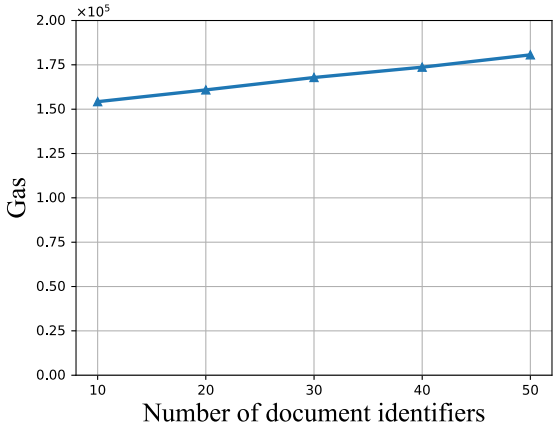


FIGURE 7. Expenditure of result verification.

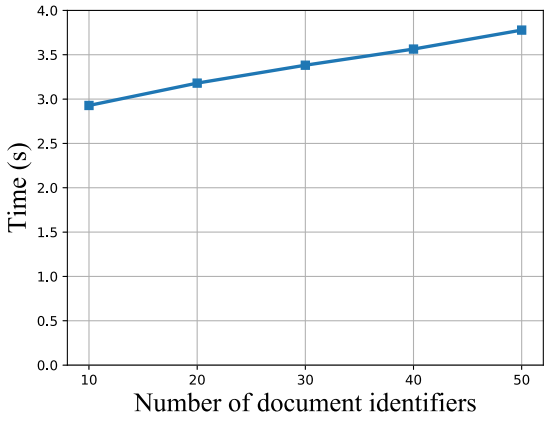


FIGURE 8. Time for verification.

(ranging from 10 to 50), and the length of the file identifier may be 18 bytes, 46 bytes and 81 bytes. The gas cost of the execution of *receiveResults* interface linearly increases with the number and size of the file identifiers. The cost of uploading 3.95 KB search results to blockchain is 117,044 $gas = 0.055$ USD.

According to the analysis, the above experimental results demonstrate the efficiency and low expenditure of BMFP.

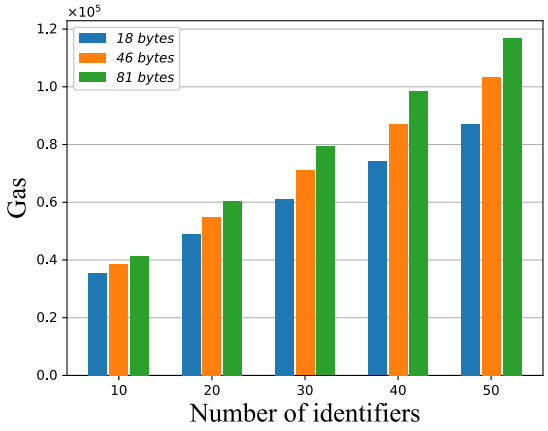


FIGURE 9. Expenditure of *receiveResults* function.

VIII. SECURITY ANALYSIS

In this section, we analyze the security of the proposed blockchain based multi-keyword ranked search with fair payment system. Specifically, the fair payment and soundness of BMFP are analyzed. A formal proof in known ciphertext model is also given.

A. FAIR PAYMENT

In traditional searchable encryption schemes, a reliable cloud service is required to honestly perform search operations, and return correct and complete search results. However, the cloud service is actually not totally honest. If the cloud platform is paid by the data user in advance, it may return incomplete or even erroneous results to lower the computation cost. On the other hand, if the search results are sent to data user before the payment, the dishonest data user may intentionally refuse to pay the service fee even though the search result is correct. The proposed BMFP scheme ensures *fair exchange* among cloud platform, data owner and data user, which is guaranteed by the reliability of smart contracts. A predefined verification mechanism is honestly performed by the smart contracts. If the server is dishonest to return the incorrect or incomplete result, the verification contract will discover the misbehavior, and the search fees are returned to the data user. If the returned search result is verified correct, the data user cannot refuse to pay the service fee to server and message fee to data owner since the fees are already locked by the fair payment contract (FPC). Thus, the fair payment is guaranteed in BMFP.

B. SOUNDNESS

In our scheme, the search result verification is achieved by the fair payment contract deployed on Ethereum. The soundness of our proposed scheme is ensured by the security of Ethereum. Since the PoW consensus mechanism used in Ethereum is a computationally intensive mathematical problem, where a large amount of miners work towards a common objective. Any adversary who wants to tamper the state in the FPC smart contract should control more than 50% of the entire Ethereum network's computation power. So it is impossible for an individual to make changes to the state and pre-defined logic of smart contract. The PoW consensus guarantees the soundness of the proposed scheme.

C. FORMAL PROOF

The access pattern can be obtained by cloud platform by recording the query trapdoor and the search results. In known ciphertext model, cloud platform does not get any additional information except for the access patterns [31].

Theorem 1: BMFP scheme is secure in the known ciphertext model [31].

The following notions are used in the security proof.

- *History* is H=($\mathcal{D}$, $\mathcal{I}$, W), where $\mathcal{D}$ is a document set, $\mathcal{I}$ is index built from $\mathcal{D}$, $W = (w_1, w_..w_m)$ is the query keyword set.
- *View* is $V(H) = (\mathcal{C}, \mathcal{EI}, token)$, where $\mathcal{C}$ is the encrypted document set (using the symmetric encryption key ek). $\mathcal{EI}$ is the encrypted index (using the key of pseudo random function sk). *token* is the encrypted query keyword set(using the key of pseudo random function sk). The contents in *V(H)* is open to cloud platform.
- *Trace of a history* is the sensitive information learnt by the cloud server, such as the access pattern. The trace of a history is defined as $Tr(H) = \{Tr(W_1), \ldots, Tr(W_n)\}$, $Tr(W_i) = \{(\delta_k, R_j)_{W_i \subset \delta_k}, 1 \le k \le |\mathcal{D}|\}$, where R_j is the ranking information of weighted zone score of the query keyword set W_i in the top-k file set δ_k.

In the known ciphertext model, given two histories $\{H, H'\}$ with same trace, it generates $V(H)$ and $V'(H')$, respectively. If $V(H)$ and $V'(H')$ are not distinguishable, the cloud server (or the attacker) cannot obtain any additional information about the index or the document set except for the access patterns. Now we prove Theorem 1.

Proof: Denote S as a simulator. Given a history H, the S can simulate a $V'(H')$ of *View* such that the cloud platform cannot distinguish *V(H)* and $V'(H')$. This purpose can be achieved by S through the following operations:

- (Simulating $\mathcal{C}'$) S randomly selects $D_i' \in \{0, 1\}^{|D_i|}$, $D_i \in \mathcal{D}$, $1 \le i \le |\mathcal{D}|$ and outputs $\mathcal{D}' = \{D_i', 1 \le i \le |\mathcal{D}|\}$
- (Simulating $\mathcal{EI}'$) To simulate the $\mathcal{EI}'$, S initializes a set of arrays $\{A_1', \cdots, A_\iota'\}$, where $\iota = C_\tau^1 + C_\tau^2 \cdots + C_\tau^n$, n is the number of query keywords, τ is the total number of the keywords extracted from files. Choose a random string ϑ of length d-bit. S computes $EF_i(W)' = (F_i(W) \oplus \vartheta)$ and sets $A_j' = (A_j[1]', \ldots, A_j[k]') = (EF_1(W)', \ldots, EF_k(W)')$. S generates an encrypted look-up table $\mathcal{T}'$ with ι entries. For $j = 1, \ldots, \iota$, S generates a three-tuple $(\tau_j', address(A_j') \oplus \nu', \rho')$ such that τ_j' is a random string of length d-bit, $address(A_j')$ is the address of the array A_j', and ν' is a random string of length d-bit and the ρ' is a random l-bit string.
- (Simulating *token'*): S chooses a l-bit random string σ', and picks τ_i' from $\mathcal{T}'$ at random, where $\mathcal{T}'[\tau_i'] = (address(A_i') \oplus \nu', \rho')$. Then, it returns the trapdoor $token = (\tau_i', \nu', \sigma')$.

Through the above operations, the encrypted index $\mathcal{EI}$ and the encrypted trapdoor *token* generate the same trace as the one that cloud platform has generated. Since the data owner uses symmetric encryption algorithm with the secret key *ek* to encrypt the documents, cloud platform cannot distinguish $\mathcal{D}$ and $\mathcal{C}'$. Moreover, the content in index $\mathcal{EI}'$, $\mathcal{EI}$, trapdoor *token*$'$, *token* are encrypted by pseudo random function. If cloud platform (or the attacker) does not have the secret key *sk* of pseudo random function, plaintext index and query keyword set cannot be recovered. Thus, cloud server cannot get any additional information about the encrypted index or documents set except for the access patterns.

IX. CONCLUSION

In this work, we propose the system model, workflow and concrete construction of blockchain-based searchable encryption scheme with fair payment mechanism. Based on inverted index data structure and smart contract, the construction supports multiple keyword search, top-*k* ranking and search result verification. The smart contract in our proposed scheme realizes a more reliable searchable encryption scheme over encrypted data than existing ones, which not only realizes the verifiability to detect incorrect results from a malicious cloud platform, but also exploits the built-in payment mechanism of blockchain to protect all the interests of data owner, data user and cloud platform. The user management contract, fair payment contract and user interface contract cooperate together to fulfill the functions of BMFP. Through the experimental results obtained on simulations, we analyze the experimental data and demonstrate the practicability of our scheme.

REFERENCES

[1] J. W. Rittinghouse and J. F. Ransome, *Cloud Computing: Implementation, Management, and Security*. Boca Raton, FL, USA: CRC Press, 2016.

[2] Y. Yang and M. Ma, "Conjunctive keyword search with designated tester and timing enabled proxy re-encryption function for E-health clouds," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 4, pp. 746–759, Apr. 2016.

[3] T. Hoang, A. A. Yavuz, and J. G. Merchan, "A secure searchable encryption framework for privacy-critical cloud storage services," *IEEE Trans. Services Comput.*, to be published.

[4] S.-F. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 763–780.

[5] X. Ge, J. Yu, H. Zhang, C. Hu, Z. Li, Z. Qin, and R. Hao, "Towards achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification," *IEEE Trans. Dependable Secure Comput.*, to be published.

[6] Z. Wan and R. H. Deng, "VPSearch: Achieving verifiability for privacy-preserving multi-keyword search over encrypted cloud data," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 6, pp. 1083–1095, Nov./Dec. 2016.

[7] S. Nakamoto. (2008). *Bitcoin: A Peer-To-Peer Electronic Cash System*. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[8] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, Apr. 2014.

[9] C. Cai, J. Weng, X. Yuan, and C. Wang, "Enabling reliable keyword search in encrypted decentralized storage with fairness," *IEEE Trans. Dependable Secure Comput.*, to be published.

[10] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2000, pp. 44–55.

[11] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.

[12] J. Wang, X. Chen, X. Huang, I. You, and Y. Xiang, "Verifiable auditing for outsourced database in cloud computing," *IEEE Trans. Comput.*, vol. 64, no. 11, pp. 3293–3303, Nov. 2015.

[13] Y. Yang, X. Liu, and R. Deng, "Multi-user multi-keyword rank search over encrypted data in arbitrary language," *IEEE Trans. Dependable Secure Comput.*, to be published.

[14] K. He, J. Guo, J. Weng, J. Weng, J. K. Liu, and X. Yi, "Attribute-based hybrid Boolean keyword search over outsourced encrypted data," *IEEE Trans. Dependable Secure Comput.*, to be published.

[15] Y. Yang, X. Liu, R. H. Deng, and Y. Li, "Lightweight sharable and traceable secure mobile health system," *IEEE Trans. Dependable Secure Comput.*, to be published.

[16] X. Liu, G. Yang, Y. Mu, and R. Deng, "Multi-user verifiable searchable symmetric encryption for cloud storage," *IEEE Trans. Dependable Secure Comput.*, to be published.

[17] J. Zhu, Q. Li, C. Wang, X. Yuan, Q. Wang, and K. Ren, "Enabling generic, verifiable, and secure data search in cloud services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 8, pp. 1721–1735, Aug. 2018.

[18] C. Manning, P. Raghavan, and H. Schütze, "Introduction to information retrieval," *Natural Lang. Eng.*, vol. 16, no. 1, pp. 100–103, 2010.

[19] Y. Yang, Y.-C. Zhang, J. Liu, X.-M. Liu, F. Yuan, and S.-P. Zhong, "Chinese multi-keyword fuzzy rank search over encrypted cloud data based on locality-sensitive hashing," *J. Inf. Sci. Eng.*, vol. 35, no. 1, pp. 137–158, Jan. 2019.

[20] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in *Proc. IEEE INFOCOM-IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 792–800.

[21] L. Chen, W.-K. Lee, C.-C. Chang, K.-K. R. Choo, and N. Zhang, "Blockchain based searchable encryption for electronic health record sharing," *Future Gener. Comput. Syst.*, vol. 95, pp. 420–429, Jun. 2019.

[22] S. Wang, Y. Zhang, and Y. Zhang, "A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems," *IEEE Access*, vol. 6, pp. 38437–38450, 2018.

[23] Y. Wu, S. Tang, B. Zhao, and Z. Peng, "BPTM: Blockchain-based privacy-preserving task matching in crowdsourcing," *IEEE Access*, vol. 7, pp. 45605–45617, 2019.

[24] Y. Zhang, R. H. Deng, J. Shu, K. Yang, and D. Zheng, "TKSE: Trustworthy keyword search over encrypted data with two-side verifiability via blockchain," *IEEE Access*, vol. 6, pp. 31077–31087, 2018.

[25] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 443–458.

[26] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, Sep. 1997.

[27] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Upper Saddle River, NJ, USA: Pearson, 2017.

[28] K. Kurosawa and Y. Ohtaki, "How to update documents verifiably in searchable symmetric encryption," in *Proc. Int. Conf. Cryptol. Netw. Secur.* Cham, Switzerland: Springer, Nov. 2013, pp. 309–328.

[29] P. Wang, H. Wang, and J. Pieprzyk, "An efficient scheme of common secure indices for conjunctive keyword-based retrieval on encrypted data," in *Proc. Int. Workshop Inf. Secur. Appl.* Berlin, Germany: Springer, Sep. 2008, pp. 145–159.

[30] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to IOPs and stateless blockchains," IACR Cryptol. ePrint Arch., Lyon, France, Tech. Rep. 2018/1188, 2018.

[31] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *J. Comput. Secur.*, vol. 19, no. 5, pp. 895–934, Jan. 2011.

[32] *The Introduction Solidity*. Accessed: May 6, 2019. [Online]. Available: https://solidity.readthedocs.io/en/latest/

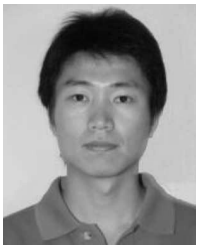
[33] *Verify Publish Contract Source Code*. Accessed: May 6, 2019. [Online]. Available: https://www.etherscan.io/verifycontract2

YANG YANG received the B.Sc. and Ph.D. degrees from Xidian University, Xi'an, China, in 2006 and 2012, respectively. She is also an Associate Professor with the College of Mathematics and Computer Science, Fuzhou University. She has published more than 100 articles in the topics of cloud security and privacy protection, including articles in the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON SERVICES COMPUTING, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, and the IEEE TRANSACTIONS ON CLOUD COMPUTING. Her research interests include the areas of information security and privacy protection. She is also a member of the CCF.

HONGRUI LIN received the B.Sc. degree from the College of Photonic and Electronic Engineering, Fujian Normal University, Fuzhou, China, in 2017. He is currently pursuing the master's degree with the College of Mathematics and Computer Science, Fuzhou University, Fuzhou. His research interests include the areas of privacy protection and blockchain.

XIMENG LIU (S'13–M'16) received the B.Sc. degree in electronic engineering and the Ph.D. degree in cryptography from Xidian University, Xi'an, China, in 2010 and 2015, respectively. He is currently a Full Professor with the College of Mathematics and Computer Science, Fuzhou University. He is also a Research Fellow with the School of Information System, Singapore Management University, Singapore. He has published more than 100 articles on the topics of cloud security and big data security, including articles in the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON SERVICES COMPUTING, and the IEEE INTERNET OF THINGS JOURNAL. His research interests include cloud security, applied cryptography, and big data security. He is also a member of the ACM and CCF. He was awarded with the Minjiang Scholars Distinguished Professor, the Qishan Scholars in Fuzhou University, and the ACM SIGSAC China Rising Star Award, in 2018.

WENZHONG GUO received the B.S. and M.S. degrees in computer science and the Ph.D. degree in communication and information system from Fuzhou University, Fuzhou, China, in 2000, 2003, and 2010, respectively, where he is currently a Full Professor with the College of Mathematics and Computer Science. His research interests include intelligent information processing, sensor networks, network computing, and network performance evaluation.



XIANGHAN ZHENG received the M.Sc. degree in distributed system and the Ph.D. degree in information communication technology from the University of Agder, Norway, in 2007 and 2011, respectively. He is currently a Professor with the College of Mathematics and Computer Sciences, Fuzhou University, China. His current research interests include new generation network with a special focus on cloud computing services and applications, and big data processing and security.

ZHIQUAN LIU received the B.S. degree from the School of Science, Xidian University, in 2012, and the Ph.D. degree from the School of Computer Science and Technology, Xidian University, in 2017. He is currently a Lecturer with the College of Information Science and Technology, Jinan University. He is also a member of the Guangdong Provincial Key Laboratory of Data Security and Privacy Protection, Guangzhou, China. His current research interests include trust management, service recommendation, and the Internet-of-Things security.