Received August 8, 2020, accepted August 28, 2020, date of publication September 1, 2020, date of current version September 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3020841

Multi-Keyword Certificateless Searchable Public Key Authenticated Encryption Scheme Based on Blockchain

XIAODONG YANG[1], (Member, IEEE), GUILAN CHEN[1], MEIDING WANG[1], TING LI[1], AND CAIFEN WANG[1,2]

[1]College of Computer Science and Engineering, Northwest Normal University, Lanzhou 730070, China
[2]College of Big Data and Internet, Shenzhen Technology University, Shenzhen 518118, China

Corresponding author: Xiaodong Yang (y200888@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61662069 and Grant 61562077, in part by the China Postdoctoral Science Foundation under Grant 2017M610817, in part by the Science and Technology Project of Lanzhou City of China under Grant 2013-4-22, and in part by the Foundation for Excellent Young Teachers by Northwest Normal University under Grant NWNU-LKQN-14-7.

ABSTRACT As a quite attractive secure search mechanism in cloud environments, searchable encryption allows encrypted files to be searched by keyword and does not reveal any information about original data files. However, most existing searchable encryption schemes only support single keyword ciphertext retrieval, and they cannot resist against inside keyword guessing attacks. Besides, the previous schemes rarely focus on integrity verification and fair transactions without any third party. Focusing on these problems, we propose a multi-keyword certificateless searchable public key authenticated encryption scheme based on blockchain. We use certificateless cryptosystem to encrypt keywords, which avoids the problems of certificate management in traditional cryptosystem and key escrow in identity-based cryptosystem. Our scheme also supports multi-keyword search, which locates encrypted files precisely and returns the desired files. Moreover, we upload the real encrypted files to the cloud server, while the encrypted indexes are put in blockchain, which ensures the anti-tampering, integrity and traceability of the encrypted indexes. The anti-tampering of blockchain also ensures that users can receive accurate search results without any third party verification. Furthermore, we utilize smart contract to track monetary rewards, which enables fair transactions between data owners and users without any trusted third party. We prove that the proposed scheme is secure against inside keyword guessing attacks in the random oracle model. Finally, our performance evaluation shows that the proposed scheme has higher computational performance than other related schemes.

INDEX TERMS Authenticated encryption, blockchain, certificateless cryptosystem, multi-keyword, searchable encryption.

I. INTRODUCTION

As big data and cloud computing develop rapidly in the recent years, an increasing number of individuals and enterprises upload data files to cloud servers due to insufficient memory capacity of application devices [1]. However, cloud servers are not completely trusted for the reason that they are not under the full supervision of enterprises and users. Once the data owners upload sensitive data to the cloud server, they cannot fully control these data. This may make malicious users, attackers, and cloud server providers (CSP) access or steal data owners' sensitive data. Therefore, the issue of data privacy preserving in cloud storage environments has become increasingly important [2], [3].

In order to protect data confidentiality, data owners often encrypt data files before they are outsourced. However, this pattern faces the problem of how to implement keyword search over encrypted files. The simplest method is to download the encrypted files, decrypt them, and then perform keyword search. But this operation is not applicable for the

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

reason that it downloads unnecessary data files and wastes a lot of network overheads [4]. Searchable encryption technology can solve this problem, which can perform keyword search in encrypted files without downloading needless data files. In addition, it does not divulge any information about original data files [5]–[7].

Searchable encryption mainly includes symmetric searchable encryption (SSE) and public key encryption with keyword search (PEKS). SSE technology is mainly used to solve the storage problem of untrusted servers, and PEKS technology is mainly used to solve the routing problem of untrusted servers [8]. In the SSE cryptosystem, data owner encrypts data files and stores ciphertext remotely on a semi-trusted cloud server, while retaining the ability to search keywords in encrypted files. Data owner can perform keyword search efficiently by utilizing the powerful computing power of the cloud server, and does not reveal any data privacy to the server. This not only protects data confidentiality, but also has high search efficiency [9]. However, the SSE schemes face risks in terms of key distribution and management for the reason that both data files encryption and trapdoor generation must be encrypted by using the same symmetric key [10].

The PEKS technology uses user's public key to encrypt data files, and the data owner does not need to perform key agreement with users. Therefore, the PEKS is quiet suitable for data sharing in real scenarios [11], [12]. Most previous PEKS schemes are constructed based on traditional cryptosystem and identity-based cryptosystem, hence they have problems with certificate management or key escrow [13]–[15]. Certificateless searchable public key encryption (CL-SPKE) is an attractive cryptosystem because it solves these problems [16]. However, the great majority of CL-SPKE schemes only support single keyword retrieval in encrypted files [17], [18]. In fact, in order to search accurately and save computing resources, users generally execute multiple keywords retrieval. In addition, most existing CL-SPKE schemes also require that cloud server, which is presumed to be "honest-but-curious", can honestly perform search operations based on user needs [19], [20]. In real situations, a malevolent cloud server may return partial or fake search results. Aiming at this problem, many scholars have proposed a series of verifiable searchable encryption schemes, whereas most verifiable schemes only detect malicious behaviors of cloud servers or users [21]–[23]. They do not have effective measures to achieve fair payments, because the malicious party has got what he or she wants. For instance, a user wants to execute keyword search in encrypted files, and he is asked to pay before the cloud server starts to search. After receiving the search results, he finds that the search results are incorrect, but the cloud server gets deposit. Obviously, the transaction is not fair.

The blockchain-based solution is a feasible method that enables users to share data files in a secure and fair manner. As a distributed database, blockchain provides a new way to record and convey value, making transactions more transparent, fair, and secure [24], [25]. Specifically, blockchain technology can ensure data integrity and achieve fair sharing of data files. Besides, the anti-tampering property of blockchain guarantees the security and authenticity of data. It also ensures that users can receive integral and correct search results without any verification. In addition, blockchain can trace information such as data authenticity and abnormal server behavior [26]–[28]. Therefore, we propose a novel certificateless searchable public key authenticated encryption (CL-SPKAE) scheme based on blockchain. In the proposed scheme, data owners store encrypted indexes in the blockchain, while the original encrypted files are stored in any public cloud server. This arrangement not only ensures that encrypted indexes are not known by cloud servers, but also detects illegal tampering by malicious cloud servers.

In this paper, we present a multi-keyword CL-SPKAE scheme based on blockchain, which can resolve the contradiction between data privacy preserving and user security sharing. The main contribution of our scheme is summarized as follows.

- We present a novel CL-SPKAE scheme. We use certificateless cryptosystem to encrypt keywords, which avoids the overhead of certificate management in traditional cryptosystem and solves the problem of key escrow in identity-based cryptosystem. In addition, our scheme encrypts keywords to generate indexes using the user's public key and the data owner's private key, which realizes keyword authentication. Therefore, the third party cannot encrypt the keyword to conduct inside keyword guessing attacks without the data owner's private key.
- Our scheme guarantees the anti-tampering, integrity and traceability of the encrypted indexes. We put the encrypted indexes into the blockchain and upload the real encrypted files to cloud servers. Apart from keeping cloud server away from the encrypted indexes, this arrangement can also monitor illegal operations by malicious cloud servers. In addition, the anti-tampering of blockchain ensures that users can receive correct search results without any third party verification.
- Our scheme achieves fair transactions. We utilize smart contract to enable fair transactions between data owners and users without the participation of any trusted third party. It ensures that the honest party always gets what he or she deserves, while the malicious party always gets nothing.
- Our scheme implements multi-keyword search over encrypted files. Compared with the single keyword model, our multi-keyword scheme can search accurately and save a lot of computing resources, which is more practical in the actual cloud environment.
- We prove that the proposed scheme is secure against keyword guessing attack in a random oracle model under type I attacker and type II attacker. Moreover, we compare feature, computational cost and communication cost with other similar schemes. The analysis results

indicate that our scheme has lower computational cost in the keyword encryption and search phase.

The rest of this paper is organized as follows. We describe the relevant work in section II and briefly present some preliminaries in section III. We give the definition and security model of the CL-SPKAE scheme in section IV. Then, we give the system model and detailed construction of our scheme in section V. We prove the security of the proposed scheme in section VI and give its performance analysis in Section VII. Finally, we draw the concluding remarks in section VIII.

II. RELATED WORK

After Song *et al.* [5] proposed the idea of searchable encryption, many scholars successively put forward a large number of provable secure SSE schemes with special properties [29]–[31]. SSE had high search efficiency, and it was simple to realize. Nevertheless, it faced risks in terms of key distribution and management. In order to address above issues, Boneh *et al.* [32] introduced searchable encryption in the asymmetric cryptosystem and constructed the first PEKS scheme. In this scheme, the sender encrypted the mail and keywords with the receiver's public key, and the receiver generated search trapdoor with its own private key. Baek *et al.* [33] proposed a new PEKS scheme, but this scheme required data owners to access all authorized users' public key when they encrypted data files. Then, Guo *et al.* [34] proposed a searchable encryption scheme based on certificate, but it only satisfied chosen plaintext security and faced the problem of complex certificate management. Zhu *et al.* [35] designed a searchable encryption scheme based on identity, which addressed the issue of certificate management. This scheme generated users' private keys with a trusted key generation center (KGC), so it existed a key escrow problem. To address the problems of certificate management and key escrow, Wu *et al.* [36] constructed a searchable encryption scheme based on certificateless cryptosystem. Later, Lu *et al.* [37] constructed a novel CL-SPKE. The scheme could resist guessing keyword attacks in the random oracle model, but it only supported single keyword ciphertext retrieval. Although single keyword retrieval schemes retrieved keyword quickly, they could not locate data files accurately, which existed certain limitations in practical applications.

Subsequently, Uwizeye *et al.* [38] designed a CL-SPKE scheme supporting conjunctive keyword search, but this scheme cannot resist against inside keyword guessing attacks. Wu *et al.* [39] constructed a novel CL-SPKAE scheme to satisfy the privacy of keyword. These schemes mentioned above required the cloud server, which was "honest-but-curious", could reliably perform search operations according to user needs. In fact, a malevolent cloud server may return partial or counterfeit search results. To solve this problem, many scholars had constructed an array of verifiable searchable encryption schemes. However, most verifiable searchable encryption schemes only detected malicious behavior and could not achieve fair payments.

To address above issue, Chen *et al.* [40] proposed a searchable encryption scheme based on blockchain and achieved fair payments by using smart contract. The scheme constructed encrypted indexes by complex logical expressions and users could search encrypted indexes by expressions, which solved the problem of only supporting single keyword retrieval in the scheme [41]. But these schemes was only applicable to symmetric cryptographic environment. As far as we know, designing an efficient blockchain-based PEKS scheme remains a challenging problem [42]. Also, there is no public multi-keyword CL-SPKAE scheme based on blockchain until now. Therefore, we are committed to constructing a new multi-keyword CL-SPKAE scheme based on blockchain in this paper.

III. PRELIMINARIES

In this section, we briefly introduce several preliminary knowledge, including smart contract, gas system, bilinear pairing and complexity assumption.

A. SMART CONTRACT

Smart contract, proposed by Nick Szabo [43], is a commitment defined in digital form, which controls digital assets and contains the rights and obligations agreed by the contract participants. Specifically speaking, a smart contract is an agreement that is executed automatically by a computer and does not require human participation. It always performs operations in accordance with rules agreed in advance. Due to the lack of a trusted execution environment, smart contracts were not applied to the actual industry when they were first proposed. Later, the application of smart contracts has been realized because of the birth of Bitcoin. That is because people realize that blockchain as the basic technology of Bitcoin can provide a trusted execution environment for smart contracts. In blockchain-based decentralized environment, the smart contract is written into the Ethereum in a digital form [44]. The characteristics of blockchain can ensure the transparency, traceability, anti-tampering and non-repudiation in the storage, reading, and execution phase of smart contracts.

In Ethereum, smart contract is a special account that consists of an account address, script code, balance, and storage space [45]. In other words, Ethereum-based smart contracts are a collection of code and data located at specific addresses in Ethereum. Participants can create smart contracts and write them into the blockchain. When certain conditions in the contract are triggered, the code defined in the contract will execute autonomously. Moreover, the execution results are also published in the blockchain. These execution results can be traced back, but they can not be changed.

There are miners in Ethereum that verify and approve all transactions in the blockchain. They add new transactions to blockchain by solving cryptographic challenges. This process is called mining new blocks. Once the new block is successfully mined, the newly created cryptocurrency is rewarded to workers. Therefore, third-party entities are encouraged to mine more blocks. In addition, the data stored in Ethereum are consistent and transparent among miners, which cannot be modified or rejected.

B. GAS SYSTEM

The gas system in Ethereum is introduced to prevent incorrect or malicious programs(such as endless loop programs). It can resist denial-of-service attacks and implement smart contracts. Each transaction has limited gas consumption in Ethereum. When the limited gas runs up, the transaction will be terminated. Moreover, users can obtain gas through currency exchange with Ethereum and workers' incomes are gas consumptions, as described by Hu *et al*. [39].

Each calculation that miners process transactions incurs a fee, which is paid for the gas consumed by the instruction. Different instruction has different gas consumption. For instance, in this paper, $Gaslsrch$ represents the gas limitation that the user has spent, and $\$gasprice$ represents the price of gas per unit. For each transaction, multiplying $Gaslsrch$ by $\$gasprice$ represents the maximum fee that the user is willing to pay to execute transactions. If the user's account has sufficient balance to pay the maximum fee, the transaction is successfully packaged and submitted to blockchain. Otherwise, the transaction is considered invalid.

C. BILINEAR PAIRING

Assuming that G_1 and G_2 are two multiplicative cyclic groups of large prime order p. A bilinear pairing map $e : G_1 \times G_1 \rightarrow G_2$ satisfies the following properties:

1) Bilinearity: $e({g_1}^a, {g_2}^b) = e(g_1, g_2)^{ab}$ hods for any $g_1, g_2 \in G_1$ and $a, b \in {Z_p}^*$.
2) Non-degeneracy: $e(g, g) \neq 1$, where g is a generator of G_1.
3) Computability: $e(g_1, g_2)$ is efficiently computable for any $g_1, g_2 \in G_1$.

D. COMPLEXITY ASSUMPTION

Given a tuple $(g, g^\alpha, g^\beta, R)$, where $\alpha, \beta \in {Z_p}^*$ and $g, R \in G_1$, the decisional Diffie-Hellman (DDH) problem is to distinguish whether R is $g^{\alpha\beta}$ or a random element in G_1.

The advantage ξ' that $\mathcal{A}$ can solve the mDDH problem is defined as $Adv_{\mathcal{A}}^{DDH} = \left|\Pr[\mathcal{A}(g, g^\alpha, g^\beta, g^{\alpha\beta}) = 1] - \Pr[\mathcal{A}(g, g^\alpha, g^\beta, R) = 1]\right| \geq \xi'$.

Definition 1 (DDH Assumption): We say the DDH assumption holds if no attacker $\mathcal{A}$ can solve the DDH problem with a non-negligible advantage.

IV. DEFINITION AND SECURITY MODEL

A. DEFINITION

Formally, the CL-SPKAE scheme is composed of six polynomial time algorithms, as follows [39]:

- **Setup**(1^λ). The algorithm takes as input a security parameter λ, and it outputs system parameters $param$ and master key msk.
- **PatialKeyGene**$(param, msk, id)$. The algorithm takes as input the identity id of an entity, $param$ and msk, and it outputs the corresponding partial private key psk.
- **KeyGene**$(param, psk)$. The algorithm takes as input $param$ and psk, and it outputs the corresponding public key pk and final private key sk.
- **EncInd**$(param, id_u, pk_u, id_o, sk_o, W, F)$. The algorithm takes as input $param$, the user's identity id_u, the user's public key pk_u, the data owner's identity id_o, the data owner's private key sk_o, the keyword set W and data file F, and it outputs encrypted indexes C_F and encrypted files CT.
- **Trapdoor**$(param, sk_u, pk_o, W')$. The algorithm takes as input $param$, the user's private key sk_u, the data owner's public key pk_o and search keywords W', and it outputs the trapdoor $T_{W'}$.
- **Search**$(param, C_F, T_{W'})$. Upon receiving $param$, C_F and $T_{W'}$, the algorithm outputs the search results.

B. SECURITY MODEL

In accordance with the security model of the CL-SPKAE presented in schemes [12], [16], [37], and [46], our scheme considers two types of adversaries: type I attacker $\mathcal{A}_1$ and type II attacker $\mathcal{A}_2$. A attacker $\mathcal{A}_1$ is permitted to replace any user's public key, while it is not capable of possessing the master key. On the contrary, A attacker $\mathcal{A}_2$ is capable of possessing the master key, while it is not permitted to replace any user's public key.

Here, the security model of a CL-SPKAE scheme is defined by the following games between a PPT attacker $\mathcal{A}_i (i = 1, 2)$ and a challenger $\mathcal{C}$.

*Game*1: This game is executed between a PPT attacker $\mathcal{A}_1$ and a challenger $\mathcal{C}$.

$\mathcal{C}$ runs **Setup** algorithm to generate the master key and system parameters. Then, $\mathcal{C}$ sends system parameters to $\mathcal{A}_1$. $\mathcal{A}_1$ can adaptively ask queries to the following oracles.

Hash-Query. $\mathcal{A}_1$ can access all hash oracles, and the corresponding hash value can be obtained.

PatialKey Query. Given an identity ID_i, $\mathcal{C}$ runs **PatialKeyGene** algorithm to generate the corresponding partial key Q_{ID_i}, and sends it to $\mathcal{A}_1$.

SecretValue Query. Upon receiving query initiated by $\mathcal{A}_1$, $\mathcal{C}$ runs **KeyGene** algorithm to generate the corresponding secret value to $\mathcal{A}_1$.

PublicKey Query. Given an identity ID_i, $\mathcal{C}$ runs **KeyGene** algorithm to generate the corresponding public key PK_{ID_i}, and sends it to $\mathcal{A}_1$.

ReplacePublicKey Query. $\mathcal{A}_1$ is permitted to replace any user's public key.

Trapdoor Query 1. $\mathcal{A}_1$ selects arbitrary keyword and initiates some trapdoor queries to $\mathcal{C}$. Then $\mathcal{C}$ runs trapdoor generation algorithm **Trapdoor** to calculate the corresponding trapdoors and returns them to $\mathcal{A}_1$.

Challenge. Upon randomly selecting two challenge keywords W_0 and W_1, $\mathcal{A}_1$ sends them to $\mathcal{C}$. Note that W_0 and W_1 have not been queried by $\mathcal{A}_1$ during trapdoor query 1 phase.

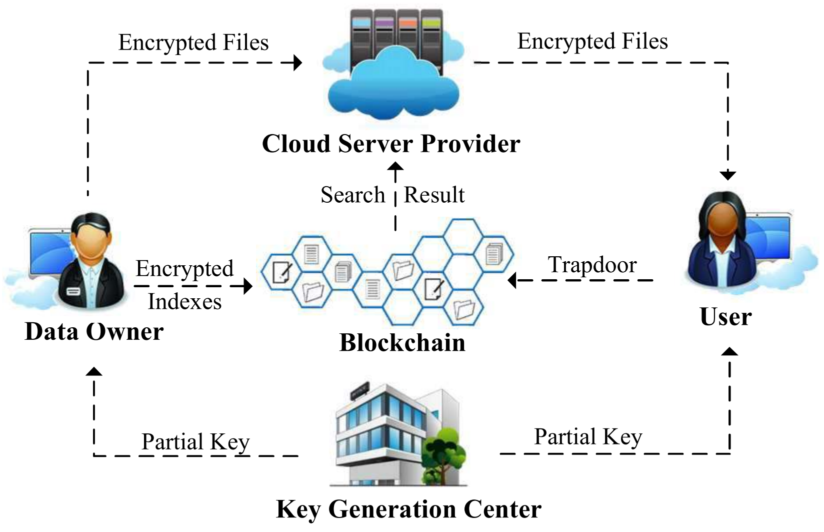

FIGURE 1. System model of the proposed CL-SPKAE scheme.

Then, $\mathcal{C}$ randomly selects $b \in \{0,1\}$ and calls the keyword encryption algorithm **EncInd** to generate corresponding challenge ciphertext of w_b. Finally, $\mathcal{C}$ send the challenge ciphertext to $\mathcal{A}_1$.

Trapdoor Query 2. $\mathcal{A}_1$ continues to request the trapdoor query. The query process is similar to trapdoor query 1, but the keyword cannot be W_0 or W_1.

Guess. $\mathcal{A}_1$ outputs b'. If b'=b, $\mathcal{A}_1$ wins the game.

We define the advantage of $\mathcal{A}_1$ in this game as $\varepsilon = \left|\Pr(b' = b) - \frac{1}{2}\right|$.

*Game*2: This game is executed between a PPT attacker $\mathcal{A}_2$ and a challenger $\mathcal{C}$.

Setup. $\mathcal{C}$ runs **Setup** algorithm to generate the master key and system parameters. Then, $\mathcal{C}$ runs **KeyGene** algorithm to generate public key pk_o and pk_u. Finally, $\mathcal{C}$ sends the master key, system parameters, pk_o and pk_u to $\mathcal{A}_2$.

$\mathcal{A}_2$ is permitted to ask **Hash-Query**,**PatialKey Query**, and **Trapdoor Query 1** as in *Game*1.

Challenge. $\mathcal{A}_2$ randomly selects two challenge keywords W_0 and W_1 and sends them to $\mathcal{C}$, where W_0 and W_1 have not been queried by $\mathcal{A}_2$ during trapdoor query 1 phase. Then $\mathcal{C}$ randomly selects $b \in \{0,1\}$ and lets w_b be a challenge keyword. Next, $\mathcal{C}$ calls the keyword encryption algorithm **EncInd** to generate corresponding ciphertext of w_b. Finally, $\mathcal{C}$ sends the challenge ciphertext to $\mathcal{A}_2$.

Trapdoor Query 2. $\mathcal{A}_2$ continues to request the trapdoor query. The query process is similar to trapdoor query 2 in *Game*1.

Guess. $\mathcal{A}_2$ outputs b'. If b'=b, $\mathcal{A}_2$ wins the game.

We define the advantage of $\mathcal{A}_2$ in this game as $\xi = \left|\Pr(b' = b) - \frac{1}{2}\right|$.

Definition 2: The proposed scheme is semantically secure against inside keyword guessing attacks if ε and ξ are negligible for any PPT attacker.

V. SYSTEM MODEL AND OUR CONSTRUCTION

A. SYSTEM MODEL

There are four entities in our CL-SPKAE scheme system model, namely: KGC, data owner, user, and CSP, as shown in Fig. 1.

- **KGC.** The KGC is responsible for the generation of the system master secret key and public parameters. Moreover, the KGC takes charges of producing a partial private key for every data owner and user, then it sends an individual partial private key to every entity through a secure channel.
- **Data Owner.** The data owner firstly extracts keyword sets from data files and builds encrypted indexes, while establishing smart contracts to describe how users search indexes. Then, the data owner puts both encrypted indexes and smart contracts in the blockchain. Finally, the data owner encrypts data files with a symmetric encryption algorithm(such as Advanced Encryption Standard) and sends encrypted files to CSP.
- **User.** The user can generate the trapdoor of multiple keywords and send it to blockchain. In addition, the user can also decrypt the encrypted files returned by CSP.
- **CSP.** The CSP is in charge of storing encrypted files and sending all matching encrypted files to users based on the search results returned by the blockchain.

B. OUR CONSTRUCTION

In this subsection, we present a novel multi-keyword CL-SPKAE scheme based on blockchain. Our CL-SPKAE scheme is described as follows.

1) SETUP

On input of the security parameter λ, the KGC performs the following steps to generate the master key and system parameters.

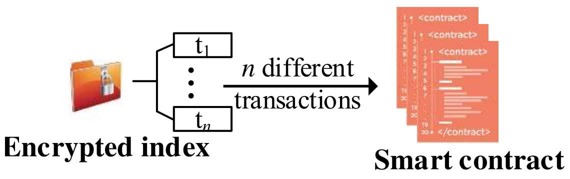

FIGURE 2. Smart contract of the proposed CL-SPKAE scheme.

- Choose two multiplicative cyclic groups G_1 and G_2 of prime order p, a generator g of group G_1, and a bilinear map $e : G_1 \times G_1 \rightarrow G_2$.
- Choose $s \in Z_p{}^*$ randomly and calculate $P{=}g^s$.
- Choose two anti-collision hash functions $H : \{0, 1\}^* \rightarrow G_1$ and $H_1 : \{0, 1\}^* \rightarrow Z_p{}^*$.
- Keep the master key $msk = s$ secretly and publish system parameters $param = (G_1, G_2, p, e, g, P, H, H_1)$.

2) PatialKeyGene

After receiving the data owner's identity id_o and user's identity id_u, KGC computes the data owner's partial private key $psk_o = H(id_o)^s$ and the user's partial private key $psk_u = H(id_u)^s$, respectively, where $Q_{ID_o} = H(id_o)$ and $Q_{ID_u} = H(id_u)$. Moreover, KGC sends psk_o and psk_u to data owner and user through a secure channel, respectively.

3) KeyGene

Upon receiving psk_o, the data owner selects $x_o \in Z_p{}^*$ randomly. Then, data owner computes his or her public key $pk_o = g^{x_o}$ and sets the corresponding private key $sk_o = (x_o, psk_o) = (x_o, H(id_o)^s)$. Similarly, the user selects $x_u \in Z_p{}^*$ randomly, computes his or her public key $pk_u = g^{x_u}$, and sets the corresponding private key $sk_u = (x_u, psk_u) = (x_u, H(id_u)^s)$.

4) EncInd

The data owner generates encrypted indexes by performing the following steps.

- Extract a set of keyword $W = \{w_i \in \psi\, |(1 \leq i \leq m)\}$, where ψ represents a collection of all keywords and m represents the number of keywords in the collection.
- Select $r \in Z_p{}^*$ randomly.
- Compute $C_1 = g^r$, $C_2 = (H(id_o)^s)^{1/x_o}$ and $C_3 = H(id_o)/H(id_u)^r$.
- Compute $E_i = (g^{H_1(w_i)} \cdot pk_u)^{r/x_o}$ if the data file contains keyword w_i, otherwise set $E_i = 1$.
- Set the encrypted indexes $C_F = (C_1, C_2, C_3, E_1, \ldots, E_m)$.

After the keyword indexes are generated, the data owner firstly predefines the reward $\$offer$ for each search and creates smart contracts, as shown in Fig. 2. Then, the data owner uploads encrypted indexes C_F and smart contracts to blockchain. Moreover, the data owner encrypts the data file F to obtain the corresponding data ciphertext CT using a symmetric encryption algorithm (such as Advanced Encryption Standard algorithm). Finally, the data owner sends CT to the CSP.

5) TRAPDOOR

The user first produces the trapdoor by performing the following steps.

- Choose a set of search keywords $W' = \{w_i \in \psi\, |(1 \leq i \leq l)\}$, where l represents the number of search keywords.
- Select $r' \in Z_p{}^*$ randomly.
- Compute $T_1 = (\prod_{\tau=1}^{l} g^{H_1(w_\tau)r'} g^{r'x_u}) \cdot H(id_u)^{sr'}$, $T_2 = P^{r'}$, and $T_3 = pk_o{}^{r'}$.
- Set the trapdoor $T_{W'} = (T_1, T_2, T_3)$.

Then, the user sets a time limitation $Time'$. Finally, the user sends $T_{W'}$ and $Time'$ to smart contracts and ask blockchain to search the encrypted indexes.

6) SEARCH

After receiving the user's search request, the smart contract searches blockchain by performing the following process.

- Check $Time < Time'$. If it times out, the search is terminated. Otherwise, the smart contract performs the following steps.
- Check $\$userdeposit > Gaslsrch \times \$gasprice + \$offer$. If the inequality is established, it means that the deposit currency $\$userdeposit$ pre-stored by the user is enough to complete a search, and smart contract performs the following steps. Otherwise, the smart contract terminates the search.
- Compute $\sigma_1 = e(C_2 \cdot \prod_{\tau=1}^{l} E_{\tau\rightarrow i}, T_3)$, $\sigma_2 = e(T_1, C_1)$ and $\sigma_3 = e(C_3, T_2)$, Where $\tau \rightarrow i$ represents the mapping relationship between the subscript of the search keywords W' and the encrypted keywords W.
- Check $\sigma_1 = \sigma_2 \cdot \sigma_3$. If the equation holds, it means that the keywords match successfully. Then, search results are returned to CSP. Otherwise, the failure messages are sent to CSP.

Note. The detailed processes that smart contracts search blockchain using trapdoor to obtain search results is shown in **Algorithm** V-B6. Here, $\$ownerB$ represents the deposit account of data owner. $\$userB$ and $\$userdeposit$ represent the user's deposit account and deposit currency, respectively. $\$offer$ represents each search price that the data owner deserves. $\$gasprice$ represents the price of gas per unit. $\$cost$ represents the total cost of each search that the user should pay. $Gaslsrch$ and $Gassrch$ represent gas limitation and cost for calling the **Search** algorithm, respectively. Smart contracts use trapdoor and previously stored encrypted indexes to perform **Search** algorithm. Then smart contracts save search results to their state. In addition, within the predefined time limitation $Time'$, the data owner receive rewards accordingly. Otherwise, the user's search request is rejected and the user's deposit is refunded.

Correctness. The correctness of matching between trapdoor and encrypted indexes is presented as follows:

$$\begin{aligned}\sigma_1 &= e((H(id_o)^{s/x_o} \prod_{\tau=1}^{l} ((g^{H_1(w_{\tau\rightarrow i})} g^{x_u})^{r/x_o}), g^{x_o r'}) \\ &= e((H(id_o)^s \prod_{\tau=1}^{l} (g^{H_1(w_{\tau\rightarrow i})} g^{x_u})^r, g^{r'})\end{aligned}$$

Algorithm 1 Search Algorithm

```
1: If current time $Time < Time'$ and $\$userdeposit > Gaslsrch$
     $\times \$gasprice + \$offer$
2:   Compute $\sigma_1 = e(C_2 \prod_{\tau=1}^{l} E_{\tau\to i}, T_3)$, $\sigma_2 = e(T_1, C_1)$,
     and $\sigma_3 = e(C_3, T_2)$.
3:   If $\sigma_1 = \sigma_2 \cdot \sigma_3$
4:     Return the search result.
5:   Else
6:     Return the failure message.
7:   End if
8:   Set $\$cost = \$offer + Gassrch \times \$gasprice$.
9:   Send $\$offer$ to $\$ownerB$ and $gassrch \times \$gasprice$ to the worker
     who executes the transaction.
10:  Set $\$userdeposit = \$userdeposit - \$cost$ and go to 1.
11: else
12:  Send $\$userdeposit$ to $\$userB$.
13: end if
```

$$= e((H(id_o), g)^{sr'} e(\prod_{\tau=1}^{l} (g^{H_1(w_{\tau\to i})} g^{x_u}), g)^{rr'},$$

$$\sigma_2 = e(\prod_{\tau=1}^{l} g^{H_1(w_\tau)r'} g^{r'x_u} \cdot H(id_u)^{sr'}, g^r)$$

$$= e(\prod_{\tau=1}^{l} (g^{H_1(w_\tau)} g^{x_u}) \cdot H(id_u)^s, g^{rr'})$$

$$= e(\prod_{\tau=1}^{l} (g^{H_1(w_\tau)} g^{x_u}), g)^{rr'} e(H(id_u), g)^{srr'},$$

$$\sigma_3 = e(H(id_o)/H(id_u)^r, P^{r'})$$

$$= e(H(id_o), g)^{r's}/e(H(id_u), g)^{rsr'}.$$

Since $\sigma_1 = \sigma_2 \cdot \sigma_3$ holds, smart contracts can successfully match the data files requested by the user.

VI. SECURITY ANALYSIS

A. FAIRNESS

Fairness is achieved by using the smart contract. In Ethereum, all transactions are paid through the purchase of gas. As long as the user performs keyword search, the data owner certainly gets the reward he or she deserves. At the same time, as long as the user pays required payment, he or she definitely gets correct search results. Moreover, the characteristics of the blockchain ensure that users can obtain accurate and complete search results without the third party verification. In other words, malicious operations is detected and dishonest parties gets nothing. In addition, the user specifies the time limit *Time′* to ensure fairness as the transactions need completing in this time. Otherwise, the user's deposit is returned.

B. SOUNDNESS

The transaction between the blockchain and the user is transparent, which can ensure that the results of each search are reliable. As long as the smart contract runs correctly on the blockchain, the search results must be correct without any third-party verification. In addition, each node of the blockchain can detect the tampering of search results by any entity. Therefore, there is no such thing as the search results are maliciously tampered with.

C. CONFIDENTIALITY

Theorem 1: Under the mDDH assumption, the proposed CL-SPKAE scheme is semantically secure against inside keyword guessing attacks in the random oracle model.

Theorem 1 can be proven by the following two lemmas.

Lemma 1: Assuming that a PPT attacker $\mathcal{A}_1$ can initiate up to q_T and q_{CUR} times trapdoor query 1 and createuser query respectively, where q_T and q_{CUR} is a positive integer. If $\mathcal{A}_1$ can win our scheme with a non-negligible advantage ε, then we can construct the algorithm $\mathcal{C}$ to solve the DDH problem with non-negligible advantage $\xi' = \varepsilon/q_{CUS} \cdot (1 - 1/q_{CUS})^{q_T}$.

Proof: Given $(g, u_1 = g^\alpha, u_2 = g^\beta, R) \in G_1^4$, $\mathcal{C}$ interacts with $\mathcal{A}_1$ by the following game for the purpose of differentiating whether R is $g^{\alpha\beta}$ or a random element in G_1. Then, $\mathcal{C}$ chooses a challenge identity $ID_D (1 < D < q_{CUS})$ and sets $P = g^\alpha$. Finally, $\mathcal{C}$ sends $param = (G_1, G_2, e, g, P, H, H_1)$ to $\mathcal{A}_1$.

Hash Query. $\mathcal{C}$ maintains a list $H - list$ of the form (ID_i, Q_{ID_i}, h_i). When $\mathcal{A}_1$ initiates some queries about (ID_i), $\mathcal{C}$ responds via the following steps.

- If there is a corresponding item (ID_i, Q_{ID_i}, h_i) of (ID_i) in $H - list$, $\mathcal{C}$ returns Q_{ID_i}.
- Else if $ID_i = ID_D$, $\mathcal{C}$ sets $Q_{ID_D} = g^\beta$. Moreover, $\mathcal{C}$ stores $(ID_D, Q_{ID_D}, \perp)$ in the table and returns Q_{ID_D} to $\mathcal{A}_1$.
- Otherwise, $\mathcal{C}$ randomly selects $h_i \in Z_p{}^*$ and computes $Q_{ID_D} = (g \cdot g^\beta)^{h_i}$. Moreover, $\mathcal{C}$ stores (ID_i, Q_{ID_i}, h_i) in the table and returns Q_{ID_i} to $\mathcal{A}_1$.

Hash1 Query. $\mathcal{C}$ maintains a list $H_1 - list$ of the form $(w_i, h1_i)$. When $\mathcal{A}_1$ initiates some queries about the keyword $w_i \in \{0, 1\}^*$, $\mathcal{C}$ responds via the following steps.

- If there is a corresponding item $(w_i, h1_i)$ of w_i in $H_1 - list$, $\mathcal{C}$ returns $h1_i$.
- Otherwise, $\mathcal{C}$ randomly selects $h1_i \in Z_p{}^*$ for each w_i that has not been queried. Finally, $\mathcal{C}$ stores $h1_i$ in the table and returns $h1_i$ to $\mathcal{A}_1$.

CreateUser Query. In order to answer these queries, $\mathcal{C}$ maintains a list $L_{user} - list$ of the form $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i}))$. Upon receiving an identity ID_i, $\mathcal{C}$ responds via the following steps.

- If there is a corresponding item $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i})$ in $L_{user} - list$, $\mathcal{C}$ returns PK_{ID_i}.
- Else if $ID_i = ID_D$, $\mathcal{C}$ randomly selects $x_{ID_D} \in Z_p{}^*$ and sets $PK_{ID_D} = g^{x_{ID_D}}$ and $PSK_{ID_D} = \perp$. Moreover, $\mathcal{C}$ stores $(ID_D, PK_{ID_D}, PSK_{ID_D}, x_{ID_D})$ in the table and returns PK_{ID_D} to $\mathcal{A}_1$.
- Otherwise, $\mathcal{C}$ randomly selects $x_{ID_i} \in Z_p{}^*$, simulates **Hash Query** to obtain h_i, and sets $PK_{ID_i} = g^{x_{ID_i}}$ and $PSK_{ID_i} = (g^\alpha)^{h_i}$. Moreover, $\mathcal{C}$ stores $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i})$ in the table and returns PK_{ID_i} to $\mathcal{A}_1$.

PatialKey Query. Upon receiving an identity ID_i, $\mathcal{C}$ terminates this game if $ID_i = ID_D$. (We use EV_1 to represent this event.) Otherwise, $\mathcal{C}$ search for ID_i in $L_{user} - list$ to find $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i}))$ and returns PSK_{ID_D} to $\mathcal{A}_1$.

SecretValue Query. Upon receiving an identity ID_i, $\mathcal{C}$ search in $L_{user} - list$ to find $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i}))$ and returns x_{ID_i} to $\mathcal{A}_1$. Note that $\mathcal{C}$ returns $\perp$ if the corresponding public key is replaced by $\mathcal{A}_1$.

PublicKey Query. Upon receiving an identity ID_i, $\mathcal{C}$ search in $L_{user} - list$ to find $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i}))$ and returns PK_{ID_i} to $\mathcal{A}_1$.

ReplacePublicKey Query. Upon receiving $(ID_i, PK_{ID_i}{}')$, $\mathcal{C}$ calls **CreateUser Query** to obtain $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i}))$. Then, $\mathcal{C}$ updates $L_{user} - list$ by replacing $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i}))$ with $(ID_i, PK_{ID_i}{}', PSK_{ID_i}, \perp))$.

Trapdoor Query 1. $\mathcal{A}_1$ initiates some trapdoor queries to $\mathcal{C}$ after selecting arbitrary keyword w', a data owner's identity ID_i and a user's identity ID_j. Then, $\mathcal{C}$ generates the corresponding trapdoor via the following steps.

- If $ID_j = ID_D$, $\mathcal{C}$ terminates game and returns failure messages. Otherwise, $\mathcal{C}$ performs the following steps. (We use EV_2 to represent this event.)
- Search (ID_i, Q_{ID_i}, h_i) and (ID_j, Q_{ID_j}, h_j) from the list $H - list$.
- Search $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i}))$ and $(ID_j, PK_{ID_j}, PSK_{ID_j}, x_{ID_j}))$ from the list $L_{user} - list$.
- Select $r' \in Z_p{}^*$ and let $r' = 1/h_j$.
- Compute trapdoors $T_1 = \prod_{\tau=1}^{l} g^{h1_i r'} g^{r' x_{ID_j}} \cdot PSK_{ID_j}{}^{r'}$, $T_2 = P^{r'}$, and $T_3 = PK_{ID_i}{}^{r'}$.
- Send $T_{w'} = (T_1, T_2, T_3)$ to $\mathcal{A}_1$.

Challenge. Upon selecting a data owner's identity ID_O, a user's identity ID_U and two challenge keywords W_0 and W_1, $\mathcal{A}_1$ sends them to $\mathcal{C}$. Note that W_0 and W_1 have not been queried by $\mathcal{A}_1$ during the trapdoor query 1 phase. Then $\mathcal{C}$ responds to $\mathcal{A}_1$ via the following steps.

- If $ID_U \neq ID_D$, $\mathcal{C}$ terminates game and returns failure messages. (We use EV_3 to represent this event.)
- Otherwise, $\mathcal{C}$ select $b \in \{0, 1\}$ and $r \in Z_p{}^*$ randomly.
- Compute $C_1 = g^r$, $C_2 = PSK_{ID_O}{}^{1/x_{ID_O}}$, $C_3 = Q_{ID_O}/(Q_{ID_u}{}^{r})$, and $E_i = (g^{h1_i} PK_{ID_U})^{r/x_{ID_O}}$.
- Send challenge ciphertext $C_F = (C_1, C_2, C_3, E_1, \ldots, E_m)$ to $\mathcal{C}$.

Trapdoor Query 2. $\mathcal{A}_1$ continues to request the trapdoor of w_j. The query process is the same as trapdoor query 1, but it is required that w_j satisfies $w_j \neq W_0$ and $w_j \neq W_1$.

Guess. $\mathcal{A}_1$ outputs b'. If b'=b, $\mathcal{A}_1$ wins the game. At this time, $\mathcal{C}$ calculates $R = \frac{T_j}{(\prod_{\tau=1}^{l} g^{h1_i r'} g^{r' x_{ID_j}})PSK_{ID_j}{}^{r'}}$. If $T_j = \prod_{\tau=1}^{l} g^{h1_i r'} g^{r' x_{ID_j}} \cdot (g \cdot g^{\beta})^{h_j \alpha r'}$, then we can smoothly infer that $R = g^{\alpha\beta}$.

Next, we analyze the probability that $\mathcal{C}$ can solve the DDH problem. We first analyze the probability of the game is not terminated in the simulation process. Then we calculate the probability that $\mathcal{C}$ can respond correctly while the game is not terminated. Finally we can estimate the advantage ξ of $\mathcal{A}_1$ to win the game. It can be known that the game has been terminated in the events EV_1, EV_2, $EV3$, respectively.

According to the event EV_3, we can know that the event $\neg EV_1$. Also, we obviously have that $\Pr[\neg EV_3] = 1/q_{CUS}$ and $\Pr[\neg EV_2] = (1 - 1/q_{CUS})^{q_T}$, respectively. Therefore, the probability that $\mathcal{C}$ terminates the game is $\Pr[\neg EV_1 \cap \neg EV_2 \cap \neg EV_3] \geq 1/q_{CUS} \cdot (1 - 1/q_{CUS})^{q_T}$.

In the light of the above analysis, the probability that $\mathcal{C}$ terminates the game is $1/q_{CUS} \cdot (1 - 1/q_{CUS})^{q_T}$. Hence the probability that $\mathcal{C}$ successfully differentiates whether R is $g^{\alpha\beta}$ or a random element in G_1 is at least $\xi' = \varepsilon/q_{CUS} \cdot (1 - 1/q_{CUS})^{q_T}$. According to the DDH assumption, we know that ξ' is negligible, and the advantage ε that $\mathcal{A}$ can break the proposed scheme is negligible.

Lemma 2: Assuming that a PPT attacker $\mathcal{A}_2$ can initiate up to q_T and q_{CUR} times trapdoor query 1 and createuser query respectively, where q_T and q_{CUR} is a positive integer. If $\mathcal{A}_2$ can win our scheme with a non-negligible advantage ξ, then we can construct the algorithm $\mathcal{C}$ to solve the DDH problem with non-negligible advantage ξ'.

Proof: Given $(g, u_1 = g^{\alpha}, u_2 = g^{\beta}, R) \in G_1^4$, $\mathcal{C}$ interacts with $\mathcal{A}_2$ by the following game for the purpose of differentiating whether R is $g^{\alpha\beta}$ or a random element in G_1. Then, $\mathcal{C}$ selects $s \in Z_p{}^*$ randomly, chooses a challenge identity $ID_D(1 < D < q_{CUS})$, and computes $P = g^s$. Finally, $\mathcal{C}$ sends $param = (G_1, G_2, e, g, P, H, H_1)$ and $msk = s$ to $\mathcal{A}_2$.

Hash Query. $\mathcal{C}$ maintains a list $H - list$ of the form (ID_i, Q_{ID_i}, h_i). When $\mathcal{A}_2$ initiates some queries about (ID_i), $\mathcal{C}$ responds via the following steps.

- If there is a corresponding item (ID_i, Q_{ID_i}, h_i) of (ID_i) in $H - list$, $\mathcal{C}$ returns Q_{ID_i}.
- Else if $ID_i = ID_D$, $\mathcal{C}$ computes $Q_{ID_D} = g^{\beta}$. Moreover, $\mathcal{C}$ stores $(ID_D, Q_{ID_D}, \perp)$ in the table and returns Q_{ID_D} to $\mathcal{A}_2$.
- Otherwise, $\mathcal{C}$ randomly selects $h_i \in Z_p{}^*$ and computes $Q_{ID_D} = (g \cdot g^{\alpha})^{h_i}$. Moreover, $\mathcal{C}$ stores (ID_i, Q_{ID_i}, h_i) in the table and returns Q_{ID_i} to $\mathcal{A}_2$.

Hash1 Query. $\mathcal{C}$ maintains a list $H_1 - list$ of the form $(w_i, h1_i)$. When $\mathcal{A}_2$ initiates some queries about the keyword $w_i \in \{0, 1\}^*$, $\mathcal{C}$ responds via the following steps.

- If there is a corresponding item $(w_i, h1_i)$ of w_i in $H_1 - list$, $\mathcal{C}$ returns $h1_i$.
- Otherwise, $\mathcal{C}$ randomly selects $h1_i \in Z_p{}^*$ for each w_i that has not been queried. Finally, $\mathcal{C}$ stores $h1_i$ in the table and returns $h1_i$ to $\mathcal{A}_2$.

CreateUser Query. In order to answer these queries, $\mathcal{C}$ maintains a list $L_{user} - list$ of the form $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i}))$. Upon receiving an identity ID_i, $\mathcal{C}$ responds via the following steps.

- If there is a corresponding item $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i})$ in $L_{user} - list$, $\mathcal{C}$ returns PK_{ID_i}.
- Else if $ID_i = ID_D$, $\mathcal{C}$ sets $PK_{ID_D} = g^{\alpha}$ and $PSK_{ID_D} = g^{\beta s}$. Moreover, $\mathcal{C}$ stores $(ID_D, PK_{ID_D}, PSK_{ID_D}, \perp)$ in the table and returns PK_{ID_D} to $\mathcal{A}_2$.

TABLE 1. A comparison of the searchable encryption scheme feature.

Schemes	Certificateless	Authenticated	Multi-keyword	Blockchain	Fair Payment
[16]	✓	✓	✗	✗	✗
[37]	✓	✓	✗	✗	✗
[38]	✓	✗	✓	✗	✗
[39]	✓	✓	✗	✗	✗
[40]	✗	✗	✓	✓	✓
[46]	✓	✓	✗	✗	✗
[47]	✓	✗	✓	✗	✗
[48]	✗	✗	✓	✗	✗
[49]	✗	✓	✗	✗	✗
[50]	✗	✗	✓	✓	✗
Ours	✓	✓	✓	✓	✓

- Otherwise, $\mathcal{C}$ randomly selects $x_{ID_i} \in Z_p{}^*$, simulates **Hash Query** to obtain h_i, and sets $PK_{ID_i} = g^{x_{ID_i}}$ and $PSK_{ID_i} = (g^s)^{h_i}$. Moreover, $\mathcal{C}$ stores $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i})$ in the table and returns PK_{ID_i} to $\mathcal{A}_2$.

PatialKey Query. Upon receiving an identity ID_i, $\mathcal{C}$ terminates this game if $ID_i = ID_D$. (We use EV_1 to represent this event.) Otherwise, $\mathcal{C}$ search for ID_i in $L_{user} - list$ to find $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i}))$ and returns PSK_{ID_D} to $\mathcal{A}_1$.

Trapdoor Query 1. $\mathcal{A}_1$ initiates some trapdoor queries to $\mathcal{C}$ after selecting arbitrary keyword w', a data owner's identity ID_i and a user's identity ID_j. Then, $\mathcal{C}$ generates the corresponding trapdoor via the following steps.

- If $ID_j = ID_D$, $\mathcal{C}$ terminates game and returns failure messages. Otherwise, $\mathcal{C}$ performs the following steps. (We use EV_2 to represent this event.)
- Search (ID_i, Q_{ID_i}, h_i) and (ID_j, Q_{ID_j}, h_j) from the list $H - list$.
- Search $(ID_i, PK_{ID_i}, PSK_{ID_i}, x_{ID_i}))$ and $(ID_j, PK_{ID_j}, PSK_{ID_j}, x_{ID_j}))$ from the list $L_{user} - list$.
- Select $r' \in Z_p{}^*$ and let $r' = \beta/sh_j$.
- Compute trapdoors $T_1 = \prod_{\tau=1}^{l} g^{h1_i r'} g^{r' x_{ID_j}} \cdot PSK_{ID_j}{}^{r'}$, $T_2 = P^{r'}$, and $T_3 = PK_{ID_i}{}^{r'}$.
- Send $T_{w'} = (T_1, T_2, T_3)$ to $\mathcal{A}_2$.

Challenge. Upon selecting a data owner's identity ID_O, a user's identity ID_U and two challenge keywords W_0 and W_1, $\mathcal{A}_2$ sends them to $\mathcal{C}$. Note that W_0 and W_1 have not been queried by $\mathcal{A}_2$ during the trapdoor query 1 phase. Then $\mathcal{C}$ responds to $\mathcal{A}_2$ via the following steps.

- If $ID_U \neq ID_D$, $\mathcal{C}$ terminates game and returns failure messages. (We use EV_3 to represent this event.)
- Otherwise, $\mathcal{C}$ select $b \in \{0, 1\}$ and $r \in Z_p{}^*$ randomly.
- Compute $C_1 = g^r$, $C_2 = PSK_{ID_O}{}^{1/x_{ID_O}}$, $C_3 = Q_{ID_O}/(Q_{ID_u}{}^r)$, and $E_i = (g^{h1_i} PK_{ID_U})^{r/x_{ID_O}}$.
- Send challenge ciphertext $C_F = (C_1, C_2, C_3, E_1, \ldots, E_m)$ to $\mathcal{C}$.

Trapdoor Query 2. $\mathcal{A}_2$ continues to request the trapdoor of w_j. The query process is the same as trapdoor query 1, but it is required that w_j satisfies $w_j \neq W_0$ and $w_j \neq W_1$.

Guess. $\mathcal{A}_2$ outputs b'. If $b'{=}b$, $\mathcal{A}_2$ wins the game. At this time, $\mathcal{C}$ calculates $R = \frac{T_j}{(\prod_{\tau=1}^{l} g^{h1_i r'} g^{r' x_{ID_j}}) PSK_{ID_j}{}^{r'}}$. If $T_j = \prod_{\tau=1}^{l} g^{h1_i r'} g^{r' x_{ID_j}} \cdot (g \cdot g^{\alpha})^{h_j s r'}$, then we can smoothly infer that $R = g^{\alpha\beta}$.

Next, We analyze the probability of the game is not terminated in the simulation process. It can be seen from the above game process that the game termination condition is the same as $Lemma 1$. Therefore, the probability that $\mathcal{C}$ terminates this game is $1/q_{CUS} \cdot (1 - 1/q_{CUS})^{q_T}$. Then we can know the probability that $\mathcal{C}$ successfully solve the DDH problem is at least $\xi' = \xi/q_{CUS} \cdot (1 - 1/q_{CUS})^{q_T}$. According to the DDH assumption, we know that ξ' is negligible, and the advantage ξ that $\mathcal{A}$ can break the proposed scheme is negligible.

VII. PERFORMANCE EVALUATION

In this section, we give a feature comparison between previous searchable encryption schemes and our scheme. Then, we implement the several related schemes, for providing a comparison of computational costs with the proposed scheme. Finally, we compare the communication cost of the proposed CL-SPKAE scheme with several related schemes.

The feature of our scheme are compared with the schemes [16], [37]–[40], [46]–[50], as shown in Table 1. Here, we use the symbol "✗" to indicate that the corresponding feature is not satisfied and "✓" indicates satisfied. The schemes [40] and [50] introduce blockchain into SSE to ensure the anti-tampering, integrity and traceability of encrypted indexes. The scheme [40] uses Boolean expressions to construct encrypted indexes and utilizes smart contracts in Ethereum to achieve fair payment. However, the schemes [40] and [50] are only applicable to symmetric cryptographic environment. The schemes [16], [37]–[39], [46], [47] and our scheme all use certificateless cryptosystem to encrypt keywords, which

TABLE 2. A comparison of the computational cost.

Schemes	KeyGen	EncInd	Trapdoor	Search
[16]	$4T_M + 2T_H$	$(m+4)T_M + 3mT_H + (2+m)T_D$	$(l+2)T_M + (l+2)T_H + 2T_A + lT_P$	$l(3T_M + 2T_H + 2T_A + 2T_P)$
[46]	$2T_M + 2T_H$	$(m+2)T_M + mT_H + T_P$	$T_M + lT_H + T_P$	lT_M
[47]	$T_M + T_H + 5T_E$	$3T_M + mT_H + (2m+6)T_E$	$(l+6)T_M + lT_H + (2l+7)T_E$	$2T_M + 4T_P$
[48]	$3T_E$	$mT_M + 3mT_H + (2m+2)T_E$	$3lT_H + (2l+1)T_E$	$T_M + T_E + 3T_P$
Ours	$2T_H + 4T_E$	$mT_M + mT_H + (2m+3)T_E$	$(l+1)T_M + lT_H + (2l+3)T_E$	$T_M + 3T_P$

solve the problems of certificate management and key escrow. The schemes [16], [37], [39], [46] and [49] only support single keyword ciphertext retrieval. The schemes [38] and [47], [48] allow encrypted files to be searched by multi-keyword, but it cannot resist inside keyword guessing attacks. The schemes [16], [37], [39], [46], [49] and our scheme solve this issue, but the scheme [49] exists key escrow problem. Furthermore, we use smart contracts to implement fair payments between data owners and users.

The computational cost of our scheme is compared with the schemes [46]–[48], and [16], as shown in Table 2. For ease of expression, let T_M, T_H, and T_A denote execution time of one multiplication operation, one hash operation, and one addition operation, respectively. Let T_E and T_P denote the execution time of one exponentiation operation and one bilinear pairing operation, respectively. Let m and l denote the number of encrypted keywords and search keywords, respectively. As can be seen from Table 2, compared with the scheme [47], our scheme requires lower computational cost. Moreover, the computational cost in our scheme is roughly the same as the scheme [48], but higher that the schemes [16] and [46]. However, with the increase of keywords, the computational cost in our scheme is lower than that of the schemes [16] and [46].

We implement simulation experiments on our scheme and the schemes [46]–[48], and [16] using the Pairing-Based Cryptography library, as shown in Fig. 3, Fig. 4, and Fig. 5. The simulation experiments are run on a desktop computer with the hardware environment of a 3.1GHz Intel Core i5-2400 processor, 4GB memory and 512GB hard disk space. The simulation program conducted with the software environment of a 64-bit Windows 10 operating system and cryptographic library PBC-0.4.7-VC.

We choose 1, 2, 5, 10, and 20 keywords to compare the computational cost of our scheme with the schemes [16], [46], [47], and [48], as shown in Fig. 3–5. Fig. 3 reflects the change of keyword encryption time with the number of keywords in the **EncInd** phase. The keyword encryption time for single keyword in the schemes [16], [46]–[48] is approximately 18.631ms, 25.778ms, 38.544ms, and 26.759ms respectively, while that in our scheme is approximately 22.158ms. When the number of keyword increases to 20, the schemes [16], [46]–[48] cost approximately 205.023ms, 189.071ms, 216.893ms, and 363.637ms respectively, while that in our scheme is approximately

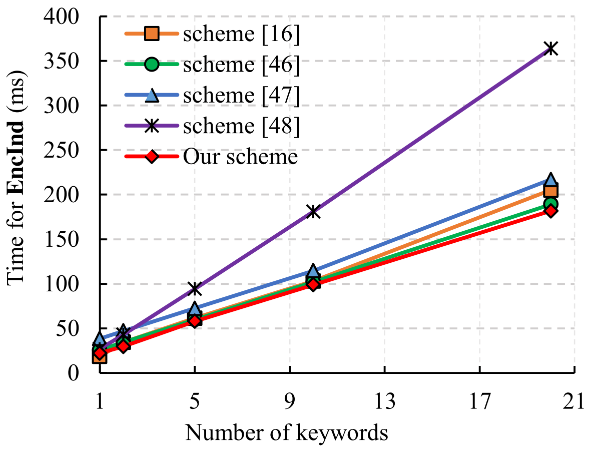

FIGURE 3. A computational cost comparison of EncInd algorithm.

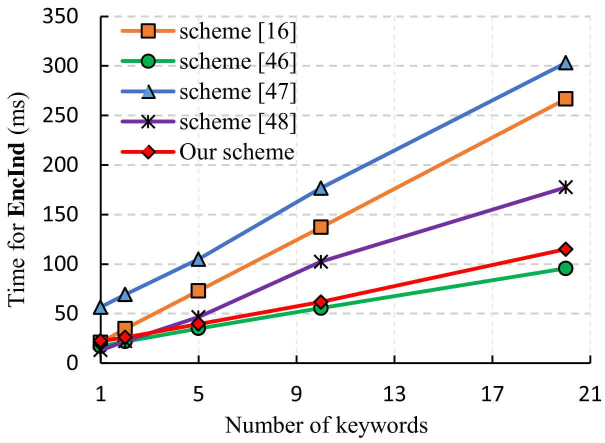

FIGURE 4. A computational cost comparison of Trapdoor algorithm.

181.762ms. We can see that our scheme always takes less time than the schemes [46], [47] and [48]. The keyword encryption time for single keyword in our scheme is more than that of the scheme [16], but our scheme takes less time than the scheme [16] when the number of keyword increases to 2. Fig. 4 reflects the change of trapdoor generation time with the number of keywords in the **Trapdoor** phase. The trapdoor generation time for single keyword in the schemes [16], [46]–[48] is approximately 20.808ms, 17.095ms, 56.517ms, and 13.044ms respectively, while that in our scheme is approximately 22.442ms. When the number

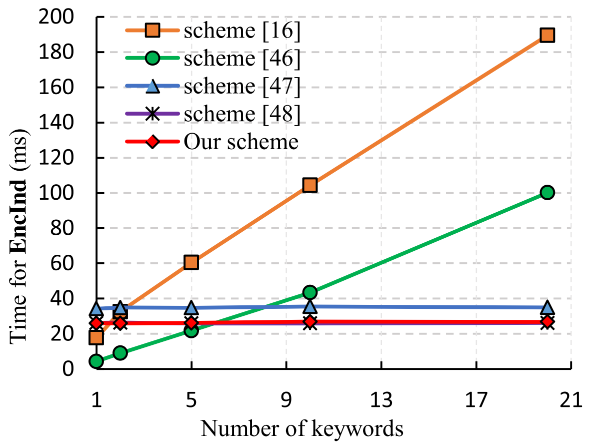

FIGURE 5. A computational cost comparison of **Search** algorithm.

TABLE 3. A comparison of the communication cost.

Schemes	PublicKey Size	**EncInd** Size	**Trapdoor** Size
[16]	$2\|G_1\|$	$(m+1)\|G_1\|$	$l\|G_2\|$
[46]	$\|G_1\|$	$(m+1)\|G_1\|$	$l\|Z_p{}^*\|$
[47]	$\|G_1\|$	$(m+4)\|G_1\|$	$5\|G_1\|$
[48]	$\|G_1\|$	$(m+2)\|G_1\|+m\|Z_p{}^*\|$	$3\|G_1\|+l\|Z_p{}^*\|$
Ours	$\|G_1\|$	$(m+3)\|G_1\|$	$3\|G_1\|$

of keyword increases to 20, the schemes [16], [46]–[48] cost approximately 266.673ms, 95.657ms, 303.111ms, and 177.519ms respectively, while that in our scheme is approximately 114.954ms. It can be seen that our scheme always takes more time than scheme [46], but takes less time than the scheme [47]. The trapdoor generation time for single keyword in our scheme is more than that of the schemes [16] and [48], but our scheme takes less time than the schemes [16] and [48] when the number of keyword increases to 3. Fig. 5 reflects the change of search time with the number of keywords in the **Search** phase. It is easy to see that the search time does not almost change with the the number of keywords in the schemes [47], [48] and our schemes, but the search time rises as the increase of keywords in the schemes [16], [46]. The search time for single keyword in the schemes [16], [46]–[48] is approximately 17.719ms, 4.34ms, 34.325ms, and 26.165ms respectively, while that in our scheme is approximately 25.995ms. When the number of keyword increases to 20, the schemes [16], [46]–[48] cost approximately 189.465ms, 100.113ms, 34.992ms, and 26.208ms respectively, while that in our scheme is approximately 26.866ms. It is clear that our scheme takes about the same time as scheme [48], but takes less time than the scheme [47]. The search time for single keyword in our scheme is more than that of the schemes [16] and [46], but our scheme takes less time than the scheme [16] and the scheme [46] when the number of keyword increases to 2 and 9, respectively.

The communication cost of our scheme is compared with the schemes [46]–[48], and [16] in terms of public key, encrypted index, and trapdoor, as shown in Table 3. For ease of expression, let $|Z_p{}^*|$, $|G_1|$, and $|G_2|$ represent the size of an element in $Z_p{}^*$, G_1, and G_2, respectively. As can be seen from Table 3, the communication cost for single keyword in our scheme is smaller than the scheme [47], but higher than the schemes [16], [46] and [48]. However, with the increase of keywords, the communication cost in our scheme is smaller than that of the schemes [16], [46], and [48].

VIII. CONCLUSION

In this paper, we propose a novel CL-SPKAE scheme based on blockchain. Our scheme achieves multi-keyword search over encrypted indexes. We use blockchain technology to ensure the anti-tampering, integrity and traceability of the encrypted indexes. Also, we utilize smart contract to realize fair and reliable transactions without any third party verification. In addition, we prove that our scheme can resist inside keyword guessing attacks in the random oracle model. Finally, we conduct simulation experiment and performance evaluation. The analysis results prove that our scheme has higher computational performances in the keyword encryption and search phases. However, our scheme cannot achieve dynamic updates over the encrypted files, which has certain limitations in practical applications. Therefore, a future interesting work is to construct a dynamic CL-SPKE scheme with forward and backward privacy.

REFERENCES

[1] G. Aceto, V. Persico, and A. Pescapé, "Industry 4.0 and health: Internet of Things, big data, and cloud computing for healthcare 4.0," *J. Ind. Inf. Integr.*, vol. 18, Jun. 2020, Art. no. 100129.

[2] H.-Y. Tran and J. Hu, "Privacy-preserving big data analytics a comprehensive survey," *J. Parallel Distrib. Comput.*, vol. 134, pp. 207–218, Dec. 2019.

[3] D. Alsmadi and V. Prybutok, "Sharing and storage behavior via cloud computing: Security and privacy in research and practice," *Comput. Hum. Behav.*, vol. 85, pp. 218–226, Aug. 2018.

[4] Y. Li and C. G. Ma, "Review of research progress on searchable encryption," *J. Netw. Inf. Secur.*, vol. 4, no. 7, pp. 13–21, 2018.

[5] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, vol. 8, May 2000, pp. 44–55.

[6] F. Xiang, C. Liu, B. Fang, C. Wang, and R. Zhong, "Research on ciphertext search for the cloud environment," *J. Commun.*, vol. 34, no. 7, pp. 143–153, 2013.

[7] X. L. Dong, J. Zhou, and Z. F. Cao, "Research progress of searchable encryption," *J. Comput. Res. Develop.*, vol. 54, no. 10, pp. 2107–2120, 2017.

[8] J. W. Li, C. F. Jia, Z. L. Liu, J. Li, and M. Li, "A review of searchable encryption technology research," *J. Softw.*, vol. 26, no. 1, pp. 109–128, 2015.

[9] M. Horvath and I. Vajda, "Searchable symmetric encryption: Sequential scan can be practical," in *Proc. SoftCOM*, Split, Croatia, 2017, pp. 1–5.

[10] P. Xu, S. He, W. Wang, W. Susilo, and H. Jin, "Lightweight searchable public-key encryption for cloud-assisted wireless sensor networks," *IEEE Trans. Ind. Informat.*, vol. 14, no. 8, pp. 3712–3723, Aug. 2018.

[11] Z. G., J. Xu, X. Y. Nie, and H. Xiong, "Overview of public key searchable encryption system," *J. Inf. Secur.*, vol. 2, no. 3, pp. 1–12, 2017.

[12] Y. Lu, J. Li, and Y. Zhang, "Privacy-preserving and pairing-free multirecipient certificateless encryption with keyword search for cloud-assisted IIoT," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 2553–2562, Apr. 2020.

[13] R. Chen, Y. Mu, G. Yang, F. Guo, and X. Wang, "Dual-server public-key encryption with keyword search for secure cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 4, pp. 789–798, Apr. 2016.

[14] Y. Lu, G. Wang, and J. Li, "Keyword guessing attacks on a public key encryption with keyword search scheme without random oracle and its improvement," *Inf. Sci.*, vol. 479, pp. 270–275, Apr. 2019.
[15] H. L. Xu and Y. Lu, "Efficient certificate-based encryption scheme with keyword search without bilinear pairings," *J. Comput. Appl.*, vol. 38, no. 2, pp. 379–385, 2018.
[16] D. He, M. Ma, S. Zeadally, N. Kumar, and K. Liang, "Certificateless public key authenticated encryption with keyword search for industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 14, no. 8, pp. 3618–3627, Aug. 2018.
[17] M. Ma, D. He, N. Kumar, K.-K.-R. Choo, and J. Chen, "Certificateless searchable public key encryption scheme for industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 14, no. 2, pp. 759–767, Feb. 2018.
[18] M. Ma, D. He, S. Fan, and D. Feng, "Certificateless searchable public key encryption scheme secure against keyword guessing attacks for smart healthcare," *J. Inf. Secur. Appl.*, vol. 50, pp. 102429–102437, Apr. 2020.
[19] N. Pakniat, "Designated tester certificateless encryption with keyword search," *J. Inf. Secur. Appl.*, vol. 49, Dec. 2019, Art. no. 102394.
[20] A. Hassan, Y. Wang, R. Elhabob, N. Eltayieb, and F. Li, "An efficient certificateless public key encryption scheme with authorized equality test in healthcare environments," *J. Syst. Archit.*, vol. 109, Oct. 2020, Art. no. 101776.
[21] W. Sun, X. Liu, W. Lou, Y. T. Hou, and H. Li, "Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data," in *Proc. IEEE INFOCOM*, Apr. 2015, pp. 2110–2118.
[22] X. Chen, J. Li, J. Weng, J. Ma, and W. Lou, "Verifiable computation over large database with incremental updates," *IEEE Trans. Comput.*, vol. 65, no. 10, pp. 3184–3195, Oct. 2016.
[23] Y. Zhang, X. Liu, X. Lang, Y. Zhang, and C. Wang, "VCLPKES: Verifiable certificateless public key searchable encryption scheme for industrial Internet of Things," *IEEE Access*, vol. 8, pp. 20849–20861, 2020.
[24] M. A. Khan and K. Salah, "IoT security: Review, blockchain solutions, and open challenges," *Future Gener. Comput. Syst.*, vol. 82, pp. 395–411, May 2018.
[25] G. Yu, X. Wang, K. Yu, W. Ni, J. A. Zhang, and R. P. Liu, "Survey: Sharding in blockchains," *IEEE Access*, vol. 8, pp. 14155–14181, 2020.
[26] 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.
[27] H. Li, H. Tian, F. Zhang, and J. He, "Blockchain-based searchable symmetric encryption scheme," *Comput. Electr. Eng.*, vol. 73, pp. 32–45, Jan. 2019.
[28] T.-T. Kuo, J. Kim, and R. A. Gabriel, "Privacy-preserving model learning on a blockchain network-of-networks," *J. Amer. Med. Inform. Assoc.*, vol. 27, no. 3, pp. 343–354, Mar. 2020.
[29] B. Zhu, "Symmetric encryption algorithm based on pairing function," *J. Nanjing Univ. Sci. Technol.*, vol. 6, pp. 696–699, 2003.
[30] A. J. Elbirt and C. Paar, "Instruction-level distributed processing for symmetric-key cryptography," in *Proc. Parallel Distrib. Process. Symp.*, 2003, vol. 2, nos. 78–87.
[31] P. A. Fouque, G. Martinet, and G. Poupard, "Practical symmetric on-line encryption," in *Proc. 10th Int. Workshop Fast Softw. Encryption*, 2003.
[32] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. EUROCRTPR*, vol. 3027. Berlin, Germany: Springer, 2004, pp. 506–522.
[33] J. Beak, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Proc. Int. Conf. Comput. Sci. Appl.* Berlin, Germany: Springer, 2008, pp. 1249–1259.
[34] Y. Guo, M. Jiang, and W. Song, "Certificate-based encryption scheme for provably secure elastic leaks," *J. Huaibei Normal Univ.*, vol. 40, no. 1, pp. 19–25, 2019.
[35] M. Zhu, Y. Chen, and Y. Hu, "Identity-based searchable encryption scheme supporting proxy re-encryption," *Comput. Eng.*, vol. 45, no. 1, pp. 129–135, 2019.
[36] T. Y. Wu, C. M. Chen, K. H. Wang, C. Meng, and E. K. Wang, "A provably secure certificateless public key encryption with keyword search," *J. Chin. Inst. Eng.*, vol. 42, no. 1, pp. 20–28, 2019.
[37] Y. Lu and J. Li, "Constructing certificateless encryption with keyword search against outside and inside keyword guessing attacks," *China Commun.*, vol. 16, no. 7, pp. 156–173, Jul. 2019.
[38] E. Uwizeye, J. Wang, Z. Cheng, and F. Li, "Certificateless public key encryption with conjunctive keyword search and its application to cloud-based reliable smart grid system," *Ann. Telecommun.*, vol. 74, nos. 7–8, pp. 435–449, Aug. 2019.
[39] L. Wu, Y. Zhang, M. Ma, N. Kumar, and D. He, "Certificateless searchable public key authenticated encryption with designated tester for cloud-assisted medical Internet of Things," *Ann. Telecommun.*, vol. 74, nos. 7–8, pp. 423–434, Aug. 2019.
[40] 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.
[41] 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. INFOCOM IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 792–800.
[42] B. Qin, Y. Chen, Q. Huang, X. Liu, and D. Zheng, "Public-key authenticated encryption with keyword search revisited: Security model and constructions," *Inf. Sci.*, vol. 516, pp. 515–528, Apr. 2020.
[43] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, pp. 1–21, 1997.
[44] A. Ma, X. Pan, L. Wu, J. F. Guo, and Q. W. Huang, "Review of blockchain technology basis and application," *J. Inf. Secur. Res.*, vol. 3, no. 11, pp. 968–983, 2017.
[45] H. Ou-Yang, Y. Xiong, and W. Huang, "A formal modeling method of token smart contract," *Comput. Eng.*, early access, 2020, doi: 10.19678/j.issn.1000-3428.0056812.
[46] N. Pakniat, D. Shiraly, and Z. Eslami, "Certificateless authenticated encryption with keyword search: Enhanced security model and a concrete construction for industrial IoT," *J. Inf. Secur. Appl.*, vol. 53, pp. 102525–102534, Aug. 2020.
[47] Q. Wu, J. Ma, H. Li, and Y. Miao, "Certificateless conjunctive keyword search over encrypted data," *J. Xidian Univ.*, vol. 44, no. 3, pp. 55–60, 2017.
[48] L. Xu, J. Li, X. Chen, W. Li, S. Tang, and H.-T. Wu, "TC-PEDCKS: Towards time controlled public key encryption with delegatable conjunctive keyword search for Internet of Things," *J. Netw. Comput. Appl.*, vol. 128, pp. 11–20, Feb. 2019.
[49] H. Li, Q. Huang, J. Shen, G. Yang, and W. Susilo, "Designated-server identity-based authenticated encryption with keyword search for encrypted emails," *Inf. Sci.*, vol. 481, pp. 330–343, May 2019.
[50] H. Li, F. Zhang, J. He, and H. Tian, "A searchable symmetric encryption scheme using BlockChain," 2017, *arXiv:1711.01030*. [Online]. Available: http://arxiv.org/abs/1711.01030

XIAODONG YANG (Member, IEEE) received the B.S. degree in mathematics from Northwest Normal University and the M.S. degree in cryptography from Tongji University, China, in 2002 and 2005, respectively, and the Ph.D. degree in cryptography from Northwest Normal University, in 2010. He is currently a Postdoctoral Fellow with the State Key Laboratory of Cryptology of China and a Professor in information and computer science with Northwest Normal University. His research interests include applied cryptography, network security, and cloud computing security. He is also a member of the Chinese Cryptology and Information Security Association.

GUILAN CHEN received the B.S. degree from Northwest Normal University, Lanzhou, China, in 2018, where she is currently pursuing the M.S. degree in computer science. Her current research interests include cloud computing security and searchable encryption.

MEIDING WANG received the B.S. degree from Northwest Normal University, Lanzhou, China, in 2018, where she is currently pursuing the master's degree in computer science. Her current research interests include cloud computing security and cloud audit.

TING LI received the B.S. degree from Zhengzhou Normal University, Zhengzhou, China, in 2018. She is currently pursuing the master's degree in computer science with Northwest Normal University. Her current research interests include network security, blockchain technology, and their applications.

CAIFEN WANG received the Ph.D. degree in cryptography from Xidian University, Xi'an, China, in 2003. She is currently a Professor in computer science with Shenzhen Technology University. Her current research interests include network security, cryptographic protocols, and security engineering. She is also a member of the Chinese Cryptology and Information Security Association.